Independence and Concurrent Separation Logic

Jonathan Hayman and Glynn Winskel
Computer Laboratory, University of Cambridge

Abstract

A compositional Petri net based semantics is given to a simple pointer-manipulating language. The model is then applied to give a notion of validity to the judgements made by concurrent separation logic that emphasizes the process-environment duality inherent in such rely-guarantee reasoning. Soundness of the rules of concurrent separation logic with respect to this definition of validity is shown. The independence information retained by the Petri net model is then exploited to characterize the independence of parallel processes enforced by the logic. This is shown to permit a refinement operation capable of changing the granularity of atomic actions.

1. Introduction

The foundational work of Hoare on parallel programming [9] identified the fact that attaching an interleaved semantics to parallel languages is problematic. Three areas of difficulty were isolated, quoted directly:

- *That of defining a "unit of action".*
- *That of implementing the interleaving on genuinely parallel hardware.*
- *That of designing programs to control the fantastic number of combinations involved in arbitrary interleaving.*

The significance of these problems increases with developments in hardware, such as multiple-core processors, that allow primitive machine actions to occur at the same time.

As Hoare went on to explain, a feature of concurrent systems in the real world is that they are often *spatially separated*, operating on completely different resources and not interacting. When this is so, the parallel processes are *independent* of each other. For instance, computer processes are spatially separated if they operate on different memory locations. The problems above are intuitively resolved if the occurrence of non-independent parallel actions is prohibited except in rare cases where we can assume atomicity, as might be enforced using the constructs proposed in [8, 3].

Independence models for concurrency allow the problems associated with an interleaved semantics to be resolved by recording when actions are independent. Independent actions can be run in either order or even concurrently with no consequence on their effect. This mitigates the increase in the state space since unnecessary interleavings of independent actions need not be considered (see *e.g.* [6] for applications to model checking). Independence models also permit easier notions of refinement which allow us to change the assumed atomicity of actions.

It is surprising that, to our knowledge, there has been no comprehensive study of the semantics of programming languages inside an independence model. The first component of this work gives such a semantics in terms of a well-known independence model, namely Petri nets. Our model isolates the specification of the control flow of programs from their effect on the shared environment.

The language that we consider is motivated by the emergence of *concurrent separation logic* [14], the rules of which form a partial correctness judgement about the execution of pointer-manipulating concurrent programs. Reasoning about such programs has traditionally proven difficult due to the problem of *variable aliasing*. For instance, Owicki and Gries' system for proving properties of parallel programs [16] essentially requires that they operate on disjoint collections of variables, thereby allowing judgements to be composed. In the presence of pointers, the same condition cannot be imposed just from the syntax of terms since distinct variables may point to the same memory location, thereby allowing arbitrary interaction between the processes.

At the core of separation logic [18, 10], initially presented for non-concurrent programs, is the *separating conjunction*, $\varphi * \psi$, which asserts that the memory may be split into two parts, one part satisfying φ and the other ψ. The separating conjunction was used by O'Hearn to adapt Owicki and Gries' system to provide a rule for parallel composition suitable for pointer-manipulating programs [14].

As we shall see, the rule for parallel composition is informally understood by splitting the initial state into two parts, one *owned* by the first process and the other by the second. Ownership can be seen as a dynamic constraint on the interference to be assumed: parallel processes always own disjoint sets of locations and only ever act on locations

that they own. As processes evolve, ownership of locations may be transferred using a system of *invariants* (an example is presented in Section 4). A consequence of this notion of ownership is that the rules discriminate between the parallel composition of processes and their interleaved expansion. For example, the logic does *not* allow the judgement

$$\vdash \{\ell \mapsto 0\}\ [\ell] := 1 \parallel [\ell] := 1\ \{\ell \mapsto 1\},$$

which informally means that the effect of two processes acting in parallel which both assign the value 1 to the memory location ℓ from a state in which ℓ holds 0 is to yield a state in which ℓ holds 1. However, if we adopt the usual rule for the nondeterministic sum of processes, the corresponding judgement *is* derivable for their interleaved expansion,

$$([\ell] := 1; [\ell] := 1) + ([\ell] := 1; [\ell] := 1).$$

The rules of concurrent separation logic contain a good deal of subtlety, and so lacked a completely formal account until the pioneering proof of their soundness due to Brookes [4]. The proof that Brookes gives is based on a form of interleaved trace semantics. The presence of pointers within the model alongside the possibility that ownership of locations is transferred means, however, that the way in which processes are separated is absolutely non-trivial, which motivates strongly the study of the language within an independence model. We therefore give a proof of soundness using our net model and then characterize entirely semantically the independence of concurrent processes in Theorem 12.

The proof technique that we employ defines validity of assertions in a way that captures the rely-guarantee reasoning [11] emanating from *ownership* in separation logic directly, and in a way that might be applied in other situations.

In [19], Reynolds argues that separation allows store actions that were assumed to be atomic, in fact, to be implemented as composite actions (seen as a change in their *granularity*) with no effect on the validity of the judgement. Independence models are suited to modeling situations where actions are not atomic, as advocated by Lamport and Pratt [17, 13]. We introduce a novel form of refinement, inspired by that of [20], and apply this to address the issue of granularity using our characterization of independence.

2. Terms and states

We begin by defining the *terms* of our language.

$$\begin{aligned} t \ ::= \ & \alpha \mid t; t \mid t \parallel t \mid \alpha.t + \alpha.t \mid \texttt{while}\ b\ \texttt{do}\ t\ \texttt{od} \\ & \mid \texttt{resource}\ x\ \texttt{do}\ t\ \texttt{od} \\ & \mid \texttt{with}\ r\ \texttt{do}\ t\ \texttt{od} \mid \texttt{with}\ x\ \texttt{do}\ t\ \texttt{od} \\ & \mid \texttt{alloc}(\ell) \mid \texttt{dealloc}(\ell) \end{aligned}$$

We use α to represent primitive *heap actions* and use b to distinguish *boolean guards* that proceed only if b holds, having no effect on the heap, but otherwise blocking. The *guarded sum* $\alpha.t + \alpha'.t'$ is a process that executes as t if α takes place or as t' if α' takes place.

A heap is an assignment of values to *allocated*, or *current*, heap locations. We denote the set of heap locations Loc and use ℓ to range over its elements, and we denote the set of values Val and use v to range over its elements. The heap model allows locations to hold pointers to other locations, so we require that $\mathrm{Loc} \subseteq \mathrm{Val}$. There is no implicit restriction that only current locations may be pointed at, thereby allowing the model to cope with 'dangling' pointers. Locations become current through $\texttt{alloc}(\ell)$, which makes a location current and sets ℓ to point at this location. For symmetry, $\texttt{dealloc}(\ell)$ makes the location pointed to by ℓ non-current if ℓ points to a current location. A heap action α does not change which locations are current.

In addition to acting on the heap, we allow the language to declare new binary semaphores, drawn from a set of *resource names* Res, and use these to protect critical regions. A critical region protected by a resource r is represented by $\texttt{with}\ r\ \texttt{do}\ t\ \texttt{od}$, which executes t if no other process is inside a critical region protected by r. The $\texttt{resource}\ x\ \texttt{do}\ t\ \texttt{od}$ construct represents that a globally unused resource is to be chosen and then used in place of x in t. It is therefore necessary to record, in addition to which resources are *available*, which resources are *current*; we shall write $\mathsf{curr}(r)$ if r is current. We say that the declaration $\texttt{resource}\ x\ \texttt{do}\ t\ \texttt{od}$ *binds* the variable x within t, and that the variable x is *free* within $\texttt{with}\ x\ \texttt{do}\ t\ \texttt{od}$. We write $\mathrm{fv}(t)$ for the set of variables free within term t and $[r/x]t$ for the term obtained by substituting the name r for the variable x within t avoiding capture. As standard, we identify terms up to α-conversion. In addition, we say that the resource name r is free within $\texttt{with}\ r\ \texttt{do}\ t\ \texttt{od}$.

These two components form the shared state in which processes execute. Motivated by the net semantics that we shall give, we define the following sets:

$$\begin{array}{llll} \mathbf{D} \stackrel{\text{def}}{=} \mathrm{Loc} \times \mathrm{Val} & & \mathbf{L} \stackrel{\text{def}}{=} \{\mathsf{curr}(\ell) \mid \ell \in \mathrm{Loc}\} \\ \mathbf{R} \stackrel{\text{def}}{=} \mathrm{Res} & & \mathbf{N} \stackrel{\text{def}}{=} \{\mathsf{curr}(r) \mid r \in \mathrm{Res}\}. \end{array}$$

A state σ is defined to be a tuple

$$(D, L, R, N)$$

where the set $D \subseteq \mathbf{D}$, the set $L \subseteq \mathbf{L}$, the set $R \subseteq \mathbf{R}$ and the set $N \subseteq \mathbf{N}$. The sets $\mathbf{D}$, $\mathbf{L}$, $\mathbf{R}$ and $\mathbf{N}$ are disjoint, so no ambiguity arises from writing, for example, $(\ell, v) \in \sigma$.

The interpretation of a state for the heap is that $(\ell, v) \in D$ if ℓ holds value v and that $\mathsf{curr}(\ell) \in L$ if L is current. For resources, $r \in R$ if the resource r is available and $\mathsf{curr}(r) \in N$ if r is current. It is clear that only certain such tuples of subsets are sensible.

Definition 1 (Consistent state) *The state* (D, L, R, N) *is* consistent *if we have*

$$R \subseteq \{r \mid \mathsf{curr}(r) \in N\}, \quad L = \{\mathsf{curr}(\ell) \mid \exists v.(\ell, v) \in D\}$$

and for any ℓ, v *and* v', *if* $(\ell, v) \in D$ *and* $(\ell, v') \in D$ *then* $v = v'$.

3. Process models

We give both an operational and a net semantics to *closed* terms. The operational semantics is presented to aid understanding of the net model, and is given by means of labelled transition relations of the forms $\langle t, \sigma\rangle \xrightarrow{\lambda} \langle t', \sigma'\rangle$ and $\langle t, \sigma\rangle \xrightarrow{\lambda} \sigma'$. As usual, the first form of transition indicates that t performs an action labelled λ in state σ to yield a resumption t' and a state σ'. The second indicates that t in state σ performs an action labelled λ to terminate and yield a state σ'. Labels follow the grammar:

$$\lambda \quad ::= \quad \mathsf{act}(D_1, D_2) \mid \mathsf{decl}(r) \mid \mathsf{end}(r) \mid \mathsf{get}(r) \mid \mathsf{rel}(r) \mid \mathsf{alloc}(\ell, v, \ell', v') \mid \mathsf{dealloc}(\ell, \ell', v).$$

We assume that we are given the semantics of primitive actions in the following form:

$$\mathcal{A}[\![\alpha]\!] \in \mathcal{P}ow(\mathcal{P}ow(\mathbf{D}) \times \mathcal{P}ow(\mathbf{D}))$$

The interpretation is that α can proceed in heap D if there are $(D_1, D_2) \in \mathcal{A}[\![\alpha]\!]$ such that, whenever D_1 is defined, D has the same value. The resulting heap is formed by updating D to have the same value as D_2 wherever it is defined. It is significant that this definition allows us to infer precisely the set of locations upon which an action depends. In order to preserve consistent markings, we shall require that D_1 and D_2 are (the graphs of) partial functions with the same domain if $(D_1, D_2) \in \mathcal{A}[\![\alpha]\!]$.

An example action is copying the value held at one location to another:

$$\mathcal{A}[\![[\ell] := [\ell']]\!] \stackrel{\text{def}}{=} \{(\{(\ell, v), (\ell', v')\}, \{(\ell, v'), (\ell', v')\}) \mid v, v' \in \mathrm{Val}\}$$

Boolean guards b are actions that wait until the boolean expression holds and may then take place. For example,

$$\mathcal{A}[\![[\ell] = v]\!] \stackrel{\text{def}}{=} \{(\{(\ell, v)\}, \{(\ell, v)\})\}$$

gives the semantics of an action that proceeds only if ℓ holds value v. We omit the specification of boolean constructs such as conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$) and further forms of equality, though these are easily definable.

The operational semantics is presented in Figure 1 (we omit the symmetric rules). Notice that special items `rel` r and `end` r are attached to the ends of terms for critical regions and to the end of scope of a resource. We write $\sigma \oplus \sigma'$ for the union of the components of two states where they are disjoint, and impose the implicit side-condition that this is defined wherever it is used. For conciseness, we do not give an error semantics to situations in which non-current locations or resources are used (*e.g.* action on a non-current location); they shall be excluded by the logic.

3.1. Net structure

The particular variant of Petri net upon which we base our model is that where conditions are marked without multiplicity (*c.f.* the 'basic' nets of [7, 22], App. A).

Within the nets that we give for processes, we distinguish two forms of condition, namely *control conditions* and *state conditions*. The marking of these sets of conditions determines the control point of the process and the state in which it is executing, respectively. When we give the net semantics, we will make use of the closure of the set of control conditions under various operations.

Definition 2 (Conditions) *Define the set of control conditions* $\mathbf{C}$*, ranged over by c, to be the least set such that:*

- $\mathbf{C}$ *contains distinguished elements* $*$ *and* $\diamond$*,*
- *if* $c \in \mathbf{C}$ *then* $r{:}c \in \mathbf{C}$ *for all* $r \in \mathrm{Res}$ *and* $i{:}c \in \mathbf{C}$ *for all* $i \in \{1, 2\}$*, and*
- *if* $c, c' \in \mathbf{C}$ *then* $(c, c') \in \mathbf{C}$.

Define the set of state conditions $\mathbf{S}$ *to be* $\mathbf{D} \cup \mathbf{L} \cup \mathbf{R} \cup \mathbf{N}$.

A state $\sigma = (D, L, R, N)$ corresponds to the marking $D \cup L \cup R \cup N$ of state conditions in the obvious way, and we continue to restrict to markings corresponding to consistent states. Similarly, if C is a marking of control conditions, the pair (C, σ) corresponds to the marking $C \cup \sigma$. We therefore use the notations interchangeably.

The nets that we form are *extensional* in the sense that two events are identified if they have the same preconditions and the same postconditions. An event is therefore regarded as a tuple,

$$e = (C, \sigma, C', \sigma'),$$

and we write ${}^\bullet e$ for $C \cup \sigma$ and $e^\bullet$ for $C' \cup \sigma'$. The control conditions in C occur as preconditions of e, as do the state conditions in σ. Similarly, the set $C' \cup \sigma'$ forms the postconditions of e. To obtain a concise notation for working with events, we write ${}^{\mathbf{C}}e$ for ${}^\bullet e \cap \mathbf{C}$, which is the set of control conditions that occur as preconditions to e. We likewise define notations $e^{\mathbf{C}}$, ${}^{\mathbf{D}}e$, ${}^{\mathbf{L}}e$ *etc.*, and call these the *components* of e by virtue of the fact that it is sufficient to define an event through defining its components.

Two (disjoint) markings of control conditions are of particular importance: those marked when the process starts executing and those marked when the process has terminated. We call these the *initial* control conditions I and *terminal* control conditions T, respectively. We shall call a net with a partition of its conditions into control and state with the subsets of control conditions I and T an *embedded net*. For an embedded net N, we write $\mathrm{Ic}(N)$ for I and $\mathrm{Tc}(N)$ for T, and we write $\mathrm{Ev}(N)$ for its set of events. Observe that no initial marking of state conditions is specified. Write $N : (C, \sigma) \xrightarrow{e} (C', \sigma')$ if the token game for nets allows the marking $C \cup \sigma$ to proceed to $C' \cup \sigma'$ by event e.

The semantics of terms shall be an embedded net, written $\mathcal{N}[\![t]\!]$. No confusion arises, so we shall write $\mathrm{Ic}(t)$ for $\mathrm{Ic}(\mathcal{N}[\![t]\!])$, and $\mathrm{Tc}(t)$ and $\mathrm{Ev}(t)$ for $\mathrm{Tc}(\mathcal{N}[\![t]\!])$ and $\mathrm{Ev}(\mathcal{N}[\![t]\!])$, respectively. The nets that we form shall always have the same sets of control and state conditions, though it

$$\frac{(D_1, D_2) \in \mathcal{A}[\![\alpha]\!] \quad D_1 \subseteq D \quad D' = (D \setminus D_1) \cup D_2}{\langle \alpha, (D, L, R, N)\rangle \xrightarrow{\mathsf{act}(D_1,D_2)} (D', L, R, N)} \qquad \frac{\langle b, \sigma\rangle \xrightarrow{\lambda} \sigma}{\langle \texttt{while } b \texttt{ do } p \texttt{ od}, \sigma\rangle \xrightarrow{\lambda} \langle p; \texttt{while } b \texttt{ do } p \texttt{ od}, \sigma\rangle} \qquad \frac{\langle \neg b, \sigma\rangle \xrightarrow{\lambda} \sigma}{\langle \texttt{while } b \texttt{ do } p \texttt{ od}, \sigma\rangle \xrightarrow{\lambda} \sigma}$$

$$\frac{\langle t_1, \sigma\rangle \xrightarrow{\lambda} \langle t_1', \sigma'\rangle}{\langle t_1 \parallel t_2, \sigma\rangle \xrightarrow{\lambda} \langle t_1' \parallel t_2, \sigma'\rangle} \qquad \frac{\langle t_1, \sigma\rangle \xrightarrow{\lambda} \sigma'}{\langle t_1 \parallel t_2, \sigma\rangle \xrightarrow{\lambda} \langle t_2, \sigma'\rangle} \qquad \begin{array}{l}\langle \texttt{with } r \texttt{ do } t \texttt{ od}, \sigma \oplus \{r\}\rangle \xrightarrow{\mathsf{get}(r)} \langle t; \texttt{rel } r, \sigma\rangle \\ \langle \texttt{rel } r, \sigma\rangle \xrightarrow{\mathsf{rel}(r)} \sigma \oplus \{r\}\end{array}$$

$$\frac{\langle t_1, \sigma\rangle \xrightarrow{\lambda} \langle t_1', \sigma'\rangle}{\langle t_1; t_2, \sigma\rangle \xrightarrow{\lambda} \langle t_1'; t_2, \sigma'\rangle} \qquad \frac{\langle t_1, \sigma\rangle \xrightarrow{\lambda} \sigma'}{\langle t_1; t_2, \sigma\rangle \xrightarrow{\lambda} \langle t_2, \sigma'\rangle} \qquad \begin{array}{l}\langle \texttt{resource } x \texttt{ do } t \texttt{ od}, \sigma\rangle \xrightarrow{\mathsf{decl}(r)} \langle [r/x]t; \texttt{end } r, \sigma \oplus \{r, \mathsf{curr}(r)\}\rangle \\ \langle \texttt{end } r, \sigma \oplus \{r, \mathsf{curr}(r)\}\rangle \xrightarrow{\mathsf{end}(r)} \sigma\end{array}$$

$$\frac{\langle \alpha_1, \sigma\rangle \xrightarrow{\lambda} \sigma'}{\langle \alpha_1.t_1 + \alpha_2.t_2, \sigma\rangle \xrightarrow{\lambda} \langle t_1, \sigma'\rangle} \qquad \begin{array}{l}\langle \texttt{alloc}(\ell), \sigma \oplus \{(\ell, v)\}\rangle \xrightarrow{\mathsf{alloc}(\ell,v,\ell',v')} \sigma \oplus \{(\ell, \ell'), (\ell', v'), \mathsf{curr}(\ell')\} \\ \langle \texttt{dealloc}(\ell), \sigma \oplus \{(\ell, \ell'), (\ell', v'), \mathsf{curr}(\ell')\}\rangle \xrightarrow{\mathsf{dealloc}(\ell,\ell',v')} \sigma \oplus \{(\ell, \ell')\}\end{array}$$

Figure 1. Operational semantics

is a trivial matter to restrict to those that are actually used.

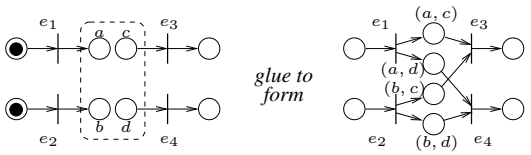
As we give the semantics of terms, we make use of some constructions on nets. For example, we wish the events of parallel processes to operate on disjoint sets of control conditions. This is conducted using a *tagging* operation on events. We define $1{:}e$ to be the event e changed so that

$$^{\mathbf{C}}(1{:}e) \stackrel{\text{def}}{=} \{1{:}c \mid c \in {}^{\mathbf{C}}e\} \qquad (1{:}e)^{\mathbf{C}} \stackrel{\text{def}}{=} \{1{:}c \mid c \in e^{\mathbf{C}}\}$$

but otherwise unchanged in its action on state conditions. We define the notations $2{:}e$ and $r{:}e$ where $r \in \mathrm{Res}$ similarly, and extend the notations pointwise to sets of events.

Another useful operation is what we call *gluing* two embedded nets together. For example, when forming the sequential composition of processes $t_1; t_2$, we want to enable the events of t_2 when t_1 has terminated. This is done by 'gluing' the two nets together, having made them disjoint on control conditions, at the terminal conditions of t_1 and the initial conditions of t_2. Therefore, rather than using a terminal condition c of $\mathrm{Tc}(t_1)$, the events of t_1 use the set of conditions $\{1{:}c\} \times (2{:}\mathrm{Ic}(t_2))$. Similarly, the events of t_2 use the set of conditions $(1{:}\mathrm{Tc}(t_1)) \times \{2{:}c'\}$ instead of an initial condition c' of $\mathrm{Ic}(t_2)$.

An example of gluing follows, indicating how gluing is used to sequentially compose events.



A variety of control properties that the nets we form possess, such as that all events have at least one pre-control condition, allow us to infer that it is impossible for an event of t_2 to occur before t_1 has terminated, and thereon it is impossible for t_1 to resume.

Assume a set $P \subseteq \mathbf{C} \times \mathbf{C}$. Useful definitions to represent gluing are:

$$\begin{array}{rcl} P \triangleleft C & \stackrel{\text{def}}{=} & \{(c_1, c_2) \mid c_1 \in C \text{ and } (c_1, c_2) \in P\} \\ & & \cup\, \{c_1 \mid c_1 \in C \text{ and } \nexists c_2.(c_1, c_2) \in P\} \end{array}$$

$$\begin{array}{rcl} P \triangleright C & \stackrel{\text{def}}{=} & \{(c_1, c_2) \mid c_2 \in C \text{ and } (c_1, c_2) \in P\} \\ & & \cup\, \{c_2 \mid c_2 \in C \text{ and } \nexists c_1.(c_1, c_2) \in P\} \end{array}$$

Extend the notation to events so that $^{\mathbf{C}}(P \triangleleft e) \stackrel{\text{def}}{=} P \triangleleft (^{\mathbf{C}}e)$ and $(P \triangleleft e)^{\mathbf{C}} \stackrel{\text{def}}{=} P \triangleleft (e^{\mathbf{C}})$, and similarly for $P \triangleright e$, and extend this to sets of events in the obvious manner. Observe that the operations of gluing and tagging affect only the control flow of events, not their effect on the marking of state conditions.

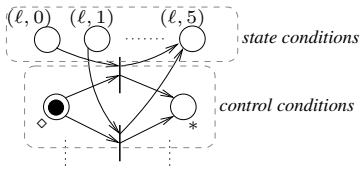
3.2. Net semantics

The net semantics that we now give for closed terms is defined by induction on their size, given in the obvious way.

Action Let $\mathsf{act}_{(C,C')}(D_1, D_2)$ denote an event e with

$$^{\mathbf{C}}e = C \quad e^{\mathbf{C}} = C' \quad ^{\mathbf{D}}e = D_1 \quad e^{\mathbf{D}} = D_2$$

and all other components empty. For an action α, we define $\mathrm{Ic}(\alpha) = \{\diamond\}$ and $\mathrm{Tc}(\alpha) = \{*\}$. The set of events $\mathrm{Ev}(\alpha)$ is the least set such that if $(D_1, D_2) \in \mathcal{A}[\![\alpha]\!]$ then it contains $\mathsf{act}_{(\{\diamond\},\{*\})}(D_1, D_2)$. The following diagram shows a net for assigning value 5 to ℓ.



Sequential composition The sequential composition of terms involves gluing the terminal marking of the net for t_1 to the initial marking of the net for t_2. The operation is therefore performed on the set

$$P = 1{:}\mathrm{Tc}(t_1) \times 2{:}\mathrm{Ic}(t_2).$$

Following the intuition above, we take

$$\mathrm{Ic}(t_1; t_2) \stackrel{\text{def}}{=} 1{:}\mathrm{Ic}(t_1) \qquad \mathrm{Tc}(t_1; t_2) \stackrel{\text{def}}{=} 2{:}\mathrm{Tc}(t_2)$$
$$\mathrm{Ev}(t_1; t_2) \stackrel{\text{def}}{=} (P \triangleleft 1{:}\mathrm{Ev}(t_1)) \cup (P \triangleright 2{:}\mathrm{Ev}(t_2)).$$

Parallel composition The control flow of the parallel composition of processes is autonomous; interaction occurs only through the state. We therefore force the events of the

two processes to work on disjoint sets of control conditions:

$$\begin{aligned}\mathrm{Ev}(t_1 \parallel t_2) &\stackrel{\text{def}}{=} 1{:}\mathrm{Ev}(t_1) \cup 2{:}\mathrm{Ev}(t_2)\\ \mathrm{Ic}(t_1 \parallel t_2) &\stackrel{\text{def}}{=} 1{:}\mathrm{Ic}(t_1) \cup 2{:}\mathrm{Ic}(t_2)\\ \mathrm{Tc}(t_1 \parallel t_2) &\stackrel{\text{def}}{=} 1{:}\mathrm{Tc}(t_1) \cup 2{:}\mathrm{Tc}(t_2)\end{aligned}$$

Guarded sum Let t be the term $\alpha_1.t_1 + \alpha_2.t_2$. The sum is formed by prefixing the actions and then gluing the sets of terminal conditions. Let $P = (1{:}\mathrm{Tc}(t_1)) \times (2{:}\mathrm{Tc}(t_2))$. Define the initial and terminal conditions

$$\mathrm{Ic}(t) \stackrel{\text{def}}{=} \{*\} \qquad \mathrm{Tc}(t) \stackrel{\text{def}}{=} P$$

and the events $\mathrm{Ev}(t)$ to be

$$\begin{aligned}&\{\mathsf{act}_{(\{\diamond\},1:\mathrm{Ic}(t_1))}(D_1, D_2) \mid (D_1, D_2) \in \mathcal{A}[\![\alpha_1]\!]\}\\ \cup\ &\{\mathsf{act}_{(\{\diamond\},2:\mathrm{Ic}(t_2))}(D_1, D_2) \mid (D_1, D_2) \in \mathcal{A}[\![\alpha_2]\!]\}\\ \cup\ &P \triangleleft (1{:}\mathrm{Ev}(t_1)) \ \cup\ P \triangleright (2{:}\mathrm{Ev}(t_2)).\end{aligned}$$

Iteration Intuitively, we glue the initial and terminal conditions of $b.t$ together and then add events to exit the loop when $\neg b$ holds. Let $P = \{\diamond\} \times 1{:}\mathrm{Tc}(t)$. Define:

$$\mathrm{Ic}(\texttt{while}\ b\ \texttt{do}\ t\ \texttt{od}) \stackrel{\text{def}}{=} P \quad \mathrm{Tc}(\texttt{while}\ b\ \texttt{do}\ t\ \texttt{od}) \stackrel{\text{def}}{=} \{*\}.$$

The set of events $\mathrm{Ev}(\texttt{while}\ b\ \texttt{do}\ t\ \texttt{od})$ is defined to be

$$\begin{aligned}&\{\mathsf{act}_{(P,1:\mathrm{Ic}(t))}(D_{\mathsf{t}}, D_{\mathsf{t}}) \mid (D_{\mathsf{t}}, D_{\mathsf{t}}) \in \mathcal{A}[\![b]\!]\}\\ \cup\ &\{\mathsf{act}_{(P,\{*\})}(D_{\mathsf{f}}, D_{\mathsf{f}}) \mid (D_{\mathsf{f}}, D_{\mathsf{f}}) \in \mathcal{A}[\![\neg b]\!]\}\\ \cup\ &P \triangleright (1{:}\mathrm{Ev}(t)).\end{aligned}$$

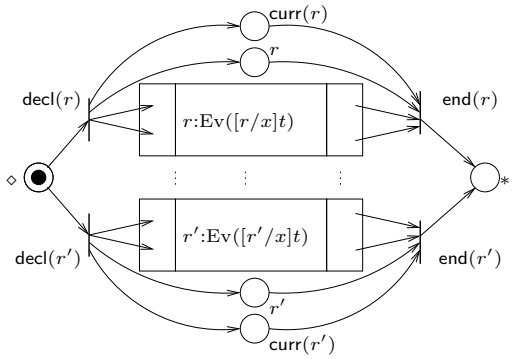
Semaphores and critical regions We introduce the following notations for resource events. These all have ${}^{\mathbf{C}}e = C$ and $e^{\mathbf{C}} = C'$, and the components other than those listed are empty. Observe that the event $\mathsf{decl}_{(C,C')}(r)$ will avoid contact, and thus be able to occur, only if the resource r is non-current.

$$\begin{aligned}\mathsf{decl}_{(C,C')}(r)&: \quad e^{\mathbf{R}} = \{r\} \text{ and } e^{\mathbf{N}} = \{\mathsf{curr}(r)\}\\ \mathsf{end}_{(C,C')}(r)&: \quad {}^{\mathbf{R}}e = \{r\} \text{ and } {}^{\mathbf{N}}e = \{\mathsf{curr}(r)\}\\ \mathsf{get}_{(C,C')}(r)&: \quad {}^{\mathbf{R}}e = \{r\}\\ \mathsf{rel}_{(C,C')}(r)&: \quad e^{\mathbf{R}} = \{r\}\end{aligned}$$

The initial conditions of the nets representing both constructs are $\{\diamond\}$, and their terminal conditions are $\{*\}$.

First consider $\texttt{resource}\ x\ \texttt{do}\ t\ \texttt{od}$. Its events form the least set containing, for each $r \in \mathrm{Res}$, where $t' = [r/x]t$:

$$\mathsf{decl}_{(\{\diamond\},\{r\,:\,\mathrm{Ic}(t')\})}(r) \cup r{:}\mathrm{Ev}(t') \cup \mathsf{end}_{(r\,:\,\mathrm{Tc}(t'),\{*\})}(r).$$



Now consider the closed term $\texttt{with}\ r\ \texttt{do}\ t\ \texttt{od}$. Its events are precisely

$$\{\mathsf{get}_{(\{\diamond\},1:\mathrm{Ic}(t))}(r)\} \cup 1{:}\mathrm{Ev}(t) \cup \{\mathsf{get}_{(1:\mathrm{Tc}(t),\{*\})}(r)\}.$$

Allocation and deallocation The command $\texttt{alloc}(\ell)$ activates, by making current and assigning an arbitrary value, a non-current location and makes ℓ point at it. For symmetry, $\texttt{dealloc}(\ell)$ deactivates the current location pointed to by ℓ.

We begin by defining two further event notations, which both have ${}^{\mathbf{C}}e = C$ and $e^{\mathbf{C}} = C'$ and empty components except those stated. First, $\mathsf{alloc}_{(C,C')}(\ell, v, \ell', v')$ is the event e such that

$${}^{\mathbf{D}}e = \{(\ell, v)\} \quad e^{\mathbf{D}} = \{(\ell, \ell'), (\ell', v')\} \quad e^{\mathbf{L}} = \{\mathsf{curr}(\ell')\},$$

which changes ℓ' from being non-current to current, gives it value v' and changes the value of ℓ from v to ℓ'. The event has concession only if the location ℓ' is not current. Second, $\mathsf{dealloc}_{(C,C')}(\ell, \ell', v')$ is the event e such that

$${}^{\mathbf{D}}e = \{(\ell, \ell'), (\ell', v')\} \quad e^{\mathbf{D}} = \{(\ell, \ell')\} \quad {}^{\mathbf{L}}e = \{\mathsf{curr}(\ell')\},$$

which does the converse of allocation.

The initial conditions of both $\texttt{alloc}(\ell)$ and $\texttt{dealloc}(\ell)$ are $\{\diamond\}$ and their terminal conditions are $\{*\}$. The events for $\texttt{alloc}(\ell)$ form the least set containing, for all $\ell' \in \mathrm{Loc}$ and $v, v' \in \mathrm{Val}$, $\mathsf{alloc}_{(\{\diamond\},\{*\})}(\ell, v, \ell', v')$. Similarly, the events for $\texttt{dealloc}(\ell)$ form the least set containing, for all $\ell' \in \mathrm{Loc}$ and $v, v' \in \mathrm{Val}$, $\mathsf{dealloc}_{(\{\diamond\},\{*\})}(\ell, \ell', v')$.

A well-known property of independence models is that they support a form of *run* in which independent actions are not interleaved: Given any sequential run of events of the net between two markings, we can swap the consecutive occurrences of any two independent events to yield a run between the same two markings. As seen in for example [22], this allows us to form an equivalence class of runs between the same markings, generating a Mazurkiewicz trace. This yields a partially ordered multiset, or *pomset*, run [17], in which the independence of event occurrences is captured through them being incomparable.

As we have progressed, the event notations introduced have corresponded to labels of the operational semantics. Write $|e|$ for the label corresponding to event e. The following theorem shows how the net and operational semantics correspond. It assumes a definition of open map bisimulation [12] based on paths as pomsets, $(N, M) \sim (N', M')$, relating paths of net N from marking M to paths of N' from M'. The bisimulations that we form respect terminal markings and markings of state conditions.

Theorem 3 *Let t be a closed term and σ be a consistent state.*

- *If $\langle t, \sigma\rangle \xrightarrow{\lambda} \sigma'$ then there exists e such that $|e| = \lambda$ and $\mathcal{N}[\![t]\!] : (\mathrm{Ic}(t), \sigma) \xrightarrow{e} (\mathrm{Tc}(t), \sigma')$.*
- *If $\langle t, \sigma\rangle \xrightarrow{\lambda} \langle t', \sigma'\rangle$ then there exists e such that $|e| = \lambda$*

and $\mathcal{N}[\![t]\!] : (\mathrm{Ic}(t), \sigma) \xrightarrow{e}\!\!\!\!\rightarrow (C', \sigma')$ *and* $(\mathcal{N}[\![t]\!], C', \sigma') \sim (\mathcal{N}[\![t']\!], \mathrm{Ic}(t'), \sigma')$.

- *If* $\mathcal{N}[\![t]\!] : (\mathrm{Ic}(t), \sigma) \xrightarrow{e}\!\!\!\!\rightarrow (C', \sigma')$ *then* $\langle t, \sigma\rangle \xrightarrow{|e|} \langle t', \sigma'\rangle$ *and* $(\mathcal{N}[\![t]\!], C', \sigma') \sim (\mathcal{N}[\![t']\!], \mathrm{Ic}(t'), \sigma')$*, or* $\langle t, \sigma\rangle \xrightarrow{|e|} \sigma'$ *and* $C' = \mathrm{Tc}(t)$.

4. Separation logic

We begin by presenting the intuition for the key judgement of concurrent separation logic, $\Gamma \vdash \{\varphi\}\ t\ \{\psi\}$:

> *If initially* φ *holds of the heap defined at the locations* owned *by the process, then, after* t *runs to completion,* ψ *holds of the heap defined at the locations owned by the process; during any such run, the process only accesses locations that it owns and preserves* invariants *in* Γ.

The fundamental rules of concurrent separation logic are presented in Figure 2; the remainder are as presented in [4, App. C]. We refer the reader to [14] for a full introduction.

Notice just from the syntax of terms that the rule for `resource` x `do` t `od` forces us to work on terms with free variables and that it inserts the variable x into the domain of Γ. We assume that the domain of Γ covers all the free variables of t, and therefore work with respect to an assignment ρ taking variables in the domain of Γ injectively to resources. We denote the net so-formed $\mathcal{N}[\![t]\!]_\rho$. To ensure that no non-current resources are accessed, we require that t has no free resource names.

The rules of separation logic are founded on the *heap logic*, the semantics of which arises as an instance of the classical 'Kripke resource monoid' semantics of Logic of Bunched Implications [15]. At the core of the heap logic are the associated notions of heap composition and the separating conjunction. Two heaps, D_1 and D_2, may be composed if they are defined over disjoint sets of locations:

$$D_1 \cdot D_2 \stackrel{\text{def}}{=} D_1 \cup D_2 \text{ if } \mathrm{dom}(D_1) \cap \mathrm{dom}(D_2) = \emptyset.$$

A heap satisfies the separating conjunction $\varphi_1 * \varphi_2$ if it can be split into two parts, one satisfying φ_1 and the other φ_2:

$$D \models \varphi_1 * \varphi_2 \quad \text{iff} \quad \exists D_1, D_2.\ D = D_1 \cdot D_2 \text{ and } D_1 \models \varphi_1 \text{ and } D_2 \models \varphi_2.$$

An instance of the separating conjunction is seen in the rule for parallel composition. Intuitively (ignoring for the moment the invariants referred to above), the rule is sound because the heap restricted to the owned locations can be split into two disjoint parts, one satisfying φ_1 and the other satisfying φ_2. The first process only accesses locations used to satisfy φ_1 and the second process only accesses locations used to satisfy φ_2. Consequently, interaction between the processes is limited.

The collection of locations that a process owns may change as the process evolves. As seen in the rule, after an allocation event has taken place, the process owns the newly current location. Similarly, deallocation of a location leads to loss of ownership.

To enable more interesting interaction between processes, the judgement environment Γ records, for each free variable of t, an *invariant*, which is a *precise* heap logic formula. A formula χ is precise if, for any D, there is at most one $D_0 \subseteq D$ such that $D_0 \models \chi$. Recall that the assignment ρ takes the variables in the domain of Γ to resources. If r is in the image of ρ, then there is an invariant in Γ associated with it; we call r *open*. We call non-open resources *closed*.

Whenever a process gains control of an open resource, it gains ownership of the locations satisfying the invariant in Γ associated with it. Reasoning relies on the fact that the invariant is satisfied if the resource is available. Whenever a process releases an open resource, ownership of a collection of locations satisfying the invariant is relinquished, and it is our obligation to guarantee that such a collection exists. Precision is used here so that at any stage we may determine which locations are owned by the process.

Notice that the rules allow ownership of locations to be transferred. Assume, for example, that the process owns location ℓ which has value 2 and there is an invariant for x that is $\ell' \mapsto 0 \vee (\ell' \mapsto 1 * \ell \mapsto 2)$. The only way in which the invariant could be satisfied disjointly from the locations that the process owns is for ℓ' to hold value 0. Consequently, as the process enters the critical region, it gains ownership of location ℓ'. If the process sets the value of ℓ' to 1, when the process leaves the critical region it would then be necessary to use ℓ to restore the invariant, so ownership of ℓ is lost.

4.1. Ownership model

We now formalize the meaning of judgements presented at the start of the previous section. We begin with the observation that the judgement will remain valid if other 'external' processes operate on the state providing they maintain invariants and do not access the locations owned by the process. In particular, external processes might soundly change the values held at unowned locations, allocate new locations and deallocate locations that they own, declare new resources and may enter critical regions thereby gaining ownership of any associated heap locations. This is all that we *rely* on when reasoning about the effect of other processes.

To model this, we construct an *interference net* for the process in Γ with ρ. This involves adding *ownership conditions* $\omega_{\mathsf{proc}}(\ell)$, $\omega_{\mathsf{inv}}(\ell)$ and $\omega_{\mathsf{oth}}(\ell)$ for each location ℓ. We restrict to markings W where precisely one of these is marked for each current location. The intuition is that $\omega_{\mathsf{proc}}(\ell)$ is marked if ℓ is owned by the process, $\omega_{\mathsf{inv}}(\ell)$ if ℓ is used to satisfy the invariant for an available open resource, and $\omega_{\mathsf{oth}}(\ell)$ if ℓ is current but owned by another process.

It is convenient to record similar information for re-

$$\frac{\text{for all } D \models \varphi \text{ and } (D_1, D_2) \in \mathcal{A}[\![\alpha]\!] : \left(\begin{array}{l} \mathrm{dom}(D_1) \subseteq \mathrm{dom}(D) \\ D_1 \subseteq D \text{ implies } (D \setminus D_1) \cup D_2 \models \psi \end{array} \right)}{\Gamma \vdash \{\varphi\}\ \alpha\ \{\psi\}} \qquad \frac{\Gamma \vdash \{\varphi_1\}\ t_1\ \{\psi_1\} \quad \Gamma \vdash \{\varphi_2\}\ t_2\ \{\psi_2\}}{\Gamma \vdash \{\varphi_1 * \varphi_2\}\ t_1 \parallel t_2\ \{\psi_1 * \psi_2\}} \qquad \frac{\Gamma \vdash \{\varphi\}\ \alpha_1; t_1\ \{\psi\} \quad \Gamma \vdash \{\varphi\}\ \alpha_2; t_2\ \{\psi\}}{\Gamma \vdash \{\varphi\}\ \alpha_1.t_1 + \alpha_2.t_2\ \{\psi\}}$$

$$\frac{\Gamma, x : \chi \vdash \{\varphi\}\ t\ \{\psi\} \quad (\chi \text{ precise})}{\Gamma \vdash \{\varphi * \chi\}\ \texttt{resource}\ x\ \texttt{do}\ t\ \texttt{od}\ \{\psi * \chi\}} \qquad \frac{\Gamma \vdash \{\varphi * \chi\}\ t\ \{\psi * \chi\}}{\Gamma, x : \chi \vdash \{\varphi\}\ \texttt{with}\ x\ \texttt{do}\ t\ \texttt{od}\ \{\psi\}}$$

$$\frac{}{\Gamma \vdash \{\ell \mapsto -\}\ \texttt{alloc}(\ell)\ \{\exists i(\ell \mapsto i * i \mapsto -)\}} \qquad \frac{}{\Gamma \vdash \{\exists i(\ell \mapsto i * i \mapsto -)\}\ \texttt{dealloc}(\ell)\ \{\exists i(\ell \mapsto i)\}}$$

Figure 2. Selected rules of concurrent separation logic

source names, so we introduce conditions $\omega_{\mathsf{proc}}(r)$, $\omega_{\mathsf{inv}}(r)$ and $\omega_{\mathsf{oth}}(r)$ for each resource r.

Table 1 defines a number of notations for events corresponding to the permitted interference described; they act according to the current ownership. The interference net is defined to comprise the following events:

- $\overline{\mathsf{act}}(D_1, D_2)$ for all D_1 and D_2 with the same domain
- $\overline{\mathsf{alloc}}(\ell, v, \ell', v')$ and $\overline{\mathsf{dealloc}}(\ell, \ell', v')$ for all locations ℓ and ℓ' and values v and v'
- $\overline{\mathsf{decl}}(r)$ and $\overline{\mathsf{end}}(r)$ for all resource names r
- $\overline{\mathsf{get}}(r)$ and $\overline{\mathsf{rel}}(r)$ for all closed resource names r
- $\overline{\mathsf{get}}(r, D_0)$ and $\overline{\mathsf{rel}}(r, D_0)$ for all open r associated with an invariant χ where $D_0 \models \chi$

The events described above describe how the locations owned by other processes is *dynamic* and how this *constrains* their action. The rule for parallel composition requires that the behaviour of the process being reasoned about conforms to these constraints, allowing its action to be seen as interference when reasoning about the other process. This requirement may be captured by *synchronizing* the events of the process with those from the interference net in the following way:

- The process event $\mathsf{act}_{(C,C')}(D, D')$ synchronizes with $\overline{\mathsf{act}}(D, D')$, and similarly for allocation, deallocation, declaration and events for critical region entry and exit where the critical region is protected by a closed resource.
- If r is an open resource associated with the invariant χ, the process event $\mathsf{get}_{(C,C')}(r)$ synchronizes with every $\overline{\mathsf{get}}(r, D_0)$ such that $D_0 \models \chi$. Similarly, $\mathsf{rel}_{(C,C')}(r)$ synchronizes with every $\overline{\mathsf{rel}}(r, D_0)$ such that $D_0 \models \chi$.

Suppose that two events synchronize, e from the process and e' from the interference net. The event e' is the event that would fire in the net for the other parallel process to simulate the event e; it is its dual. Let $e \cdot e'$ be the event formed by taking the union of the preconditions of e and e', other than using $\omega_{\mathsf{proc}}(\ell)$ in place of $\omega_{\mathsf{oth}}(\ell)$, and similarly $\omega_{\mathsf{proc}}(r)$ in place of $\omega_{\mathsf{oth}}(r)$.

Definition 4 *$\mathcal{W}[\![t, \Gamma]\!]_\rho$ is the net formed with the previous definitions of control, state and ownership conditions, and events:*

- *Every event from the interference net.*
- *Every event $e \cdot e'$ where e is an event of $\mathcal{N}[\![t]\!]_\rho$ and e' from the interference net such that e and e' synchronize.*

Let M be a marking of $\mathcal{W}[\![t, \Gamma]\!]_\rho$. We say that the process has violated its guarantees, or M is *violating*, if there exists an event e of $\mathcal{N}[\![t]\!]_\rho$ that has concession in marking M but there is no event e' from the interference net that synchronizes with e such that $e \cdot e'$ has concession. If no violation is ever encountered, the behaviour of $\mathcal{W}[\![t, \Gamma]\!]_\rho$ encapsulates all that of $\mathcal{N}[\![t]\!]_\rho$.

The following example shows how release of an open resource will cause a violation if the invariant is not restored.

Example 5 *Let r be an open resource associated with the invariant $\ell \mapsto 0$ and $D_0 = \{(\ell, 0)\}$. As only $D_0 \models \ell \mapsto 0$, the only associated interference event for release in the environment is $\overline{\mathsf{rel}}(r, D_0)$. This synchronizes with an event $\mathsf{rel}_{(C,C')}(r)$ in $\mathcal{N}[\![t]\!]_\rho$ to form the event e in $\mathcal{W}[\![t, \Gamma]\!]_\rho$ with*

$$\begin{array}{rcl} {}^\bullet e & = & \{\omega_{\mathsf{proc}}(r), \omega_{\mathsf{proc}}(\ell), (\ell, 0)\} \cup C \\ e^\bullet & = & \{\omega_{\mathsf{inv}}(r), r, \omega_{\mathsf{inv}}(\ell), (\ell, 0)\} \cup C'. \end{array}$$

This will not have concession if the invariant is not re-established, whereas the event $\mathsf{rel}_{(C,C')}(r)$ may have.

4.2. Soundness and validity

The rule for parallel composition tells us that the ownership of the heap is initially split between the two processes, so that what one process owns is seen as owned by an external process by the other.

Definition 6 (Ownership split) *Let W be a marking of ownership conditions. W_1 and W_2 form an* ownership split *of W if (with the same constraints for each r):*

- *$\omega_{\mathsf{oth}}(\ell) \in W$ iff $\omega_{\mathsf{oth}}(\ell) \in W_1$ and $\omega_{\mathsf{oth}}(\ell) \in W_2$.*
- *$\omega_{\mathsf{inv}}(\ell) \in W$ iff $\omega_{\mathsf{inv}}(\ell) \in W_1$ and $\omega_{\mathsf{inv}}(\ell) \in W_2$.*
- *$\omega_{\mathsf{proc}}(\ell) \in W$ iff $\omega_{\mathsf{proc}}(\ell) \in W_1$ and $\omega_{\mathsf{oth}}(\ell) \in W_2$, or $\omega_{\mathsf{proc}}(\ell) \in W_2$ and $\omega_{\mathsf{oth}}(\ell) \in W_1$.*

Following Brookes' lead, we are now able to prove the lemma upon which the proof of soundness lies. The effect of this lemma is that we can determine the terminal states of

Abbreviation	Preconditions	Postconditions
$\overline{\mathsf{act}}(D_1, D_2)$	$D_1 \cup \{\omega_{\mathsf{oth}}(\ell) \mid \exists v.(\ell, v) \in D_1\}$	$D_2 \cup \{\omega_{\mathsf{oth}}(\ell) \mid \exists v.(\ell, v) \in D_2\}$
$\overline{\mathsf{alloc}}(\ell, v, \ell', v')$	$\{\omega_{\mathsf{oth}}(\ell), (\ell, v)\}$	$\{\omega_{\mathsf{oth}}(\ell), \omega_{\mathsf{oth}}(\ell'), \mathsf{curr}(\ell'), (\ell, \ell'), (\ell', v')\}$
$\overline{\mathsf{dealloc}}(\ell, \ell', v')$	$\{\omega_{\mathsf{oth}}(\ell), \omega_{\mathsf{oth}}(\ell'), \mathsf{curr}(\ell'), (\ell, \ell'), (\ell', v')\}$	$\{\omega_{\mathsf{oth}}(\ell), (\ell, \ell')\}$
$\overline{\mathsf{decl}}(r)$	$\{\}$	$\{\omega_{\mathsf{oth}}(r), \mathsf{curr}(r), r\}$
$\overline{\mathsf{end}}(r)$	$\{\omega_{\mathsf{oth}}(r), \mathsf{curr}(r), r\}$	$\{\}$
$\overline{\mathsf{get}}(r)$	$\{\omega_{\mathsf{oth}}(r), r\}$	$\{\omega_{\mathsf{oth}}(r)\}$
$\overline{\mathsf{rel}}(r)$	$\{\omega_{\mathsf{oth}}(r)\}$	$\{\omega_{\mathsf{oth}}(r), r\}$
$\overline{\mathsf{get}}(r, D_0)$	$\{\omega_{\mathsf{inv}}(r), r\} \cup D_0 \cup \{\omega_{\mathsf{inv}}(\ell) \mid \exists v.(\ell, v) \in D_0\}$	$\{\omega_{\mathsf{oth}}(r)\} \cup D_0 \cup \{\omega_{\mathsf{oth}}(\ell) \mid \exists v.(\ell, v) \in D_0\}$
$\overline{\mathsf{rel}}(r, D_0)$	$\{\omega_{\mathsf{oth}}(r)\} \cup D_0 \cup \{\omega_{\mathsf{oth}}(\ell) \mid \exists v.(\ell, v) \in D_0\}$	$\{\omega_{\mathsf{inv}}(r), r\} \cup D_0 \cup \{\omega_{\mathsf{inv}}(\ell) \mid \exists v.(\ell, v) \in D_0\}$

Table 1. Interference events

parallel processes simply by observing the terminal states of its components if we split the ownership of the initial state correctly. For convenience, the lemma is stated without intimating the particular event that takes place on the net transition relation.

Lemma 7 (Parallel decomposition) *Let* $M = (1{:}C_1 \cup 2{:}C_2, \sigma, W)$ *be a marking of the net* $\mathcal{W}[\![t_1 \parallel t_2, \Gamma]\!]_\rho$*, and let* W_1 *and* W_2 *form an ownership split of* W*.*

If, for both $i \in \{1, 2\}$*,* $M_i = (C_i, \sigma, W_i)$ *is not violating in* $\mathcal{W}[\![t_i, \Gamma]\!]$*, then* M *is not violating in* $\mathcal{W}[\![t_1 \parallel t_2, \Gamma]\!]$*.*

Furthermore, if $(1{:}C_1 \cup 2{:}C_2, \sigma, W) \twoheadrightarrow (1{:}C_1' \cup 2{:}C_2', \sigma', W')$ *in* $\mathcal{W}[\![t_1 \parallel t_2, \Gamma]\!]_\rho$ *then there exist* W_1' *and* W_2' *forming an ownership split of* W' *such that* $(C_1, \sigma, W_1) \twoheadrightarrow (C_1', \sigma', W_1')$ *in* $\mathcal{W}[\![t_1, \Gamma]\!]_\rho$ *and* $(C_2, \sigma, W_2) \twoheadrightarrow (C_2', \sigma', W_2')$ *in* $\mathcal{W}[\![t_2, \Gamma]\!]_\rho$*.*

Say that a state σ with an ownership marking W satisfies the formula φ and the invariants in Γ if the heap restricted to the owned locations satisfies φ and the invariants are met for all the available resources. In the formalization of this, we write $\mathrm{inv}(\Gamma, \rho, R)$ for the formula $\chi_1 * \ldots * \chi_n$ where $x_i : \chi_i \in \Gamma$ and x_i is available in R, *i.e.* $\rho(x_i) \in R$. We also use the notation

$$D \restriction_W \mathsf{proc} \stackrel{\text{def}}{=} \{(\ell, v) \mid \omega_{\mathsf{proc}}(\ell) \in W\}$$

for the heap at owned locations, and define similar notations $D \restriction_W \mathsf{inv}$ and $D \restriction_W \mathsf{oth}$.

Definition 8 *A marking* $(C, (D, L, R, N), W)$*, where* (D, L, R, N) *is consistent,* satisfies φ in Γ *if:*

- $D \restriction_W \mathsf{proc} \models \varphi$ *and* $D \restriction_W \mathsf{inv} \models \mathrm{inv}(\Gamma, \rho, R)$
- *all the open resource constants are current in* N
- *the resource* r *is owned as an invariant if* r *is open and available in* R

There is no restriction on how the ownership of the non-open and held resource names is split.

Definition 9 (Validity) *Let* t *be a term containing no free names with free variables contained in the domain of* Γ*.*

$\Gamma \models \{\varphi\}\ t\ \{\psi\}$ *if for any marking* $(\mathrm{Ic}(t), \sigma, W)$ *that satisfies* φ *in* Γ *and any assignment* ρ*, in the net* $\mathcal{W}[\![t, \Gamma]\!]_\rho$ *no violating marking is reachable and any reachable marking* $(\mathrm{Tc}(t), \sigma', W')$ *satisfies* ψ *in* Γ*.*

An analysis essentially following that in Brookes' proof allows us to arrive at:

Theorem 10 (Soundness)

$$\Gamma \vdash \{\varphi\}\ t\ \{\psi\} \textit{ implies } \Gamma \models \{\varphi\}\ t\ \{\psi\}.$$

Corollary 11 *Let* t *be a closed term with no free resource names. If* $\emptyset \vdash \{\varphi\}\ t\ \{\psi\}$ *then, from any state in which the heap satisfies* φ*, the process* t *never accesses a non-current heap location or resource, and, whenever the process terminates, the resulting heap satisfies* ψ*.*

We conclude this section by characterizing the independence of parallel processes, which implies Brookes' race freedom result. Say that two events are *control independent* if they share no common pre- or post-control condition, and observe that two such events may be enabled only if they arise from different components of a parallel composition.

Theorem 12 (Separation) *Suppose that* $\emptyset \vdash \{\varphi\}\ t\ \{\psi\}$ *and that* σ *is a state in which the heap satisfies* φ*. If* M *is a marking reachable from* $(\mathrm{Ic}(t), \sigma)$ *in* $\mathcal{N}[\![t]\!]$ *and* e_1 *and* e_2 *are control independent events then:*

- *If* $M \xrightarrow{e_1} M_1 \xrightarrow{e_2} M'$ *then either* e_1 *and* e_2 *are independent or* e_1 *releases a resource name, resource or a location that* e_2 *correspondingly binds, takes or allocates.*
- *If* $M \xrightarrow{e_1} M_1$ *and* $M \xrightarrow{e_2} M_2$ *then either* e_1 *and* e_2 *are independent or* e_1 *and* e_2 *compete either to declare the same resource name, take the same resource or to allocate the same location.*

The first part of the preceding theorem tells us how the event occurrences of parallel processes *causally depend* on each other: the way in which the ability of one process to affect the global state in a particular way is dependent on events of the other process. The second part tells us how the enabled events of parallel processes *conflict* with each other in a state: the way in which one parallel process can prevent the other acting in a particular way on the global state.

Observe that, although there is neither conflict nor causal dependence arising from heap events (and hence the processes are race-free in the sense of Brookes), there may be interaction through allocation and deallocation events. One

may therefore give judgements for parallel processes that interact *without* using critical regions. Suppose, for example, that we have a heap $\{(\ell_0, \ell_1), (\ell_1, v), (\ell_2, v')\}$. If we place the process $t_1; \mathtt{dealloc}(\ell_0)$ in parallel with

$$\mathtt{alloc}(\ell_2); \quad \mathtt{while}\ [\ell_2] \neq \ell_1\ \mathtt{do}\ \mathtt{alloc}(\ell_2)\ \mathtt{od}; \quad t_2,$$

the process t_2 only takes place once t_1 has terminated.

5. Refinement

As we remarked in the introduction, the atomicity assumed of primitive actions, also called their *granularity*, is of significance when considering parallel programs. For example, suppose that an assignment $[\ell] := [\ell'] + 1$ is in fact executed as $[\ell] := [\ell']$ followed by $[\ell] := [\ell] + 1$. Now suppose that there is process running concurrently that performs some action if ℓ and ℓ' hold the same value. Running from a state in which they do not, the first interpretation falsely suggests that this may never happen. In [19], Reynolds proposes that the absence of races allows our semantics to be insensitive to the atomicity assumed of actions. We now provide a form of *refinement*, similar to but more general than that of [20], that captures these ideas. We relate the nets representing processes by regarding them as alternative substitutions into a *context*.

We begin with the observation that our embedded nets have certain properties: the sets of initial and terminal control conditions are disjoint and nonempty; all events have at least one precondition and one postcondition; and if any reachable marking of control conditions C contains the terminal conditions T of the net, then $C = T$ and no event has concession on its control conditions.

Definition 13 *Define a* context K *to be an embedded net with a distinguished event* $[-]$. *The event* $[-]$ *is such that* ${}^\bullet[-]^\bullet \subseteq \mathbf{C}$ *and its pre- and postconditions form disjoint, nonempty sets.*

Let K *and* N *have disjoint sets of control conditions and the same sets of state conditions. Define the sets*

$$P_{\mathsf{i}} \stackrel{def}{=} {}^\bullet[-] \times \mathrm{Ic}(N) \qquad P_{\mathsf{t}} \stackrel{def}{=} [-]^\bullet \times \mathrm{Tc}(N)$$

and let $K[N]$ *be the net formed with events*

$$\begin{aligned} \mathrm{Ev}(K[N]) &\stackrel{def}{=} (P_{\mathsf{i}} \cup P_{\mathsf{t}}) \triangleleft (\mathrm{Ev}(K) \setminus \{[-]\}) \\ &\cup \quad (P_{\mathsf{i}} \cup P_{\mathsf{t}}) \triangleright \mathrm{Ev}(N). \end{aligned}$$

To investigate the properties of substitutions, we write $N : \sigma \Downarrow \sigma'$ if there exists a run from the marking $\mathrm{Ic}(N) \cup \sigma$ to $\mathrm{Tc}(N) \cup \sigma'$ within N. We also define a notion of equivalence $\simeq$ as:

$$N_1 \simeq N_2 \quad \text{iff} \quad (\forall \sigma, \sigma')\, N_1 : \sigma \Downarrow \sigma' \iff N_2 : \sigma \Downarrow \sigma'.$$

Intuitively, if the substituend N were an atomic event, it would start running only if the conditions P_{i} were marked and P_{t} were not. There are two distinct ways in which the context K can affect the execution of N. Firstly, it might affect the marking of conditions in P_{i} or P_{t} whilst N is running. Secondly, it might change the marking of state conditions in a way that affects the execution of N. An instance of the latter form of interference can be inferred from the example at the start of this section. We now define a form of constrained substitution, guided by Theorem 12, so that N is not subject to these forms of interference.

Say that a control condition c is *internal* to N within $K[N]$ if c is a condition of N not in $\mathrm{Ic}(N)$ or $\mathrm{Tc}(N)$. Given a marking M of $K[N]$, say that N is *active* if $P_{\mathsf{i}} \subseteq M$ or there exists an internal condition of N in M.

Definition 14 *For a given marking of state conditions* σ, *we say that* $K[N]$ *is a* non-interfering substitution *if, for all markings* M *reachable from* $(\mathrm{Ic}(K[N]), \sigma)$*:*

- *if* $P_{\mathsf{i}} \subseteq M$ *then* $P_{\mathsf{t}} \cap M = \emptyset$, *and*
- *if* N *is active in* M *then no enabled event of* K *has a pre- or postcondition in* P_{i} *or* P_{t}, *and*
- *if* $M \stackrel{e_1}{\longrightarrow\!\!\!\!\!\rightarrow} M_1 \stackrel{e_2}{\longrightarrow\!\!\!\!\!\rightarrow} M'$, *one of* e_1 *and* e_2 *is from* N *and the other from* K *and* N *is active in* M *and* M_1, *then* e_1 *and* e_2 *are independent.*

Theorem 15 *If* $N_1 \simeq N_2$ *and* $K[N_1]$ *and* $K[N_2]$ *are non-interfering substitutions from state* σ, *then, for any* σ'*:*

$$K[N_1] : \sigma \Downarrow \sigma' \quad \textit{iff} \quad K[N_2] : \sigma \Downarrow \sigma'.$$

The proof of the preceding theorem relies on the fact that, since consecutive actions of K and N are independent, their occurrence in any run may be swapped (in fact, this is all we require; independence is not strictly necessary). Consequently, we only need to reason about runs in which there is no interleaving of events of K with events of N.

Suppose that we have $\emptyset \models \{\varphi\}\ t\ \{\psi\}$ and that there is an occurrence of α inside $\mathcal{N}[\![t]\!]$ not at the head of a sum or as the boolean of a while loop. It follows from Theorem 12 that this forms a non-interfering substitution with the rest of the net for t. If N is an embedded net such that $\mathcal{N}[\![\alpha]\!] \simeq N$ that accesses the same locations as α along any run between two states, it follows that this also forms a non-interfering substitution. Thus Corollary 11 also holds for the net where the occurrence of α is replaced by N.

6. Related work and conclusions

The first component of this work provides an inductive definition of the semantics of command terms as a net. This is a relatively novel technique, but has in the past been applied to give the semantics of a language for investigating security protocols, SPL [7], though our language involves a richer collection of constructs. Other independence models for terms include the Box calculus [1] and the event structure and net semantics of CCS [21, 22], though these model interaction as synchronized communication rather than occurring through shared state.

The proof of soundness of separation logic here is led by Brookes' earlier work [4]. There are a few minor differences in the syntax of processes, including that we allow the dynamic binding of resource variables. More notably, our store model does not include the *stack variable*, which may be seen as a particular form of memory location to which other locations may not point. In Brookes' model, interference of parallel processes through stack variables is constrained by the use of a side condition on the rule rather than using the concept of ownership; an area of current research on 'permissions' [2] promises a uniform approach. We have chosen not to include stack variables in our model in order to highlight the concept of ownership. To obtain a more useable programming language without weakening this concept, the language could be extended with a $\texttt{let}\ x = e\ \texttt{in}\ t$ construct, where e may evaluate to a location. This would come at the cost of a detailed, though straightforward, technical analysis of open terms (which proves convenient for providing Hoare's law of existential elimination).

At the core of Brookes' work is a 'local enabling relation', which gives the semantics of programs over a restricted set of 'owned' locations. Our notion of validity involves maintaining a record of ownership and using this to constrain the occurrence of events in the interference net augmented to the process. This allows the intuition of ownership in O'Hearn's introduction of concurrent separation logic [14] to be seen directly as constraining interference. Though the relationship between our model and Brookes' is fairly obvious, we believe that our approach leads to a clearer parallel decomposition lemma, upon which the proof of soundness critically stands.

The most significant difference between our work and Brookes' is that the net model captures, as a primitive property, the independence of parallel processes enforced by the logic. We have used this property to apply a straightforward, yet general, form of refinement suited to proving that race freedom obviates the problem of granularity. This is in contrast to [5], which provides a different form of trace semantics to tackle the issue of granularity, tailored very specifically to reasoning about processes that are race-free. Furthermore, our separation result is much stronger than the existing proof of race freedom, for example showing that interaction between parallel processes may occur through allocation and deallocation. This is significant, as such interaction leads to examples of the incompleteness of concurrent separation logic.

There are a number of areas for further research. At present, we are investigating semantic models that deal more elegantly with name binding and how local reasoning may be used to establish liveness properties.

Acknowledgements We would like to thank Peter O'Hearn and Matthew Parkinson for helpful discussions and the anonymous referees for constructive suggestions.

References

[1] E. Best, R. Devillers, and J. G. Hall. The box calculus: A new causal algebra with multi-label communication. In *Advances in Petri Nets*, volume 609 of *LNCS*. Springer Verlag, 1992.

[2] R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *Proc. POPL '05*. ACM Press, 2005.

[3] P. Brinch Hansen. Structured multiprogramming. *Comm. ACM*, 15(7):574–578, 1972.

[4] S. Brookes. A semantics for concurrent separation logic. In *Proc. CONCUR '04*, volume 3170 of *LNCS*. Springer Verlag, 2004.

[5] S. Brookes. A grainless semantics for parallel programs with shared mutable data. In *Proc. MFPS XXI*, ENTCS, 2005.

[6] E. M. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *Int. Journal on Software Tools for Technology Transfer*, 2(3), 1999.

[7] F. Crazzolara and G. Winskel. Events in security protocols. In *Proc. CCS '01*, New York, 2001. ACM Press.

[8] E. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*. Academic Press, 1968.

[9] C. A. R. Hoare. Towards a theory of parallel programming. In C. A. R. Hoare and R. H. Perrot, editors, *Operating Systems Techniques*. Academic Press, 1972.

[10] S. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. POPL '01*. ACM Press.

[11] C. B. Jones. Specification and design of (parallel) programs. In R. E. A. Mason, editor, *Information Processing 83: Proc. IFIP Congress*, pages 321–332, 1983.

[12] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. In *Proc. LICS '93*, volume 127(2) of *Information and Computation*. Elsevier, 1993.

[13] L. Lamport. On interprocess communication. *Distributed Computing*, 1(2):77–101, June 1986.

[14] P. W. O'Hearn. Resources, concurrency and local reasoning. *Theoretical Computer Science*, 2004. To appear.

[15] P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2), 1999.

[16] S. Owicki and D. Gries. Verifying properties of parallel programs: An axiomatic approach. *Comm. ACM*, 19(5):279–285, 1976.

[17] V. Pratt. Modeling concurrency with partial orders. *Int. Journal of Parallel Programming*, 15(1), 1986.

[18] J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*, 2000.

[19] J. C. Reynolds. Towards a grainless semantics for shared variable concurrency. In *Proc. FSTTCS '04*, volume 3328 of *LNCS*. Springer Verlag, 2004.

[20] R. J. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In *Proc. MFCS '89*, volume 379 of *LNCS*. Springer Verlag, 1989.

[21] G. Winskel. Event structure semantics for CCS and related languages. In *Proc. ICALP '82*, volume 140 of *LNCS*. Springer Verlag, 1982.

[22] G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic and the Foundations of Computer Science*, volume 4, pages 1–148. OUP, 1995.

A. Petri nets

A basic net is a five-tuple,

$$(B, E, {}^\bullet(-), (-)^\bullet, M_0).$$

The set B comprises the *conditions* of the net, the set E consists of the *events* of the net, and M_0 is the subset of B of *marked conditions* (the *initial marking*). The maps

$${}^\bullet(-), (-)^\bullet : E \to \mathcal{P}ow(B)$$

are the *precondition* and *postcondition* maps, respectively. Nets are normally drawn with:

- circles to represent conditions,
- bold lines to represent events,
- arrows from conditions to events to represent the precondition map,
- arrows from events to conditions to represent the postcondition map, and
- tokens (dots) inside conditions to represent the marking.

Action within nets is defined according to a *token game*. This defines how the marking of the net changes according to *firing* of the events. An event e can fire if all its preconditions are marked and, following their un-marking, all the postconditions are not marked. That is, in marking M,

$$(1) \qquad {}^\bullet e \subseteq M$$

$$(2) \qquad (M \setminus {}^\bullet e) \cap e^\bullet = \emptyset.$$

Such an event is said to have *concession* or to be *enabled*. We write

$$M \xrightarrow{e} M'$$

where

$$M' = (M \setminus {}^\bullet e) \cup e^\bullet.$$

If constraint (2) does not hold, there is said to be *contact* in the marking.

For any event $e \in E$, define the notation

$${}^\bullet e^\bullet \stackrel{\text{def}}{=} {}^\bullet e \cup e^\bullet.$$

The appropriate notion of independence within this form of Petri net is to say that two events e_1 and e_2 are independent, written $e_1 I e_2$, if, and only if,

$${}^\bullet e_1{}^\bullet \cap {}^\bullet e_2{}^\bullet = \emptyset.$$

B. Size of terms

$$\begin{aligned}
\mathsf{size}(\alpha) &\stackrel{\text{def}}{=} 1 \\
\mathsf{size}(t_1; t_2) &\stackrel{\text{def}}{=} \mathsf{size}(t_1) + \mathsf{size}(t_2) \\
\mathsf{size}(t_1 \parallel t_2) &\stackrel{\text{def}}{=} \mathsf{size}(t_1) + \mathsf{size}(t_2) \\
\mathsf{size}(\alpha_1.t_1 + \alpha_2.t_2) &\stackrel{\text{def}}{=} 2 + \mathsf{size}(t_1) + \mathsf{size}(t_2) \\
\mathsf{size}(\texttt{resource}\ x\ \texttt{do}\ t\ \texttt{od}) &\stackrel{\text{def}}{=} 1 + \mathsf{size}(t) \\
\mathsf{size}(\texttt{with}\ x\ \texttt{do}\ t\ \texttt{od}) &\stackrel{\text{def}}{=} 1 + \mathsf{size}(t) \\
\mathsf{size}(\texttt{with}\ r\ \texttt{do}\ t\ \texttt{od}) &\stackrel{\text{def}}{=} 1 + \mathsf{size}(t) \\
\mathsf{size}(\texttt{alloc}(\ell)) &\stackrel{\text{def}}{=} 1 \\
\mathsf{size}(\texttt{dealloc}(\ell)) &\stackrel{\text{def}}{=} 1
\end{aligned}$$

C. Rules of concurrent separation logic

- Action

$$\frac{\begin{array}{c}\text{for all } D \models \varphi \text{ and } (D_1, D_2) \in \mathcal{A}[\![\alpha]\!] : \\ \left(\begin{array}{l}\mathrm{dom}(D_1) \subseteq \mathrm{dom}(D) \\ D_1 \subseteq D \implies (D \setminus D_1) \cup D_2 \models \psi\end{array}\right)\end{array}}{\Gamma \vdash \{\varphi\}\ \alpha\ \{\psi\}}$$

- Allocation:

$$\frac{}{\Gamma \vdash \{\ell \mapsto -\}\ \texttt{alloc}(\ell)\ \{\exists i(\ell \mapsto i * i \mapsto -)\}}$$

- Deallocation:

$$\frac{}{\Gamma \vdash \{\exists i(\ell \mapsto i * i \mapsto -)\}\ \texttt{dealloc}(\ell)\ \{\exists i(\ell \mapsto i)\}}$$

- Sequential composition:

$$\frac{\Gamma \vdash \{\varphi\}\ t_1\ \{\varphi'\} \quad \Gamma \vdash \{\varphi'\}\ t_2\ \{\psi\}}{\Gamma \vdash \{\varphi\}\ t_1; t_2\ \{\psi\}}$$

- Sum:

$$\frac{\Gamma \vdash \{\varphi\}\ \alpha_1; t_1\ \{\psi\} \quad \Gamma \vdash \{\varphi\}\ \alpha_2; t_2\ \{\psi\}}{\Gamma \vdash \{\varphi\}\ \alpha_1.t_1 + \alpha_2.t_2\ \{\psi\}}$$

- Loop:

$$\frac{\Gamma \vdash \{\varphi\}\ b\ \{\varphi'\} \quad \Gamma \vdash \{\varphi'\}\ \texttt{while}\ b\ \texttt{do}\ t\ \texttt{od}\ \{\varphi\} \quad \Gamma \vdash \{\varphi\}\ \neg b\ \{\psi\}}{\Gamma \vdash \{\varphi\}\ \texttt{while}\ b\ \texttt{do}\ t\ \texttt{od}\ \{\psi\}}$$

- Resource declaration:

$$\frac{\Gamma, x : \chi \vdash \{\varphi\}\ t\ \{\psi\} \quad (\chi \text{ precise})}{\Gamma \vdash \{\varphi * \chi\}\ \texttt{resource}\ x\ \texttt{do}\ t\ \texttt{od}\ \{\psi * \chi\}}$$

- Critical region:

$$\frac{\Gamma \vdash \{\varphi * \chi\}\ t\ \{\psi * \chi\}}{\Gamma, x : \chi \vdash \{\varphi\}\ \texttt{with}\ x\ \texttt{do}\ t\ \texttt{od}\ \{\psi\}}$$

- Parallel composition:

$$\frac{\Gamma \vdash \{\varphi_1\}\ t_1\ \{\psi_1\} \quad \Gamma \vdash \{\varphi_2\}\ t_2\ \{\psi_2\}}{\Gamma \vdash \{\varphi_1 * \varphi_2\}\ t_1 \parallel t_2\ \{\psi_1 * \psi_2\}}$$

- Frame:

$$\frac{\Gamma \vdash \{\varphi\}\ t\ \{\psi\}}{\Gamma \vdash \{\varphi * \varphi'\}\ t\ \{\psi * \varphi'\}}$$

- Consequence:

$$\frac{\varphi \implies \varphi' \quad \Gamma \vdash \{\varphi'\}\ t\ \{\psi'\} \quad \psi' \implies \psi}{\Gamma \vdash \{\varphi\}\ t\ \{\psi\}}$$

- Existential: (As stated, uses a program variable)

$$\frac{\Gamma \vdash \{[x/i]\varphi\}\ t\ \{[x/i]\psi\}}{\Gamma \vdash \{\exists i.\varphi\}\ t\ \{\exists i.\psi\}}(x \text{ fresh})$$

- Conjunction:

$$\frac{\Gamma \vdash \{\varphi_1\}\ t\ \{\psi_1\} \quad \Gamma \vdash \{\varphi_2\}\ t\ \{\psi_2\}}{\Gamma \vdash \{\varphi_1 \wedge \varphi_2\}\ t\ \{\psi_1 \wedge \psi_2\}}$$

- Disjunction:

$$\frac{\Gamma \vdash \{\varphi_1\}\ t\ \{\psi_1\} \quad \Gamma \vdash \{\varphi_2\}\ t\ \{\psi_2\}}{\Gamma \vdash \{\varphi_1 \vee \varphi_2\}\ t\ \{\psi_1 \vee \psi_2\}}$$

- Expansion:

$$\frac{\Gamma \vdash \{\varphi\}\ t\ \{\psi\}}{\Gamma, \Gamma' \vdash \{\varphi\}\ t\ \{\psi\}}$$

- Contraction:

$$\frac{\Gamma, \Gamma' \vdash \{\varphi\}\ t\ \{\psi\}}{\Gamma \vdash \{\varphi\}\ t\ \{\psi\}}(\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma))$$