Working Paper Series
ISSN 1170-487X

ATOMIC COMPONENTS

Steve Reeves and David Streader

Working Paper: 01/2004
February 2004

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Atomic Components

Steve Reeves and David Streader
Department of Computer Science
University of Waikato
Hamilton, New Zealand
{dstr,stever}@cs.waikato.ac.nz

Abstract

There has been much interest in components that combine the best of state-based and event-based approaches. The interface of a component can be thought of as its specification and substituting components with the same interface cannot be observed by any user of the components. Here we will define the semantics of atomic components where both states and event can be part of the interface. The resulting semantics is very similar to that of (event only) processes. But it has two main novelties: one, it does not need recursion or unique fixed points to model nontermination; and two, the behaviour of divergence is modelled by abstraction, i.e. *the construction of the observational semantics.*

Keywords: state and action, components, refinement, labelled transition systems, Z

1 Introduction

Industry is looking to create a market in reliable "plug-and-play" components. To do this the *interface* [2] of a component needs to be defined in a way that makes it safe to substitute components with the same interface.

Microsoft approaches this issue using Abstract State Machines (ASM) as a starting point and have noted [4] that it would be very useful to combine the event-based process algebras, which have modular reasoning built in, with the descriptive ability of the state-based ASM. To this end some process algebra features have been added to ASM [4] but many conceptual difficulties remain. An alternative approach is to start with a process algebra and enrich its descriptive ability to be more like ASM while retaining the desired modularity. The work of [9] can be interpreted as an example of this approach.

Here we are going to consider only simple components with atomic states and atomic actions. Because of the simple semantics of our components they are easily recognisable as a small extension of processes. Our goal is to model states and events in as direct a style as we can and to define an equality that is congruent with respect to component composition and abstraction, where abstraction simplifies the component while preserving its interface.

To achieve these goals we make the following design decisions:

1 states and events are either in the interface or internal;
2 use operational not denotational semantics;
3 "observational" congruences ignore the internal;
4 model divergence with out chaos;
5 model nonterminating process without using recursion or unique fixed points.

The first of our design decisions reflects our desire to model components not processes, the second arises from the well-known [21, 8, 17] isomorphism between state-based relational semantics and event-based operational semantics.

We reject models based on bisimulation semantics such as CCS [16] or ACP [1] as few of the hidden actions, *i.e.* τs, can be removed. With models based on failure semantics such as CSP, CFFD [22] and NDFD [14] all τ actions can be removed, although to obtain congruence the denotational semantics is complicated by adding both divergence and stability. We reject CSP's failure-divergence semantics for two reasons: one, it does not attempt to model how divergent processes behave— "once a process can diverge, we assume (whether it is true or not) that it can then perform any trace, refuse anything, and diverge on any later trace" [19][p. 95]; and two, abstraction does not distribute through choice. We reject the use of denotational semantics in CFFD and NDFD, but find that our operational definitions result in an equivalence that, with some small restrictions, is equivalent to NDFD.

There is a natural, from an operational perspective, decomposition of observational equivalences into two parts: one, abstraction (the construction of the observational semantics); and two, a definition of a strong, *i.e.* non-observational, equivalence (see Section 2.3). The main contribution of this work is defining abstraction so that it mod-

els what can be seen of a process that diverges.

The separation of observational equivalences into abstraction (the construction of an observational semantics) and a strong equivalence has been well exploited in [16, 1], where they use bisimulation and a definition of abstraction that ignores divergence. It is well known that for failure-equivalence divergence cannot be ignored [3].

To be consistent with our first design decision we have used choice from [25] not one the more well known process algebras [12, 19, 11, 16, 1] although for terminating process this specialise to choice as in [1, 16].

It is possible to take previously defined models, one for states and one for events, and then glue them together [20, 8, 26]. This has the advantage of making tool reuse easy but requires accepting each model on an all-or-nothing basis. We are trying to define a model that treats states and events on an equal footing by taking what best fits our needs from a range of models.

This work is novel in two ways: one, we do not use recursion to define nonterminating processes; and two, our operational semantics models what can be seen of divergent processes as nondeterminism.

The discussion above has been rather general and conceptual, but our motivation has been to provide a semantics that permits modular reasoning *in practice*. So, in an attempt to give some assurance that our framework is of practical use, we use it (in Section 5) to model and reason about a simple example and briefly compare our model with other, more well-known, models from the literature.

2 Component specifications

Components consist of atomic states and events. Both can either be a part of the component's *interface* or are internal to the component.

We will write $\mathsf{A}, \mathsf{B} \ldots$ for components and will assume a universe of observable action names $\mathsf{div}, \mathsf{a}, \mathsf{b} \ldots \in Act$ and τ for internal actions and $Act^\tau \stackrel{\text{def}}{=} Act \cup \{\tau\}$. We also assume a universe of state propositions Π (see [9]) which include propositions *s* for the start and *e* for end of the component. We can now view components as given by certain transition systems.

Definition 1 *Component transition system (CTS)*

$$\mathsf{A} \stackrel{\text{def}}{=} (N_\mathsf{A}, Os_\mathsf{A}, T_\mathsf{A}) \textit{ is a CTS where}$$

N_A *is a finite set of nodes*
$T_\mathsf{A} \subseteq \{(n, a, m) \mid n, m \in N_\mathsf{A} \wedge a \in Act^\tau\}$.
$Os_\mathsf{A} : N_\mathsf{A} \to 2^\Pi$ *(the observable states of* A*).* ∘

As usual we can define $e_\mathsf{A} \stackrel{\text{def}}{=} \{n \mid e \in Os_\mathsf{A}(n)\}$ and $s_\mathsf{A} \stackrel{\text{def}}{=} \{n \mid s \in Os_\mathsf{A}(n)\}$. This shows we allow *sets* of ending (final) and starting (initial) states. The intended meaning of the set of initial states is that the specification can nondeterministically be started in any of these states.

If $Os_\mathsf{A}(n) = \varnothing$ then *n* is an *internal* state, else it is a part of the interface. We write $n \xrightarrow{a} m$ for $(n, a, m) \in T_\mathsf{A}$ and $n \xrightarrow{a}$ for $\exists_m . (n, a, m) \in T_\mathsf{A}$.

Then $\pi(s) \stackrel{\text{def}}{=} \{a \mid s \xrightarrow{a}\}$ and $\alpha(\mathsf{A}) \stackrel{\text{def}}{=} \{a \mid n \xrightarrow{a} m \in T_\mathsf{A}\}$

2.1 Processes are special components.

A process can easily be seen as a component with state propositions restricted to propositions for the start and end of the process. Clearly process $(N_\mathsf{A}, Os_\mathsf{A}, T_\mathsf{A})$ defines and is defined by $(N_\mathsf{A}, s_\mathsf{A}, e_\mathsf{A}, T_\mathsf{A})$, which is just a labelled transition system (LTS) as usually understood. So, we can say that every CTS can be viewed as an LTS, which we shall do in the sequel, especially when being able to relate back to the standard world of LTS is desirable.

Although we regard our atomic components as a minor extension of processes these processes are slightly different from those of CSP, CCS and ACP in that they have a set of start states.

2.2 Component Equivalences

Failures are usually defined with traces that are sequences of action names. But we wish to take account of state observation and the fact that we have a set of start states.

We write $n_1 \xrightarrow{Os(n_1)\mathsf{x}_1 Os(n_2)\mathsf{x}_2 \ldots \mathsf{x}_k Os(n_{k+1})} n_{k+1}$ when $\exists_{n_1, \ldots n_{k+1}} . (n_1, \mathsf{x}_1, n_2), \ldots (n_k, \mathsf{x}_k, n_{k+1}) \in T_\mathsf{A}$ and let θ range over alternating sequences of state observations $Os(n_i)$ and actions x_i.

$Tr(\mathsf{A}) \stackrel{\text{def}}{=} \{\theta \mid m \xrightarrow{\theta} n \wedge s \in Os_\mathsf{A}(m)\}$

$F(\mathsf{A}) \stackrel{\text{def}}{=} \{\langle \theta, X \rangle \mid m \xrightarrow{\theta} n \wedge s \in Os_\mathsf{A}(m) \wedge \forall \mathsf{x} . n \xrightarrow{\mathsf{x}} \not\rightarrow\}$.

2.3 Component Abstraction

The CTS in Definition 1 take no account of τ actions being unobservable, so we would call it a **strong semantics** ($\rightarrow$) and an equivalence based on it a **strong equivalence** ($=_X$) . From an **observational semantics** ($\Rightarrow_a$) we define an abstraction function $Abs_a(\mathsf{A}) \stackrel{\text{def}}{=} \langle N_\mathsf{A}, Os_\mathsf{A}, \{n \xrightarrow{x} m \mid n \stackrel{x}{\Longrightarrow}_a m\}\rangle$ that builds a strong semantics and define **observational equivalence**s ($=_{aX}$) as:

$$\mathsf{A} =_{aX} \mathsf{C} \stackrel{\text{def}}{=} Abs_a(\mathsf{A}) =_X Abs_a(\mathsf{C})$$

We will use a lower case prefix to depict the abstraction function and an upper case suffix to depict the strong equivalence (*e.g.* F for failure below).

Definition 2 *Observational semantics* $\stackrel{a}{\Longrightarrow}_o$*:*

$s \stackrel{\tau}{\Longrightarrow} t \stackrel{\text{def}}{=} s_0 \stackrel{\tau}{\longrightarrow} s_1, s_1 \stackrel{\tau}{\longrightarrow} s_2, \ldots s_{n-1} \stackrel{\tau}{\longrightarrow} s_n \wedge s = s_1 \wedge t = s_n \wedge \forall_{i \leq n} . Os(s_0) = Os(s_i)$

$n \stackrel{a}{\Longrightarrow}_o m \stackrel{\text{def}}{=} n \stackrel{\tau}{\Longrightarrow} n', n' \stackrel{a}{\longrightarrow} m', m' \stackrel{\tau}{\Longrightarrow} m \wedge (a \in Act \vee Os(n') \neq Os(m'))$

$$\mathsf{A} =_{oF} \mathsf{C} \stackrel{\text{def}}{=} Abs_o(\mathsf{A}) =_F Abs_o(\mathsf{C}) \quad \bullet$$

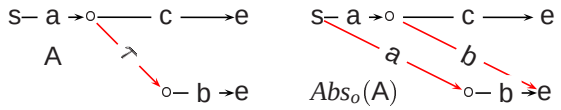


Figure 1. Action abstraction

We extend $Abs_\circ$ to cope with nonterminating components by replacing τ loops with nondeterminism. This can be done assuming divergence is observable or is indistinguishable from deadlock.

Process A in Fig. 2 is an example of what we call *optional divergence*, it could either act fairly and always eventually perform a or alternatively it could stay forever in the s state. What is more which it does is not determined by any outside agent.

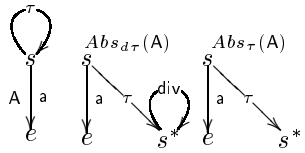


Figure 2. Divergence

Hence we model this divergence as the nondeterministic choice between performing a and being trapped in the s^* state (see Fig. 2). If divergence or internal activity is observable (the green light of [24]) then we would use $Abs_{d\tau}$, else Abs_τ.

Definition 3 *Let* $\mathsf{A} \stackrel{\text{def}}{=} (N_\mathsf{A}, Os_\mathsf{A}, T_\mathsf{A})$,

$D_\tau(N_\mathsf{A}) = N_\mathsf{A} \cup \{d^* \mid s \stackrel{\tau}{\Longrightarrow} t\}$,

$D_\tau(T_\mathsf{A}) = T_\mathsf{A} \cup \{d \stackrel{\tau}{\longrightarrow} d^* \mid s \stackrel{\tau}{\Longrightarrow} t\}$ *and*

$D_d(T_\mathsf{A}) = T_\mathsf{A} \cup \{d^* \stackrel{\text{div}}{\longrightarrow} d^* \mid d \stackrel{\tau^+}{\longrightarrow} d\}$ *then*

$$Abs_\tau(\mathsf{A}) \stackrel{\text{def}}{=} (D_\tau(N_\mathsf{A}), Os_\mathsf{A}, Abs_o(D_\tau(T_\mathsf{A})))$$

$$Abs_{d\tau}(\mathsf{A}) \stackrel{\text{def}}{=} (D_\tau(N_\mathsf{A}), Os_\mathsf{A}, D_d(Abs_o(D_\tau(T_\mathsf{A})))) \quad \bullet$$

The congruence of process algebraic equivalences give them their much sought-after modularity. With this in mind we define component operators, taken from the process literature, in order that our observational equivalences are congruent. We adopt the operators from the process literature: $\sqcap$ internal choice; $\oplus$ choice; ; sequential composition; and $\|_\gamma$ parallel composition. The transitions of nonterminating components can be defined using a set of linear recursive equations $\{X = aX + bY, Y = cZ\}$ (see ACP [1]).

The above set of equalities can be interpreted as giving a mutually recursive definition of $\{X, Y, Z\}$. Alternatively $\{X, Y, Z\}$ can simply be interpreted as a set of states.

An equality can be interpreted as defining the set of actions with the same pre-state. Each pair of the name and state on the right of the equality defines the name and post-state of a transition. For example B in Fig. 3 can be defined by $N_\mathsf{B} \stackrel{\text{def}}{=} \{X, Y, Z\}$ (seen as $\{s, \circ, \text{\ae}\}$), $s_\mathsf{B} \stackrel{\text{def}}{=} \{X, Z\}$, $e_\mathsf{B} \stackrel{\text{def}}{=} \{Z\}$ and $T_\mathsf{B} \stackrel{\text{def}}{=} \{X = aY, Y = cZ, Z = bY\}$.

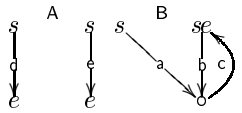


Figure 3. A **and** B

3 A component algebra

We are interested in choice, sequential composition and parallel composition between components. But, how we would wish these operators to affect the state propositions depends upon the meaning of the propositions. Thus the approach in [9], where parallel composition is parameterised by "proposition rules", seems appropriate. Because we wish to compare our operators with those in the process literature we restrict ourselves to the two propositions, one for start and one for end, *i.e.* essentially to LTSs.

Choice has been defined ([1, 25]) on operational semantics with one start state by gluing the two start states together. Here we glue together two sets of start states.

As a first step we define what it means to have a set of nodes in a transition:

$$n \stackrel{a}{\longrightarrow} \{s_1 \ldots s_n\} \stackrel{\text{def}}{=} \{n \stackrel{a}{\longrightarrow} s_1 \ldots n \stackrel{a}{\longrightarrow} s_n\}$$

and use this to define how to glue together two sets of states *e.g.* s_A and s_B.

Let $S' \stackrel{\text{def}}{=} \{s'_1, s'_2 \ldots s'_m\}$ and $S \stackrel{\text{def}}{=} \{s_1, s_2 \ldots s_n\}$ and define the n substitutions $\{s_i / \{(s_i, s'_1), \ldots (s_i, s'_m)\}\}$ for $s_i \in S$ to be written $\{\{S/S \times S'\}\}$. Also, we define the $n + m$ substitutions $\{\{S/S \times S'\}\} \cup \{\{S'/S \times S'\}\}$ to be written $\{\{SS'/S \times S'\}\}$.

Consequently $\{\{s_\mathsf{A} s_\mathsf{B} / s_\mathsf{A} \times s_\mathsf{B}\}\}$ will identify the two sets of nodes s_A and s_B as $s_\mathsf{A}\{\{s_\mathsf{A} s_\mathsf{B} / s_\mathsf{A} \times s_\mathsf{B}\}\}$ and $s_\mathsf{B}\{\{s_\mathsf{A} s_\mathsf{B} / s_\mathsf{A} \times s_\mathsf{B}\}\}$ are both the $n \times m$ set of nodes $s_\mathsf{A} \times s_\mathsf{B}$.

Definition 4 *Operations* $\oplus$, ; τ_A δ_A *and* $\|_\gamma$ *on LTSs* A *and* B

$N_{\mathsf{A}\|_\gamma\mathsf{B}} \stackrel{\text{def}}{=} N_\mathsf{A} \times N_\mathsf{B}$, $e_{\mathsf{A}\|_\gamma\mathsf{B}} \stackrel{\text{def}}{=} e_\mathsf{A} \times e_\mathsf{B}$, $s_{\mathsf{A}\|_\gamma\mathsf{B}} \stackrel{\text{def}}{=} s_\mathsf{A} \times s_\mathsf{B}$ *and*

(where γ *is a partial function from* $Act \times Act$ *to* Act*)* $T_{\mathsf{A}\|_\gamma\mathsf{B}}$ *is defined by*

$$\frac{n \stackrel{a}{\longrightarrow}_\mathsf{A} n'}{\forall_{x \in N_\mathsf{B}} (n, x) \stackrel{a}{\longrightarrow} (n', x)} \qquad \frac{m \stackrel{a}{\longrightarrow}_\mathsf{B} m'}{\forall_{x \in N_\mathsf{A}} (x, m) \stackrel{a}{\longrightarrow} (x, m')}$$

$$\frac{n \stackrel{a}{\longrightarrow}_\mathsf{A} n', m \stackrel{b}{\longrightarrow}_\mathsf{B} m', \gamma(a,b) = c}{(n, m) \stackrel{c}{\longrightarrow} (n', m')}$$

$N_{\mathsf{A}\tau_S} = N_\mathsf{A}$, $s_{\mathsf{A}\tau_S} = s_\mathsf{A}$,$e_{\mathsf{A}\tau_S} = e_\mathsf{A}$,

$$\frac{n \xrightarrow{\mathsf{a}}_\mathsf{A} n', \mathsf{a} \notin S}{n \xrightarrow{\mathsf{a}}_{\mathsf{A}\tau_S} n'} \qquad \frac{n \xrightarrow{\mathsf{a}}_\mathsf{A} n', \mathsf{a} \in S}{n \xrightarrow{\tau}_{\mathsf{A}\tau_S} n'}$$

$N_{\mathsf{A}\delta_S} = N_\mathsf{A}$, $s_{\mathsf{A}\delta_S} = s_\mathsf{A}$,$e_{\mathsf{A}\delta_S} = e_\mathsf{A}$, $\dfrac{n \xrightarrow{\mathsf{a}}_\mathsf{A} n', \mathsf{a} \notin S}{n \xrightarrow{\mathsf{a}}_{\mathsf{A}\delta_S} n'}$

$\mathsf{A} \oplus \mathsf{B} \stackrel{\text{def}}{=} (N_\mathsf{A} \cup N_\mathsf{B}, s_\mathsf{A}, e_\mathsf{A} \cup e_\mathsf{B}, (T_\mathsf{A} \cup T_\mathsf{B}))\{\{s_\mathsf{A}s_\mathsf{B}/s_\mathsf{A} \times s_\mathsf{B}\}\}$

Let $s^*_\mathsf{B} = \{s^*_i \mid s_i \in s_\mathsf{B}\}$ *and* $s^*e^*_\mathsf{B} = \{s^*_i \mid s_i \in s_\mathsf{B} \cap e_\mathsf{B}\}$

$\mathsf{A};\mathsf{B} \stackrel{\text{def}}{=} (N_\mathsf{A} \cup N_\mathsf{B} \cup s^*_\mathsf{B}, s_\mathsf{A}, e_\mathsf{B} \cup s^*e^*_\mathsf{B}, T_\mathsf{A} \cup T_\mathsf{B} \cup \{s^*_i \xrightarrow{\mathsf{x}} n \mid s_i \in s_\mathsf{B} \wedge s_i \xrightarrow{\mathsf{x}} n\})\{\{e_\mathsf{A}s^*_\mathsf{B}/e_\mathsf{A} \times s^*_\mathsf{B}\}\}$

$\mathsf{A} \sqcap \mathsf{B} \stackrel{\text{def}}{=} (N_\mathsf{A} \cup N_\mathsf{B}, s_\mathsf{A} \cup s_\mathsf{B}, e_\mathsf{A} \cup e_\mathsf{B}, T_\mathsf{A} \cup T_\mathsf{B})$ ∘

Our definition of choice, see $\mathsf{A} \oplus \mathsf{B}$ Fig. 4, is based on that from [25] where the start states of both processes are simply glued together. This is unlike that of ACP where the processes are first root unwound. We amend the definition of [25] in the obvious way to cope with a set of start states.

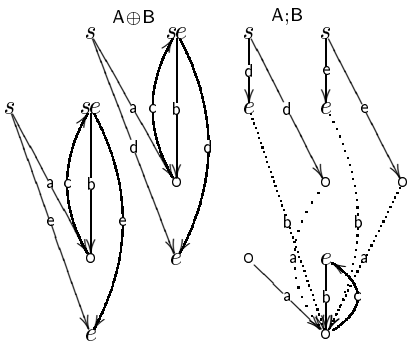


Figure 4. $\mathsf{A} \oplus \mathsf{B}$ **and** $\mathsf{A};\mathsf{B}$

In our definition of sequential composition a set of transitions is added to those of the operands. In the example $\mathsf{A};\mathsf{B}$ Fig. 4 these added transitions are shown with dotted arrows.

The details of sequential composition is dependent upon the details of successful termination but CSP and ACP treat successful termination differently. Our definitions are based on ACP not CSP (adding the actions $\{n \xrightarrow{\mathsf{x}} s \mid e \in e_\mathsf{A} \wedge s \in s_\mathsf{B} \wedge n \xrightarrow{\mathsf{x}} e\}$ would result in a more CSP-like definition). For further discussion see Section 4.3 later.

We show that either of our two definitions of abstraction Abs_τ and $Abs_{d\tau}$ preserve congruence w.r.t. $\{\sqcap, ;, \oplus, \|_\gamma\}$ (proofs in Appendix).

Lemma 1 *If* $=_X$ *is congruent w.r.t.* $\{\sqcap, ;, \oplus, \|_\gamma\}$ *then so are* $=_{\tau X}$ *and* $=_{d\tau X}$.

Lemma 2 $=_{\tau F^s}$ *and* $=_{d\tau F^s}$ *are congruent w.r.t.* $\sqcap, ;, \oplus, \|_\gamma$.

4 Comparison

Hoare and He say [13][p.198] "The main distinguishing feature of CSP is to define a hiding operator that succeeds in total concealment of internal actions." Here we have constructed an operational semantics consistent with this.

4.1 Choice

In CSP (but not CCS/ACP) τ actions $s \xrightarrow{\tau} n_2$ and $s \xrightarrow{\tau} n_1$ can model "the process could be in state n_1 or n_2 but we cannot know which" (see $\mathsf{a} \sqcap \mathsf{b}$ in Fig. 5). We interpret CSP external choice $\Box$ and ACP choice $+$ to be the same and use the different role of τ actions in CSP and ACP/CCS to explain why $\mathsf{A} \Box \mathsf{c} \neq \mathsf{A} + \mathsf{c}$ in Fig. 6.

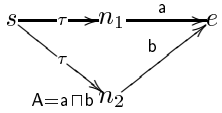
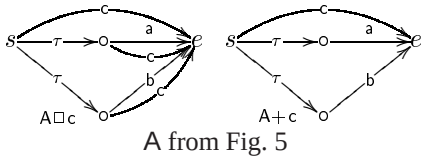


Figure 5. CSP $\sqcap$



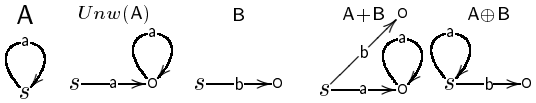
A from Fig. 5

Figure 6. CSP $\Box$ **and ACP** $+$

The renaming of observable actions as τ actions, $\tau_{\{\mathsf{a}\}}$, does not distribute through CSP choice, whereas $\tau_{\{\mathsf{a}\}}$ does distribute through CCS/ACP choice and our $\oplus$.

The internal or τ actions are unobservable in that when performed they cannot synchronise with an (observing) action of some context. The denotational semantics of CSP, CFFD and NDFD all use *stability* to define congruence w.r.t. choice whereas our observational semantics and that of CCS and ACP keep $s_\mathsf{A} \xrightarrow{\tau} \circ$. Although we believe this to be of little importance it does introduce small discrepancies in what would other wise be the same equivalences.

Root unwinding or not. We have motivated our congruence by using distinguishing states that are in the interface from those that are not. This introduces a question: what happens if a process returns to one of the start states that is in the interface?



$\mathsf{A} =_F Unw(\mathsf{A})$ $\mathsf{A} \neq_{F^s} Unw(\mathsf{A})$ and $\mathsf{A} \neq_{F^+} Unw(\mathsf{A})$

Figure 7. Choice + or $\oplus$

Choice on LTS as defined in [23, 1] is $+$ (see Fig. 7). It first *root unwinds* the LTS then identifies start states. An alternative in [3] is to not allow cycles to return to the initial node, that is to say consider only unwound graphs. Root

unwinding allows us to view loops as mere "sugar" for their true meaning as an acyclic LTS.

Here choice is modelled by gluing together the root nodes of two LTSs without performing root unwinding. This is not new: it appears in [25] where such a definition of choice is given as limits in categories of LTSs and Petri nets.

By changing the definition of choice, what is required of a congruence is changed. With the semantics in [25] $\mathsf{A} \neq Unw(\mathsf{A})$, as we would expect from our desire to distinguish states that are in the interface from those that are not.

We will show later (see Fig. 10) that by not assuming that cyclic semantics is equivalent to their unwound semantics allows a natural decomposition of some components; root unwinding would prevent this.

4.2 Divergence and abstraction

Action-based approaches frequently use a single syntactic class (of actions) and use recursion to define nonterminating components, which are given a fixed point semantics. The need to have a unique fixed point semantics has had a strong influence on the semantics of CSP [19, p215] and unifying theories of programming [13, 2.6, 2.7]. Alternatively, a very powerful argument, for state-based systems, has been made [10] in support of using refinement semantics rather than fixed point semantics.

When modelling State-and-Action systems it is natural to use two syntactic classes, one for states and one for actions. Using such a formalism, recursion and fixed points are not needed to define nonterminating components.

Both divergence and abstraction have been treated in various ways in the literature.

CCS and ACP: τ actions are not observable nor refusable and divergence can be ignored. This is quite natural if we apply a *fair choice* interpretation, see process A, Fig. 2. The fact that CCS and ACP need to keep some of the τ actions in their definition of observational semantics somewhat detract from the claim that τ actions are unobservable. This is needed for observational bisimulation to be congruent with respect to an interleaving interpretation of parallel composition. But these τ actions would not be needed had they used failure equivalence as their strong equivalence (see [22]);

CSP eager abstraction: τ actions are not observable nor refusable and divergence becomes chaos. This has been described as: "of rather limited use as a means of abstraction" [19][p. 296] and "not really adequate with respect to the operational interpretation of this phenomenon" [15];

CSP lazy abstraction: τ actions are not observable but are **refusable** and divergence can be ignored (see [19] [p.297]);

Here: τ actions are not observable nor refusable and *optional divergence* (see Fig. 2) becomes the nondeterministic choice between eventually performing an action and remaining for ever live-locked.

4.3 Sequential composition

Sequential composition is defined using an explicit representation of the successful *termination* of a process. In CSP termination *SKIP* "can always be chosen" when offered, *i.e.* $SKIP\Box\mathsf{a} \rightarrow STOP = (SKIP\Box\mathsf{a} \rightarrow STOP) \sqcap SKIP$. This is quite different from ACP termination ϵ which cannot always be chosen. This can be seen in the construction of $((\mathsf{a};\epsilon)+\epsilon) \parallel (\mathsf{a};\epsilon)$ (see [1] [p. 76]) where the ability of one of the components to initially terminate is simply lost. Because of these differences we will avoid comparing congruence w.r.t. sequential composition from CSP and our definition which follows that of ACP.

4.4 NDFD Divergence without chaos

In [22, 23][1] they construct a denotational semantics without interpreting divergence as chaos. Stability and divergence are defined on the strong operational semantics. Failure semantics is defined on the observational semantics ($\Rightarrow_o$) and finally stability, divergence and failures are all used in the definition of *NDFD*.

Let A and B be LTSs.

$sta(\mathsf{A}) \stackrel{\text{def}}{=} s \stackrel{\tau}{\not\longrightarrow} \wedge\ s \in s_{\mathsf{A}}$ and

$div(\mathsf{A}) \stackrel{\text{def}}{=} \{\theta \mid s \stackrel{\theta}{\Longrightarrow}_o n \wedge s \in s_{\mathsf{A}} \wedge n \xrightarrow{\tau\tau^*} n\}$

$fail(\mathsf{A}) \stackrel{\text{def}}{=} \{(\theta, X) \mid \exists_n s \stackrel{\theta}{\Longrightarrow}_o n \wedge s \in s_{\mathsf{A}} \wedge \forall_{\mathsf{x} \in X}\ n \stackrel{\mathsf{x}}{\not\Longrightarrow}_o\}$

$dfail(\mathsf{A}) \stackrel{\text{def}}{=} \{(\theta, X) \mid (\theta, X) \in fail(\mathsf{A}) \vee \theta \in div(\mathsf{A})\}$

$\mathsf{A} =_{NDFD} \mathsf{B} \stackrel{\text{def}}{=} sta(\mathsf{A}) = sta(\mathsf{B}) \wedge dfail(\mathsf{A}) = dfail(\mathsf{B}) \wedge div(\mathsf{A}) = div(\mathsf{B})$

A LTS is *well-terminating* if $n \xrightarrow{a}$ implies $n \notin e_{\mathsf{A}}$.

Lemma 3 *For stable, unwound and well-terminating processes.*

$$\mathsf{A} =_{d\tau F} \mathsf{B} \Leftrightarrow \mathsf{A} =_{NDFD} \mathsf{B}$$

We have provided an operational interpretation of action abstraction that transforms divergence into nondeterminism. The above result tells us that computing failure equivalence on the observational semantics gives the same result as computing NDFD equivalence on the strong semantics. The restriction to stable, unwound processes is explained in Section 4.1 and the restriction to well-terminating processes is explained in Section 4.3.

[1] We do not need traces as we consider only finite state processes.

5 Z components

We will use Z schemas to define both operations and state. Unfortunately Z leaves as informal any attempt to localise state or action, so here we informally follow the convention of simply allowing schemas to be grouped together. We call a group containing the schemas *State, init, final* and *OP* (a set of named operation schemas) a component. We adopt the conventions, from process algebra, that state is local to a process and τ operations are internal, *i.e.* under local control and not observable from outside.

As these Z components can be given a LTS semantics (see [21, 8, 18, 17] for details) we can apply process operators to them to build a LTS specification in a modular fashion.

5.1 Example

A vending machine accepts an electronic money card, then allows the user to request cups of tea if the card has sufficient funds, and finally to remove the card. We feel it to be natural to decompose this into these three parts: one-Insert_card; two-Desp_tea; and three-Remove_card. We can then define the composition of the whole vending machine as the sequential composition of the three components.

After inserting the card the machine can be in one of two states, *i.e.* sufficient funds or insufficient funds. Because the next process, Desp_tea, acts differently depending on which state the machine is in we choose to model these two states by giving Desp_tea two initial states and let our definition of sequential composition (Definition 4) introduce the nondeterminism.

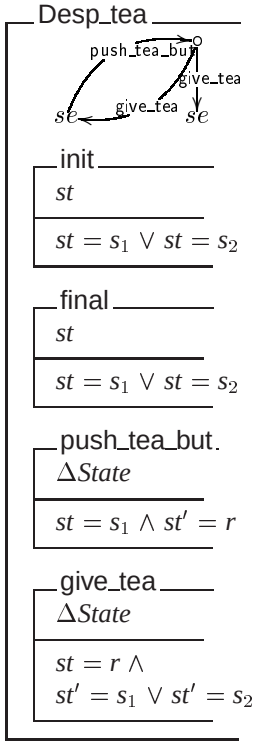


Figure 8. Desp_tea

For brevity, in Fig. 9 and Fig. 10, we assume *State* is defined as the obvious enumerated data type and where needed we assume init $\stackrel{\text{def}}{=}$ $[State \mid st = s]$ and final $\stackrel{\text{def}}{=}$ $[State \mid st = e]$.

The definition of Desp_tea in Fig. 9 would not be possible had we used the CCS, ACP or CSP semantics that equates processes with their root unwinding. We have defined the effect of the process operator on the operational semantics of the Z processes; what we have not done is define these operators directly for Z.

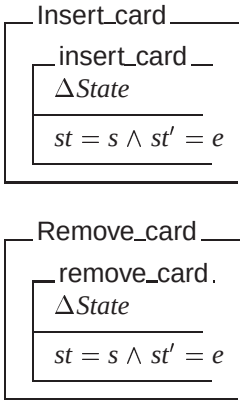


Figure 9. Insert_card and Remove_card

We can construct the operational semantics of Insert_card; Desp_tea; Remove_card which can be simplified by pruning unreachable operations and identifying bisimilar nodes, in order to arrive at the LTS Card_tea in Fig. 10. The Z text can then be constructed from the LTS.

We can define Card_tea_value(t,v), a value passing version of our vending machine. It is easy to verify that, for all values of v and t, Card_tea_value(t,v) is a Z data refinement of Card_tea. To verify that two cards cannot be inserted without an intervening removal of a card we can use the more abstract Card_tea because Z refinement is singleton failure refinement and hence trace subset refinement. We can simplify Card_tea (as all we are interested in are the actions insert_card and remove_card) by abstracting the other actions. Finally the answer is obvious from the simplified operational semantics. But we could not have used CSP's hiding, *i.e.* eager abstraction, and obtained the correct answer. And we would not, in general, wish to use CSP's lazy abstraction as we interpret τ actions as under local control, *i.e.* as not refusable.

6 Conclusion

To model components we give an equal status to states and events. We require that our components have an interface and that components with the same interface are "observationally" equivalent. In particular we allow both states and events to be a part of the interface. To this end we use two syntactic classes, one for states and one for events, and consequently we do not need recursion or unique fixed points to define the semantics of nonterminating components.

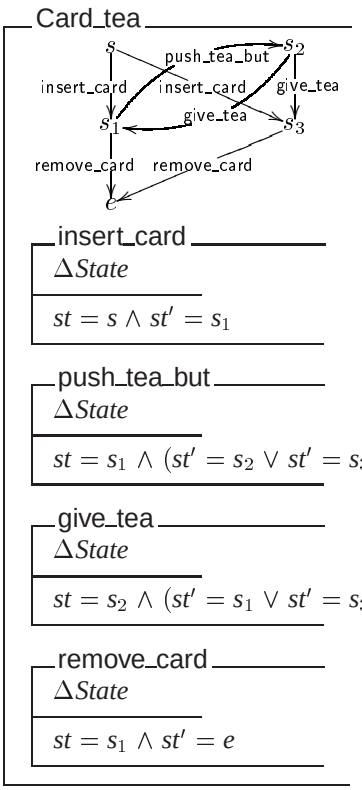


Figure 10. Card_tea

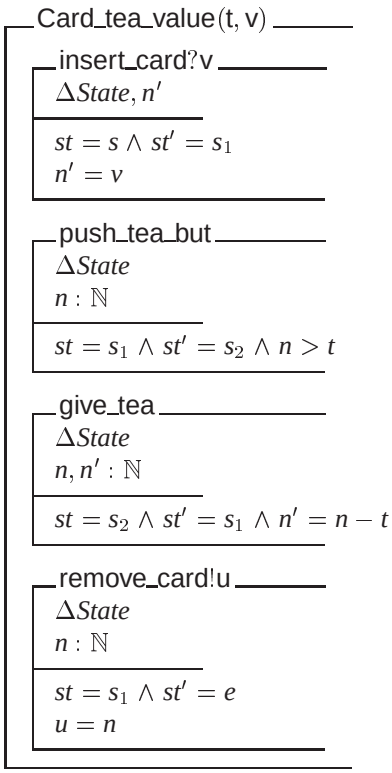


Figure 11. Card_tea_value(t, v)

In order to preserve a component's interface we reject the root unwinding built into the semantics of many process algebras. In this regard our approach is based on that of Winskel and Nielson [25]. We demonstrate some practical advantages of this approach in Section 5.

To take advantage of the well-known [21, 8, 17] isomorphism between state-based relational semantics and event-based operational semantics we use operational rather than denotational semantics. There is a natural way of defining observational equivalences in two steps: first, build an observational semantics from the strong semantics; then, apply a strong equivalence to the newly built observational semantics Section 2.3. We take advantage of this and define an observational semantics that models what can be observed of divergent processes. This can subsequently be applied with any strong equivalence.

In [5] they define a singleton failures semantics for ADTs but hiding has to be restricted to exclude the possibility of considering divergent ADTs. We can extend this work to consider nonterminating processes by first applying our definition of abstraction and then applying their definitions to the resulting observational semantics. This would result in a singleton version of NDFD in place of their singleton version of failure semantics.

The work in [17] gives testing characterisations that "explain" the difference between several known refinements including LOTOS's extension [6], conformance [7], may and must testing [11], failure refinement and singleton failure refinement [5]. But all these refinements ignore divergence and hence apply only to terminating processes. We can construct the observational semantics defined here and subsequently apply the work in [17] to the observational semantics. This extends the original work to cover nonterminating processes where divergence is not ignored.

References

[1] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, 1990.

[2] M. Barnett and W. Schulte. The ABCs of Specification: AsmL, Behavior, and Components. *Informatica*, 25(4), November 2001.

[3] J. A. Bergstra, J. W. Klop, and E.-R. Olderog. Failures without chaos: A new process semantics for fair abstraction. In M. Wirsing, editor, *Formal Description of Programming Concepts*, volume III of *IFIP*, pages 77–103. Elsevier, 1987.

[4] T. Bolognesi and E. Borger. Abstract State Processes. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Ab-*

stract State Machines, Advances in Theory and Practice, 10th International Workshop, ASM 2003, Taormina, Italy, March 3-7, Proceedings, volume 2589 of *Lecture Notes in Computer Science*, pages 218–228, 2003.

[5] C. Bolton and J. Davies. A singleton failures semantics for Communicating Sequential Processes. Research Report PRG-RR-01-11, Oxford University Computing Laboratory, 2001.

[6] E. Brinksma and G. Scollo. Formal notions of implementation and conformance in LOTOS. Technical Report INF-86-13, Twente University of Technology, Department of Informatics, Enschede, The Netherlands, 1986.

[7] E. Brinksma, G. Scollo, and C. Steenbergen. LOTOS specifications, their implementation and their tests. In B. Sarikaya and G. V. Bochmann, editors, *Protocol Specification, Testing and Verification*, volume VI, pages 349–360. North-Holland, 1986.

[8] J. Derrick and E. Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. Formal Approaches to Computing and Information Technology. Springer, May 2001.

[9] H. Hansen, H. Virtanen, and A. Valmari. Merging state-based and action-based verification. In *Proceedings of the Third International Conference on Application of Concurrency to System Design (ACSD'03)*, Guimara es, Portugal, June 2003. IEEE Computer Society.

[10] E. C. R. Hehner and A. M. Gravell. Refinement semantics and loop rules. In *World Congress on Formal Methods (2)*, pages 1497–1510, 1999.

[11] M. Hennessy. *Algebraic Theory of Processes*. The MIT Press, 1988.

[12] C. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science, 1985.

[13] C. Hoare and H. Jifeng. *Unifying Theories of Programming*. Prentice Hall International Series in Computer Science, 1998.

[14] R. Kaivola and A. Valmari. The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear temporal logic. In *International Conference on Concurrency Theory LNCS 630*, pages 207–221, 1992.

[15] G. Leduc. Failure-based Congruences, Unfair Divergences and New Testing Theory. In S. T. Vuong and S. T. Chanson, editors, *PSTV*, volume 1 of *IFIP Conference Proceedings*. Chapman & Hall, 1994.

[16] R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989.

[17] S. Reeves and D. Streader. Comparison of Data and Process Refinement. In J. S. Dong and J. C. P. Woodcock, editors, *ICFEM 2003*, LNCS 2885, pages 266–285. Springer-Verlag, 2003.

[18] S. Reeves and D. Streader. State-based and process-based value passing. In *Proceedings of St.Eve @ FM'03*. 2003. Available at www.cs.waikato.ac.nz/˜stever/06-Reeves-Streader.pdf.

[19] A. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall International Series in Computer Science, 1997.

[20] G. Smith. A Fully Abstract Semantics of Classes for Object-Z. *Formal Aspects of Computing*, 7(3):289–313, 1995.

[21] G. Smith. A Semantic Integration of Object-Z and CSP for the Specification of Concurrent Systems. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313, pages 62–81. Springer-Verlag, 1997.

[22] A. Valmari and M. Tienari. An improved failure equivalence for finite-state systems with a reduction algorithm. In *Protocol Specification, Testing and Verification*, IFIP XI. North-Holland, 1991.

[23] A. Valmari and M. Tienari. Compositional Failure-based Semantics Models for Basic LOTOS. *Formal Aspects of Computing*, 7(4):440–468, 1995.

[24] R. L. van Glabbeek. The linear time - branching time spectrum I. the semantics of concrete sequential processes. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier Science, Amsterdam, The Netherlands, 2001.

[25] G. Winskel and M. Nielsen. Models for concurrency. Technical Report DAIMI PB 429, Computer Science Dept. Aarhus Universty, 1992.

[26] J. C. P. Woodcock and A. L. C. Cavalcanti. The Semantics of Circus. In M. C. H. Didier Ber, Jonathan P. Bowen and K. Robinson, editors, *ZB 2002 Formal Specification and Development in Z and B LNCS 2272*, pages 184–203. Springer-Verlag, 2002.

Appendix

Lemma 4 *For stable, unwound and well-terminating processes* A *and* B.

$$\mathsf{A} =_{d\tau F} \mathsf{B} \Leftrightarrow \mathsf{A} =_{NDFD} \mathsf{B} \qquad \textit{Lemma 3 in text}$$

Proof $\mathsf{A} =_{NDFD} \mathsf{B} \stackrel{\text{def}}{=} dfail(\mathsf{A}) = dfail(\mathsf{B}) \wedge div(\mathsf{A}) = div(\mathsf{B})$

Part1. $\mathsf{A} =_{d\tau F} \mathsf{B} \Rightarrow \stackrel{\text{def}}{=} dfail(\mathsf{A}) = dfail(\mathsf{B}) \wedge div(\mathsf{A}) = div(\mathsf{B})$

If $\theta \in div(\mathsf{A}) \Leftrightarrow \theta\mathsf{div} \in Tr(Abs_{d\tau}(\mathsf{A}))$ and $\theta\mathsf{div} \in Tr(Abs_{d\tau}(\mathsf{B})) \Leftrightarrow \theta \in div(\mathsf{B})$.

Hence $div(\mathsf{A}) = div(\mathsf{B})$ [1]

If $(\theta, X) \in dfail(\mathsf{A})$

Case. 1 $\theta \notin div(\mathsf{A})$ clearly $(\theta, X \cup \{\mathsf{div}\}) \in fail(Abs_{d\tau}(\mathsf{A}))$ hence $(\theta, X \cup \{\mathsf{div}\}) \in fail(Abs_{d\tau}(\mathsf{B}))$ and $(\theta, X) \in dfail(\mathsf{B})$.

Case. 2 $\theta \in div(\mathsf{A})$ from [1] $\theta \in div(\mathsf{B})$ and hence $(\theta, X) \in dfail(\mathsf{B})$

hence from case 1 and 2 $(\theta, X) \in dfail(\mathsf{B})$

Part 2. $\mathsf{A} =_{d\tau F} \mathsf{B} \Leftarrow \stackrel{\text{def}}{=} dfail(\mathsf{A}) = dfail(\mathsf{B}) \wedge div(\mathsf{A}) = div(\mathsf{B})$

If $(\theta, X) \in fail(Abs_{d\tau}(\mathsf{A}))$

Case. 1 $\theta \notin div(\mathsf{A})$: clearly $(\theta, X) \in dfail(\mathsf{A})$ hence $(\theta, X) \in dfail(\mathsf{B})$ and $(\theta, X) \in fail(\mathsf{B})$ and $(\theta, X) \in fail(Abs_{d\tau}(\mathsf{B}))$.

Case. 2 $\theta \in div(\mathsf{A})$: clearly $\theta \in div(\mathsf{B})$ and hence $(\theta, X) \in fail(Abs_{d\tau}(\mathsf{B}))$

hence from case 1 and 2 $\mathsf{A} =_{d\tau F} \mathsf{B}$

Finally from parts 1 and 2. •

Properties about $=_{NDFD}$ that can be found from [23, 14] include the congruence of $=_{NDFD}$ with respect to LOTOS operators and that it preserves next-time-less linear temporal logic. Hence $=_{d\tau F}$ is congruent with respect to LOTOS operators.

Congruence results for and $=_{F^s}$

These results are quite standard. Adding $Os(\_)$ to the trace changes little for the strong semantics. The only points of slight interest are that adding sets of start states does not change the congruence results. nor does adding $Os(\_)$ change the congruence result for $\oplus$.

Lemma 5 *Let* $op \in \{||_\gamma, \oplus, ;, \sqcap\}$

If $\mathsf{A} =_{F^s} \mathsf{A}^\star$ *and* $\mathsf{B} =_{F^s} \mathsf{B}^\star$ *then* $\mathsf{A}op\mathsf{B} =_{F^s} \mathsf{A}^\star op \mathsf{B}^\star$

Proof The results follows from $(\theta^s, X) \in F^s(\mathsf{A}op\mathsf{B}) \Leftrightarrow (\theta^s, X) \in F^s(\mathsf{A}^\star op \mathsf{B}^\star)$.

Part 1 - $||_\gamma$. Let $\gamma = \varnothing$. From the construction if $(\theta^s, X) \in F^s(\mathsf{A} \;||_\gamma\; \mathsf{B})$ then $(s_a, s_b) \xrightarrow{\theta^s} (n_a, n_b)$ and $s_a \xrightarrow{\theta^s_a} n_a \in T_\mathsf{A} \wedge s_b \xrightarrow{\theta^s_b} n_b \in T_\mathsf{B}$ where (a) the action of θ^s are an interleaving of the action of θ^s_a and θ^s_b; and where (b) $Os((n_a, n_b)) \stackrel{\text{def}}{=} Os(n_a) \cap Os(n_b)$.

As $\pi((n_a, n_b)) = \pi(n_a) \cup \pi(n_b)$ then $(\theta^s, X) \in F^s(\mathsf{A} \;||_\gamma \mathsf{B})$ if and only if $(\theta^s_a, X_a) \in F^s(\mathsf{A})$, $(\theta^s_b, X_b) \in F^s(\mathsf{B})$ and $X = X_a \cap X_b$

As $(\theta^s_a, X_a) \in F^s(\mathsf{A}) \Leftrightarrow (\theta^s_a, X_a) \in F^s(\mathsf{A}^*)$ and $(\theta^s_b, X_b) \in F^s(\mathsf{B}) \Leftrightarrow (\theta^s_b, X_b) \in F^s(\mathsf{B}^*)$ hence $(\theta^s, X) \in F^s(\mathsf{A}^* \;||_\gamma \mathsf{B}^*)$

Let $\gamma \neq \varnothing$. It is easy to see that the existence of any synchronisation actions can be computed from the failure sets of the components. Let $\gamma(\mathsf{a}, \mathsf{b}) = \mathsf{c}$ then if $(\theta^s, X) \in F^s(\mathsf{A} \;||_\gamma \mathsf{B})$ and $(\theta^s_a, X_a) \in F^s(\mathsf{A})$, $(\theta^s_b, X_b) \in F^s(\mathsf{B})$ then $\mathsf{c} \in X \Leftrightarrow \mathsf{a} \in X_a \vee \mathsf{a} \in X_b \vee \mathsf{b} \in X_a \vee \mathsf{b} \in X_b$.

Part 2 - $\oplus$. Let $(\theta^s, X) \in F^s(\mathsf{A} \oplus \mathsf{B})$ then by case analysis on $\mid \theta^s \mid$ the number of action names in θ^s.

Case 1. $\mid \theta^s \mid = 0$ $((), X) \in F^s(\mathsf{A} \oplus \mathsf{B}) \Leftrightarrow \exists\, s \in s_{\mathsf{A}\oplus\mathsf{B}}. \forall \mathsf{x} \in X. s \xrightarrow{\mathsf{x}} \not\rightarrow$) from the construction $\exists\, s^a \in s_\mathsf{A}, s^b \in s_\mathsf{B}. \forall \mathsf{x} \in X. s^a \xrightarrow{\mathsf{x}} \not\rightarrow \wedge\; s^b \xrightarrow{\mathsf{x}} \not\rightarrow)$. Clearly for $\mid \theta^s \mid = 0$ $F^s(\mathsf{A} \oplus \mathsf{B}) = F^s(\mathsf{A}) \cap F^s(\mathsf{B})$ and the result follows.

Case 2. $\mid \theta^s \mid > 0$ Let $\exists_{n_1 \ldots n_{i+1}}.(n_1, \theta^s \mid_1, n_2), \ldots (n_i, \theta^s \mid_i, n_{i+1}) \in T_{\mathsf{A}\oplus\mathsf{B}}$

case-a. If $\forall_{2 \leq x \leq i+1}\, s \notin Os(n_x)$ then clearly $(\theta^s, X) \in F^s(\mathsf{A} \oplus \mathsf{B})$ implies $(\theta^s, X) \in F^s(\mathsf{A})$ or $(\theta^s, X) \in F^s(\mathsf{B})$.

As $\mathsf{A} =_{F^s} \mathsf{A}^\star$ and $\mathsf{B} =_{F^s} \mathsf{B}^\star$ we have

$(\theta^s, X) \in F^s(\mathsf{A}^\star)$ or $(\theta^s, X) \in F^s(\mathsf{B}^\star)$.

and similarly $(\theta^s, X) \in F^s(\mathsf{A}^\star \oplus \mathsf{B}^\star)$.

case-b. If $\neg\; \forall_{2 \leq x \leq i+1}\, s \notin Os(n_x)$ then because we only need consider finite traces their we can use $\{n_j \mid s \in Os(n_j)\}$ to impose a finite partition on θ^s and for each partition we can reason exactly as for case-a.

Part 3 - ;. $(\theta^s, X) \in F^s(\mathsf{A};\mathsf{B}) \Leftrightarrow \exists\, n.(s \xrightarrow{\theta^s} n \wedge s \in s_\mathsf{A} \wedge \forall \mathsf{x} \in X. n \xrightarrow{\mathsf{x}} \not\rightarrow)$ and by case analysis on $n \in N_\mathsf{A}$, $n \in e_\mathsf{A} \times s_\mathsf{B}$ or $n \in N_\mathsf{B}$.

Case 1. $n \in N_\mathsf{A}$ Let $(\theta^s, X) \in F^s(\mathsf{A};\mathsf{B})$ As a trace can never have left nodes of A and from the construction of ; $\pi(n)$ in $\mathsf{A};\mathsf{B}$ must, in this case, be the same as in A. Hence $(\theta^s, X) \in F^s(\mathsf{A})$. As $\mathsf{A} =_{F^s} \mathsf{A}^\star$ we have $(\theta^s, X) \in F^s(\mathsf{A}^\star)$ and similarly $(\theta^s, X) \in F^s(\mathsf{A}^\star;\mathsf{B}^\star)$.

Case 2. $n = (e, s) \in e_\mathsf{A} \times s_\mathsf{B}$ Let $(\theta^s, X) \in F^s(\mathsf{A}op\mathsf{B})$ A trace can never have left nodes of A and the only node that has been changed in the construction of $\mathsf{A};\mathsf{B}$ is the last i.e. $n = (e, s)$ hence the only change to ρ could be between $Os(e)$ and $Os(n)$. We now show that any change is the same for all failure equivalent B. Clearly $Os(e) - \{e\} = Os(n)$ unless $s \in e_\mathsf{B}$ then $e \in Os(n)$. But as $\mathsf{B} =_{F^s} \mathsf{B}^\star$ we know $s \in e_\mathsf{B}$ implies $\exists_{s'} \in e_{\mathsf{B}^*} \cap s_{\mathsf{B}^*}$ hence $\exists_{n'=(e,s')}.e \in Os(n')$.

Hence if $s^a \xrightarrow{\theta^s} (e, s) \wedge s^a \in s_\mathsf{A}$ in $\mathsf{A};\mathsf{B}$ then $s^a \xrightarrow{\theta^s} (e, s') \wedge s^a \in s_{\mathsf{A}^*}$ in $\mathsf{A}^*;\mathsf{B}^*$.

By construction $\pi((e, s))$ in $\mathsf{A};\mathsf{B}$ must be $\pi_\mathsf{A}(e) \cup \pi_\mathsf{B}(s)$ hence $X \in Refusal(\rho, \mathsf{A};\mathsf{B})$ then $\exists_{X_\mathsf{A}, X_\mathsf{B}}.X = X_\mathsf{A} \cap X_\mathsf{B} \wedge X_\mathsf{A} \in Refusal(\rho, \mathsf{A}) \wedge X_\mathsf{B} \in Refusal((), \mathsf{B})$. Hence $X \in Refusal(\rho, \mathsf{A}^*;\mathsf{B}^*)$ and $X \in Refusal(\rho, \mathsf{A}^*;\mathsf{B}^*)$. Hence $(\rho, X) \in F^s(\mathsf{A}^* op \mathsf{B}^*)$

Case 3. $n \in N_\mathsf{B}$ Any trace of $\mathsf{A};\mathsf{B}$ that ends at $n \in N_\mathsf{B}$ can be decomposed into an initial component σ in A and a final component ρ in B. When $(\sigma^s \theta^s, X) \in F^s(\mathsf{A};\mathsf{B})$ then $\exists\, \sigma^{s'} \theta^{s'}.\sigma^{s'} \in Ctr(\mathsf{A}) \wedge (\theta^{s'}, X) \in F^s(\mathsf{B})$.

From $\mathsf{A} =_{F^s} \mathsf{A}^\star$ we get $\sigma' \in Ctr(\mathsf{A}^\star)$ and from $\mathsf{B} =_{F^s} \mathsf{B}^\star$ we get $(\theta^{s'}, X) \in F^s(\mathsf{B}^\star)$. Finally we have $(\sigma^s \theta^s, X) \in F^s(\mathsf{A}^\star;\mathsf{B}^\star)$.

$s_\mathsf{B} \in e_\mathsf{B}$ implies $s_{\mathsf{B}^\star} \in e_{\mathsf{B}^\star}$

Part 4- $\sqcap$. From definition. •

Congruence results for $=_{d\tau F^s}$ and $=_{\tau F^s}$

The main use of $Os(\_)$ is to give congruence results for the observational semantics. It does this by preventing abstraction from removing $n \xrightarrow{\tau} m$ where $Os(n) \neq Os(m)$.

Lemma 6 *If* $=_X$ *is congruent w.r.t. op and* Abs_x *distributed through op then* $=_{xX}$ *is congruent w.r.t. op.*

Proof Assume $\mathsf{A}_\mathsf{i} =_{xX} \mathsf{B}_\mathsf{i}$ then $Abs_x(\mathsf{A}_\mathsf{i}) =_X Abs_x(\mathsf{B}_\mathsf{i})$ def $=_{xX}$

then $op(Abs_x(\mathsf{A}_\mathsf{i})) =_X op(Abs_x(\mathsf{B}_\mathsf{i}))$ $=_X$ is congruent w.r.t. *op*

then $(Abs_x(op\mathsf{A}_\mathsf{i})) =_{xX} (Abs_x(op(\mathsf{B}_\mathsf{i}))$ Abs_x distributed through *op*

implies $op(\mathsf{A}_\mathsf{i}) =_{xX} op(\mathsf{B}_\mathsf{i})$ def $=_{xX}$ •

Lemma 7 *Let* $op \in \{\sqcap, ;, \oplus, \|_\gamma\}$

$$Abs_\tau(\mathsf{A} op \mathsf{B}) =_{oX} Abs_\tau(\mathsf{A}) op Abs_\tau(\mathsf{B})$$
$$Abs_{d\tau}(\mathsf{A} op \mathsf{B}) =_{oX} Abs_{d\tau}(\mathsf{A}) op Abs_{d\tau}(\mathsf{B})$$

Proof There is an obvious bijection between the nodes on the LTS on either side of the equality. Equally obviously this bijection relates nodes that were divergent to nodes that were divergent and hence:

$$D_\tau(\mathsf{A} op \mathsf{B}) =_{oX} D_\tau(\mathsf{A}) op D_\tau(\mathsf{B})$$
$$D_d(\mathsf{A} op \mathsf{B}) =_{oX} D_d(\mathsf{A}) op D_\tau(\mathsf{B})$$

The final step that needs a little thought is:

$$Abs_\circ(\mathsf{A} op \mathsf{B}) =_{\circ X} Abs_\circ(\mathsf{A}) op Abs_\circ(\mathsf{B})$$

This can be seen to be true for ; and $\oplus$ because all τ actions in A / B that abstraction might use with actions in B / A are not removed by abstraction and hence can be abstracted in the computation of $=_{\circ X}$.
Nothing need to be said about $\sqcap$ and $\|_\gamma$ is standard from the literature.

•

Lemma 8 *If* $\mathsf{A} =_{\tau F^s} \mathsf{A}^1$ *then* $\tau_I(\mathsf{A}) =_{\tau F^s} \tau_I(\mathsf{A}^1)$

Proof It is well known that: if $\mathsf{A} =_{oF^s} \mathsf{A}^1$ then $\tau_I(\mathsf{A}) =_{oF^s} \tau_I(\mathsf{A}^1)$ is true for terminating processes but problems arise with τ loops. As D_τ removes all τ loops all we need to show is:

If $\mathsf{A} =_{F^s} \mathsf{A}^1$ then $D_\tau(\mathsf{A}) =_{F^s} D_\tau(\mathsf{A}^1)$.

This is clear from definitions. •

Lemma 9 *If* $=_X$ *is congruent w.r.t.* $\{\sqcap, ;, \oplus, \|_\gamma\}$ *then so are* $=_{\tau X}$ *and* $=_{d\tau X}$

Lemma 1 in text

Proof from Lemma 6 and Lemma 7. •

Lemma 10 $=_{\tau F^s}$ *and* $=_{d\tau F^s}$ *are congruent w.r.t.* $\sqcap, ; , \oplus, \|_\gamma$.

Lemma 2 in text

Proof from Lemma 5, Lemma 9. •