Università di Pisa
Dipartimento di Informatica

Technical Report : TR-97-18

Failures, Finiteness, and Full Abstraction

Franck van Breugel

September 09, 1997

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

Failures, Finiteness, and Full Abstraction

Franck van Breugel

Università di Pisa, Department of Computer Science
Corso Italia 40, 56125 Pisa, Italy

franck@di.unipi.it

Abstract

For a simple CCS-like language a trace semantics and a failure semantics are presented. The failure semantics is shown to be fully abstract with respect to the trace semantics *if and only if* the set of internal actions is infinite.

Introduction

In this paper we focus on the *full abstraction* problem. Already in the early seventies this issue was raised by Milner [Mil73, page 168]. For a discussion of its importance we refer the reader to the introduction of Stoughton's monograph [Sto88].

In the early eighties, Brookes, Hoare, and Roscoe [BHR84] introduced *failures* to provide a semantics for CSP [Hoa85]. These failures give rise to a fully abstract semantics for CSP as was shown by Bergstra, Klop, and Olderog [BKO88, Corollary 7.3.1] and Main [Mai87, Section 4.3]. It is well known that failures fail to be fully abstract for a variety of concurrent languages based on asynchronous communication (see, e.g., [BKPR91]). However, our observation that failures are not always fully abstract for a synchronous CCS-like language [Mil80] seems to be new. Whether the failure semantics is fully abstract depends on the cardinality of the set of internal actions. If this set is *finite* then the failure semantics is not fully abstract. Otherwise, it is.

For a simple CCS-like language we present a *trace semantics*. Its definition is based on a *labelled transition system* following Plotkin's structural approach [Plo81]. As one can easily verify, the trace semantics lacks *compositionality*. The search for a compositional semantics for the language leads us to a *failure semantics*. Also this semantics is defined by means of the labelled transition system. Because the failure semantics is shown to be compositional and to make more distinctions than the trace semantics, the failure semantic is called *correct* with respect to the trace semantics. We call the failure semantics *complete* if it does not make too many distinctions. That is, if the statements s_1 and s_2 are distinguished by the failure semantics, then we should be able to construct a larger statement from s_1, denoted by $C[s_1]$, such that if we extend s_2 similarly—this extension is denoted by $C[s_2]$—then the two extensions $C[s_1]$ and $C[s_2]$ are not identified by the trace semantics. We prove that the failure semantics is complete with respect to the trace semantics if and only if the set of internal actions is infinite. In the completeness proof it is essential that one can choose an internal action that is different from a finite set of actions that play a significant role. The assumption that the set of internal actions is infinite allows us to find such a fresh action (cf., e.g., [BG87, Definition 5.3]). Combining the above we can conclude that the failure semantics is *fully abstract*, i.e. correct and complete, with respect to the trace semantics if and only if we have infinitely many internal actions.

Similar full abstraction studies have been carried out by, e.g., De Bakker and De Vink [BV96, Chapter 17], Bergstra, Klop, and Olderog [BKO88], Horita [Hor93], Main [Mai87], and Rutten [Rut88]. This paper builds on their work.

The rest of this paper is organized as follows. In Section 1, we introduce the language. For this language a labelled transition system is given in Section 2. Based on this labelled transition system, a trace semantics and a failure semantics are developed in Section 3 and 4. In Section 5, the failure semantics is shown to be correct with respect to the trace semantics. The completeness of the failure semantics is studied in Section 6. In this section, some new results are presented. We generalize some results by De Bakker and De Vink [BV96, Section 17.3]. Our Theorem 6.5 strengthens [BV96, Lemma 17.21] and Corollary 6.9 proves the conjecture of [BV96, page 504]. Some conclusions are drawn in the final section.

Acknowledgements

I am thankful to the members of the Amsterdam Concurrency Group, in particular Jaco de Bakker and Jan Rutten, for their comments on [Bre94]. I am grateful to Mike Mislove for the stimulating discussions which led to the proof of Corollary 6.9, the main result of this paper. My thanks also to Furio Honsell for his insightful questions when I presented this material to him. Maurizio Gabbrielli provided some comments on a preliminary version of this paper and Rocco De Nicola made me aware of some related work, for which I am thankful. My thanks also to the referees for pointing out an error and for their comments which improved the presentation.

1 Language

In this section, we introduce the syntax of a very simple CCS-like language. We presuppose a set $(i \in) IAct$ of *internal actions* with a distinguished action τ. We assume that these internal actions are observable. The action τ we use to model synchronization. This τ will be visible in the semantics. Furthermore, we presuppose a nonempty set $(c \in) SAct$ of *synchronization actions*. The synchronization actions are assumed not to be observable. We also presuppose a (bijective) function $\bar{\cdot} : SAct \rightarrow SAct$ with for all $c \in SAct$, $\bar{\bar{c}} = c$. It yields for every synchronization action c its synchronization partner $\bar{c}$. The set $(a \in) Act$ of *actions* is given by $Act = IAct \cup SAct$. Finally, we presuppose a nonempty set $(x \in) SVar$ of *statement variables*. These statement variables add recursion to the language.

Definition 1.1 The set $(s \in) Stat$ of *statements* is defined by

$$s ::= nil \mid a.s \mid s + s \mid s \parallel s \mid x.$$

┘

Roughly, the meaning of the statement constructions is as follows. The statement nil cannot do anything but terminate. The prefixing $a.s$ first performs the action a and then behaves like s. The nondeterministic choice $s_1 + s_2$ behaves either as s_1 or as s_2. The parallel composition $s_1 \parallel s_2$ represents s_1 and s_2 performing concurrently, possibly synchronizing. To each statement variable a declaration (see Definition 1.2) assigns a (guarded) statement. A statement variable behaves like the statement associated to it by the declaration. We restrict ourselves to guarded recursion[1]. That is, to each statement variable the declaration assigns a guarded statement.

Definition 1.2 The set $(g \in) GStat$ of *guarded statements* is defined by

$$g ::= nil \mid a.s \mid g + g \mid g \parallel g.$$

The set $(d \in) Decl$ of *declarations* is defined by

$$Decl = SVar \rightarrow GStat.$$

┘

[1] Unguarded recursion gives rise to unbounded nondeterminacy. It is well known that unbounded nondeterminacy complicates the search for a fully abstract semantics (see, e.g., [AP86]). If we were to consider unguarded recursion, then Theorem 6.5 would not hold any more.

We assume some fixed declaration d as given, and all considerations in any argument refer to this declaration.

The language we have introduced above is a minimal one. Apart from the statement variables, all the constructions of the language are used in the contexts constructed in the proof of Theorem 6.5 (cf. Definition 6.3 and Lemma 6.4). The statement variables, which allow us to specify recursion, are crucial for the results of Subsection 6.2. In [BV96, BKO88, Rut88] sequential composition instead of prefixing is used. Also restriction and renaming are present in the language studied in [BKO88]. We are confident that the main results of the present paper also hold if we replace prefixing by sequential composition and add restriction and renaming.

2 Labelled transition system

The trace semantics and the failure semantics are both based on the *labelled transition system* presented below. It is shown that the system is finitely branching. We will exploit this fact in Section 6.1.

The configurations of the labelled transition system are statements and the labels are actions. The transition relation is presented in

DEFINITION 2.1 The *transition relation* $\rightarrow$ is defined by the following axiom and rules.

(1) $a.s \xrightarrow{a} s$

(2) $\dfrac{s_1 \xrightarrow{a} s_1'}{s_1 + s_2 \xrightarrow{a} s_1'}$ $\qquad$ $\dfrac{s_2 \xrightarrow{a} s_2'}{s_1 + s_2 \xrightarrow{a} s_2'}$

(3) $\dfrac{s_1 \xrightarrow{a} s_1'}{s_1 \parallel s_2 \xrightarrow{a} s_1' \parallel s_2}$ $\qquad$ $\dfrac{s_2 \xrightarrow{a} s_2'}{s_1 \parallel s_2 \xrightarrow{a} s_1 \parallel s_2'}$

(4) $\dfrac{s_1 \xrightarrow{c} s_1' \qquad s_2 \xrightarrow{\bar{c}} s_2'}{s_1 \parallel s_2 \xrightarrow{\tau} s_1' \parallel s_2'}$

(5) $\dfrac{d(x) \xrightarrow{a} s}{x \xrightarrow{a} s}$

┘

Every configuration has only finitely many outgoing transitions as is shown in

PROPOSITION 2.2 *The labelled transition system is finitely branching.*

PROOF By structural induction, one can show that all guarded statements have only finitely many outgoing transitions. Subsequently, one can prove, again by structural induction, that all statements have finitely many outgoing transitions. □

3 Trace semantics

From the labelled transition system given in the previous section we extract the *trace semantics*. It is shown that this semantics is not compositional.

In the trace semantics we do not record the transitions labelled by synchronization actions, which may be viewed as unsuccessful attempts at synchronization, unless a statement can only make transitions labelled by synchronization actions. In that case, we say that the statement deadlocks. We consider successfully terminating computations

$$s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots \xrightarrow{i_{n-1}} s_n \not\rightarrow,^2$$

nonterminating computations

$$s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots,$$

and deadlocking computations

$$s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots \xrightarrow{i_{n-1}} s_n \text{ deadlocks.}$$

Note that all computations are maximal, i.e. they cannot be extended. These three types of computations are modelled by finite sequences of internal actions, infinite sequences of internal actions, and finite sequences of internal actions followed by a δ, respectively.

Definition 3.1 The set $(T \in)\, \mathbb{T}$ of *trace sets* is defined by

$$\mathbb{T} = \mathcal{P}\,(IAct^* \cup IAct^\omega \cup IAct^* \cdot \{\delta\}).$$

$\lrcorner$

The trace semantics assigns to each statement a trace set as follows.

Definition 3.2 The *trace semantics* $\mathcal{T} : Stat \rightarrow \mathbb{T}$ is defined by

$$\begin{aligned}\mathcal{T}\,(s) = \{\, & i_1 i_2 \cdots i_{n-1} \mid s = s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots \xrightarrow{i_{n-1}} s_n \not\rightarrow \} \cup \\ \{\, & i_1 i_2 \cdots \mid s = s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots \} \cup \\ \{\, & i_1 i_2 \cdots i_{n-1}\delta \mid s = s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots \xrightarrow{i_{n-1}} s_n \text{ deadlocks} \},\end{aligned}$$

where s_n *deadlocks* if $s_n \rightarrow$ and $s_n \xrightarrow{i_n} s_{n+1}$ for no i_n and s_{n+1}. $\lrcorner$

It is important to notice that like in, e.g., [BV96] (and in contrast to, e.g., [Mai87]) in the trace semantics synchronization (τ) and deadlock (δ) are observable and unsuccessful attempts at synchronization (synchronization actions) are not. Note also that the trace semantics does not rule out unfair computations. This semantics is not compositional as is demonstrated in

Proposition 3.3 *$\mathcal{T}$ is not compositional.*

Proof Towards a contraction, assume that $\mathcal{T}$ is compositional. Then

$$\begin{aligned}\tau &\in \mathcal{T}\,(c.nil \parallel \bar{c}.nil) \\ &= \mathcal{T}\,(c.nil) \parallel \mathcal{T}\,(\bar{c}.nil) \quad [\mathcal{T} \text{ is compositional}] \\ &= \mathcal{T}\,(c.c.nil) \parallel \mathcal{T}\,(\bar{c}.nil) \quad [\mathcal{T}\,(c.nil) = \{\delta\} = \mathcal{T}\,(c.c.nil)] \\ &= \mathcal{T}\,(c.c.nil \parallel \bar{c}.nil) \quad [\mathcal{T} \text{ is compositional}]\end{aligned}$$

$\square$

[2] In this paper we use the convention $(n \in)\, \mathbb{N} = \{1, 2, \ldots\}$.

4 Failure semantics

Also the *failure semantics* is defined in terms of the labelled transition system of Section 2. In contrast to the trace semantics, this semantics is compositional. It is furthermore shown how the trace semantics can be derived from the failure semantics.

To construct a compositional semantics for the language we make more distinctions than we did in the trace semantics. We do this by also recording the transitions labelled by synchronization actions and by administrating which synchronization actions a deadlocking statement refuses to do—rather than just signaling deadlock as we did in the trace semantics. This leads to considering successfully terminating computations

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_n \not\rightarrow,$$

nonterminating computations

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots,$$

and deadlocking computations

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_n \text{ deadlocks and refuses to do the set } X \text{ of synchronization actions.}$$

We use finite sequences of actions, infinite sequences of actions, and finite sequences of actions followed by a refusal set X of synchronizations actions to model these three types of computations.

Definition 4.1 The set $(F \in) \mathbb{F}$ of *failure sets* is defined by

$$\mathbb{F} = \mathcal{P}\,(Act^* \cup Act^\omega \cup Act^* \cdot \mathcal{P}\,(SAct)).$$

┘

These failure sets are assigned to the statements as follows.

Definition 4.2 The *failure semantics* $\mathcal{F} : Stat \to \mathbb{F}$ is defined by

$$\begin{aligned}\mathcal{F}\,(s) = {} & \{\, a_1 a_2 \cdots a_{n-1} \mid s = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_n \not\rightarrow \} \cup \\ & \{\, a_1 a_2 \cdots \mid s = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \} \cup \\ & \{\, a_1 a_2 \cdots a_{n-1} X \mid s = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_n \text{ deadlocks and refuses } X \,\},\end{aligned}$$

where s_n *refuses* X if for all $c \in X$, $s_n \xrightarrow{c} s_{n+1}$ for no s_{n+1}. ┘

Note that in the above introduced failure semantics synchronization (τ) is observable, like, e.g., in [BV96]. This should be contrasted with, e.g., testing semantics [Hen88] where synchronization is usually not visible.

Next we show that this failure semantics is compositional. For that purpose we first give characterizations of the prefixing operator, the nondeterministic choice, and the parallel composition. These characterizations will also be exploited in the proof of Lemma 6.7. In the characterization of the parallel composition we make use of *schedulers*. These schedulers are closely related to oracles which are used by, e.g., Park [Par81, Section 6] to describe the fair merge. A scheduler σ tells us whether the i-th transition of a parallel composition $s_{\mathrm{L}} \parallel s_{\mathrm{R}}$

* is taken by the left statement s_{L} in which case $\sigma\,(i) = \mathrm{L}$,
* is taken by the right statement s_{R} in which case $\sigma\,(i) = \mathrm{R}$,
* or arises as the result of the statements s_{L} and s_{R} synchronizing by means of the actions c and $\bar{c}$ in which case $\sigma\,(i) = c$.

The contribution of the left statement s_{L} to the i-th transition labelled by a of the parallel composition $s_{\text{L}} \parallel s_{\text{R}}$ is given by $\sigma_{\text{L}}(a, i)$ and the contribution of the right statement s_{R} is denoted by $\sigma_{\text{R}}(a, i)$, where

$$\sigma_{\text{L}}(a, i) = \begin{cases} a & \text{if } \sigma(i) = \text{L} \\ \epsilon & \text{if } \sigma(i) = \text{R} \\ c & \text{if } \sigma(i) = c \text{ and } a = \tau \end{cases}$$

and

$$\sigma_{\text{R}}(a, i) = \begin{cases} \epsilon & \text{if } \sigma(i) = \text{L} \\ a & \text{if } \sigma(i) = \text{R} \\ \bar{c} & \text{if } \sigma(i) = c \text{ and } a = \tau. \end{cases}$$

A similar characterization of the merge has been given by, e.g., Meyer [Mey86, Section 2.2].

Lemma 4.3

1. *$a_1 \ldots a_n \in \mathcal{F}(a.s)$ if and only if $a_1 = a$ and $a_2 \ldots a_n \in \mathcal{F}(s)$.*
2. *$a_1 a_2 \ldots \in \mathcal{F}(a.s)$ if and only if $a_1 = a$ and $a_2 a_3 \ldots \in \mathcal{F}(s)$.*
3. *$X \in \mathcal{F}(a.s)$ if and only if $a \in SAct$ and $a \notin X$.*
4. *$a_1 \ldots a_n X \in \mathcal{F}(a.s)$ if and only if $a_1 = a$ and $a_2 \ldots a_n X \in \mathcal{F}(s)$.*
5. *$a_1 \ldots a_{n-1} \in \mathcal{F}(s_1 + s_2)$ if and only if $a_1 \ldots a_{n-1} \in \mathcal{F}(s_1)$ or $a_1 \ldots a_{n-1} \in \mathcal{F}(s_2)$.*
6. *$a_1 a_2 \ldots \in \mathcal{F}(s_1 + s_2)$ if and only if $a_1 a_2 \ldots \in \mathcal{F}(s_1)$ or $a_1 a_2 \ldots \in \mathcal{F}(s_2)$.*
7. *$X \in \mathcal{F}(s_1 + s_2)$ if and only if $X \in \mathcal{F}(s_1)$ and $X \in \mathcal{F}(s_2)$.*
8. *$a_1 \ldots a_n X \in \mathcal{F}(s_1 + s_2)$ if and only if $a_1 \ldots a_n X \in \mathcal{F}(s_1)$ or $a_1 \ldots a_n X \in \mathcal{F}(s_2)$.*
9. *$a_1 \ldots a_{n-1} \in \mathcal{F}(s_1 \parallel s_2)$ if and only if there exists a function $\sigma : \{1, \ldots, n-1\} \to (\{\text{L}, \text{R}\} \cup SAct)$ such that $\sigma_{\text{L}}(a_1, 1) \ldots \sigma_{\text{L}}(a_{n-1}, n-1) \in \mathcal{F}(s_1)$ and $\sigma_{\text{R}}(a_1, 1) \ldots \sigma_{\text{R}}(a_{n-1}, n-1) \in \mathcal{F}(s_2)$.*
10. *$a_1 a_2 \ldots \in \mathcal{F}(s_1 \parallel s_2)$ if and only if there exists a function $\sigma : \mathbb{N} \to (\{\text{L}, \text{R}\} \cup SAct)$ such that $\sigma_{\text{L}}(a_1, 1)\sigma_{\text{L}}(a_2, 2) \ldots$ is a prefix of an element in $\mathcal{F}(s_1)$ and $\sigma_{\text{R}}(a_1, 1)\sigma_{\text{R}}(a_2, 2) \ldots$ is a prefix of an element in $\mathcal{F}(s_2)$.*
11. *$a_1 \ldots a_{n-1} X \in \mathcal{F}(s_1 \parallel s_2)$ if and only if there exists a function $\sigma : \{1, \ldots, n-1\} \to (\{\text{L}, \text{R}\} \cup SAct)$ and X_{L}, $X_{\text{R}} \in \mathcal{P}(SAct)$ such that*
    - $\sigma_{\text{L}}(a_1, 1) \ldots \sigma_{\text{L}}(a_{n-1}, n-1) X_{\text{L}} \in \mathcal{F}(s_1)$,
    - $\sigma_{\text{R}}(a_1, 1) \ldots \sigma_{\text{R}}(a_{n-1}, n-1) X_{\text{R}} \in \mathcal{F}(s_2)$,
    - *$c \notin X_{\text{L}}$ and $\bar{c} \notin X_{\text{R}}$ for no $c \in SAct$, and*
    - $X \subseteq X_{\text{L}} \cap X_{\text{R}}$.

Proof We only show the tenth case. The other cases are simpler or can be proved similarly.

Assume $a_1 a_2 \ldots \in \mathcal{F}(s_1 \parallel s_2)$. From the rules defining the transition relation we can deduce that there exist statements s_i^{L} and s_i^{R}, for $i \in \mathbb{N}$, such that $s_1^{\text{L}} = s_1$, $s_1^{\text{R}} = s_2$, and for all $i \in \mathbb{N}$,

$$s_i^{\text{L}} \parallel s_i^{\text{R}} \xrightarrow{a_i} s_{i+1}^{\text{L}} \parallel s_{i+1}^{\text{R}}$$

satisfying one (and only one) of the following three conditions.

1. $s_i^{\mathrm{L}} \xrightarrow{a_i} s_{i+1}^{\mathrm{L}}$ and $s_i^{\mathrm{R}} = s_{i+1}^{\mathrm{R}}$.
2. $s_i^{\mathrm{R}} \xrightarrow{a_i} s_{i+1}^{\mathrm{R}}$ and $s_i^{\mathrm{L}} = s_{i+1}^{\mathrm{L}}$.
3. $s_i^{\mathrm{L}} \xrightarrow{c} s_{i+1}^{\mathrm{L}}$, $s_i^{\mathrm{R}} \xrightarrow{\bar{c}} s_{i+1}^{\mathrm{R}}$, for some $c \in SAct$, and $a_i = \tau$.

Depending on which condition is satisfied we define $\sigma(i)$ as follows.

1. $\sigma(i) = \mathrm{L}$.
2. $\sigma(i) = \mathrm{R}$.
3. $\sigma(i) = c$.

One can easily verify that $\sigma_{\mathrm{L}}(a_1, 1)\sigma_{\mathrm{L}}(a_2, 2)\ldots$ is a prefix of an element in $\mathcal{F}(s_1)$ and that $\sigma_{\mathrm{R}}(a_1, 1)\sigma_{\mathrm{R}}(a_2, 2)\ldots$ is a prefix of an element in $\mathcal{F}(s_2)$.

Let $\sigma : \mathbb{N} \to (\{\mathrm{L}, \mathrm{R}\} \cup SAct)$ be such that $\sigma_{\mathrm{L}}(a_1, 1)\sigma_{\mathrm{L}}(a_2, 2)\ldots$ is a prefix of an element in $\mathcal{F}(s_1)$ and that $\sigma_{\mathrm{R}}(a_1, 1)\sigma_{\mathrm{R}}(a_2, 2)\ldots$ is a prefix of an element in $\mathcal{F}(s_2)$. Then there exist statements s_i^{L} and s_i^{R}, for $i \in \mathbb{N}$, such that $s_1^{\mathrm{L}} = s_1$, $s_1^{\mathrm{R}} = s_2$, and for all $i \in \mathbb{N}$,

$$\begin{array}{ll}
s_i^{\mathrm{L}} \xrightarrow{a_i} s_{i+1}^{\mathrm{L}} \text{ and } s_i^{\mathrm{R}} = s_{i+1}^{\mathrm{R}} & \text{if } \sigma(i) = \mathrm{L} \\
s_i^{\mathrm{L}} = s_{i+1}^{\mathrm{L}} \text{ and } s_i^{\mathrm{R}} \xrightarrow{a_i} s_{i+1}^{\mathrm{R}} & \text{if } \sigma(i) = \mathrm{R} \\
s_i^{\mathrm{L}} \xrightarrow{c} s_{i+1}^{\mathrm{L}}, s_i^{\mathrm{R}} \xrightarrow{\bar{c}} s_{i+1}^{\mathrm{R}}, \text{ and } a_i = \tau & \text{if } \sigma(i) = c.
\end{array}$$

From the rules defining the transition relation we can deduce that for all $i \in \mathbb{N}$,

$$s_i^{\mathrm{L}} \parallel s_i^{\mathrm{R}} \xrightarrow{a_i} s_{i+1}^{\mathrm{L}} \parallel s_{i+1}^{\mathrm{R}}.$$

Consequently, $a_1 a_2 \ldots \in \mathcal{F}(s_1 \parallel s_2)$. □

The third condition of the last case tells us that the refusal sets X_{L} and X_{R} rule out synchronization. The fourth condition states that a parallel composition refuses those synchronization actions which both components refuse to do.

From the above characterizations we can derive that the failure semantics is compositional.

Corollary 4.4 *$\mathcal{F}$ is compositional.*

Proof From Lemma 4.3 one can extract the definitions of the semantic operators. For example, the semantic prefixing operator $a.\cdot : \mathbb{F} \to \mathbb{F}$ is given by

$$\begin{array}{rll}
a.F = & \{\, a a_1 a_2 \ldots a_{n-1} & \mid a_1 a_2 \ldots a_{n-1} \in F \,\} \cup \\
& \{\, a a_1 a_2 \ldots & \mid a_1 a_2 \ldots \in F \,\} \cup \\
& \{\, a a_1 a_2 \ldots a_{n-1} X & \mid a_1 a_2 \ldots a_{n-1} X \in F \,\} \cup \\
& \{\, X & \mid a \in SAct \text{ and } a \notin X \,\}.
\end{array}$$

One can easily verify that $\mathcal{F}(a.s) = a.\mathcal{F}(s)$. □

Similar semantic operators have been given in [BV96, Definition 17.14], [BKO88, Section 5.2] [BHR84, Section 4], and [Rut88, Definition 4.4].

We conclude this section by showing that the trace semantics can be derived from the failure semantics. For that purpose we introduce an abstraction operator which assigns to each failure set the corresponding trace set.

Definition 4.5 The function $abs : \mathbb{F} \to \mathbb{T}$ is defined by

$$\begin{array}{rll} abs\,(F) = & \{\, i_1 \ldots i_{n-1} & \mid i_1 \ldots i_{n-1} \in F \,\} \cup \\ & \{\, i_1 i_2 \ldots & \mid i_1 i_2 \ldots \in F \,\} \cup \\ & \{\, i_1 \ldots i_{n-1} \delta & \mid i_1 \ldots i_{n-1} X \in F \,\}. \end{array}$$

This abstraction operator links the trace and failure semantics.

Proposition 4.6 *$\mathcal{T} = abs \circ \mathcal{F}$.*

Proof Trivial. □

This proposition will be exploited in the proofs of Theorem 5.2 and 6.8.

5 Correctness

The failure semantics $\mathcal{F}$ is shown to be *correct* with respect to the trace semantics $\mathcal{T}$, i.e. for all statements s_1 and s_2 and for all contexts $C\,[\cdot]$,

if $\mathcal{F}\,(s_1) = \mathcal{F}\,(s_2)$ then $\mathcal{T}\,(C\,[s_1]) = \mathcal{T}\,(C\,[s_2])$.

A context $C\,[\cdot]$ is a statement with holes. These holes are represented by the special symbol $[\cdot]$. By $C\,[s]$ we denote the statement obtained by replacing all the holes $[\cdot]$ in the context $C\,[\cdot]$ by the statement s. The contexts are introduced in

Definition 5.1 The set $(C\,[\cdot] \in)$ *Cont* of *contexts* is defined by

$$C\,[\cdot] ::= [\cdot] \mid nil \mid a.C\,[\cdot] \mid C\,[\cdot] + C\,[\cdot] \mid C\,[\cdot] \parallel C\,[\cdot] \mid x.$$

From Corollary 4.4 and Proposition 4.6 we can conclude

Theorem 5.2 *$\mathcal{F}$ is correct with respect to $\mathcal{T}$.*

Proof For all s_1, $s_2 \in Stat$ and $C\,[\cdot] \in Cont$,

$$\begin{array}{ll} \mathcal{F}\,(s_1) = \mathcal{F}\,(s_2) & \\ \quad \Rightarrow \ \mathcal{F}\,(C\,[s_1]) = \mathcal{F}\,(C\,[s_2]) & \text{[Corollary 4.4]} \\ \quad \Rightarrow \ \mathcal{T}\,(C\,[s_1]) = \mathcal{T}\,(C\,[s_2]) & \text{[Proposition 4.6]} \end{array}$$

□

6 Completeness

Is the failure semantics $\mathcal{F}$ also *complete* with respect to the trace semantics $\mathcal{T}$? This question boils down to checking whether for all statements s_1 and s_2,

if $\mathcal{F}\,(s_1) \neq \mathcal{F}\,(s_2)$ then $\mathcal{T}\,(C\,[s_1]) \neq \mathcal{T}\,(C\,[s_2])$ for some context $C\,[\cdot]$. (1)

Below we will show that $\mathcal{F}$ is complete with respect to $\mathcal{T}$ *if and only if* the set *IAct* of internal actions is infinite. First, we demonstrate that if the set *IAct* is infinite then $\mathcal{F}$ is complete with respect to $\mathcal{T}$. Second, we prove that this is not the case if *IAct* is a finite set.

6.1 $IAct$ is infinite

In this subsection we assume the set $IAct$ of internal actions to be infinite. Under this assumption we prove (1) as follows. Let s_1 and s_2 be statements with $\mathcal{F}(s_1) \neq \mathcal{F}(s_2)$. Without loss of any generality we can assume that there exists a $w \in Act^* \cup Act^\omega \cup Act^* \cdot \mathcal{P}(SAct)$ such that $w \in \mathcal{F}(s_1)$ and $w \notin \mathcal{F}(s_2)$. As we will see below, we can restrict our attention to finite sequences of actions possibly followed by a finite refusal set, that is $w \in Act^* \cup Act^* \cdot \mathcal{P}_f(SAct)$. Given such a w, we can construct a context $[\cdot] \parallel test_i(w)$ such that $\mathcal{T}(s_1 \parallel test_i(w)) \neq \mathcal{T}(s_2 \parallel test_i(w))$. The internal action i in the statement $test_i(w)$ should be fresh, i.e. neither the statement s_1 nor the statement s_2 should be able to perform this action in its first $|w|$ transitions, where $|w|$ denotes the length of the finite sequence w. Because the labelled transition system is finitely branching and the set $IAct$ is infinite, we can always find such a fresh i as we will see below.

In the rest of this subsection we fill in the details of the proof sketched above. The set $act(\mathcal{F}(s)[n])$ consists of those actions the statement s can perform in its first n transitions.

Proposition 6.1 *For all $s \in Stat$ and $n \in \mathbb{N} \cup \{0\}$, the set*

$$act(\mathcal{F}(s)[n]) = \{\, a_k \mid s = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_k} s_{k+1} \text{ and } k \leq n \,\}$$

is finite.

Proof Induction on n exploiting Proposition 2.2. □

Since the sets $act(\mathcal{F}(s_1)[|w|])$ and $act(\mathcal{F}(s_2)[|w|])$ are finite and the set $IAct$ is infinite, we can always find an internal action i which is neither in $act(\mathcal{F}(s_1)[|w|])$ nor in $act(\mathcal{F}(s_2)[|w|])$. We exploit this fact in the proof of Theorem 6.5.

The fact that we only have to consider sequences in $Act^* \cup Act^* \cdot \mathcal{P}_f(SAct)$ follows from

Proposition 6.2 *Let s_1, $s_2 \in Stat$.*

1. *For all $w \in Act^\omega$, if $w \in \mathcal{F}(s_1)$ and $w \notin \mathcal{F}(s_2)$ then there exists a finite prefix of w which is not a prefix of any element in $\mathcal{F}(s_2)$.*
2. *For all $wX \in Act^* \cdot \mathcal{P}(SAct)$, if $wX \in \mathcal{F}(s_1)$ and $wX \notin \mathcal{F}(s_2)$ then there exists a finite subset Y of X such that $wY \in \mathcal{F}(s_1)$ and $wY \notin \mathcal{F}(s_2)$.*

Proof By means of König's lemma and Proposition 2.2 one can prove that for all $w \in Act^\omega$, if every finite prefix of w is a prefix of an element in $\mathcal{F}(s_2)$ then $w \in \mathcal{F}(s_2)$. From this we can conclude 1.

Let $wX \in Act^* \cdot \mathcal{P}(SAct)$. Assume that $wX \in \mathcal{F}(s_1)$ and $wX \notin \mathcal{F}(s_2)$. Take $Y = X \cap act(\mathcal{F}(s_2)[|wX|])$. According to Proposition 6.1, Y is a finite subset of X. One can easily verify that $wY \in \mathcal{F}(s_1)$ and $wY \notin \mathcal{F}(s_2)$, and hence we can conclude 2. □

The contexts we construct are of the form $[\cdot] \parallel test_i(w)$. The statement $test_i(w)$ is designed in such a way that we can derive from $\mathcal{T}(s \parallel test_i(w))$ whether $w \in \mathcal{F}(s)$. To detect this we construct the sequence $result_i(w)$. The details are provided in Lemma 6.4. Recall that the synchronization actions and the refusals sets of the failure semantics are not observable in the trace semantics. A synchronization action c performed by the statement s can be made visible by the test performing its synchronization partner $\bar{c}$, because the two can synchronize resulting in the observable action τ. However, we have to distinguish this synchronization of the statements s and $test_i(w)$ from synchronizations occurring within the statement s (as we will see no synchronizations occur within the test). This is done by pre- and postfixing the synchronization action $\bar{c}$ by a fresh internal action i. The synchronization of the statement s and the test now results in $i\tau i$, whereas a synchronization within the statement s can never give rise to $i\tau i$ since i is fresh. If the statement s refuses $\{c_1, \ldots, c_n\}$ then its parallel composition with $\bar{c}_1.nil + \cdots + \bar{c}_n.nil$ signals deadlock in the trace semantics. In this way we can make also the refusal sets of the failure semantics visible in the trace semantics.

Definition 6.3 Let $i \in IAct$. The function[3] $test_i : (Act^* \cup Act^* \cdot \mathcal{P}_f(SAct)) \rightarrow Stat$ is defined by

$$
\begin{array}{ll}
test_i(\epsilon) & = nil \\
test_i(X) & = \begin{cases} nil & \text{if } X = \emptyset \\ \bar{c}_1.nil + \cdots + \bar{c}_n.nil & \text{if } X = \{c_1, \ldots, c_n\} \end{cases} \\
test_i(aw) & = \begin{cases} test_i(w) & \text{if } a \in IAct \\ i.\bar{a}.i.test_i(w) & \text{if } a \in SAct. \end{cases}
\end{array}
$$

The function $result_i : (Act^* \cup Act^* \cdot \mathcal{P}_f(SAct)) \rightarrow (IAct^* \cup IAct^* \cdot \{\delta\})$ is defined by

$$
\begin{array}{ll}
result_i(\epsilon) & = \epsilon \\
result_i(X) & = \delta \\
result_i(aw) & = \begin{cases} a\, result_i(w) & \text{if } a \in IAct \\ i\tau i\, result_i(w) & \text{if } a \in SAct. \end{cases}
\end{array}
$$

$\lrcorner$

Note that the above construction of the test only works for finite action sequences possibly followed by a finite refusal set.

The key property of $test_i$ and $result_i$ is stated in the next lemma. This lemma is crucial in the proof of Theorem 6.5.

Lemma 6.4 *For all $s \in Stat$, $w \in Act^* \cup Act^* \cdot \mathcal{P}_f(SAct)$, and $i \in IAct$, with $i \notin act(\mathcal{F}(s)[|w|])$ and $i \neq \tau$,*

1. *$w \in \mathcal{F}(s)$ if and only if $result_i(w) \in \mathcal{T}(s \parallel test_i(w))$, and*
2. *w is a prefix of an element in $\mathcal{F}(s)$ if and only if $result_i(w)$ is a prefix of an element in $\mathcal{T}(s \parallel test_i(w))$.*

Proof Induction on the length of w. $\square$

The assumption $i \notin act(\mathcal{F}(s)[|w|])$ is essential in the above lemma. For example, let $s = i.\tau.nil$ and $w = c$. Then we have that $test_i(w) = i.\bar{c}.i.nil$ and $result_i(w) = i\tau i$. Clearly, $i\tau i$ is a prefix of an element in $\mathcal{T}(i.\tau.nil \parallel i.\bar{c}.i.nil)$, but c is not a prefix of any element in $\mathcal{F}(i.\tau.nil)$. Also the assumption $i \neq \tau$ is necessary. For example, assume $s = c.\tau.nil$ and $w = \tau c$. Then we have that $test_\tau(w) = \tau.\bar{c}.\tau.nil$ and $result_\tau(w) = \tau\tau\tau\tau$. Obviously, we have that $\tau\tau\tau\tau \in \mathcal{T}(c.\tau.nil \parallel \tau.\bar{c}.\tau.nil)$, but $\tau c \notin \mathcal{F}(c.\tau.nil)$.

Combining the above results we can prove

Theorem 6.5 *$\mathcal{F}$ is complete with respect to $\mathcal{T}$.*

Proof We have to prove (1). Let $s_1, s_2 \in Stat$ such that $\mathcal{F}(s_1) \neq \mathcal{F}(s_2)$. Without loss of generality we can assume that there exists a $w \in Act^* \cup Act^\omega \cup Act^* \cdot \mathcal{P}(SAct)$ such that $w \in \mathcal{F}(s_1)$ and $w \notin \mathcal{F}(s_2)$. We distinguish the following three cases.

- $w \in Act^*$ Let $i \in IAct$, with $i \notin act(\mathcal{F}(s_1)[|w|]) \cup act(\mathcal{F}(s_2)[|w|])$ and $i \neq \tau$. We can always find such an i since the set $IAct$ is assumed to be infinite and the set $act(\mathcal{F}(s_1)[|w|]) \cup act(\mathcal{F}(s_2)[|w|])$ is finite according to Proposition 6.1. We take $C[\cdot] = [\cdot] \parallel test_i(w)$. From Lemma 6.4.1 we can conclude that $result_i(w) \in \mathcal{T}(C[s_1])$ but $result_i(w) \notin \mathcal{T}(C[s_2])$.
- $w \in Act^\omega$ According to Proposition 6.2.1 there exists a finite prefix v of w which is not a prefix of any element in $\mathcal{F}(s_2)$. Let $i \in IAct$ such that $i \notin act(\mathcal{F}(s_1)[|v|]) \cup act(\mathcal{F}(s_2)[|v|])$ and $i \neq \tau$. In this case we take $C[\cdot] = [\cdot] \parallel test_i(v)$. From Lemma 6.4.2 we can deduce that $result_i(v)$ is a prefix of an element in $\mathcal{T}(C[s_1])$ but $result_i(v)$ is not a prefix of any element in $\mathcal{T}(C[s_2])$.

[3] Note that $test_i$ is not really a function. Since any ordering of the synchronization actions $c_1, \ldots, c_n$ will serve our purposes, we just choose one.

$w = vX$ According to Proposition 6.2.2 there exists a finite subset Y of X such that $vY \in \mathcal{F}(s_1)$ and $vY \notin \mathcal{F}(s_2)$. Let $i \in \mathit{IAct}$ such that $i \notin \mathit{act}(\mathcal{F}(s_1)[|vY|]) \cup \mathit{act}(\mathcal{F}(s_2)[|vY|])$ and $i \neq \tau$. In this case we take $C[\cdot] = [\cdot] \parallel \mathit{test}_i(vY)$. From Lemma 6.4.1 we can derive that $\mathit{result}_i(vY) \in \mathcal{T}(C[s_1])$ but $\mathit{result}_i(vY) \notin \mathcal{T}(C[s_2])$.

$\square$

The above proof is based on the proofs of [BV96, Lemma 17.21], [BKO88, Theorem 7.2.1], the one presented in [Bre94], and [Rut88, Theorem 6.13]. It generalizes [BV96, Lemma 17.21]. The condition that the set of statement variables is finite has been dropped.

6.2 *IAct* is finite

In this subsection we assume that the set of internal actions is finite. Let $\mathit{IAct} = \{\tau, i_1, \ldots, i_{n-1}\}$. Under this assumption we show that (1) does not hold. Let $x \in \mathit{SVar}$. For the rest of this subsection we fix a declaration d satisfying

$$d(x) = \tau.x + i_1.x + \cdots + i_{n-1}.x.$$

Furthermore, we take

$$s_1 = x \text{ and } s_2 = x + c.x.$$

The transition graphs of these two statements are depicted below.

The only difference between the two statements is that s_2 can start with a transition labelled by c and s_1 cannot. Hence, the statements are not identified by the failure semantics. Note that, since the statements x and $x + c.x$ are both not deadlocking, the failure sets associated to them do not contain refusal sets. As we will see below (cf. Lemma 6.7), refusal sets do not play a role in the incompleteness result presented in this subsection.

PROPOSITION 6.6 $\mathcal{F}(x) \neq \mathcal{F}(x + c.x)$.

PROOF We have that $c\tau^\omega \in \mathcal{F}(x + c.x)$ but $c\tau^\omega \notin \mathcal{F}(x)$. $\square$

This difference between the statements in the failure semantics cannot be brought about in the trace semantics by putting the statements in parallel with $\bar{c}.\mathit{nil}$. Also a parallel composition with $i.\bar{c}.i.\mathit{nil}$, where i is some internal action, does not distinguish the two in the trace semantics. As we will see, the trace semantics identifies the statements in every context, disproving (1). To show this we first compare the failure semantics of s_1 and s_2 in every context.

LEMMA 6.7 *For all $C[\cdot] \in \mathit{Cont}$, if $w \in \mathcal{F}(C[x + c.x])$ and $w \notin \mathcal{F}(C[x])$ then $w = w_1 c w_2$ or $w = w_1 \tau w_2$ for some $w_1 \in \mathit{Act}^*$ and $w_2 \in \mathit{Act}^\omega$ such that for all $u \in \mathit{IAct}^\omega$, $w_1 u \in \mathcal{F}(C[x])$.*

PROOF We prove this lemma by structural induction on $C[\cdot]$. We only consider the context $C[\cdot] \parallel s$.[4] The other contexts can be handled similarly. We distinguish the following three cases.

[4] Here we exploit the folklore result that only contexts with one hole need to be considered.

1. Towards a contradiction, assume $w = a_1 \ldots a_n$. Since $w \in \mathcal{F}(C[x + c.x] \parallel s)$, by Lemma 4.3.9 there exists a function $\sigma : \{1, \ldots n\} \to (\{\text{L}, \text{R}\} \cup SAct)$ such that $w_{\text{L}} = \sigma_{\text{L}}(a_1, 1) \ldots \sigma_{\text{L}}(a_n, n) \in \mathcal{F}(C[x + c.x])$ and $\sigma_{\text{R}}(a_1, 1) \ldots \sigma_{\text{R}}(a_n, n) \in \mathcal{F}(s)$. Because $w \notin \mathcal{F}(C[x] \parallel s)$, again by Lemma 4.3.9, $w_{\text{L}} \notin \mathcal{F}(C[x])$. By induction, $w_{\text{L}} = w_1^{\text{L}} c w_2^{\text{L}}$ or $w_{\text{L}} = w_1^{\text{L}} \tau w_2^{\text{L}}$ for some $w_1^{\text{L}} \in Act^*$ and $w_2^{\text{L}} \in Act^\omega$, and hence $w_{\text{L}} \in Act^\omega$, a contradiction.

2. Assume $w = a_1 a_2 \ldots$. Because $w \in \mathcal{F}(C[x + c.x] \parallel s)$, by Lemma 4.3.10 there exists a function $\sigma : \mathbb{N} \to (\{\text{L}, \text{R}\} \cup SAct)$ such that $w_{\text{L}} = \sigma_{\text{L}}(a_1, 1)\sigma_{\text{L}}(a_2, 2) \ldots$ is a prefix of $v_{\text{L}} \in \mathcal{F}(C[x + c.x])$ and $w_{\text{R}} = \sigma_{\text{R}}(a_1, 1)\sigma_{\text{R}}(a_2, 2) \ldots$ is a prefix of an element in $\mathcal{F}(s)$. Since $w \notin \mathcal{F}(C[x] \parallel s)$, again by Lemma 4.3.10, $v_{\text{L}} \notin \mathcal{F}(C[x])$. By induction we have one of the following two cases.

   2.1. Let $v_{\text{L}} = v_1^{\text{L}} c v_2^{\text{L}}$ for some $v_1^{\text{L}} \in Act^*$ and $v_2^{\text{L}} \in Act^\omega$ with for all $u \in IAct^\omega$, $v_1^{\text{L}} u \in \mathcal{F}(C[x])$. Towards a contradiction, assume that w_{L} is a prefix of v_1^{L}. Then w_{L} is a prefix of an element in $\mathcal{F}(C[x])$ and by Lemma 4.3.10, we have that $w \in \mathcal{F}(C[x] \parallel s)$, a contradiction. Consequently, $w_{\text{L}} = v_1^{\text{L}} c v^{\text{L}}$ for some $v^{\text{L}} \in Act^* \cup Act^\omega$. Assume we have that $v_1^{\text{L}} = \sigma_{\text{L}}(a_1, 1) \ldots \sigma_{\text{L}}(a_{j-1}, j - 1)$ and $v^{\text{L}} = \sigma_{\text{L}}(a_{j+1}, j + 1)\sigma_{\text{L}}(a_{j+1}, j + 2) \ldots$. We distinguish two cases.

   2.1.1. Assume $\sigma(j) = c$. Then $a_j = \tau$. Hence, $w = v_1 \tau v$ where $v_1 = a_1 \ldots a_{j-1}$ and $v = a_{j+1} a_{j+2} \ldots$.

   2.1.2. Assume $\sigma(j) = \text{L}$. Then $a_j = c$. Hence, $w = v_1 c v$ where $v_1 = a_1 \ldots a_{j-1}$ and $v = a_{j+1} a_{j+2} \ldots$.

   Let $u = i_1 i_2 \ldots \in IAct^\omega$. We define

   $$\sigma'(h) = \begin{cases} \sigma(h) & \text{if } 1 \leq h < j \\ \text{L} & \text{if } h \geq j \end{cases}$$

   Since

   $$\begin{aligned} &\sigma'_{\text{R}}(a_1, 1) \ldots \sigma'_{\text{R}}(a_{j-1}, j - 1)\sigma'_{\text{R}}(a_j, j)\sigma'_{\text{R}}(a_{j+1}, j + 1) \ldots \\ &\quad = \sigma_{\text{R}}(a_1, 1) \ldots \sigma_{\text{R}}(a_{j-1}, j - 1) \end{aligned}$$

   is a prefix of w_{R} which is a prefix of an element in $\mathcal{F}(s)$ and

   $$\begin{aligned} &\sigma'_{\text{L}}(a_1, 1) \ldots \sigma'_{\text{L}}(a_{j-1}, j - 1)\sigma'_{\text{L}}(a_j, j)\sigma'_{\text{L}}(a_{j+1}, j + 1) \ldots \\ &\quad = \sigma_{\text{L}}(a_1, 1) \ldots \sigma_{\text{L}}(a_{j-1}, j - 1) i_1 i_2 \ldots \\ &\quad = v_1^{\text{L}} u \end{aligned}$$

   is an element in $\mathcal{F}(C[x])$, by Lemma 4.3.10,

   $$a_1 \ldots a_{j-1} i_1 i_2 \ldots = v_1 u \in \mathcal{F}(C[x] \parallel s).$$

   2.2. Let $v_{\text{L}} = v_1^{\text{L}} \tau v_2^{\text{L}}$ for some $v_1^{\text{L}} \in Act^*$ and $v_2^{\text{L}} \in Act^\omega$ such that for all $u \in IAct^\omega$, $v_1^{\text{L}} u \in \mathcal{F}(C[x])$. Similar to case 2.1.2.

3. Assume $w = a_1 \ldots a_n X$. Similar to case 1.

□

Note that the scheduler σ' in the above proof is unfair. From the above lemma we can conclude

THEOREM 6.8 *For all* $C[\cdot] \in Cont$, $\mathcal{T}(C[x]) = \mathcal{T}(C[x + c.x])$.

PROOF Let $C[\cdot] \in Cont$. Clearly, $\mathcal{F}(x) \subseteq \mathcal{F}(x + c.x)$. Because all semantic operators are monotone, $\mathcal{F}(C[x]) \subseteq \mathcal{F}(C[x + c.x])$. Since *abs* is monotone, $\mathcal{T}(C[x]) \subseteq \mathcal{T}(C[x + c.x])$ by Proposition 4.6. To conclude that $\mathcal{T}(C[x]) = \mathcal{T}(C[x + c.x])$ it suffices to show that if $w \in \mathcal{F}(C[x + c.x])$ and $w \notin \mathcal{F}(C[x])$ then $w \notin IAct^* \cup IAct^\omega \cup IAct^* \cdot \mathcal{P}(SAct)$. According to Lemma 6.7 we only have to consider the following two cases.

1. Assume $w = w_1 c w_2$ for some $w_1 \in Act^*$ and $w_2 \in Act^\omega$. Then $w \notin IAct^* \cup IAct^\omega \cup IAct^* \cdot \mathcal{P}(SAct)$.
2. Assume $w = w_1 \tau w_2$ for some $w_1 \in Act^*$ and $w_2 \in Act^\omega$. Towards a contradiction, assume that $w \in IAct^* \cup IAct^\omega \cup IAct^* \cdot \mathcal{P}(SAct)$. Then $w_1 \in IAct^*$ and $w_2 \in IAct^\omega$. According to Lemma 6.7, $w_1 \tau w_2 \in \mathcal{F}(C[x])$, a contradiction.

□

Combining the above results we arrive at a proof of the conjecture of [BV96, page 504].

Corollary 6.9 *$\mathcal{F}$ is not complete with respect to $\mathcal{T}$.*

Proof Immediate consequence of Proposition 6.6 and Theorem 6.8. □

Conclusion

From Theorem 5.2 and 6.5 and Corollary 6.9 we can conclude that the failure semantics is correct and complete, and hence fully abstract, with respect to the trace semantics if and only if the set of internal actions is infinite—the result announced in the abstract. This is an example of a result which shows that the choice of a finite or an infinite set of actions does have (theoretical) implications. Note that we do not claim that this result tells us whether one should choose for finitely or infinitely many actions. Both choices have their merits and demerits (see [Con96]).

The problem of finding the fully abstract semantics for the language with finitely many internal actions is still open. We only know that it should make more distinctions than the trace semantics but less than the failure semantics, and that it should identify statements like s_1 and s_2 given in Subsection 6.2.

By changing the trace semantics—for example, by observing also the unmatched synchronization actions—the failure semantics is fully abstract with respect to this modified trace semantics, no matter whether the set of internal actions is finite or infinite (see [Hor93, Chapter 4]).

Instead of specifying recursion by means of declarations (cf. Definition 1.2), one can also introduce it by adding the construct $\mu x.g$, where g is a guarded statement (see Definition 1.2), to the clause defining the set of statements in Definition 1.1. In this modified setting we can also consider contexts of the form $\mu x.C[\cdot]$. Although we are confident that the main results presented in this paper still hold, several of their proofs have to be changed considerably. For example, to prove Corollary 4.4 we have to add to the set $\mathbb{F}$ of failure sets some additional structure (e.g., a partial order or a metric) to express $\mathcal{F}(\mu x.g)$ as a fixed point of $\mathcal{F}(g)$.

In [MO95], Mislove and Oles address the question of extending a fully abstract semantics for a language without recursion to the language with recursion. To obtain their results they assume the strongly order fully abstractness hypothesis. They cannot prove their results without this hypothesis, nor do they have a counterexample showing that the results do not hold without it. We believe that our study provides such a counterexample. Assume $IAct = \{\tau, i\}$. From Corollary 6.9 we can conclude that the failure semantics is not fully abstract with respect to the trace semantics. However, if we leave out recursion, the failure semantics is fully abstract. This fact can be shown along the lines of the proof of Theorem 6.5. Instead of contexts of the form $[\cdot] \parallel test_i(w)$ we use $[\cdot] \parallel test_{i^m}(w)$, where m is the maximal length of a sequence in the failure semantics of the two statements to be distinguished.

References

[AP86] K.R. Apt and G.D. Plotkin. Countable Nondeterminism and Random Assignment. *Journal of the ACM*, 33(4):724–767, October 1986.

[BG87] J.C.M. Baeten and R.J. van Glabbeek. Merge and Termination in Process Algebra. In K.V. Nori, editor, *Proceedings of the 7th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 287 of *Lecture Notes in Computer Science*, pages 153–172, Pune, December 1987. Springer-Verlag.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(3):560–599, July 1984.

[BKO88] J.A. Bergstra, J.W. Klop, and E.-R. Olderog. Readies and Failures in the Algebra of Communicating Processes. *SIAM Journal on Computing*, 17(6):1134–1177, December 1988.

[BKPR91] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. The Failure of Failures in a Paradigm for Asynchronous Communication. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of CONCUR'91*, volume 527 of *Lecture Notes in Computer Science*, pages 111–126, Amsterdam, August 1991. Springer-Verlag.

[Bre94] F. van Breugel. A Non Fully Abstract Model for a Language with Synchronization. Unpublished lecture notes, September 1994.

[BV96] J.W. de Bakker and E.P. de Vink. *Control Flow Semantics.* Foundations of Computing Series. The MIT Press, Cambridge, 1996.

[Con96] Finite/Infinite Action Sets. Discussion on the concurrency forum, October 1996. Available at `theory.lcs.mit.edu` as `/pub/people/meyer/concurrency.new`.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* Foundations of Computing Series. The MIT Press, Cambridge, 1988.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice Hall International, London, 1985.

[Hor93] E. Horita. *Fully Abstract Models for Concurrent Languages.* PhD thesis, Vrije Universiteit, Amsterdam, September 1993.

[Mai87] M.G. Main. Trace, Failure and Testing Equivalences for Communicating Processes. *International Journal of Parallel Programming*, 16(5):383–400, 1987.

[Mey86] J.-J.Ch. Meyer. Merging Regular Processes by means of Fixed-Point Theory. *Theoretical Computer Science*, 45(2):193–260, 1986.

[Mil73] R. Milner. Processes: a Mathematical Model of Computing Agents. In H.E. Rose and J.C. Shepherdson, editors, *Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 157–173, Bristol, July 1973. North-Holland.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[MO95] M.W. Mislove and F.J. Oles. Full Abstraction and Recursion. *Theoretical Computer Science*, 151(1):207–256, November 1995.

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In P. Deussen, editor, *Proceedings of 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Karlsruhe, March 1981. Springer-Verlag.

[Plo81] G.D. Plotkin. A Structural Approach to Operational Semantics. Report DAIMI FN-19, Aarhus University, Aarhus, September 1981.

[Rut88] J.J.M.M. Rutten. Correctness and Full Abstraction of Metric Semantics for Concurrency. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings of the School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 628–659, Noordwijkerhout, May/June 1988. Springer-Verlag.

[Sto88] A. Stoughton. *Fully Abstract Models of Programming Languages.* Research Notes in Theoretical Computer Science. Pitman, London, 1988.