
FAILURE TRACE SEMANTICS FOR A PROCESS ALGEBRA WITH TIME-OUTS

ROB VAN GLABBEEK

Data61, CSIRO, Sydney, Australia

School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
e-mail address: rvg@cs.stanford.edu

ABSTRACT. This paper extends a standard process algebra with a time-out operator, thereby increasing its absolute expressiveness, while remaining within the realm of untimed process algebra, in the sense that the progress of time is not quantified. Trace and failures equivalence fail to be congruences for this operator; their congruence closure is characterised as failure trace equivalence.

1. MOTIVATION

This work has four fairly independent motivations, in the sense that it aims to provide a common solution to four different problems. How it yields a solution to the last two of these problems is left for future work, however.

1.1. The passage of time in processes modelled as labelled transition systems.

The standard semantics of untimed non-probabilistic process algebras is given in terms of labelled transition systems (LTSs), consisting of a set of states, with action-labelled transitions between them. The behaviour of a system modelled as a state in an LTS can be visualised as a token, starting in this initial state, that travels through the LTS by following its transitions. A question that has plagued me since I first heard of this model, in December 1984, is where exactly (and how) does time progress when this token travels: in the states, during the transitions, or maybe both?

In some sense this question is more of a philosophical than an practical nature. Process algebra has flourished, without any convincing answer to this question sitting in its way.

One reasonable attitude is that in untimed progress algebra we *abstract* from time, which makes the entire question meaningless. While this does make sense, I still desire seeing an untimed process as a special case of a timed process, one where any period of time that might be specified in a timed formalism is instantiated by a nondeterministic choice allowing *any* amount of time. Such a point of view would certainly assist in relating

Key words and phrases: Concurrency, process algebra, time-outs, priority, CCSP, labelled transition systems, semantic equivalences, linear time, branching time, failure trace semantics, absolute expressiveness, full abstraction, safety properties, may testing.

This paper is dedicated to Jos Baeten, at the occasion of his retirement.

timed and untimed process algebras. It is from this perspective that my question is in need of an answer.

The common answer given by the process algebra community is that there is no need to model transitions that take time, for a durational transition can simply be modelled as a pair of instantaneous transitions with a time-consuming state in between. Here I follow this train of thought, and regard transitions as occurring instantaneously. This implies that time must elapse in states, if at all.

Now visualise a system reaching a state s in an LTS, that has a single outgoing transition u leading to a desired goal state. Assume, moreover, that this transition cannot be blocked by the environment. A fundamental assumption made in most process algebraic formalisms is that the system will eventually reach this goal state, that is, that it will not stay forever in state s . In [GH19] this assumption is called *progress*. Without it it is not possible to prove meaningful *liveness properties* [Lam77], saying that “something [good] must eventually happen”.

When our LTS would model all activities of the represented system, it is hard to believe how the system, upon reaching state s , comes to a rest, spends a finite amount of time in state s while doing nothing whatsoever, and then suddenly, and without any clear reason, takes the transition to the desired goal state. It appears more plausible that if we allow the system to stay in state s for a finite amount of time, it can (and perhaps even must) stay there for an infinite amount of time.

The way out of this paradox is that our system, being discrete, necessarily abstracts from a lot of activity. The abstracted activity that is relevant for the visit of the system to state s must be some durational activity, an amount of work to be done, that for a while sits in the way of taking transition u . As soon as that work is done, and nothing further sits in the way, transition u takes place.

In this paper this abstracted activity is made explicit, in the shape of a time-out transition \xrightarrow{t} . Similar to the internal transition $\xrightarrow{\tau}$, modelling the occurrence of an instantaneous action from which we abstract, the time-out transition \xrightarrow{t} models the end of a time-consuming activity from which we abstract. A state s in which the system is supposed to spend some time—a positive and finite, but otherwise unquantified amount—can now be modelled as a pair of states s_1 and s_2 with a time-out transition between them; the outgoing transitions of s are then moved to s_2 . Time will be spent in state s_1 only, where some abstracted time-consuming activity takes place. The only thing in the model that testifies to this activity is the time-out transition $s_1 \xrightarrow{t} s_2$, that is guaranteed to occur as soon as the time-consuming work is done. Immediately afterwards, the system will take one of the outgoing transitions of s_2 , unless they are all blocked by the environment in which our reactive system is running.

An alternative proposal on how systems spend finite amounts of time in states is essentially due to Milner [Mil90]. It says that the states in an LTS model *reactive systems* that merely react on stimuli from the environment. This reaction may be instantaneous, but the environment itself takes time between issuing stimuli. Usually, each outgoing transition of a state s is labelled by a visible action a , and only when the environment chooses to synchronise on channel a will such a transition occur. Normally, it may take a while before the environment is ready to synchronise with any outgoing transition of state s , and this is exactly what accounts for the time spent in state s . A special case occurs when state s has an outgoing τ -transition. Since such transitions do not require synchronisation with

the environment, it appears that the represented system is not allowed to linger in such a state.

The current paper combines the time-out transitions proposed above with the reactive viewpoint adopted by Milner. So any time the system spends in a state is either due to the environment not allowing an outgoing transition, or the system waiting for a time-out transition to occur (or both).

Admitting time-out transitions in an LTS not only allows the specification of states in which the system spends a positive but finite amount of time, as described above; it also allows the specification of states in which the system spends no time whatsoever, unless forced by the environment—this is achieved by not using any time-out transitions leaving that state. Moreover, we can model states s in which some outgoing transitions have to wait for a time-out to occur, and others do not.¹ This is done by modelling s as $s_1 \xrightarrow{t} s_2$, where the outgoing transitions of s_2 have to wait for the time-out, and the ones of s_1 do not. In that case the time-out will occur only if the environment fails to synchronise in time with any of the outgoing transitions from s_1 . This implements a priority mechanism, in which from the system's perspective the outgoing transitions of s_1 have priority over the ones of s_2 .

1.2. Failure trace semantics. In [Gla01, Gla93] I classified many semantic equivalences on processes, and proposed testing scenarios for these semantics in terms of button-pushing experiments on reactive and generative machines, such that two processes are inequivalent iff their difference can be detected through the associated testing scenario. Figure 1 shows the various equivalences in the absence of internal transitions τ (or t). For equivalences such as simulation equivalence (point S in Figure 1), or bisimilarity (B), the testing scenarios require a replication facility, allowing the experimenter to regularly make copies of a system *in its current state*, and expose each of those copies to further tests. This facility could be regarded as fairly unrealistic, in the sense that one would not expect an actual implementation of such a facility to be available. When one takes this opinion, (bi)similarity makes distinctions between processes that are not well justified. Likewise, readiness equivalence (R^*) has a testing scenario that allows us, in certain states, to see the menu of all available actions the environment (= tester) can synchronise with. It may be deemed unrealistic to assume such a menu to be generally available. Or for systems in which such a menu is available, one might say that it has to be specified as part of the system behaviour, so that we do not need it explicitly in a testing scenario. When one takes this opinion, readiness equivalence, and its finer variants, also makes distinctions between processes that are not well justified.

Following this line of reasoning, the finest (= most discriminating) semantic equivalence that does have a realistic testing scenario is *failure trace semantics* (FT). Its testing scenario allows the tester to control which actions may take place (are available for synchronisation) and which are not. It also allows the tester to record sequences of actions that take place during a run of the system, interspersed with periods of idling, where each idle period is annotated with the set of actions made available for synchronisation by the tester during this period. Such a sequence is a *failure trace*, and two systems are distinguished iff one has a failure trace that the other has not. In Figure 1 two variants of failure trace equivalence

¹Enriching the model with this kind of states is not merely a luxury in which I indulge; such states arise naturally through parallel composition. In my interleaving semantics it will turn out that $t.a||t.b \xrightarrow{t} a||t.b = a.t.b + t.(a.b + b.a)$.

are recorded (FT^* and FT^∞), depending on whether one only allows finite observations, leading to *partial* failure traces, or also infinite ones.

Failures semantics (F^*) is the default semantics of the process algebra CSP [BHR84, Hoa85]. It is coarser than failure trace semantics, in the sense that it makes more identifications. Its testing scenario is the variant of the one for failure trace semantics described above, in which the environment/tester cannot alter the set of allowed actions once the system idles (= reaches a state of deadlock). Thus, when two systems are failures inequivalent, one can think of an environment such that when placed in that environment one of them deadlocks and the other does not. An important insight is that such an environment can always be built from some basic process algebraic operators. So when P and Q are failures inequivalent, there is process algebraic context $\mathcal{C}[-]$, such that $\mathcal{C}[P]$ has a deadlock which $\mathcal{C}[Q]$ has not (or vice versa).

Given that two processes P and Q are failure trace inequivalent iff, under a realistic testing scenario, an environment can see their difference, one would expect, analogously to the situation with failures semantics, that one can build an environment from process algebraic operators that exploits that difference, i.e., that one can find a context $\mathcal{C}[-]$, such that the difference between $\mathcal{C}[P]$ and $\mathcal{C}[Q]$ becomes much more tangible. However, standard process algebras lack the expressiveness to achieve this.

So far the search for suitable process algebraic operators that allow us to exploit the difference between failure trace inequivalent processes has not born fruit. Most operators fall short in doing so, whereas others have too great a discriminating power. A prime example of the latter is the priority operator of Baeten, Bergstra & Klop [BBK87]. It allows building contexts that distinguish processes whenever they are *ready trace* equivalent. Here ready trace semantics (RT^*) is a bit finer than failure trace semantics (FT^*). When analysing how priority operators distinguish processes that are failure trace equivalent, one finds that the priority operator in fact has unrealistic powers, and is not likely implementable.

The present paper shows that the addition of a time-out operator to a standard process algebra suffices to exploit the difference between failure trace inequivalent processes. Here the time-out operator is simply an instance of the traditional action prefixing operator $\alpha._$, taking for α the time-out action t . I show that with this operator one can build, for any two failure trace inequivalent processes P and Q , a context $\mathcal{C}[-]$ such that $\mathcal{C}[P]$ and $\mathcal{C}[Q]$ are trace inequivalent. In fact, P and Q can be distinguished with *may testing*, as proposed by De Nicola & Hennessy [DH84]. Without the time-out operator, may testing merely distinguishes processes when they are trace inequivalent.

1.3. Capturing liveness properties while assuming justness. In [Gla19] I present a research agenda aiming at laying the foundations of a theory of concurrency that is equipped to ensure liveness properties of distributed systems without making fairness assumptions. The reason is that fairness assumptions, while indispensable for some applications, in many situations lead to false conclusions [GH19, Gla19]. Merely assuming progress, on the other hand, is often insufficient to obtain intuitively valid liveness properties. As an alternative to fairness assumptions, [GH19] proposes the weaker assumption of *justness*, that does not lead to false conclusions.

As pointed out in [Gla19], when assuming justness but not fairness, liveness properties are not preserved by strong bisimilarity, let alone by any of the other equivalences in the linear time – branching time spectrum. An adequate treatment of liveness therefore calls

for different semantic equivalences, ones that do not make all the identifications of strong bisimilarity.

Whereas adapting the notion of bisimilarity to take justness considerations properly into account is far from trivial, it appears that a version of failure trace semantics that preserves liveness properties when assuming justness but not fairness comes naturally, and is in a denotational approach to semantics hard to avoid. Here it is crucial that the semantics is based on *complete* failure traces, modelling potentially infinite observations of systems.

The present paper paves the way for such an approach by considering *partial* failure traces. This yields a semantic equivalence that is coarser than strong bisimilarity, and can be justified from an operational point of view, although it completely fails to respect liveness properties. The move from partial to complete failure trace equivalence is left for future work.

1.4. Absolute expressiveness. In comparing the expressiveness of process algebraic languages, an important distinction between absolute and relative expressiveness is made [Par08]. Relative expressiveness deals with the question whether an operator in one language can be faithfully mimicked by a context or open term in another. It is usually studied by means of valid encodings between languages [Gor10, Gla18]. As there is a priori no upper bound on how convoluted an operator one can add to a process algebra, it may not be reasonable to aim for a universally expressive language, in which all others can be expressed, unless a thorough discipline is proclaimed on which class of operators is admissible.

Absolute expressiveness deals with the question whether there are systems that can be denoted by a closed term in one language but not in another. It can for instance be argued that up to strong bisimilarity the absolute expressiveness of a standard process algebra like CCS with guarded recursion is not decreased upon omitting the parallel composition (although its relative expressiveness surely is). Namely, up to strong bisimilarity, each transition system that can be denoted by a closed CCS expression can already be denoted by such a CCS expression that does not employ parallel composition. This consideration occurs in the proof of Theorem 2.3. When using absolute expressiveness, the goal of a universally expressive process algebra becomes much less unrealistic. In fact, when not considering time, probabilities, or other features that are alien to process algebras like CCS, it could be argued, and is perhaps widely believed, that the version of CCS with arbitrary infinite sums and arbitrary systems of recursive equations is already universally expressive. The reason is that all that can be expressed by CCS-like languages are states in LTSs, and up to strong bisimilarity, each state in each LTS can already be denoted by such a CCS expression.

This conclusion loses support when factoring in justness, as discussed in Section 1.3, for in this setting strong bisimilarity becomes too coarse an equivalence to base a theory of expressiveness upon. In fact, the idea that standard process algebras like CCS are universally expressive has been challenged in [Gla05] and [GH15], where concrete and useful distributed systems are proposed that can not be modelled in such languages. In [Gla05] this concerns a system that stops executing a repeating task after a time-out goes off, whereas the systems considered in [GH15] are fair schedulers and mutual exclusion protocols. In both cases a proper treatment of justness appears to be a prerequisite for the correct modelling of such systems, and for this reason this task falls outside the scope of the current paper. However, additionally, both examples need something extra beyond what CCS has to offer, and both papers indicate that a simple priority mechanism would do the job.

Table 1: Structural operational interleaving semantics of CCSP_t

| | | | |
|---|---|---|---|
| $\alpha.x \xrightarrow{\alpha} x$ | $\frac{x \xrightarrow{\alpha} x'}{x + y \xrightarrow{\alpha} x'}$ | $\frac{y \xrightarrow{\alpha} y'}{x + y \xrightarrow{\alpha} y'}$ | $\frac{x \xrightarrow{\alpha} x'}{\mathcal{R}(x) \xrightarrow{\beta} \mathcal{R}(x')} \left(\begin{array}{l} \alpha=\beta=\tau \\ \vee \alpha=\beta=t \\ \vee (\alpha,\beta) \in \mathcal{R} \end{array} \right)}$ |
| $\frac{x \xrightarrow{\alpha} x'}{x \parallel_S y \xrightarrow{\alpha} x' \parallel_S y} (\alpha \notin S)$ | $\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'} (a \in S)$ | $\frac{y \xrightarrow{\alpha} y'}{x \parallel_S y \xrightarrow{\alpha} x \parallel_S y'} (\alpha \notin S)$ | |
| $\frac{x \xrightarrow{\alpha} x'}{\tau_I(x) \xrightarrow{\alpha} \tau_I(x')} (\alpha \notin I)$ | $\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')} (a \in I)$ | $\frac{\langle \mathcal{S}_X \mathcal{S} \rangle \xrightarrow{\alpha} y}{\langle X \mathcal{S} \rangle \xrightarrow{\alpha} y}$ | |

The current paper offers such a simple priority mechanism, and thereby paves the way for expressing the systems that have been proposed as witnesses for the lack of universal expressiveness of standard process algebras.

2. THE PROCESS ALGEBRA CCSP_t

Let A and V be countably infinite sets of *visible actions* and *variables*, respectively. The syntax of CCSP_t is given by

$$E ::= 0 \mid \alpha.E \mid E + E \mid E \parallel_S E \mid \tau_I(E) \mid \mathcal{R}(E) \mid X \mid \langle X | \mathcal{S} \rangle \text{ (with } X \in V_S)$$

with $\alpha \in \text{Act} := A \uplus \{\tau, t\}$, $S, I \subseteq A$, $\mathcal{R} \subseteq A \times A$, $X \in V$ and \mathcal{S} a *recursive specification*: a set of equations $\{Y = \mathcal{S}_Y \mid Y \in V_S\}$ with $V_S \subseteq V$ (the *bound variables* of \mathcal{S}) and \mathcal{S}_Y a CCSP_t expression.

The constant 0 represents a process that is unable to perform any action. The process $\alpha.E$ first performs the action α and then proceeds as E . The process $E + F$ will behave as either E or F . \parallel_S is a partially synchronous parallel composition operator; actions $a \in S$ must synchronise—they can occur only when both arguments are ready to perform them—whereas actions $\alpha \notin S$ from both arguments are interleaved. τ_I is an abstraction operator; it conceals the actions in I by renaming them into the hidden action τ . The operator \mathcal{R} is a relational renaming: it renames a given action $a \in A$ into a choice between all actions b with $(a, b) \in \mathcal{R}$. I require that all sets $\{b \mid (a, b) \in \mathcal{R}\}$ are finite. Finally, $\langle X | \mathcal{S} \rangle$ represents the X -component of a solution of the system of recursive equations \mathcal{S} . A CCSP_t expression E is *closed* if every occurrence of a variable X is *bound*, i.e., occurs in a subexpression $\langle Y | \mathcal{S} \rangle$ of E with $X \in V_S$.

The interleaving semantics of CCSP_t is given by the labelled transition relation $\rightarrow \subseteq \mathbb{P} \times \text{Act} \times \mathbb{P}$ on the set \mathbb{P} of closed CCSP_t terms or *processes*, where the transitions $P \xrightarrow{\alpha} Q$ are derived from the rules of Table 1. Here $\langle E | \mathcal{S} \rangle$ for E an expression and \mathcal{S} a recursive specification denotes the expression E in which $\langle Y | \mathcal{S} \rangle$ has been substituted for the variable Y , for all $Y \in V_S$.

The language CCSP is a common mix of the process algebras CCS [Mil90] and CSP [BHR84, Hoa85]. It first appeared in [Old87], where it was named following a suggestion by M. Nielsen. The family of parallel composition operators \parallel_S stems from [OH86], and incorporates the two CSP parallel composition operators from [BHR84]. The relation renaming operators $\mathcal{R}(_)$ stem from [Vaa93]; they combine both the (functional) renaming operators that are common to CCS and CSP , and the inverse image operators of CSP . The remaining constructs are common to CCS and CSP . The syntactic form of inaction 0, action prefixing

$\alpha.E$ and choice $E + F$ follows CCS, whereas the syntax of abstraction $\tau_I(_)$ and recursion $\langle X|\mathcal{S} \rangle$ follows ACP [BW90]. The only addition by me is the prefixing operator $t._$; so far it is merely an action that admits no synchronisation, concealment, or renaming.

Definition 2.1. A *strong bisimulation* is a symmetric relation \mathcal{R} on \mathbb{P} , such that, for all $(P, Q) \in \mathcal{R}$ and $\alpha \in Act$,

- if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some Q' with $P' \mathcal{R} Q'$.

Processes $P, Q \in \mathbb{P}$ are *strongly bisimilar*, $P \simeq Q$, if $P \mathcal{R} Q$ for some strong bisimulation \mathcal{R} .

Strong bisimilarity lifts in the standard way to *open* CCSP_t expressions, containing free with variables: $E \simeq F$ iff $P \simeq Q$ for each pair of closed substitution instances P and Q of E and F .

The common *strong bisimulation semantics* of process algebras like CCSP_t interprets closed expressions as \simeq -equivalence classes of processes, thereby identifying strongly bisimilar processes. Operators, or more generally open terms E , then denote n -ary operations on such equivalence classes, with n the number of free variables occurring in E . To make sure that the meaning of the operators is independent of the choice of representative processes within their equivalence classes, it is essential that \simeq is a congruence for all n -ary operators f :

$$\text{if } P_i \simeq Q_i \text{ for all } i = 1, \dots, n \text{ then } f(P_1, \dots, P_n) \simeq f(Q_1, \dots, Q_n).$$

This property holds for CCSP_t since the structural operation semantics of the recursion-free fragment of the language, displayed in Table 1, fits the *tyft/tyxt format* of [GV92]. Likewise, to make sure that also the meaning of the recursion construct is independent of the choice of representative expressions within their equivalence classes, \simeq needs to be *full congruence* for recursion [Gla17]:

$$\text{if } \mathcal{S}_Y \simeq \mathcal{S}'_Y \text{ for all } Y \in V_{\mathcal{S}} = V_{\mathcal{S}'}, \text{ then } \langle X|\mathcal{S} \rangle \simeq \langle X|\mathcal{S}' \rangle$$

for all recursive specifications \mathcal{S} and \mathcal{S}' and variables X with $X \in V_{\mathcal{S}} = V_{\mathcal{S}'}$. Again, this property holds since the semantics of Table 1 fits the *tyft/tyxt format with recursion* of [Gla17].

Definition 2.2. Given a recursive specification \mathcal{S} , write $X \xrightarrow{u} Y$, for variables $X, Y \in V_{\mathcal{S}}$, if Y occurs in the expression \mathcal{S}_X outside of all subexpressions $\alpha.E$ of \mathcal{S}_X . \mathcal{S} is *guarded* iff there is no infinite chain $X_0 \xrightarrow{u} X_1 \xrightarrow{u} X_2 \xrightarrow{u} \dots$. A process is *guarded* if all recursive specifications called by it are guarded.

Often, one restricts attention to guarded processes. The axioms of Table 2 are *sound* for \simeq , meaning that writing \simeq for $=$, and substituting arbitrary expressions for the variables x, y, z , or the meta-variables P_i and Q_j , turns them into true statements. In these axioms α, β range over Act and a, b over A . All axioms involving variables are equations. The axiom involving P and Q is a template that stands for a family of equations, one for each fitting choice of P and Q . This is the CCSP_t version of the *expansion law* from [Mil90]. The axiom $\langle X|\mathcal{S} \rangle = \langle \mathcal{S}_X|\mathcal{S} \rangle$ is the *Recursive Definition Principle* (RDP) [BW90]. It says that recursively defined processes $\langle X|\mathcal{S} \rangle$ satisfy their set of defining equations \mathcal{S} . In particular, this entails that each recursive specification has a solution. The axiom RSP [BW90] is a conditional equation, with as antecedents the equations of a guarded recursive specification \mathcal{S} . It says that the X -component of any solution of \mathcal{S} —a vector of processes substituted for the variables $V_{\mathcal{S}}$ —equals $\langle X|\mathcal{S} \rangle$. This is equivalent to the statement that the solutions of guarded recursive specifications must be unique.

Table 2: A complete axiomatisation for strong bisimilarity on guarded CCSP_t processes

| | | |
|--|---|--|
| $x + (y + z) = (x + y) + z$ | $\tau_I(x + y) = \tau_I(x) + \tau_I(y)$ | $\mathcal{R}(x + y) = \mathcal{R}(x) + \mathcal{R}(y)$ |
| $x + y = y + x$ | $\tau_I(\alpha.x) = \alpha.\tau_I(x)$ if $\alpha \notin I$ | $\mathcal{R}(\tau.x) = \tau.\mathcal{R}(x)$ |
| $x + x = x$ | $\tau_I(\alpha.x) = \tau.\tau_I(x)$ if $\alpha \in I$ | $\mathcal{R}(t.x) = t.\mathcal{R}(x)$ |
| $x + 0 = 0$ | $\langle X \mathcal{S} \rangle = \langle \mathcal{S}_X \mathcal{S} \rangle$ | $\mathcal{R}(a.x) = \sum_{\{b (a,b) \in \mathcal{R}\}} b.\mathcal{R}(x)$ |
| If $P = \sum_{i \in I} \alpha_i.P_i$ and $Q = \sum_{j \in J} \beta_j.Q_j$ then | | |
| $P \parallel_S Q = \sum_{i \in I, \alpha_i \notin \mathcal{S}} \alpha_i.(P_i \parallel_S Q) + \sum_{j \in J, \beta_j \notin \mathcal{S}} \beta_j.(P \parallel_S Q_j) + \sum_{i \in I, j \in J, \alpha_i = \beta_j \in \mathcal{S}} \alpha_i.(P_i \parallel_S Q_j)$ | | |
| Recursive Specification Principle (RSP) $\mathcal{S} \Rightarrow X = \langle X \mathcal{S} \rangle$ (\mathcal{S} guarded) | | |

Theorem 2.3. For guarded $P, Q \in \mathbb{P}$, one has $P \Leftrightarrow Q$ iff $P = Q$ is derivable from the axioms of Table 2.

Proof Sketch. “If”, the *soundness* of the axiomatisation of Table 2, is an immediate consequence of the soundness of the individual axioms.

“Only if”, the *completeness* of the axiomatisation: Using the axioms from the first box of Table 2 any guarded CCSP_t process P can be brought in the form $\sum_{i \in I} \alpha_i.P_i$ —a *head normal form*. In fact, $\{(\alpha_i, P_i) \mid i \in I\}$ can be chosen to be $\{(\alpha_i, P_i) \mid P \xrightarrow{\alpha} P_i\}$. Given a process P , let $\text{reach}(P)$ be the smallest set of processes containing P , such that $R \xrightarrow{\alpha} R'$ with $R \in \text{reach}(P)$ implies $R' \in \text{reach}(P)$. Now dedicate to each $R \in \text{reach}(P)$ a variable X_R , and consider the recursive specification \mathcal{S} with $V_{\mathcal{S}}$ the set of all those variables, and as equations $X_R = \sum_{k \in K} \alpha_k.X_{R_k}$, using the head normal form of R . Using RSP, we derive $P = \langle X_P | \mathcal{S} \rangle$. Likewise, we can derive $Q = \langle X_Q | \mathcal{S}' \rangle$ for a recursive specification \mathcal{S}' , consisting of equations of the form $X_S = \sum_{l \in L} \beta_l.X_{S_l}$.

Once we have two such recursive specifications, as described for instance in [Mil90] we can create a combined recursive specification \mathcal{S}'' such that both P and Q are the X-components of solutions of \mathcal{S}'' . With RSP one then derives $P = Q$. \square

The above proof idea stems from Milner [Mil90] and can be found in various places in the literature, but always applied to finite-state processes, or some other restrictive subset of a calculus like CCSP_t . Since the set of true statements $P \Leftrightarrow Q$, with P and Q processes in a process algebra like CCSP_t , is well-known to be undecidable, and even not recursively enumerable, it was widely believed that no sound and complete axiomatisation of strong bisimilarity could exist. Only in March 2017, Kees Middelburg [Mid17] observed (in the setting of the process algebra ACP [BW90]) that the above standard proof applies verbatim to arbitrary guarded processes. His result does not contradict the non-enumerability of the set of true statements $P \Leftrightarrow Q$, due to the fact that RSP is a proof rule with infinitely many premises.

3. ADDING A TIME-OUT TO CCSP

The process algebra CCSP, just like CCS, CSP and ACP, is an *untimed* process algebra, meaning that its semantics abstracts from a quantification of the amounts of time that elapse during or between the execution of the actions. Untimed process algebras do not deny that the execution of processes takes time; they abstract from timing merely in not

specifying how much time elapses here and there. An untimed process could therefore be seen as a timed process, in which each occurrence of a particular period of time is replaced by a nondeterministic choice allowing *any* amount of time.

From this perspective, one could wonder whether time elapses during the execution of the actions $\alpha \in Act$, or between the actions, that is, in the states. In some works, notably [GV87], it is assumed that time happens during the execution of actions. This leads to a rejection of equations like $a \parallel b = a.b + b.a$,² which are fundamental for *interleaving semantics* [Mil90, BHR84, Hoa85, BW90, OH86], for the left-hand side allows an overlap in time of the actions a and b , whereas the right-hand side does not. The default opinion, however, is that the execution of actions is instantaneous, so that time must elapse between the actions. The following formulation of this assumption is taken from Hoare [Hoa85, Page 24]:

The actual occurrence of each event in the life of an object should be regarded as an instantaneous or an atomic action without duration. Extended or time-consuming actions should be represented by a pair of events, the first denoting its start and the second denoting its finish.

In [GV97] it is pointed out that simply replacing each occurrence of a durational action α by a sequence $\alpha^+.\alpha^-$ of two instantaneous actions, one denoting the start and one the end of α , is insufficient to adequately capture the durational nature of actions—it even makes a difference whether actions are split into two or into three parts. The problem arises from the possibility that two actions α may occur independently in parallel, and the above splitting allows for the possibility that the end of one such action is confused with the end of the other. A possible solution is to model a durational action α as a process $\sum_{i \in I} \alpha_i^+.\alpha_i^-$, where each start action α^+ is equipped with a tag $i \in I$ that has to be matched by the corresponding end action α^- . When the choice I of tags is large enough, this suffices to model durational actions in terms of instantaneous actions and durational states. It is for that reason that in the present paper I focus on time elapsing in states only.

Let s be the state between the a - and τ -transitions in the process $a.\tau.P$. Assume that time may elapse in state s , in the sense that after the execution of a and before the execution of τ , the modelled system, for some positive amount of time, does nothing whatsoever. Then it is hard to imagine what could possibly trigger the system to resume activity after this amount of time. A system that does nothing whatsoever may be regarded as dead, and remains dead.

The philosophy of Milner [Mil90] appears to be that once a system reaches state s , it will proceed with the τ -transition immediately. However, once it reaches a state $\sum_{i \in I} a_i.P_i$, where the a_i are visible actions, it remains idle until the environment of the system triggers the execution of one of the actions a_i . The system is *reactive*, in the sense that visible actions occur only in reaction to stimuli from the environment. The latter could be modelled as the environment being a user of the system that may press buttons labelled a , for $a \in A$. In a state where the system is not able to engage in an a -transition, the user may exercise pressure on the a button, but it will not go down. In this philosophy, all time that elapses is due to the system waiting on its environment.

An unsatisfactory aspect of the above philosophy is that there is an asymmetry between a system and its environment. The latter can spontaneously cause delays, but the former may delay only in reaction to the latter. When we would model the environment also as a closed process algebraic expression, and put it in parallel with the system, in such a way

²I abbreviate $a.0$ by a and \parallel_\emptyset by \parallel . Moreover, $+$ binds weaker than $a._$, and \parallel_S binds weakest of all.

that no action depends on triggers from outside the system/environment composition, then the resulting composition performs all its actions instantaneously, possibly until it reaches a deadlock, from which it cannot recover.

The present paper offers an alternative account on the passage of time, with a symmetric view on systems and their environments. In fact, it tries to be as close as possible to the philosophy of Milner described above, but adds the expressiveness to model delays caused by a system rather than its environment. Here there are two overriding design decisions. First, amounts of time will not be quantified, for I aim to remain in the realm of untimed process algebras. Second, the assumption of *progress* [GH19] needs to be maintained. Progress says that from state s above one will sooner or later reach the process P , i.e., an infinite amount of waiting is ruled out. This assumption is essential for the formulation and verification of liveness properties, saying that a system will reach some goal eventually [GH19].

Following [Gla01, Gla93], I explicitly allow the environment of a system to exercise pressure on multiple buttons at the same time. The actions for which, at a given time, the environment is exercising pressure, are *allowed* at this time, whereas the others are *blocked*. Instead of a model with buttons, one may imagine a model with switches for each visible action, where the environment, at discrete points in time, can toggle the switches of any action between *allowed* and *blocked*. In particular the occurrence of a visible action may trigger the environment to rearrange its switches.

To model time elapsing in a state, I assume that the state may be equipped with one or more time-outs. A *time-out* is a kind of clock that performs some internal activity, from which I abstract, and at the end of this activity emits a signal that causes a state-change. The time-out always runs for an unspecified, but positive and finite amount of time. For each time-out of a state there is an outgoing transition labelled t that models this state-change. A t -transition is not observable by the environment. So, apart from the associated time-out, it is similar to a τ -transition.

The state $a.P + b.Q + t.R$, for instance, has one outgoing t -transition, corresponding with one time-out. If, upon reaching this state, the environment is allowing one of a or b , this action will happen immediately, reaching the follow-up state P or Q ; in case the environment allows both a and b , the choice between them is made nondeterministically, i.e., by means that are not part of my model. If, upon reaching the state $a.P + b.Q + t.R$, the environment is not allowing a or b , the system idles, until either the environment changes and allows a or b after all, or the time-out goes off, triggering the t -transition.

A state like $a.P + t.Q + t.R$ simply has two time-outs; which one goes off first is not predetermined. If the transition to R occurs, the other time-out is simply discarded. A state like $a.P + b.Q$ has no time-outs. Upon reaching this state the system idles until the environment allows a or b to happen.

Note that in $a.P + b.Q + t.R$ the action t should not be seen as a durational τ -action, for that would suggest that as soon as that action starts, one has lost the option to perform a or b . Instead, the options a and b remain open until the instantaneous transition to R occurs.

The expressive power of time-outs. To model that a process like $a.\tau.P$ simply spends some time in the state s right before the τ -action, one could write $a.t.\tau.P$. Furthermore, a durational action α , contemplated above, could be modelled as $\sum_{i \in I} \alpha_i^+.t.\alpha_i^-$. Both uses do not involve placing a process $t.P$ in a $+$ -context. The latter is useful for modelling a simple priority mechanism. Suppose one wants to model a process like $a.P + b.Q$, which

can react on a - and b -stimuli from the environment. However one has a preference for $a.P$, in the sense that if the environment allows both possibilities, $a.P$ should be chosen. This can be modelled as $a.P + t.b.Q$ or $a.P + t.(a.P + b.Q)$. In an environment allowing both a and b , this system will perform $a.P$. But in an environment just allowing b , first some idling occurs, and then $b.Q$.

Three crucial laws. The most fundamental law concerning time-outs, which could be added as an axiom to Table 2, is

$$\tau.P + t.Q = \tau.P . \quad (3.1)$$

It says that a choice between $\tau.P$ and $t.Q$ will always be resolved in favour of $\tau.P$, because the τ -action is not contingent on the environment and thus can happen immediately, before the time-out associated with $t.Q$ has a chance to go off.

A process $t.(\tau.P + b.Q)$ will, after waiting for some time, make a choice between $\tau.P$ and $b.Q$. In case the environment allows b , this choice is nondeterministic. In case the environment blocks b at the time the time-out associated with the t -transition goes off, the system will immediately take the $\tau.P$ branch, thereby discarding $b.Q$. Henceforth I will substitute $\tau_{\{b\}}(P)$ for P , to indicate that this is a process that cannot perform a b -action. Now consider two such processes placed in parallel, synchronising on the action b . Each of these processes acts as the environment in which the other will be placed. In this paper I assume that here the synchronisation on b can *not* occur:

$$t.(\tau.\tau_{\{b\}}(P) + b.Q) \parallel_{\{b\}} t.(\tau.\tau_{\{b\}}(S) + b.T) = t.\tau.\tau_{\{b\}}(P) \parallel_{\{b\}} t.\tau.\tau_{\{b\}}(S) . \quad (3.2)$$

Intuitively, the reason is that the time-outs on the left and on the right go off at random points in real time. I explicitly assume no causal link of any kind between the relative timing of these events. Hence the probability that the two time-outs go off at the very same moment is 0, and this possibility can be discarded. So either the left-hand side in the parallel composition reaches the state $L := \tau.\tau_{\{b\}}(P) + b.Q$ before the right-hand process reaches the state $R := \tau.\tau_{\{b\}}(S) + b.T$, or vice versa. For reasons of symmetry I only consider the former case. When the state L is reached, the environment of that process is not yet ready to synchronise on b , so $\tau.\tau_{\{b\}}(P)$ will happen instead. This forces the right-hand side, when reaching the state R , to choose $\tau.\tau_{\{b\}}(S)$, for $\tau_{\{b\}}(P)$ cannot synchronise on b .

Algebraically, (3.2) can be derived from (3.1) and the axioms of Table 2, when taking for granted the obvious identity $\tau_{\{b\}}(x) \parallel_{\{b\}} t.(y + b.z) = \tau_{\{b\}}(x) \parallel_{\{b\}} t.y$ (whose closed instances with guarded recursion are derivable from the axioms of Table 2):

$$\begin{aligned} & t.L \parallel_{\{b\}} t.R \stackrel{\text{(Expansion Law)}}{=} \\ & t. \left(L \parallel_{\{b\}} t.R \right) + t. \left(t.L \parallel_{\{b\}} R \right) \stackrel{\text{(Expansion Law)}}{=} \\ & t. \left(\tau. \left(\tau_{\{b\}}(P) \parallel_{\{b\}} t.R \right) + t. \left(L \parallel_{\{b\}} R \right) \right) + t. \left(t.L \parallel_{\{b\}} R \right) \stackrel{(3.1)}{=} \\ & t. \left(\tau. \left(\tau_{\{b\}}(P) \parallel_{\{b\}} t.R \right) \right) + t. \left(t.L \parallel_{\{b\}} R \right) \stackrel{\text{(obvious identity; same reasoning on the right)}}{=} \\ & t. \left(\tau. \left(\tau_{\{b\}}(P) \parallel_{\{b\}} t.\tau.\tau_{\{b\}}(S) \right) \right) + t. \left(\tau. \left(t.\tau.\tau_{\{b\}}(P) \parallel_{\{b\}} \tau_{\{b\}}(S) \right) \right) \stackrel{\text{(same way back)}}{=} \\ & t.\tau.\tau_{\{b\}}(P) \parallel_{\{b\}} t.\tau.\tau_{\{b\}}(S) . \end{aligned}$$

To reject (3.2) one would need to take into account seriously the possibility that the two time-outs take exactly equally long. This corresponds to the addition of the rule

$$\frac{x \xrightarrow{t} x' \quad y \xrightarrow{t} y'}{x \parallel_S y \xrightarrow{t} x' \parallel_S y'}$$

to the structural operational semantics of CCSP_t , and the summand

$$+ \sum_{i \in I, j \in J, \alpha_i = \beta_j = t} t.(P_i \parallel_S Q_j)$$

to the expansion law of CCSP_t . While this could very well lead to an interesting alternative treatment of timeouts, I will not pursue this option here.

Another law, that could also be added as an axiom to Table 2, is

$$a.P + t.(Q + \tau.R + a.S) = a.P + t.(Q + \tau.R) . \quad (3.3)$$

When the process $a.P + t.(Q + \tau.R + a.S)$ runs in an environment that allows a , the summand $t.(Q + \tau.R + a.S)$ will not be taken, so it does not matter if there is a summand $a.S$ after the t -action. And when this process runs in an environment that keeps blocking a long enough, the summand $a.S$ will not be taken. So the only way to execute $a.S$ is in an environment that initially blocks a , but then makes a change to allow a . In this case we need to consider two independently chosen points in time: time t_e where the environment starts allowing a , and time t_s where the time-out goes off that makes the system take the (unobservable) t -transition to the state $Q + \tau.R + a.S$. Following the same reasoning as for law (3.2), the probability that $t_e = t_s$ is 0, so this possibility can be ignored. In case t_e occurs before t_s , the system will never reach the state $Q + \tau.R + a.S$. In case t_e occurs after t_s , the system will, in state $Q + \tau.R + a.S$, choose $\tau.R$ (or a branch from Q) over $a.S$. In either case, the summand $a.S$ can just as well be omitted.

Time guardedness. CCSP_t allows expressions such as $\langle X | X = \tau.X \rangle$ or $\langle X | X = a.X \rangle$ where infinitely many actions could happen instantaneously. When this is felt as a drawback, one could impose syntactic restrictions on the language to rule out such behaviour. The following is an example of such a restriction.

Definition 3.1. Given a recursive specification \mathcal{S} , write $X \xrightarrow{tu} Y$, for variables $X, Y \in V_{\mathcal{S}}$, if Y occurs in expression \mathcal{S}_X outside of all subexpressions $t.E$ of \mathcal{S}_X . \mathcal{S} is *time guarded* iff there is no infinite chain $X_0 \xrightarrow{tu} X_1 \xrightarrow{tu} \dots$. A process is *time guarded* if all recursive specifications called by it are time guarded.

Clearly, the time guarded CCSP_t expressions, a subset of the guarded CCSP_t expressions, allow only finitely many actions to occur between any two time-out transitions.

4. TRACE AND FAILURES EQUIVALENCE FAIL TO BE CONGRUENCES

The paper [Gla01] presents a spectrum of semantic equivalences on processes encountered in the literature. It only deals with *concrete* processes, not featuring the internal action τ . Of course, the time-out action t does not occur in [Gla01] either. Figure 1 reproduces the spectrum of [Gla01, Figure 9], while omitting those semantic equivalences that fail to be congruences for the τ -free fragment of CCSP —those were coloured red in [Gla01, Figure 9].

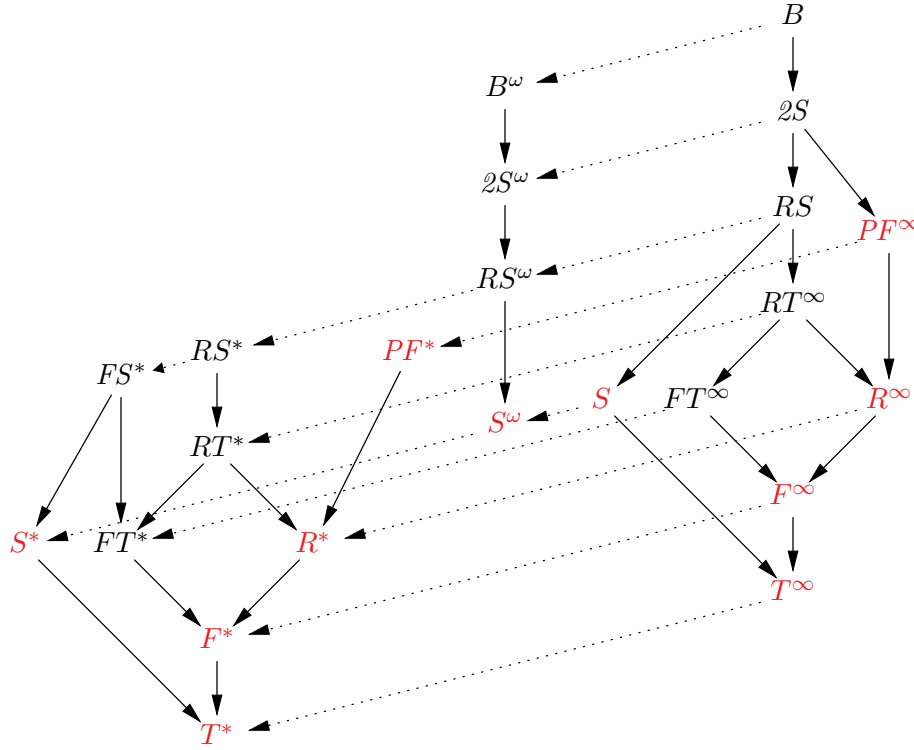


Figure 1: The linear time – branching time spectrum [Gla01]

(Strong) bisimilarity (B) [Mil90, Par81], cf. Definition 2.1, is the finest semantics of Figure 1, i.e., making the fewest identifications between processes, or yielding the smallest equivalence classes. It sits at the *branching time* side of the spectrum, as it distinguishes processes based on the timing of choices between different execution sequences, e.g., $a.(b.c + b.d) \neq a.b.c + a.b.d$.

(Partial) trace semantics (T^*) [Hoa80] is the coarsest semantics of Figure 1, i.e., making the most identifications. It sits at the *linear time* side of the spectrum, as it abstracts a process into the sequences of action occurrences that can be observed during its runs, its *traces*, and identifies two processes when they have the same traces.

Failures semantics (F^*) is the default semantics of the process algebra CSP [BHR84, Hoa85]. When restricting to concrete processes, it falls neatly between T^* and B . Similar to bisimulation semantics, failures semantics rejects the equation $a.(b + c) = a.b + a.c$ that holds in trace semantics. The main reason for doing so is that in an environment where action c remains blocked, the left-hand side will surely do the b -action, whereas the right-hand side may deadlock instead. However, no such argument distinguishes $a.(b.c + b.d)$ and $a.b.c + a.b.d$, and hence these processes are identified in failures semantics.

Failure trace semantics (FT^*) was proposed in [Gla01], and first presented in [Bae86], as the natural completion of the square suggested by failures, readiness [OH86] and ready trace semantics [BBK87]. For finitely branching processes it coincides with *refusal semantics*, introduced by Phillips in [Phi87]. Like failure semantics it rejects the identity $a.(b + c) = a.b + a.c$, but accepts $a.(b.c + b.d) = a.b.c + a.b.d$. The standard example of two processes that are failures equivalent but not failure trace equivalent is displayed in Figure 2.

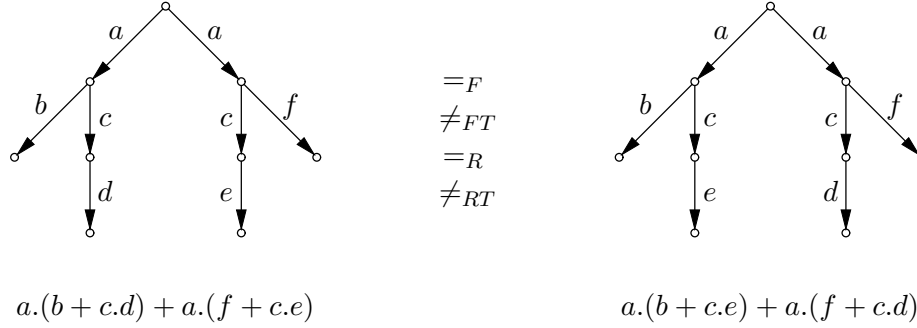


Figure 2: Failures and ready equivalent, but not failure trace or ready trace equivalent [Gla01]

The following argument shows that neither trace nor failures equivalence are congruences for the operators of CCSP_t . That is, there are failures equivalent processes P and Q (that are therefore certainly trace equivalent) and a recursion-free CCSP_t -context $\mathcal{C}[-]$ such that $\mathcal{C}[P]$ and $\mathcal{C}[Q]$ are trace inequivalent.

Example 4.1. Take $P := a.(b + c.d) + a.(f + c.e)$ and $Q := a.(b + c.e) + a.(f + c.d)$. Use the context $\mathcal{C}[-] := \tau_{\{a,b,c\}}(a.(b + t.c) \parallel_{\{a,b,c,f\}} -)$. Then only $\mathcal{C}[Q]$ can ever perform the action d . Namely, the context, enforcing synchronisation on the actions a , b , c and f , implements a priority of b over c (while blocking f altogether); whenever its argument process P or Q faces a choice between a b - and a c -transition, within this context it must take the b -transition. This explains why $\mathcal{C}[P]$ can never reach the action d . The abstraction operator $\tau_{\{a,b,c\}}$ is not redundant, although $\tau_{\{b\}}$ would have worked as well. Without such an operator, the process $\mathcal{C}[P]$ might run in an environment in which b is blocked, and here the d could occur after all.

Interestingly, I could make this argument without even providing definitions of trace and failures equivalence on LTSs that feature time-out transitions. The only two things we need to know about such equivalences are that they must (a) distinguish a process that can never do the action d from one that can, and (b) identify the two processes of Figure 2. Property (b) is satisfied when merely requiring that on concrete process (without τ and t) these equivalences agree with the definitions in [Gla01]. The argument is thus also independent of the question how these equivalences are extended to a setting with the hidden action τ .

The same example, with its above analysis, also shows that ready equivalence (R^*) fails to be a congruence for CCSP_t , for it also identifies the processes P and Q .

In Figure 1 I have coloured red all those semantics that can be seen to fail the congruence requirement by a similar argument. For the infinitary versions of trace, failures and readiness semantics (T^∞ , F^∞ and R^∞), the same example applies. For possible-futures equivalence (PW^* and PW^∞) one takes as witness the two processes of [Gla01, Counterexample 7], in combination with the obvious context that gives priority of b over a as well as c . Simulation equivalence (S) and its finitary variants (S^* and S^ω) make the identification $a.b + a = a.b$ [Gla01, Counterexample 2]. Using this, the following argument shows that these semantics also fail to be congruences for CCSP_t .

Example 4.2. Take $P := a.b + a$ and $Q = a.b$, and use the context $\mathcal{C}[-] := \tau_{\{a,b\}}(a.(b + t.d) \parallel_{\{a,b\}} -)$. Then only $\mathcal{C}[P]$ can ever perform the action d . Here d can be done by the

context only, rather than by the processes P or Q . However, after synchronising on a , the context will proceed towards d only when the environment of $b + t.d$, which is the process P or Q synchronising on b , blocks the b -transition. This is possible for P but not for Q . In this example the possibility to deadlock (or the impossibility to perform a certain action) is made observable through a context.

It follows that failure trace semantics is in fact the coarsest semantics reviewed in [Gla01] that could be a congruence for the operators of CCSP_t . By the arguments in Section 1.2 it also is the finest semantics with a convincing testing scenario, in the sense that all finer semantics make distinctions that are hard to justify. Together, this tells us that failure trace semantics may be the only reasonable semantics for CCSP_t .

The next section proposes an extension of the definition of failure trace semantics (in [Gla01] only given for concrete processes, without the special actions τ and t) to arbitrary LTSs featuring τ and t , and thereby to CCSP_t . Of course on LTSs with τ but without t it coincides with the definition given in [Gla93].

Then Section 6 shows that, when restricting the choice operator $+$ of CCSP_t to guarded choice, failure trace equivalence is in fact a congruence for the operators of CCSP_t . To obtain a congruence for the $+$, a *rooted* version of failure trace equivalence is defined that records more information on the behaviour of process around their initial states. This situation is the same as for weak bisimilarity [Mil90] and virtually all other semantic equivalences that partially abstract from the hidden action τ .

In Section 8 I then show that for any two failure trace inequivalent processes P and Q one can find a recursion-free CCSP_t -context $\mathcal{C}[_]$ such that $\mathcal{C}[P]$ and $\mathcal{C}[Q]$ are trace inequivalent. This means that the arguments from Examples 4.1 and 4.2 not only apply to the semantics surveyed in [Gla01], but in fact to *any* semantics that is incomparable with or coarser than failure trace semantics.

5. PARTIAL FAILURE TRACE SEMANTICS OF CCSP_t

A *finite failure trace* is a sequence $\sigma \in (A \cup \mathcal{P}(A))^*$ of actions and sets of actions. It represents an observation of a system. Each action $a \in A$ occurring in σ represents the observation of the instantaneous occurrence of that action. Each set $X \subseteq A$ occurring in σ represents the observation of a period of idling (meaning that no actions occur) during which the environment allows (i.e., is ready to synchronise with) exactly the actions in X . Such a set is called a *refusal set* [BHR84, Gla01], because it corresponds with an offer X from the environment that is refused by the system. A *partial failure trace* $\sigma \top$ consists of a finite failure trace σ followed by the tag \top . It represents a partial observation of a system. The symbol \top indicates the act, on the part of the observer, of ending the observation.

Table 3: Operational failure trace semantics of CCSP_t

| | | |
|---|--|---|
| $\top \in FT^*(x)$ | $\frac{x \xrightarrow{a} y \quad \rho \in FT^*(y)}{a\rho \in FT^*(x)}$ | $\frac{x \xrightarrow{\tau} y \quad \rho \in FT^*(y)}{\rho \in FT^*(x)}$ |
| $\frac{x \not\xrightarrow{\alpha} \text{ for all } \alpha \in X \cup \{\tau\} \quad \rho \in FT^*(x)}{X\rho \in FT^*(x)}$ | $\frac{x \not\xrightarrow{\alpha} \text{ for all } \alpha \in X \cup \{\tau\} \quad x \xrightarrow{t} y \quad X\rho \in FT^*(y)}{X\rho \in FT^*(x)}$ | $\frac{x \not\xrightarrow{\alpha} \text{ for all } \alpha \in X \cup \{\tau\} \quad x \xrightarrow{t} y \quad a\rho \in FT^*(y)}{Xa\rho \in FT^*(x)} \quad (a \in X)$ |

Table 3 derives the set $FT^*(P)$ of partial failure traces of a process P . Here concatenation of sequences is denoted by juxtaposition. The first rule testifies that it is always possible to end the observation and just record \top . The next two rules say that visible actions are observed, whereas hidden actions are not. The fourth rule says that, when the environment allows the set of actions X , idling in a given state x is possible iff from that state no actions $a \in X$, nor the hidden action τ , can occur. These four rules stem in essence from [Gla01, Gla93]. The last two rules deal with the time-out t . To follow a transition $x \xrightarrow{t} y$, the system must be idling in state x until the time-out occurs. Let X be the set of actions allowed by the environment at this time. As explained with Law (3.3), I ignore the possibility that the environment changes the set of allowed actions at the exact same time as the occurrence of the time-out. So right after the time-out occurs, X is still the set of actions allowed by the environment. At this time there are two possibilities. Either the system keeps idling, namely when none of the actions from X can occur in state y ; or the system performs one of the actions $a \in X$ (possibly after some τ -transitions). These two possibilities correspond with the remaining two rules.

Definition 5.1. Processes $P, Q \in \mathbb{P}$ are *partial failure trace equivalent*, $P \equiv_{FT}^* Q$, iff $FT^*(P) = FT^*(Q)$.

A path-based characterisation of partial failure traces.

Definition 5.2. A finite *path* $\pi : P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_\ell} P_\ell$ is an alternating sequence of states/processes and transitions, starting and ending with a state, each transition going from the state before it to the state after it. One says it is a *path of* $P \in \mathbb{P}$ if its first state P_0 is P .

A process $P \in \mathbb{P}$ is *stable*, notation $stable(P)$, if $P \not\xrightarrow{\tau}$, i.e., if it has no outgoing τ -transitions. The set $\mathcal{I}(P)$ of *initial visible actions* of a process P is $\{a \in A \mid P \xrightarrow{a}\}$, provided P is stable. Here $P \xrightarrow{a}$ means that $P \xrightarrow{a} Q$ for some $Q \in \mathbb{P}$. If P is unstable, $\mathcal{I}(P) := \perp$.

The *concrete ready trace* $crt(\pi)$ of a finite path π as above is $\mathcal{I}(P_0)\alpha_1\mathcal{I}(P_1)\alpha_2 \dots \alpha_\ell\mathcal{I}(P_\ell)$. The (abstract) *ready trace* $rt(\pi)$ of π is obtained from $crt(\pi)$ by leaving out all occurrences of τ and \perp .

Note that each partial failure trace σ of a process P is generated by a path π of P , and is in fact completely determined by $rt(\pi)$. Each occurrence of action a in σ originates from a transition \xrightarrow{a} occurring in π (by application of the second rule of Table 3) and each occurrence of a refusal set X in σ originates (by application of the fourth or sixth rule) from a stable process P_i in π such that $\mathcal{I}(P_i) \cap X = \emptyset$. The last symbol \top of σ originates from the last state P_ℓ of π (by application of the first rule). The mapping from the action occurrences in σ to the A -labelled transitions in π must be bijective, and any two elements (actions, refusal sets or \top) of σ occur in the same order in σ as their originators occur in π .

Let ξ be the occurrence of an action, refusal set or \top in σ , originating from $\xrightarrow{\alpha_i}$ or P_j . If ξ is the first element of σ , the *realm* of ξ is the prefix of π ending in P_j . In case a or Y is the symbol in σ preceding ξ , with a originating from $\xrightarrow{\alpha_i}$ or Y originating from P_i , then $j \geq i$ and the *realm* of ξ is the subpath $P_i \xrightarrow{\alpha_{i+1}} \dots \xrightarrow{\alpha_j} P_j$. Thus, σ partitions π into realms, with each two adjacent realms having exactly one state in common.

Each transition in the realm of \top must be labelled τ , matching an application of the third rule of Table 3. Each transition $\xrightarrow{\alpha_k}$ in the realm of a refusal set X must be labelled t

or τ , matching applications of the third and fifth rules; if $\alpha_k = t$ then P_{k-1} must be stable and $\mathcal{I}(P_{k-1}) \cap X = \emptyset$. Only the first transition in the realm of an action occurrence a could be labelled t ; if it is then in σ action a is preceded with a refusal set X such that $a \in X$. This matches an application of the sixth rule.

In particular, a path featuring a t -labelled transition leaving from an unstable state does not generate any partial failure traces.

System- versus environment-ended periods of idling.

Observation 5.3. $\sigma X \rho \in FT^*(P) \Leftrightarrow \sigma X X \rho \in FT^*(P)$.

An occurrence of a set X in a partial failure trace σ denotes a period p of idling, during which X is the set of actions allowed by the environment. During period p the system idles because none of the actions in X is enabled by the system, and none of the others is allowed by the environment. Such a period can end in exactly two ways: either by a time-out within the modelled system, or by the environment changing the set of actions it allows. The latter can be thought of as the occurrence of time-out outside the modelled system. In this paper I ignore the possibility that a time-out within the system occurs at the exact same moment as a time-out outside the system. Hence there can never be ambiguity on which party ends a period of idling.

I will now show how from the shape of a partial failure trace one can deduce which party ends a given period of idling. If, within σ , X is followed by a set $Y \neq X$, the period p is followed by a period q of idling, this time with Y the set of actions allowed by the environment. The transfer from period p to period q must therefore be triggered by the environment, namely by changing the set of actions it allows to occur. If, within σ , X is followed by an action $a \in X$, the state of the system in which action a occurs must be different from the state of the system in which X is refused, and action a not enabled. It follows that period p must be ended by a time-out occurring within the system. If, within σ , X is followed by an action $a \notin X$, period p must be ended by the environment. For if the set of actions allowed by the environment is not changed after period p , the action $a \notin X$ would still not be allowed.

Definition 5.4. An occurrence of X in a partial failure trace σ is *system-ended* iff within σ , X is followed by an action $a \in X$.

6. CONGRUENCE PROPERTIES

In this section I show that partial failure trace equivalence is a congruence for the operators of $CCSP_t$, with the exception of the choice operator $+$. I also characterise the coarsest equivalence included in \equiv_{FT}^* that also is a congruence for $+$.

Definition 6.1. Let $F \subseteq (A \cup \mathcal{P}(A))^* \top$, meaning F is a set of partial failure traces with the end tag \top . Then $col(F)$ is the smallest set containing F such that $\sigma X X \rho \in col(F) \Rightarrow \sigma X \rho \in col(F)$.

Theorem 6.2. \equiv_{FT}^* is a congruence for the parallel composition operators \parallel_S .

Proof. I need to show that $P \equiv_{FT}^* P'$ and $Q \equiv_{FT}^* Q'$ implies $P \parallel_S Q \equiv_{FT}^* P' \parallel_S Q'$. This is equivalent to showing that $FT^*(P \parallel_S Q)$ is completely determined by $FT^*(P)$, S and $FT^*(Q)$.

Below I define, for each $S \subseteq A$, a binary operator \parallel_S that takes as arguments sets of partial failure traces, and produces again a set of partial failure traces. In the appendix—Proposition A.1—I prove that

$$FT^*(P \parallel_S Q) = col(FT^*(P) \parallel_S FT^*(Q)) . \quad (6.1)$$

This yields Theorem 6.2. □

In order to define the operator \parallel_S , I analyse how a partial failure trace of $P \parallel_S Q$ decomposes into partial failure traces of P and Q .

Definition 6.3. A *decomposition* of a partial failure trace $\sigma = \sigma_1 \sigma_2 \cdots \sigma_n \top \in (A \cup \mathcal{P}(A))^* \top$ w.r.t. a set $S \subseteq A$ is a pair σ_L, σ_R of partial failure traces, obtained by leaving out all \star -elements from sequences $\sigma^L = \sigma_1^L \sigma_2^L \cdots \sigma_n^L \top$ and $\sigma^R = \sigma_1^R \sigma_2^R \cdots \sigma_n^R \top \in (A \cup \{\star\} \cup \mathcal{P}(A))^* \top$, such that, for all $i = 1, \dots, n$,

- if $\sigma_i \in A \setminus S$ then either $\sigma_i^L = \sigma_i$ and $\sigma_i^R = \star$, or $\sigma_i^R = \sigma_i$ and $\sigma_i^L = \star$,
- if $\sigma_i \in S$ then $\sigma_i^L = \sigma_i^R = \sigma_i$, and
- if $\sigma_i \in \mathcal{P}(A)$ then $\sigma_i^L \in \mathcal{P}(A)$ and $\sigma_i^R \in \mathcal{P}(A)$.

This decomposition is *valid* if, for each $i = 1, \dots, n$ for which $\sigma_i \in \mathcal{P}(A)$, one has

$$\sigma_i \text{ is system-ended in } \sigma \Leftrightarrow ((\sigma_i^L \text{ is system-ended in } \sigma_L) \vee (\sigma_i^R \text{ is system-ended in } \sigma_R)) \quad (6.2)$$

$$\neg((\sigma_i^L \text{ is system-ended in } \sigma_L) \wedge (\sigma_i^R \text{ is system-ended in } \sigma_R)) \quad (6.3)$$

$$\sigma_i^L \setminus S = \sigma_i \setminus S = \sigma_i^R \setminus S \quad \text{and} \quad (\sigma_i^L \cap S) \cup (\sigma_i^R \cap S) = \sigma_i \cap S . \quad (6.4)$$

For $F, G \subseteq (A \cup \mathcal{P}(A))^* \top$ sets of partial failure traces, let $F \parallel_S G \subseteq (A \cup \mathcal{P}(A))^* \top$ be the set of partial failure traces σ , such that for some valid decomposition one has $\sigma_L \in F$ and $\sigma_R \in G$.

Definition 6.3 says that in any partial failure trace $\sigma \in FT^*(P \parallel_S Q)$, each occurrence of an action $a \notin S$ in σ must stem from either P or Q , whereas each occurrence of an action $a \in S$ must stem from both. A set X occurring in σ denotes a period of idling; necessarily both P and Q idle during this period. I justify (6.2)–(6.4) informally below; formally, these requirements are fully justified by the fact that they allow me to prove (6.1).

In a parallel composition $P \parallel_S Q$ there are three parties that can block actions, or end idle periods: P , Q and the global environment \mathcal{E} of the composition. The environment of P consists of Q and \mathcal{E} .

An idle period of $P \parallel_S Q$ ends through a single time-out. If this timeout occurs in $P \parallel_S Q$ itself, rather than in its environment, it must occur in exactly one of the components P and Q . And if the time-out stems from the environment it occurs in neither component. This explains conditions (6.2) and (6.3).

An action $a \notin S$ can, from the perspective of one component, not be blocked by the other component. So it is blocked by the environment of that component iff it is blocked by the environment \mathcal{E} of the parallel composition. Hence $a \in \sigma_i^L \Leftrightarrow a \in \sigma_i \Leftrightarrow a \in \sigma_i^R$. This gives the first formula of (6.4).

An action $a \in S$ requires cooperation of all three parties, P , Q and \mathcal{E} , so it can be blocked by any of them. If such an action is in σ_i^L (resp. σ_i^R), it is not blocked by the

environment of P (resp. Q), and thus certainly not by \mathcal{E} . This gives direction \subseteq of the second formula of (6.4).

To justify the other direction, note that each set X occurring in a partial failure trace of a process R corresponds with path $R_0 \xrightarrow{\alpha_1} R_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} R_n$ (its realm), where R_0 is reachable from R , such that (i) all α_j are either τ or t , and (ii) for each $j \in \{0, \dots, n-1\}$ such that $\alpha_{j+1} = t$, and also for $j = n$, one has $R_j \xrightarrow{\beta} \not\rightarrow$ for all $\beta \in X \cup \{\tau\}$. In the special case that $R = P \parallel_S Q$, and where $X (= \sigma_i)$ is decomposed into $X_L (= \sigma_i^L)$ and $X_R (= \sigma_i^R)$, each such R_j has the form $P_j \parallel_S Q_j$, with $P_j \xrightarrow{\beta} \not\rightarrow$ for $\beta \in X_L \cup \{\tau\}$ and $Q_j \xrightarrow{\beta} \not\rightarrow$ for $\beta \in X_R \cup \{\tau\}$. Since, for $a \in S$, $P_j \parallel_S Q_j \xrightarrow{a} \not\rightarrow$ iff either $P_j \xrightarrow{a} \not\rightarrow$ or $Q_j \xrightarrow{a} \not\rightarrow$, one can always choose the decomposition of X into X_L and X_R in such a way that $a \in X$ implies $a \in X_L \vee a \in X_R$. This gives direction \supseteq of the second formula of (6.4).

There is one circumstance, however, where the above argument breaks down, namely if different choices of $j \in \{0, \dots, n\}$ give rise to a different decomposition of X into X_L and X_R . However, in such a case one can simply write the occurrence of X as $XX \dots X$, such that now each occurrence of X has a unique decomposition. The original failure trace is then recovered by the closure operator col in (6.1).

The following example shows the necessity of Condition (6.2).

Example 6.4. Consider the process $a \parallel_{\emptyset} t.b$. Here $\{b\}a\top \in FT^*(a)$ and $\{b\}b\top \in FT^*(t.b)$. So without Condition (6.2) one would obtain $\{b\}ab\top \in FT^*(a \parallel_{\emptyset} t.b)$. Yet $a \parallel_{\emptyset} t.b$ has no such partial failure trace.

Two similar examples, without and with synchronisation, show the necessity of Condition (6.3).

Example 6.5. Consider the process $t.a \parallel_{\emptyset} t.b$. Here $\{a,b\}a\top \in FT^*(t.a)$ and $\{a,b\}b\top \in FT^*(t.b)$. So without Condition (6.3) one would obtain $\{a,b\}ab\top \in FT^*(t.a \parallel_{\emptyset} t.b)$. Yet $t.a \parallel_{\emptyset} t.b$ has no such partial failure trace.

Example 6.6. Consider the process $t.(\tau + b) \parallel_{\{b\}} t.(\tau + b)$. Here $\{b\}b\top \in FT^*(t.(\tau + b))$. So without Condition (6.3) one would obtain $\{b\}b\top \in FT^*(t.(\tau + b) \parallel_{\{b\}} t.(\tau + b))$. Yet, as explained at (3.2), $t.(\tau + b) \parallel_{\{b\}} t.(\tau + b)$ has no such partial failure trace.

The next example shows that the operation col in (6.1) cannot be omitted.

Example 6.7. One has $\sigma := \{b\}\{b\}a\top \in FT^*(b + t.a) \parallel_{\{a,b\}} FT^*(t.(b + t.a))$, taking $\sigma_L := \emptyset\{b\}a\top$ and $\sigma_R := \{b\}\emptyset a\top$. Namely $\top \in FT^*(0)$, $a\top \in FT^*(a)$, $\{b\}a\top \in FT^*(a)$, $\emptyset\{b\}a\top \in FT^*(a)$, so $\emptyset\{b\}a\top \in FT^*(b + t.a)$, and $\emptyset a\top \in FT^*(a)$, $\emptyset a\top \in FT^*(b + t.a)$, $\emptyset a\top \in FT^*(t.(b + t.a))$, so $\{b\}\emptyset a\top \in FT^*(t.(b + t.a))$. Yet $\{b\}a\top \notin FT^*(b + t.a) \parallel_{\{a,b\}} FT^*(t.(b + t.a))$.

Theorem 6.8. \equiv_{FT}^* is a congruence for the abstraction operators τ_I .

Proof. A partial failure trace ρ survives abstraction from $I \subseteq A$ iff

- (i) each set X occurring in ρ satisfies $I \subseteq X$,
- (ii) each subsequence $Xc_0 \dots c_n Y$ with $c_0, \dots, c_n \in I$ satisfies $X = Y$, and
- (iii) each subsequence $Xc_0 \dots c_n a$ with $c_0, \dots, c_n \in I$ and $a \notin I$ satisfies $a \in X$.

If this is the case, then $\tau_I(\rho)$ is the result of contracting any subsequence $Xc_0 \dots c_n X$ of ρ to X , and omitting all remaining occurrences of actions $c \in I$. If this is not the case $\tau_I(\rho)$

is undefined. Let $\sigma \cup I$ denote the partial failure trace obtained from a partial failure trace σ by replacing each occurrence of X in σ by $X \cup I$. I claim that

$$\sigma \in FT^*(\tau_I(P)) \Leftrightarrow \exists \rho \in FT^*(P). \tau_I(\rho) = \sigma \cup I. \quad (6.5)$$

This shows that $FT^*(\tau_I(P))$ is completely determined by $FT^*(P)$ and I , which yields Theorem 6.8. A formal proof of (6.5) is given in the appendix—Proposition A.2. \square

Intuitively, an occurrence of X in σ denotes a period of idling in which X is the set of actions allowed by the environment \mathcal{E} of $\tau_I(P)$. The environment \mathcal{E}_P of P when P is placed in a $\tau_I(_)$ -context always allows the actions from I , and for the rest is like \mathcal{E} . This explains (i) and the use of $\sigma \cup I$ in (6.5). Conditions (ii) and (iii) express that the instantaneous occurrence of a sequence of internal actions does not constitute an opportune moment for the environment to change its mind on which actions are allowed. Since $c_0 \in X$ (by (i)), the period X of idling ends through a time-out action of the system, right before the occurrence of c_0 . Whereas in P the action c_0 could synchronise with the environment, and thus cause a change in the set of allowed actions, this option is no longer available for $\tau_I(P)$. So after the sequence of internal actions, the same set of actions X is allowed by the environment, either resulting in further idling, or in the execution of an action $a \in X$.

An occurrence of X in σ imposes the requirement on some reachable states $\tau_I(P')$ of $\tau_I(P)$ that $\tau_I(P') \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X \cup \{\tau\}$. It is satisfied iff $P' \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X \cup I \cup \{\tau\}$. The particular way of hiding actions $c \in I$ in $\tau_I(\rho)$, together with Conditions (ii) and (iii), exactly matches the two rules for bridging a time-out transition in Table 3.

Theorem 6.9. \equiv_{FT}^* is a congruence for the relational renaming operators \mathcal{R} .

Proof. For $X \subseteq A$, let $\mathcal{R}^{-1}(X) := \{a \in A \mid \exists b \in X. (a, b) \in \mathcal{R}\}$. Furthermore, let $\mathcal{R}^{-1}(\sigma)$ denote the set of partial failure traces obtained from σ by (i) replacing each occurrence of a set X by the set $\mathcal{R}^{-1}(X)$, and (ii) replacing each occurrence of an action b by some action a with $(a, b) \in \mathcal{R}$, subject to the condition that if the occurrence of b is preceded by an occurrence of a set X , then $b \in X$ iff $a \in \mathcal{R}^{-1}(X)$. Now, as shown by Proposition A.3 in the appendix,

$$\sigma \in FT^*(\mathcal{R}(P)) \Leftrightarrow \exists \rho \in FT^*(P). \rho \in \mathcal{R}^{-1}(\sigma). \quad (6.6)$$

The side condition ensures that for each period of idling, the party that ends it (system or environment) does not differ between σ and ρ . This shows that $FT^*(\mathcal{R}(P))$ is completely determined by \mathcal{R} and P , which yields Theorem 6.9. \square

The following example shows that the side condition in the definition of $\mathcal{R}^{-1}(\sigma)$ cannot be omitted.

Example 6.10. Let $P = t.a$ and $\mathcal{R} = \{(a, b), (a, c)\}$. Then $\rho = \{a\}a\top \in FT^*(P)$, so without the side condition one would obtain $\sigma = \{b\}c\top \in FT^*(\mathcal{R}(P))$. However, $\mathcal{R}(P)$ has no such failure trace, for if the environment allows just b when the time-out occurs, $\mathcal{R}(P)$ will proceed with b rather than c .

Rooted partial failure trace semantics. Partial failure trace equivalence, however, like all default equivalences that abstract from internal activity [Mil90], fails to be a congruence for the choice operator $+$. In particular, $FT^*(b) = FT^*(\tau.b)$, yet $FT^*(a+b) \neq FT^*(a+\tau.b)$, for only $a+\tau.b$ has a partial failure trace $\{a\}\top$, and only $a+b$ has a partial failure trace $\emptyset a\top$. The classical way to solve this problem for languages like CCSP is to add one bit

of information to the semantics of processes. Two CCSP processes could be called *partial failure trace congruent* iff $FT^*(P) = FT^*(Q) \wedge \text{stable}(P) \Leftrightarrow \text{stable}(Q)$. This turns out to be a congruence for CCSP, and moreover the coarsest congruence finer than partial failure trace equivalence. In the presence of time-outs this one-bit modification of partial failure trace equivalence is insufficient, for $FT^*(t.b) = FT^*(t.t.b)$, yet $FT^*(a+t.b) \neq FT^*(a+t.t.b)$. Namely, only $a + t.t.b$ has a partial failure trace $\{b\}\{a, b\}b$.

To make partial failure trace equivalence a congruence, one needs additionally to keep track of partial failure traces of the form $X\sigma$ that are lifted by the fifth rule of Table 3 through an initial t -transition. I also use an additional bit, telling that a stable state is reachable along a nonempty path of τ -transitions.

Definition 6.11. Let $FT^{r*}(P) :=$

$$FT^*(P) \cup \{\text{STAB} \mid P \xrightarrow{\tau} \} \cup \{tX\sigma \mid \exists P'. P \xrightarrow{t} P' \wedge X\sigma \in FT^*(P') \wedge P \xrightarrow{\tau} \wedge \mathcal{I}(P) \cap X = \emptyset\} \\ \cup \{\text{POSTSTAB} \mid P \xrightarrow{\tau} \wedge \emptyset \top \in FT^*(P)\}.$$

Processes $P, Q \in \mathbb{P}$ are *rooted partial failure trace equivalent*, $P \equiv_{FT}^{r*} Q$, iff $FT^{r*}(P) = FT^{r*}(Q)$.

One has $t.b \not\equiv_{FT}^{r*} t.t.b$ because $t\{a, b\}b \in FT^{r*}(t.t.b)$ but $t\{a, b\}b \notin FT^{r*}(t.b)$.

For the purpose of defining \equiv_{FT}^{r*} , the POSTSTAB bit is redundant, as it is clearly determined by the other elements of $FT^{r*}(P)$. However, it will turn out to be useful in defining a rooted partial failure trace preorder \sqsubseteq_{FT}^{r*} in Section 7.

Observation 6.12. Rooted partial failure trace equivalence is finer than partial failure trace equivalence.

Let $\text{ini}_Z(F)$, for $Z \subseteq A$ and $F \subseteq (A \cup \mathcal{P}(A))^* \top$, be the least set such that (a) $F \subseteq \text{ini}_Z(F)$, (b) $\top \in \text{ini}_Z(F)$, and (c) if $\sigma \in \text{ini}_Z(F)$ and $X \cap Z = \emptyset$ then $X\sigma \in \text{ini}_Z(F)$. It represents the set of partial failure traces of a stable process P derivable by the first and fourth rule of Table 3 when $F \subseteq FT^*(P)$ and $\mathcal{I}(P) = Z$.

Theorem 6.13. Rooted partial failure trace equivalence (\equiv_{FT}^{r*}) is a congruence for the operators of CCSP_t .

Proof. It suffices to show that $FT^{r*}(P + Q)$ is fully determined by $FT^{r*}(P)$ and $FT^{r*}(Q)$, that $FT^{r*}(\alpha.P)$ is fully determined by α and $FT^{r*}(P)$, and similarly for $FT^{r*}(P \parallel_S Q)$, $FT^{r*}(\tau_I(P))$ and $FT^{r*}(\mathcal{R}(P))$. The following equations establish this.

$$FT^{r*}(a.P) = \text{ini}_{\{a\}}(\{a\sigma \mid \sigma \in FT^*(P)\}) \cup \{\text{STAB}\} \quad (6.7)$$

$$FT^{r*}(\tau.P) = FT^*(P) \cup \{\text{POSTSTAB} \mid \emptyset \top \in FT^*(P)\} \quad (6.8)$$

$$FT^{r*}(t.P) = \text{ini}_{\emptyset}(\{X\sigma \mid X\sigma \in FT^*(P)\} \cup \{Xa\sigma \mid a\sigma \in FT^*(P) \wedge a \in X\}) \\ \cup \{\text{STAB}\} \cup \{tX\sigma \mid X\sigma \in FT^*(P)\} \quad (6.9)$$

$$FT^{r*}(P + Q) = \begin{cases} FT^{r*}(P) \cup FT^{r*}(Q) & \text{if } \neg \text{stable}(P) \wedge \neg \text{stable}(Q) \\ \{a\sigma \mid a\sigma \in FT^{r*}(P)\} \cup FT^{r*}(Q) & \text{if } \text{stable}(P) \wedge \neg \text{stable}(Q) \\ FT^{r*}(P) \cup \{a\sigma \mid a\sigma \in FT^{r*}(Q)\} & \text{if } \neg \text{stable}(P) \wedge \text{stable}(Q) \\ \text{ini}_{\mathcal{I}(P) \cup \mathcal{I}(Q)}(H) \cup \{\text{STAB}\} \cup K & \text{if } \text{stable}(P) \wedge \text{stable}(Q) \end{cases} \quad (6.10)$$

$$FT^{r*}(P \parallel_S Q) = \text{col}(FT^{r*}(P) \parallel_S FT^{r*}(Q)) \quad (6.11)$$

$$\begin{aligned}
FT^{r*}(\tau_I(P)) = & \{\sigma \mid (\text{STAB} \neq \sigma \neq \text{POSTSTAB}) \wedge \exists \rho \in FT^{r*}(P). \tau_I(\rho) = \sigma \cup I\} \\
& \cup \{\text{STAB} \mid \text{STAB} \in FT^{r*}(P) \wedge \mathcal{I}(P) \cap I = \emptyset\} \\
& \cup \{\text{POSTSTAB} \mid (\exists c_0 c_1 \dots c_n I\top \in FT^{r*}(P) \text{ where } n \geq 0 \text{ and all } c_i \in I) \\
& \quad \vee (\text{POSTSTAB} \in FT^{r*}(P) \wedge I\top \in FT^{r*}(P))\}
\end{aligned} \tag{6.12}$$

$$FT^{r*}(\mathcal{R}(P)) = \{\sigma \mid \exists \rho \in FT^{r*}(P). \rho \in \mathcal{R}^{-1}(\sigma)\} \tag{6.13}$$

Here $H := \{a\sigma \mid a\sigma \in FT^{r*}(P) \cup FT^{r*}(Q)\} \cup$
 $\{X\sigma \mid tX\sigma \in FT^{r*}(P) \cup FT^{r*}(Q) \wedge (\mathcal{I}(P) \cup \mathcal{I}(Q)) \cap X = \emptyset\} \cup$
 $\{Xa\sigma \mid Xa\sigma \in FT^{r*}(P) \cup FT^{r*}(Q) \wedge a \in X \wedge (\mathcal{I}(P) \cup \mathcal{I}(Q)) \cap X = \emptyset\}$
and $K = \{tX\sigma \mid tX\sigma \in FT^{r*}(P) \cup FT^{r*}(Q) \wedge (\mathcal{I}(P) \cup \mathcal{I}(Q)) \cap X = \emptyset\}$.

The first four equations follow immediately from Definition 6.11 and Table 3. In (6.9) the argument of ini_{\emptyset} generates the partial failure traces contributed by the fifth and sixth rule of Table 3; the operation ini_{\emptyset} then closes this set under applications of the first and fourth rule. In the fourth case of (6.10), the three lines of H generate the partial failure traces contributed by the second, fifth and sixth rule of Table 3, respectively; the operation $ini_{\mathcal{I}(P) \cup \mathcal{I}(Q)}$ then closes this set under applications of the first and fourth rule.

To see that (6.7)–(6.10) fulfil their intended purpose, note that $FT^*(P)$ is fully determined by $FT^{r*}(P)$, namely by dropping the bits STAB and POSTSTAB , and all traces $t\sigma$. Moreover $stable(P)$ holds iff $\text{STAB} \in FT^{r*}(P)$ and for stable P the set $\mathcal{I}(P)$ can be computed by $\mathcal{I}(P) = \{a \in A \mid \exists \sigma. a\sigma \in FT^{r*}(P)\}$.

To explain (6.11), the definition of decomposition needs to be extended to rooted partial failure traces $tX\sigma$. In the first bullet-point of Definition 6.3, “ $\sigma_i \in A \setminus S$ ” should now be read as “ $\sigma_i \in (A \setminus S) \cup \{t\}$ ”. The definition of $F \parallel_S G$ is unchanged, except that rooted partial failure traces $tX\sigma$ are dropped from $F \parallel_S G$ unless $\text{STAB} \in F$ and $\text{STAB} \in G$; furthermore STAB is deemed to be in $F \parallel_S G$ iff it is in both F and G , and POSTSTAB is deemed to be in $F \parallel_S G$ if either $\text{POSTSTAB} \in F \wedge \text{POSTSTAB} \in G$, or $\text{POSTSTAB} \in F \wedge \text{STAB} \in G$, or $\text{STAB} \in F \wedge \text{POSTSTAB} \in G$. Moreover, the operator col extends to sets of rooted partial failure traces. In the appendix it is shown that with these definitions (6.11) holds—see Proposition A.4.

In (6.12), the concept “survives abstraction from $I \subseteq A$ ” from the proof of Theorem 6.8 is applied verbatim to rooted partial failure traces $tX\sigma$, as is the operation τ_I . The validity of (6.12) is shown in the appendix—Proposition A.5.

In (6.13), the operator \mathcal{R}^{-1} extends verbatim to rooted partial failure traces. The validity of (6.12) is shown in the appendix—Proposition A.6. \square

As remarked in [Gla94], the countable choice operator $\sum_{i \in \mathbb{N}} P_i$ is expressible in terms of the binary choice operator $+$ and recursion: $\sum_{i \in \mathbb{N}} P_i = \langle X_0 \mid \mathcal{S} \rangle$, where $X_i = P_i + X_{i+1}$ for $i \in \mathbb{N}$. An equation similar to (6.10) shows that \equiv_{FT}^{r*} also is a congruence for the countable choice.

Theorem 6.14. Rooted partial failure trace equivalence (\equiv_{FT}^{r*}) is the largest congruence for the operators of CCSP_t that is included in \equiv_{FT}^* .

Proof. In view of Observation 6.12 and Theorem 6.13 it suffices to find, for any two rooted partial failure trace inequivalent processes $P \not\equiv_{FT}^{r*} Q$, a CCSP_t -context \mathcal{C} such that $\mathcal{C}(P) \not\equiv_{FT}^* \mathcal{C}(Q)$.

First of all, one can apply a bijective renaming operator on P and Q , mapping A to $A \setminus \{f\}$ for some action $f \in A$. Consequently, I may assume that f is *fresh* in the sense that

for each process R reachable from P or Q one has $R \not\stackrel{f}{\rightarrow}$. For any such R one has

$$\sigma X\rho \in FT^*(R) \Leftrightarrow \sigma(X \cup \{f\})\rho \in FT^*(R). \quad (*)$$

Let $\sigma \in FT^{r*}(Q) \setminus FT^{r*}(P)$. (The other case proceeds by symmetry.) The only relevant cases are that $\sigma = (\text{POST})_{\text{STAB}}$ or $\sigma = \text{t}X\rho$, since in all other cases one immediately has $\sigma \in FT^*(Q) \setminus FT^*(P)$.

Let $\sigma = \text{STAB}$, that is, Q is stable but P is not. Then $\emptyset f \top \in FT^*(Q + f)$ but $\emptyset f \top \notin FT^*(P + f)$. So it suffices to take the context $\mathcal{C}(_) = _ + f$.

Now let $\sigma = \text{t}X\rho$. So $Q \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X \cup \{\tau\}$, $Q \xrightarrow{\text{t}} Q'$ and $X\rho \in FT^*(Q')$. I may assume that P is stable, as the case that Q is stable but P is not has been treated already. By Observation 5.3 one has $XX\rho \in FT^*(Q')$, and (*) yields $X^-X^+\rho \in FT^*(Q')$, where $X^- := X \setminus \{f\}$ and $X^+ := X \cup \{f\}$. The fifth rule of Table 3 gives $X^-X^+\rho \in FT^*(Q+f)$, using that $Q+f \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X^- \cup \{\tau\}$. It suffices to show that $X^-X^+\rho \notin FT^*(P+f)$, for then the context $\mathcal{C}(_) = _ + f$ finishes the proof.

So assume, towards a contradiction, that $X^-X^+\rho \in FT^*(P+f)$. This could not have been derived by the fourth rule of Table 3, since surely $X^+\rho \notin FT^*(P+f)$. Hence the fifth rule must have been used, so that $P \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X \cup \{\tau\}$, $P \xrightarrow{\text{t}} P'$ and $X^-X^+\rho \in FT^*(P')$. Now $XX\rho \in FT^*(P')$ by (*), and $X\rho \in FT^*(P')$ by Observation 5.3. Thus $\text{t}X\rho \in FT^{r*}(P)$, yielding the required contradiction.

Finally, let $\sigma = \text{POSTSTAB}$. Then $\{f\}\top \in FT^*(Q + f)$, yet $\{f\}\top \notin FT^*(P + f)$. \square

In the quest for a congruence, an alternative solution to changing partial failure trace equivalence into rooted partial failure trace equivalence, is to restrict the expressiveness of CCSP_t . Let CCSP_t^g be the version of CCSP_t where the binary operator $+$ is replaced by guarded sums $\sum_{i=1}^n \alpha_i.P_i$, for $n \in \mathbb{N} \cup \{\infty\}$. Here each guarded sum $\sum_{i=1}^n \alpha_i._$ counts as a separate n -ary operator. The constant 0 and action prefixing $\alpha.P$ are the special cases with $n = 0$ and $n = 1$. The congruence property for guarded sums demands that $P_i \equiv_{FT}^* Q_i$ for $i = 1, \dots, n$ implies $\sum_{i=1}^n \alpha_i.P_i = \sum_{i=1}^n \alpha_i.Q_i$. That this holds is an immediately consequence of Theorem 7.2 below.

7. PARTIAL FAILURE TRACE PREORDERS

Here I introduce (rooted) partial failure trace preorders \sqsubseteq_{FT}^* and \sqsubseteq_{FT}^{r*} in such a way that (rooted) partial failure trace equivalence is its kernel, i.e., $P \equiv_{FT}^* Q \Leftrightarrow (P \sqsubseteq_{FT}^* Q \wedge Q \sqsubseteq_{FT}^* P)$ and, likewise, $P \equiv_{FT}^{r*} Q \Leftrightarrow (P \sqsubseteq_{FT}^{r*} Q \wedge Q \sqsubseteq_{FT}^{r*} P)$. These preorders should be defined in such a way that they are precongruences.

Definition 7.1. Write $P \sqsubseteq_{FT}^* Q$, for processes $P, Q \in \mathbb{P}$, iff $FT^*(P) \supseteq FT^*(Q)$.

The orientation of the symbol \sqsubseteq_{FT}^* aligns with that of the safety preorder, discussed in the next section. The following proposition says that the preorder \sqsubseteq_{FT}^* is a *precongruence* for the operators of CCSP_t^g , or that recursion-free CCSP_t^g -contexts are *monotone* w.r.t. \sqsubseteq_{FT}^* .

Theorem 7.2. Let \mathcal{C} be a unary recursion-free CCSP_t^g -context.

If $P \sqsubseteq_{FT}^* Q$ then $\mathcal{C}(P) \sqsubseteq_{FT}^* \mathcal{C}(Q)$.

Proof. Let $G := \sum_{i \in I} \alpha_i.P_i$. In case $\alpha_i \neq \tau$ for all $i \in I$, then (6.7)–(6.10) imply (or generalise to)

$$FT^*(G) = ini_{I(G)} \left(\bigcup_{\substack{i \in I \\ \alpha_i \in A}} \{\alpha_i \sigma \mid \sigma \in FT^*(P_i)\} \cup \bigcup_{\substack{i \in I \\ \alpha_i = t}} \left(\left\{ \begin{array}{l} X\sigma \mid X\sigma \in FT^*(P_i) \wedge \mathcal{I}(G) \cap X = \emptyset \\ Xa\sigma \mid a\sigma \in FT^*(P_i) \wedge \mathcal{I}(G) \cap X = \emptyset \\ \wedge a \in X \end{array} \right\} \cup \right) \right).$$

Here $\mathcal{I}(G) := \{\alpha_i \mid i \in I\} \setminus \{t\}$. Moreover, if $\alpha_i = \tau$ for at least one $i \in I$, then

$$FT^*(G) = \bigcup_{\substack{i \in I \\ \alpha_i \in A}} \{\alpha_i \sigma \mid \sigma \in FT^*(P_i)\} \cup \bigcup_{\substack{i \in I \\ \alpha_i = \tau}} FT^*(P_i).$$

These equations immediately imply the monotonicity of the guarded choice operators w.r.t. \sqsubseteq_{FT}^* . The monotonicity of $\|_S$, τ_I and \mathcal{R} follows from Properties (6.1), (6.5) and (6.6). \square

Definition 7.3. Write $P \sqsubseteq_{FT}^{r*} Q$, for $P, Q \in \mathbb{P}$, iff $FT^{r*}(P) \supseteq FT^{r*}(Q)$.

In this definition, the `POSTSTAB` bit of Definition 6.11 plays a crucial rôle. Without it, we would have $b \sqsubseteq_{FT}^{r*} \tau.b$, since $FT^{r*}(b) = FT^{r*}(\tau.b) \uplus \{\text{STAB}\}$. This would cause a failure of monotonicity, as $a + b \not\sqsubseteq_{FT}^{r*} a + \tau.b$, for only the latter process has a partial failure trace $\{a\}\top$. With the `POSTSTAB` bit one obtains $b \not\sqsubseteq_{FT}^{r*} \tau.b$, since `POSTSTAB` $\in FT^{r*}(\tau.b)$, yet `POSTSTAB` $\notin FT^{r*}(b)$. The preorder \sqsubseteq_{FT}^{r*} still relates processes of which only one is stable; for instance $b \sqsubseteq_{FT}^{r*} \langle X \mid X = b + \tau.X \rangle$.

Lemma 7.4. If $P \sqsubseteq_{FT}^{r*} Q$ (or $P \sqsubseteq_{FT}^* Q$) and P and Q are both stable, then $\mathcal{I}(P) = \mathcal{I}(Q)$.

Proof. If $a \in \mathcal{I}(Q)$ then $a\top \in FT^{r*}(Q) \subseteq FT^{r*}(P)$, so $a \in \mathcal{I}(P)$.

Moreover, if $a \notin \mathcal{I}(Q)$ then $\{a\}\top \in FT^{r*}(Q) \subseteq FT^{r*}(P)$, so $a \notin \mathcal{I}(P)$. \square

Theorem 7.5. Let \mathcal{C} be a unary recursion-free CCSP_t-context.

If $P \sqsubseteq_{FT}^{r*} Q$ then $\mathcal{C}(P) \sqsubseteq_{FT}^{r*} \mathcal{C}(Q)$.

Proof. The monotonicity of $\|_S$, τ_I and \mathcal{R} follows from Properties (6.11)–(6.13), using Lemma 7.4. The monotonicity of α_- follows from (6.7)–(6.9), using that $FT^{r*}(P) \supseteq FT^{r*}(Q)$ implies $FT^*(P) \supseteq FT^*(Q)$.

The monotonicity of $+$ can be formulated as $P \sqsubseteq_{FT}^{r*} Q \Rightarrow P + R \sqsubseteq_{FT}^{r*} Q + R$ (the requirement $R + P \sqsubseteq_{FT}^{r*} R + Q$ then follows by symmetry). So let $P \sqsubseteq_{FT}^{r*} Q$, i.e., $FT^{r*}(P) \supseteq FT^{r*}(Q)$.

- Let $\neg\text{stable}(P)$ and $\neg\text{stable}(R)$. Then `STAB` $\notin FT^{r*}(P) \supseteq FT^{r*}(Q)$, so $\neg\text{stable}(Q)$. Hence $FT^{r*}(P + R) = FT^{r*}(P) \cup FT^{r*}(R) \supseteq FT^{r*}(Q) \cup FT^{r*}(R) = FT^{r*}(Q + R)$.
- The case $\neg\text{stable}(P)$ and $\text{stable}(R)$ proceeds likewise.

In all remaining cases I assume $\text{stable}(P)$. Note that `POSTSTAB` $\notin FT^{r*}(P) \supseteq FT^{r*}(Q)$.

- Let $\text{stable}(Q)$ and $\neg\text{stable}(R)$. Then $FT^{r*}(P + R) = \{a\sigma \mid a\sigma \in FT^{r*}(P)\} \cup FT^{r*}(R) \supseteq \{a\sigma \mid a\sigma \in FT^{r*}(Q)\} \cup FT^{r*}(R) = FT^{r*}(Q + R)$.
- Let $\neg\text{stable}(Q)$ and $\neg\text{stable}(R)$. Now all $\sigma \in FT^{r*}(Q)$ must have the form \top or $a\zeta$. Namely if $X\zeta \in FT^{r*}(Q)$ then one would have `POSTSTAB` $\in FT^{r*}(Q)$. Hence $FT^{r*}(P + R) = \{a\sigma \mid a\sigma \in FT^{r*}(P)\} \cup FT^{r*}(R) \supseteq FT^{r*}(Q) \cup FT^{r*}(R) = FT^{r*}(Q + R)$.
- Let $\text{stable}(Q)$ and $\text{stable}(R)$. Then $\mathcal{I}(P) = \mathcal{I}(Q)$ by Lemma 7.4. Using this, the desired monotonicity property follows from the fourth case of (6.10).
- Let $\neg\text{stable}(Q)$ and $\text{stable}(R)$. Again all $\sigma \in FT^{r*}(Q)$ must have the form \top or $a\zeta$. Let H and K be as in (6.10), but with R substituted for Q . Now

$$\begin{aligned} FT^{r*}(P + R) &= ini_{\mathcal{I}(P) \cup \mathcal{I}(Q)}(H) \cup \{\text{STAB}\} \cup K \supseteq H \cup \{\top\} \\ &\supseteq FT^{r*}(Q) \cup \{a\sigma \mid a\sigma \in FT^{r*}(R)\} = FT^{r*}(Q + R). \end{aligned} \quad \square$$

Theorem 7.6. \sqsubseteq_{FT}^{r*} is the largest precongruence for the operators of CCSP_t that is included in \sqsubseteq_{FT}^* .

Proof. Theorem 7.5 shows that \sqsubseteq_{FT}^{r*} is a precongruence for the operators of CCSP_t . By Definition 6.11, \sqsubseteq_{FT}^{r*} is included in \sqsubseteq_{FT}^* , i.e., $P \sqsubseteq_{FT}^{r*} Q$ implies $P \sqsubseteq_{FT}^* Q$. Thus it suffices to find, for any two processes P and Q with $P \not\sqsubseteq_{FT}^{r*} Q$, a CCSP_t -context \mathcal{C} such that $\mathcal{C}(P) \not\sqsubseteq_{FT}^* \mathcal{C}(Q)$. This proceeds exactly as in the proof of Theorem 6.14. Of course the symmetric case “ $\sigma \in FT^{r*}(P) \setminus FT^{r*}(Q)$ ” is skipped, as it is not needed here. \square

8. THE COARSEST PRECONGRUENCE RESPECTING SAFETY PROPERTIES

In [Gla10] I proposed a way to define refinement preorders \sqsubseteq on processes, where $P \sqsubseteq Q$ says that for all practical purposes under consideration, Q is at least as suitable as P , i.e., it will never harm to replace P by Q . In the stepwise design of systems, P may be closer to a specification, and Q to an implementation. Each such a refinement preorder also yields a semantic equivalence \equiv , with $P \equiv Q$ saying that for practical purposes P and Q are equally suitable; i.e., one can be replaced by the other without untoward side effects. Naturally, $P \equiv Q$ iff both $P \sqsubseteq Q$ and $Q \sqsubseteq P$.

The method of [Gla10] equips the choice of \sqsubseteq with two parameters: a class \mathcal{G} of good properties of processes, ones that may be required of processes in a given range of applications, and a class \mathcal{O} of useful operators for combining processes. The first requirement on \sqsubseteq is $P \sqsubseteq Q \Rightarrow \forall \varphi \in \mathcal{G}. (P \models \varphi \Rightarrow Q \models \varphi)$ where $P \models \varphi$ denotes that process P has the property φ . If this holds, \sqsubseteq *respects* or *preserves* the properties in \mathcal{G} . The second requirement is that \sqsubseteq is a *precongruence* for \mathcal{O} : $P \sqsubseteq Q \Rightarrow \mathcal{C}[P] \sqsubseteq \mathcal{C}[Q]$ for any context $\mathcal{C}[-]$ built from operators from \mathcal{O} . Given the choice of \mathcal{G} and \mathcal{O} , the preorder recommended by [Gla10] is the *coarsest*, or largest, one satisfying both requirements. This preorder is called *fully abstract* w.r.t. \mathcal{G} and \mathcal{O} , and is characterised by

$$P \sqsubseteq Q \Leftrightarrow \forall \mathcal{O}\text{-context } \mathcal{C}[-]. \forall \varphi \in \mathcal{G}. (\mathcal{C}[P] \models \varphi \Rightarrow \mathcal{C}[Q] \models \varphi).$$

The corresponding semantic equivalence identifies processes only when this is enforced by the two requirements above.

Naturally, increasing the class \mathcal{O} of operators makes the resulting fully abstract preorder *finer*, or smaller. The same holds when increasing the class \mathcal{G} of properties. In [Gla10] I employed a class \mathcal{O} of operators that was effectively equivalent to the operators of CCSP^g . Here I use the operators of CCSP_t^g , thus adding time-outs. In this section, following [Gla10, Section 3], I take as class \mathcal{G} of good properties the *safety properties*, saying that something bad will never happen. What exactly counts as a safety property could be open to some debate. However, the *canonical safety property*, proposed in [Gla10], is definitely in this class. I first characterise the preorder $\sqsubseteq_{\text{safety}}$ that is fully abstract w.r.t. the operators of CCSP_t^g and this single safety property. It turns out to be \sqsubseteq_{FT}^* . Then I argue that adding any other safety properties to the class \mathcal{G} could not possible make the resulting preorder any finer. An immediately corollary of this and Theorem 7.6 is that \sqsubseteq_{FT}^{r*} is fully abstract for the operators of CCSP_t and the class of safety properties.

Full abstraction w.r.t. the canonical safety property. To formulate the canonical safety property, assume that the alphabet A of visible actions contains one specific action b , whose occurrence is *bad*. The property now says that b **will never happen**.

Definition 8.1. A process P satisfies the *canonical safety property*, notation $P \models \text{safety}(b)$, if no partial failure trace of P contains the action b .

Lemma 8.2. If $a\eta \in FT^*(P)$ then $P \xrightarrow{a}$ or $P \xrightarrow{\tau}$.

Proof. That $a\eta \in FT^*(P)$ can be derived only from the second or third rule of Table 3. \square

Corollary 8.3. If $Xa\eta \in FT^*(P)$ is obtained from the fourth rule of Table 3, then $a \notin X$.

Proof. The fourth rule requires that $P \not\xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$, and that $a\eta \in FT^*(P)$. The latter implies $P \xrightarrow{a}$ or $P \xrightarrow{\tau}$ by Lemma 8.2. So $P \xrightarrow{a}$ and $a \notin X$. \square

As encountered already in the proof of Theorem 6.14, I call an action b *fresh* for a process P if for each process R reachable from P one has $R \not\xrightarrow{b}$. In that case $P \models \text{safety}(b)$. I call it *fresh* for a partial failure trace $\sigma \in (A \cup \mathcal{P}(A))^* \top$ if b does not occur in σ , either as action or inside a refusal set in σ .

Theorem 8.4. For each $\sigma \in (A \cup \mathcal{P}(A))^* \top$ for which b is fresh, there exists a CCSP_t^g process T_σ such that

$$\tau_{A \setminus \{b\}}(T_\sigma \parallel_{A \setminus \{b\}} P) \not\models \text{safety}(b) \Leftrightarrow \sigma \in FT^*(P)$$

for each CCSP_t^g process P for which b is fresh.

Proof. Let $B := A \setminus \{b\}$, the set of all visible actions except b . Define T_σ with structural induction on σ :

$$T_\top := t.b \quad T_{c\rho} := \tau + c.T_\rho \quad T_{X\eta} := t.T_\eta + \sum_{a \in X} a \quad T_{Xd\rho} := t.(d.T_\rho + \sum_{a \in X \setminus \{d\}} a) + \sum_{a \in X} a$$

where, $c, d \in B$, $X \subseteq B$, $d \in X$, and η is not of the form $d\rho$ with $d \in X$. Note that $\tau_B(T_\sigma \parallel_B P) \not\models \text{safety}(b)$ iff there is a path $\pi := \tau_B(T_\sigma \parallel_B P) \xrightarrow{\alpha_1} Q_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} Q_n \xrightarrow{b} Q'$, and this path gives rise to a partial failure trace $\beta b \top \in FT^*(\tau_B(T_\sigma \parallel_B P))$ by application of the rules of Table 3. Due to the operator τ_B , each of the α_i must be τ or t .

To obtain “ \Rightarrow ” I assume $\tau_B(T_\sigma \parallel_B P) \not\models \text{safety}(b)$ and apply structural induction on σ .

- Let $\sigma = \top$. Then $\sigma \in FT^*(P)$ for all P .
- Let $\sigma = c\rho$. For the process $\tau_B(T_\sigma \parallel_B P)$ to ever reach a b -transition, the component $T_\sigma = \tau + c.T_\rho$ must take the c -transition to T_ρ . So the beginning of π must have the form

$$\tau_B(T_\sigma \parallel_B P) \xrightarrow{\alpha_1} \tau_B(T_\sigma \parallel_B P_1) \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \tau_B(T_\sigma \parallel_B P_n) \xrightarrow{\tau} \tau_B(T_\rho \parallel_B P')$$

for some $n \geq 0$, where $\tau_B(T_\rho \parallel_B P') \not\models \text{safety}(b)$. Let $P_0 := P$. All of these α_i must be τ , for t -transitions lose out from the outgoing τ -transition from $\tau_B(T_\sigma \parallel_B P_{i-1})$. Technically, if $\alpha_i = t$, for $1 \leq i \leq n$, then the fifth and sixth rules of Table 3 do not allow any partial failure trace of $\tau_B(T_\rho \parallel_B P_i)$ to give rise to a partial failure trace of $\tau_B(T_\sigma \parallel_B P_{i-1})$. The above path must stem from a path

$$T_\sigma \parallel_B P \xrightarrow{\beta_1} T_\sigma \parallel_B P_1 \xrightarrow{\beta_2} \dots \xrightarrow{\beta_n} T_\sigma \parallel_B P_n \xrightarrow{c} T_\rho \parallel_B P'$$

that in turns stems from a path $P \xrightarrow{\beta_1} P_1 \xrightarrow{\beta_2} \dots \xrightarrow{\beta_n} P_n \xrightarrow{c} P'$. All of these β_i must be τ , as the synchronisation on B stops all visible actions stemming from P .

By induction $\rho \in FT^*(P')$. The second and third rules of Table 3 yield $\sigma \in FT^*(P)$.

- Let $\sigma = X\eta$, where $X \subseteq B$ and η is not of the form $d\rho$ with $d \in X$. For the process $\tau_B(T_\sigma \parallel_B P)$ to ever reach a b -transition, the component $T_\sigma = t.T_\eta + \sum_{a \in X} a$ must take the t -transition to T_η . So the beginning of π must have the form

$$\tau_B(T_\sigma \parallel_B P) \xrightarrow{\alpha_1} \tau_B(T_\sigma \parallel_B P_1) \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \tau_B(T_\sigma \parallel_B P_n) \xrightarrow{t} \tau_B(T_\eta \parallel_B P_n)$$

for some $n \geq 0$, where $\tau_B(T_\eta \parallel_B P_n) \not\Leftarrow \text{safety}(b)$. By induction $\eta \in FT^*(P_n)$. As above, again using that visible actions from P are blocked by \parallel_B , it must be that $P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} P_n$. I proceed with induction on n .

Base case: It must be that $P \xrightarrow{\alpha} \not\Leftarrow$ for all $\alpha \in X \cup \{\tau\}$, for otherwise $\tau_B(T_\sigma \parallel_B P_n)$ would have an outgoing τ -transition that wins from the t -transition to $\tau_B(T_\eta \parallel_B P_n)$. The fourth rule of Table 3 yields $\sigma \in FT^*(P)$.

Induction step: By induction $\sigma \in FT^*(P_1)$. In case $\alpha_1 = \tau$ one obtains $\sigma \in FT^*(P)$ by the third rule of Table 3. In case $\alpha_1 = t$, it must be that $P \xrightarrow{\alpha} \not\Leftarrow$ for all $\alpha \in X \cup \{\tau\}$, for otherwise $\tau_B(T_\sigma \parallel_B P)$ would have an outgoing τ -transition that wins from the t -transition to $\tau_B(T_\sigma \parallel_B P_1)$. The fifth rule of Table 3 yields $\sigma \in FT^*(P)$.

- Let $\sigma = Xd\rho$ with $d \in X$. Writing R for $d.T_\rho + \sum_{a \in X \setminus \{d\}} a$, the first part of π must have the form

$$\begin{aligned} \tau_B(T_\sigma \parallel_B P) &\xrightarrow{\alpha_1} \tau_B(T_\sigma \parallel_B P_1) \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \tau_B(T_\sigma \parallel_B P_n) \xrightarrow{t} \tau_B(R \parallel_B P_n) \\ \tau_B(R \parallel_B P_n) &\xrightarrow{\alpha_{n+1}} \tau_B(R \parallel_B P_{n+1}) \xrightarrow{\alpha_{n+2}} \dots \xrightarrow{\alpha_m} \tau_B(R \parallel_B P_m) \xrightarrow{\tau} \tau_B(T_\rho \parallel_B P') \end{aligned}$$

for some $m \geq n \geq 0$, where $\tau_B(T_\rho \parallel_B P') \not\Leftarrow \text{safety}(b)$. By induction $\rho \in FT^*(P')$. As above, it must be that $P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} P_n \xrightarrow{\alpha_{n+1}} P_{n+1} \xrightarrow{\alpha_{n+2}} \dots \xrightarrow{\alpha_m} P_m \xrightarrow{d} P'$. Necessarily, $P_n \xrightarrow{\alpha} \not\Leftarrow$ for all $\alpha \in X \cup \{\tau\}$, for otherwise $\tau_B(T_\sigma \parallel_B P_n)$ would have an outgoing τ -transition that wins from the t -transition to $\tau_B(R \parallel_B P_n)$. In particular, $P_n \xrightarrow{d} \not\Leftarrow$, whereas $P_m \xrightarrow{d}$, so $n \neq m$. Moreover, $\alpha_{n+1} = t$, for $\alpha_{n+1} = \tau$ contradicts with $P_n \xrightarrow{\tau} \not\Leftarrow$. Let $k \leq m$ be the highest index such that $\alpha_k = t$. Then $d\rho \in FT^*(P_k)$, by the second and third rules of Table 3. For each index i with $\alpha_i = t$ one must have $P_{i-1} \xrightarrow{\alpha} \not\Leftarrow$ for all $\alpha \in X \cup \{\tau\}$, for the same reason as above. The sixth rule of Table 3 yields $Xd\rho \in FT^*(P_{k-1})$. Successive applications of the third and fifth rules yield $Xd\rho \in FT^*(P_j)$ for all $j < k$, so in particular $\sigma \in FT^*(P)$.

To obtain “ \Leftarrow ”, I will show that $\sigma \in FT^*(P)$ implies $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P))$. Here A is the set of all visible actions. In doing so, I apply induction on the derivation of $\sigma \in FT^*(P)$.

Suppose $\sigma = \top \in FT^*(P)$ is derived from the first rule of Table 3. Since $T_\sigma = t.b.0$ one has $\tau_B(T_\sigma \parallel_B P) \xrightarrow{t} \tau_B(b.0 \parallel_B P) \xrightarrow{b} \tau_B(0 \parallel_B P)$. Hence $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P))$, using the first, second and sixth rules of Table 3, and using that $\tau_B(T_\sigma \parallel_B P) \xrightarrow{\alpha} \not\Leftarrow$ for all $\alpha \in A \cup \{\tau\}$.

Suppose $\sigma = c\rho \in FT^*(P)$ is derived from the second rule of Table 3. Then $P \xrightarrow{c} P'$ and $\rho \in FT^*(P')$. By induction, $Ab\top \in FT^*(\tau_B(T_\rho \parallel_B P'))$. Since $\tau_B(T_\sigma \parallel_B P) \xrightarrow{\tau} \tau_B(T_\rho \parallel_B P')$, it follows with the third rule of Table 3 that $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P))$.

Suppose $\sigma \in FT^*(P)$ is derived from the third rule of Table 3. Then $P \xrightarrow{\tau} P'$ and $\sigma \in FT^*(P')$. By induction, $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P'))$. Since $\tau_B(T_\sigma \parallel_B P) \xrightarrow{\tau} \tau_B(T_\sigma \parallel_B P')$, it follows with the third rule of Table 3 that $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P))$.

Suppose $\sigma = X\eta \in FT^*(P)$ is derived from the fourth rule of Table 3. Then $P \xrightarrow{\alpha} \not\Leftarrow$ for all $\alpha \in X \cup \{\tau\}$ and $\eta \in FT^*(P)$. In the special case that $\eta = d\rho$, by Corollary 8.3 $d \notin X$. By induction $Ab\top \in FT^*(\tau_B(T_\eta \parallel_B P))$. Since $\tau_B(T_\sigma \parallel_B P) \xrightarrow{t} \tau_B(T_\eta \parallel_B P)$, and $\tau_B(T_\sigma \parallel_B P) \xrightarrow{\alpha} \not\Leftarrow$ for all $\alpha \in A \cup \{\tau\}$, the fifth rule of Table 3 yields $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P))$.

Suppose $\sigma = X\eta \in FT^*(P)$ is derived from the fifth rule of Table 3. Then $P \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$, $P \xrightarrow{t} P'$ and $\sigma \in FT^*(P')$. So $\tau_B(T_\sigma \parallel_B P) \not\rightarrow^\alpha$ for all $\alpha \in A \cup \{\tau\}$, and $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P'))$ by induction. Since $\tau_B(T_\sigma \parallel_B P) \xrightarrow{t} \tau_B(T_\sigma \parallel_B P')$, the fifth rule yields $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P))$.

Suppose $\sigma = Xd\rho \in FT^*(P)$ is derived from the sixth rule of Table 3. Then $P \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$, $d \in X$, $P \xrightarrow{t} P'$ and $d\rho \in FT^*(P')$. So $\tau_B(T_\sigma \parallel_B P) \not\rightarrow^\alpha$ and $\tau_B(R \parallel_B P) \not\rightarrow^\alpha$ for all $\alpha \in A \cup \{\tau\}$, where $R := d.T_\rho + \sum_{a \in X \setminus \{d\}} a$. By induction $Ab\top \in FT^*(\tau_B(Td\rho \parallel_B P'))$. As $\tau_B(T_\sigma \parallel_B P) \xrightarrow{t} \tau_B(R \parallel_B P) \xrightarrow{t} \tau_B(R \parallel_B P')$, two applications of the fifth rule of Table 3 yield $Ab\top \in FT^*(\tau_B(T_\sigma \parallel_B P))$. \square

Corollary 8.5. \sqsubseteq_{FT}^* is fully abstract w.r.t. the operators of $CCSP_t^g$ and the canonical safety property.

Proof. By Theorem 7.2, \sqsubseteq_{FT}^* is a precongruence for the operators of $CCSP_t^g$. By definition, $P \sqsubseteq_{FT}^* Q$ implies $P \models \text{safety}(b) \Rightarrow Q \models \text{safety}(b)$. It remains to show that \sqsubseteq_{FT}^* is the coarsest preorder with these properties. To this end it suffices to find for any processes P and Q with $P \not\sqsubseteq_{FT}^* Q$ a recursion-free $CCSP_t^g$ -context $\mathcal{C}[_]$ such that $\mathcal{C}[P] \models \text{safety}(b)$, yet $\mathcal{C}[Q] \not\models \text{safety}(b)$.

So assume $P \not\sqsubseteq_{FT}^* Q$. By applying a bijective renaming operator on P and Q , mapping A to $A \setminus \{b\}$, I may assume that b is fresh for P and Q . Take $\sigma \in FT^*(Q) \setminus FT^*(P)$. By Property (*) in the proof of Theorem 6.14 I may assume that b is fresh for σ . Now Theorem 8.4 yields $\tau_{A \setminus \{b\}}(T_\sigma \parallel_{A \setminus \{b\}} P) \models \text{safety}(b)$, yet $\tau_{A \setminus \{b\}}(T_\sigma \parallel_{A \setminus \{b\}} Q) \not\models \text{safety}(b)$. \square

Full abstraction w.r.t. general safety properties. The above says that the unique preorder $\sqsubseteq_{\text{safety}}$ that is fully abstract w.r.t. the operators of $CCSP_t^g$ and the canonical safety property coincides with \sqsubseteq_{FT}^* . Without the time-out operator, $\sqsubseteq_{\text{safety}}$ would be way less discriminating, and coincides with reverse weak partial trace inclusion [Gla10]. I now check whether one could get an even finer preorder than \sqsubseteq_{FT}^* by also considering other safety properties.

To arrive at a general concept of safety property for LTSs, the paper [Gla10] assumes that some notion of *bad* is defined. This induces the safety property saying that this bad thing will never happen. To judge whether a process P satisfies this safety property, one should judge whether P can reach a state in which one would say that this bad thing had happened. But all observable behaviour of P that is recorded in an LTS until one comes to such a verdict, is the sequence of visible actions performed until that point. Thus the safety property is completely determined by the set sequences of visible actions that, when performed by P , lead to such a judgement. Therefore, [Gla10] proposes to define the concept of a safety property in terms of such a set: a safety property is given by a set $B \subseteq A^*$ of *bad* partial traces. A process P *satisfies* this property, notation $P \models \text{safety}(B)$, if P has no partial trace in B . It follows immediately from this definition that partial trace equivalent processes satisfy the same safety properties of this kind.

The present paper allows for a stronger concept of safety property. Here the observable behaviour of a process that is recorded until one comes to the verdict that something bad has happened can be modelled as a partial failure trace. Based on this, my definition is analogous to the one in [Gla10]:

Definition 8.6. A *safety property* of CCSP_t processes is given by a set $B \subseteq (A \cup \mathcal{P}(A))^* \top$ of *bad* partial failure traces. A process P *satisfies* this property, notation $P \models \text{safety}(B)$, if $FT^*(P) \cap B = \emptyset$.

It follows immediately that \sqsubseteq_{FT}^* respects all safety properties. Consequently, as in [Gla10], for the definition of $\sqsubseteq_{\text{safety}}$ it does not matter whether all safety properties are considered, or only the canonical one.

Trace equivalence and its congruence closure. A possible definition of a (weak) partial trace is simply a partial failure trace from which all refusal sets have been omitted. Define two processes to be (weak) (partial) trace equivalent iff they have the same partial traces. Now Corollary 8.5 implies that \equiv_{FT}^* is the congruence closure of trace equivalence, that is, the coarsest congruence for the operators of CCSP_t that is finer than trace equivalence.

May testing. *May testing* was proposed by De Nicola & Hennessy in [DH84] for the process algebra CCS. Translated to CCSP_t it works as follows. A *test* is a CCSP_t process that instead of visible actions from the alphabet A , uses visible actions from the alphabet $A \uplus \{\omega\}$, where ω is a fresh action reporting “success”. A process P *may pass* a test T iff $\tau_A(T \parallel_A P)$ has some partial failure trace $\beta\omega\top$, or, equivalently, has a partial trace containing the action ω . A process with such a trace represents a system with an execution that eventually reports success. Now define the preorder \sqsubseteq_{may} on CCSP_t processes by

$$P \sqsubseteq_{\text{may}} Q \text{ iff each test } T \text{ that } P \text{ may pass, may also be passed by } Q.$$

May testing contrasts with *must testing*, which requires that *each* execution leads to a state that can report success.

Theorem 8.7. $P \sqsubseteq_{\text{may}} Q$ iff $Q \sqsubseteq_{FT}^* P$.

Proof. Note that R has some partial failure trace $\beta\omega\top$ iff $R \not\models \text{safety}(\omega)$. Using this, Theorem 8.7 is an immediate consequence of Theorem 8.4, with ω in the rôle of b . The side conditions of b being fresh in Theorem 8.4 are redundant in this context, as ω is fresh by construction. \square

So the may-preorder is the converse of the safety preorder. This is related to calling ω a success action, instead of b a bad action. The may-preorder preserves the property that a process *can* do something good, whereas the safety preorder preserves the property that a process *can not* do something bad.

9. CONCLUSION

This paper extended the model of labelled transition systems with time-out transitions $s_1 \xrightarrow{t} s_2$. Such a transition is assumed to be instantaneous and unobservable by the environment, just like a transition $s_1 \xrightarrow{\tau} s_2$, but becomes available only a positive but finite amount of time after the represented system reaches state s_1 . This extension allows the implementation of a simple priority mechanism, thereby increasing the expressiveness of the model.

To denote such labelled transition systems I extended the standard process algebra CCSP with the action prefixing operator $t._$. Many semantic preorders \sqsubseteq have been defined on labelled transition systems, and thereby on the expressions in CCSP and other process algebras. $P \sqsubseteq Q$ says that process Q is a *refinement* of process P , where P is closer to the

specification of a system, and Q to its implementation. The least one may wish to require from such a preorder is

- (1) that it is a precongruence for the *static* operators of CCSP: partially synchronous parallel composition, renaming, and abstraction by renaming visible actions into the hidden action τ ,
- (2) and that it respects (basic) safety properties, meaning that any safety property of a process P also holds for its refinements Q .

The coarsest preorder with these properties is well known to be the reverse weak partial trace inclusion (see *e.g.* [Gla10]). Accordingly, weak partial trace inclusion is the preorder generated by *may testing* [DH84].

After the extension with time-out transitions, the weak partial trace preorder is no longer a precongruence for the static operators of CCSP. I here characterised the coarsest preorder satisfying (1) and (2) above as the *partial failure trace* preorder. Again, its converse is the preorder generated by may testing. All the above also applies to semantic equivalences, arising as the kernels of these preorders.

To obtain a (pre)congruence for all CCSP operators, including the CCS choice $+$, one needs to use a *rooted* version of the partial failure trace preorder, as is common in the study of preorders that abstract from the hidden action τ .

The work reported here remains in the realm of untimed process algebra, in the sense that the progress of time is not quantified. In the study of Timed CSP [RR87, DS95, Sch95, RR99], similar work has been done in a setting where the progress of time is quantified. Also there a form of failure trace semantics was found to be the right equivalence, and the connection with may testing was made in [Sch95]. Since the bookkeeping of time in [RR87, DS95, Sch95, RR99] is strongly interwoven with the formalisation of failure traces, it is not easy to determine whether my semantics can in some way be seen as an abstraction of the one for Timed CCSP, for instance by instantiating each quantified passage of time with a nondeterministic choice allowing *any* amount of time. This appears to be a question worthy of further research. A contribution of the present work is that the transformation of a failures based semantics of CSP [BHR84, Hoa85] to a failure trace based semantics of Timed CSP [RR87, DS95, Sch95, RR99] is not really necessitated by quantification of time—often seen as the main difference between CSP and Timed CSP—but rather by the introduction of a time-out operator (quantified or not).

Future work includes proving a congruence result for recursion, finding complete axiomatisations, and extending the approach from partial to complete failure traces, so that liveness properties will be respected. In that setting the expressiveness questions of Section 1.4 can be studied. The adaption of strong bisimilarity to the setting with time-out transitions is studied in [Gla20].

REFERENCES

- [Bae86] J.C.M. Baeten (1986): *Procesalgebra*. Programmatuurkunde, Kluwer, Deventer. In Dutch.
- [BBK87] J.C.M. Baeten, J.A. Bergstra & J.W. Klop (1987): *Ready-trace semantics for concrete process algebra with the priority operator*. *Computer Journal* 30(6), pp. 498–506, doi:10.1093/comjnl/30.6.498.
- [BHR84] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes*. *Journal of the ACM* 31(3), pp. 560–599, doi:10.1145/828.833.
- [BW90] J.C.M. Baeten & W.P. Weijland (1990): *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, doi:10.1017/CB09780511624193.

- [DH84] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes*. *Theoretical Computer Science* 34, pp. 83–133, doi:10.1016/0304-3975(84)90113-0.
- [DS95] J. Davies & S.A. Schneider (1995): *A brief history of Timed CSP*. *Theoretical Computer Science* 138, pp. 243–271, doi:10.1016/0304-3975(94)00169-J.
- [GH15] R.J. van Glabbeek & P. Höfner (2015): *CCS: It's not fair! Fair schedulers cannot be implemented in CCS-like languages even under progress and certain fairness assumptions*. *Acta Informatica* 52(2-3), pp. 175–205, doi:10.1007/s00236-015-0221-6. Available at <http://arxiv.org/abs/1505.05964>.
- [GH19] R.J. van Glabbeek & P. Höfner (2019): *Progress, Justness and Fairness*. *ACM Computing Surveys* 52(4):69, doi:10.1145/3329125. Available at <https://arxiv.org/abs/1810.07414>.
- [Gla93] R.J. van Glabbeek (1993): *The Linear Time – Branching Time Spectrum II; The semantics of sequential systems with silent moves*. In E. Best, editor: *Proceedings 4th International Conference on Concurrency Theory, CONCUR'93, LNCS 715, Springer*, pp. 66–81, doi:10.1007/3-540-57208-2_6.
- [Gla94] R.J. van Glabbeek (1994): *On the expressiveness of ACP (extended abstract)*. In A. Ponse, C. Verhoef & S.F.M. van Vlijmen, editors: *Proceedings First Workshop on the Algebra of Communicating Processes, ACP'94, Workshops in Computing, Springer*, pp. 188–217, doi:10.1007/978-1-4471-2120-6_8.
- [Gla01] R.J. van Glabbeek (2001): *The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes*. In J.A. Bergstra, A. Ponse & S.A. Smolka, editors: *Handbook of Process Algebra*, chapter 1, Elsevier, pp. 3–99, doi:10.1016/B978-044482830-9/50019-9.
- [Gla05] R.J. van Glabbeek (2005): *On Specifying Timeouts*. In L. Aceto & A.D. Gordon, editors: *Short Contributions from the Workshop on Algebraic Process Calculi: The First Twenty Five Years and Beyond, PA'05, Electronic Notes in Theoretical Computer Science* 162, Elsevier, pp. 112–113, doi:10.1016/j.entcs.2005.12.083.
- [Gla10] R.J. van Glabbeek (2010): *The Coarsest Precongruences Respecting Safety and Liveness Properties*. In C.S. Calude & V. Sassone, editors: *Proceedings 6th IFIP TC 1/WG 2.2 International Conference on Theoretical Computer Science (TCS'10); held as part of the World Computer Congress, IFIP 323, Springer*, pp. 32–52, doi:10.1007/978-3-642-15240-5_3. Available at <http://arxiv.org/abs/1007.5491>.
- [Gla17] R.J. van Glabbeek (2017): *Lean and Full Congruence Formats for Recursion*. In: *Proceedings 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'17, IEEE Computer Society Press*, doi:10.1109/LICS.2017.8005142. Available at <https://arxiv.org/abs/1704.03160>.
- [Gla18] R.J. van Glabbeek (2018): *A Theory of Encodings and Expressiveness*. Technical Report, Data61, CSIRO. Available at <https://arxiv.org/abs/1805.10415>. Extended abstract in: C. Baier and U. Dal Lago, editors: *Proceeding 21st International Conference on Foundations of Software Science and Computational Structures, FoSSaCS'18; held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS'18, LNCS 10803, Springer*, pp. 183–202, doi:10.1007/978-3-319-89366-2_10.
- [Gla19] R.J. van Glabbeek (2019): *Ensuring liveness properties of distributed systems: Open problems*. *Journal of Logical and Algebraic Methods in Programming* 109:100480, doi:10.1016/j.jlamp.2019.100480. Available at <http://arxiv.org/abs/1912.05616>.
- [Gla20] R.J. van Glabbeek (2020): *Reactive Bisimulation Semantics for a Process Algebra with Time-Outs*. In I. Konnov & L. Kovács, editors: *Proceedings 31st International Conference on Concurrency Theory, CONCUR'20, Leibniz International Proceedings in Informatics (LIPIcs)* 171, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, doi:10.4230/LIPIcs.CONCUR.2020.6.
- [Gor10] D. Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. *Information and Computation* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.
- [GV87] R.J. van Glabbeek & F.W. Vaandrager (1987): *Petri net models for algebraic theories of concurrency (extended abstract)*. In J.W. de Bakker, A.J. Nijman & P.C. Treleaven, editors: *Proceedings PARLE, Parallel Architectures and Languages Europe, Eindhoven, The Netherlands, June 1987, Vol. II: Parallel Languages, LNCS 259, Springer*, pp. 224–242, doi:10.1007/3-540-17945-3_13.
- [GV92] J.F. Groote & F.W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. *Information and Computation* 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6.

- [GV97] R.J. van Glabbeek & F.W. Vaandrager (1997): *The Difference Between Splitting in n and $n+1$* . *Information and Computation* 136(2), pp. 109–142, doi:10.1006/inco.1997.2634.
- [Hoa80] C.A.R. Hoare (1980): *Communicating sequential processes*. In R.M. McKeag & A.M. Macnaghten, editors: *On the construction of programs – an advanced course*, Cambridge University Press, pp. 229–254.
- [Hoa85] C.A.R. Hoare (1985): *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs.
- [Lam77] L. Lamport (1977): *Proving the correctness of multiprocess programs*. *IEEE Transactions on Software Engineering* 3(2), pp. 125–143, doi:10.1109/TSE.1977.229904.
- [Mid17] K. Middelburg (2017): Personal communication.
- [Mil90] R. Milner (1990): *Operational and algebraic semantics of concurrent processes*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, doi:10.1007/3-540-10235-3.
- [OH86] E.-R. Olderog & C.A.R. Hoare (1986): *Specification-oriented semantics for communicating processes*. *Acta Informatica* 23, pp. 9–66, doi:10.1007/BF00268075.
- [Old87] E.-R. Olderog (1987): *Operational Petri net semantics for CCSP*. In G. Rozenberg, editor: *Advances in Petri Nets 1987*, LNCS 266, Springer, pp. 196–223, doi:10.1007/3-540-18086-9_27.
- [Par81] D.M.R. Park (1981): *Concurrency and automata on infinite sequences*. In P. Deussen, editor: *5th GI Conference*, LNCS 104, Springer, pp. 167–183, doi:10.1007/BFb0017309.
- [Par08] J. Parrow (2008): *Expressiveness of Process Algebras*. *Electronic Notes in Theoretical Computer Science* 209, pp. 173–186, doi:10.1016/j.entcs.2008.04.011.
- [Phi87] I.C.C. Phillips (1987): *Refusal testing*. *Theoretical Computer Science* 50, pp. 241–284, doi:10.1016/0304-3975(87)90117-4.
- [RR87] G.M. Reed & A.W. Roscoe (1987): *Metric Spaces as Models for Real-Time Concurrency*. In M.G. Main, A. Melton, M.W. Mislove & D.A. Schmidt, editors: *Proceedings 3rd Workshop on Mathematical Foundations of Programming Language Semantics*, Tulane University, New Orleans, Louisiana, USA, April 1987, LNCS 298, Springer, pp. 331–343, doi:10.1007/3-540-19020-1_17.
- [RR99] G.M. Reed & A.W. Roscoe (1999): *The Timed Failures-Stability Model for CSP*. *Theoretical Computer Science* 211(1-2), pp. 85–127, doi:10.1016/S0304-3975(98)00214-X.
- [Sch95] S.A. Schneider (1995): *An Operational Semantics for Timed CSP*. *Information and Control* 116(2), pp. 193–213, doi:10.1006/inco.1995.1014.
- [Vaa93] F.W. Vaandrager (1993): *Expressiveness Results for Process Algebras*. In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: *Proceedings REX Workshop on Semantics: Foundations and Applications*, Beekbergen, The Netherlands, 1992, LNCS 666, Springer, pp. 609–638, doi:10.1007/3-540-56596-5_49.

APPENDIX A. PROOFS OF CONGRUENCE PROPERTIES

This appendix contains the proofs of the explicit characterisations of the operators \parallel_S , τ_I and \mathcal{R} on (rooted) partial failure trace sets: Properties (6.1), (6.5), (6.6), (6.11), (6.12) and (6.13). These are the central parts of the proofs showing that \equiv_{FT}^* and \equiv_{FT}^{r*} are congruences for these operators.

Proposition A.1. $FT^*(P \parallel_S Q) = col(FT^*(P) \parallel_S FT^*(Q))$.

Proof. “ \supseteq ”: Let $\sigma \in FT^*(P) \parallel_S FT^*(Q)$, and fix a valid decomposition of σ such that $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$. With Observation 5.3 it suffices to show that $\sigma \in FT^*(P \parallel_S Q)$. I proceed with structural induction on the derivations of $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$ from the rules in Table 3, making a case distinction on the first symbol of σ . The case $\sigma = \top$ is trivial.

- Let $\sigma = a\rho$ with $a \in S$. Then $\sigma_L = a\rho_L$ and $\sigma_R = a\rho_R$.

Assume $\sigma_L \in FT^*(P)$ is obtained from the third rule of Table 3. Then $P \xrightarrow{\tau} P'$ and $\sigma_L \in FT^*(P')$. So $P \parallel_S Q \xrightarrow{\tau} P' \parallel_S Q$ by Table 1, and $\sigma \in FT^*(P' \parallel_S Q)$ by induction. It follows that $\sigma \in FT^*(P \parallel_S Q)$.

The case that $\sigma_R \in FT^*(Q)$ is obtained from the third rule of Table 3 proceeds likewise.

So assume $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$ are both obtained from the second rule of Table 3. Then $P \xrightarrow{a} P'$, $Q \xrightarrow{a} Q'$, $\rho_L \in FT^*(P')$ and $\rho_R \in FT^*(Q')$. So $P \parallel_S Q \xrightarrow{a} P' \parallel_S Q'$ by Table 1, and $\rho \in FT^*(P' \parallel_S Q')$ by induction. It follows that $\sigma \in FT^*(P \parallel_S Q)$.

- Let $\sigma = a\rho$ with $a \notin S$. Assume that $\sigma_L = a\rho_L$ and $\sigma_R = \rho_R$ —the other case, that $\sigma_L = \rho_L$ and $\sigma_R = a\rho_R$, will follow by symmetry.

Assume $\sigma_L \in FT^*(P)$ is obtained from the third rule of Table 3. Then $P \xrightarrow{\tau} P'$ and $\sigma_L \in FT^*(P')$. So $P \parallel_S Q \xrightarrow{\tau} P' \parallel_S Q$ by Table 1, and $\sigma \in FT^*(P' \parallel_S Q)$ by induction. It follows that $\sigma \in FT^*(P \parallel_S Q)$.

Now assume $\sigma_L \in FT^*(P)$ is obtained from the second rule of Table 3. Then $P \xrightarrow{a} P'$ and $\rho_L \in FT^*(P')$. So $P \parallel_S Q \xrightarrow{a} P' \parallel_S Q$ by Table 1, and $\rho \in FT^*(P' \parallel_S Q)$ by induction. It follows that $\sigma \in FT^*(P \parallel_S Q)$.

- Let $\sigma = X\rho$. Then $\sigma_L = X_L\rho_L$ and $\sigma_R = X_R\rho_R$. Both these statements must be derived from the fourth to sixth rule of Table 3. Hence $P \xrightarrow{\alpha} P'$ for all $\alpha \in X_L \cup \{\tau\}$, and $Q \xrightarrow{\alpha} Q'$ for all $\alpha \in X_R \cup \{\tau\}$. Note that (6.2)–(6.4) hold with X , X_L and X_R in the rôles of σ_i , σ_i^L and σ_i^R . From (6.4) it follows that $P \parallel_S Q \xrightarrow{\alpha} P' \parallel_S Q'$ for all $\alpha \in X \cup \{\tau\}$.

Assume $\sigma_L \in FT^*(P)$ is obtained from the fifth rule of Table 3. Then $P \xrightarrow{t} P'$ and $\sigma_L \in FT^*(P')$. So $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q$ by Table 1, and $\sigma \in FT^*(P' \parallel_S Q)$ by induction. It follows that $\sigma \in FT^*(P \parallel_S Q)$.

The case that $\sigma_R \in FT^*(Q)$ is obtained from the fifth rule of Table 3 proceeds likewise.

Assume $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$ are both obtained from the fourth rule of Table 3. Then $\rho_L \in FT^*(P)$ and $\rho_R \in FT^*(Q)$. So $\rho \in FT^*(P \parallel_S Q)$ by induction. Hence $\sigma \in FT^*(P \parallel_S Q)$.

Assume $\sigma_L \in FT^*(P)$ is obtained from the sixth rule of Table 3 and $\sigma_R \in FT^*(Q)$ from the fourth. Then $P \xrightarrow{t} P'$, $\rho_L \in FT^*(P')$ and ρ_L has the form $a\eta_L$ with $a \in X_L \subseteq X$ —the latter using (6.4). Moreover, $\rho_R \in FT^*(Q)$. Now $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q$ by Table 1, and $\rho \in FT^*(P' \parallel_S Q)$ by induction. Since X_L is system-ended in σ_L , by (6.2) X must be system-ended in σ . Hence ρ has the form $b\eta$ with $b \in X$. Using the sixth rule of Table 3 it follows that $\sigma \in FT^*(P \parallel_S Q)$.

The case that $\sigma_L \in FT^*(P)$ is obtained from the fourth rule of Table 3 and $\sigma_R \in FT^*(Q)$ from the sixth proceeds likewise.

In case $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$ are both obtained from the sixth rule of Table 3, X_L is system-ended in σ_L and X_R is system-ended in σ_R . So this case is excluded by condition (6.3).

“ \subseteq ”: Let $FT^e(x)$ be defined exactly as $FT^*(x)$ —see Table 3—except that the conclusion of the fifth rule reads “ $XX\rho \in FT^e(x)$ ”. Now $FT^*(P \parallel_S Q) \subseteq col(FT^e(P \parallel_S Q))$, so, using that col is monotonous, it suffices to show that $FT^e(P \parallel_S Q) \subseteq FT^*(P) \parallel_S FT^*(Q)$.

So let $\sigma \in FT^e(P \parallel_S Q)$. With structural induction on the derivation of $\sigma \in FT^e(P \parallel_S Q)$ from the (amended) rules of Tables 1 and 3 I show that $\sigma \in FT^*(P) \parallel_S FT^*(Q)$. This means I have to give a valid decomposition of σ such that $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$.

- Let $\sigma = \top$, with $\sigma \in FT^e(P \parallel_S Q)$ derived from the first rule of Table 3. Then σ has only one decomposition, which is valid. Trivially, $\sigma_L = \top \in FT^*(P)$ and $\sigma_R = \top \in FT^*(Q)$.

- Let $\sigma = a\rho$, with $\sigma \in FT^e(P \parallel_S Q)$ derived from the second rule. Then $P \parallel_S Q \xrightarrow{a} P' \parallel_S Q'$ and $\rho \in FT^e(P' \parallel_S Q')$. By induction there is a valid decomposition of ρ such that $\rho_L \in FT^*(P')$ and $\rho_R \in FT^*(Q')$. Depending on which rule of Table 1 was employed to derive $P \parallel_S Q \xrightarrow{a} P' \parallel_S Q'$,

- (i) either $a \notin S$, $P \xrightarrow{a} P'$ and $Q' = Q$,
- (ii) or $a \in S$, $P \xrightarrow{a} P'$ and $Q \xrightarrow{a} Q'$,
- (iii) or $a \notin S$, $P' = P$ and $Q \xrightarrow{a} Q'$.

In case (i) take $\sigma_L := a\rho_L$ and $\sigma_R := \rho_R \in FT^*(Q)$. As the given decomposition of ρ is valid, so is the newly constructed one of σ . The second rule of Table 3 yields $\sigma_L \in FT^*(P)$. In case (ii) $\sigma_L := a\rho_L \in FT^*(P)$ and $\sigma_R := a\rho_R \in FT^*(Q)$. Also this decomposition is valid. In case (iii) $\sigma_L := \rho_L \in FT^*(P)$ and $\sigma_R := a\rho_R \in FT^*(Q)$. Also this decomposition is valid.

- Suppose $\sigma \in FT^e(P \parallel_S Q)$ is derived from the third rule of Table 3. Then $P \parallel_S Q \xrightarrow{\tau} P' \parallel_S Q'$ and $\sigma \in FT^e(P' \parallel_S Q')$. By induction there is a valid decomposition of σ such that $\sigma_L \in FT^*(P')$ and $\sigma_R \in FT^*(Q')$. Depending on which rule of Table 1 was employed to derive $P \parallel_S Q \xrightarrow{\tau} P' \parallel_S Q'$,

- (i) either $P \xrightarrow{\tau} P'$ and $Q' = Q$,
- (ii) or $P' = P$ and $Q \xrightarrow{\tau} Q'$.

In either case the third rule of Table 3 yields $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$.

- Let $\sigma = X\rho$, with $\sigma \in FT^e(P \parallel_S Q)$ derived from the fourth rule of Table 3. Then $P \parallel_S Q \xrightarrow{\alpha} P' \parallel_S Q'$ for all $\alpha \in X \cup \{\tau\}$, and $\rho \in FT^e(P \parallel_S Q)$. By induction there is a valid decomposition of ρ such that $\rho_L \in FT^*(P)$ and $\rho_R \in FT^*(Q)$. Let $X_L := (X \setminus S) \cup \{a \in X \cap S \mid P \xrightarrow{a}\}$ and $X_R := (X \setminus S) \cup \{a \in X \cap S \mid Q \xrightarrow{a}\}$. Considering that, for $a \in X \cap S$, $P \parallel_S Q \xrightarrow{a}$ iff $P \xrightarrow{a} \vee Q \xrightarrow{a}$, Condition (6.4), with X , X_L and X_R in the rôles of σ_i , σ_i^L and σ_i^R , is satisfied. Moreover, $P \xrightarrow{\alpha}$ for all $\alpha \in X_L \cup \{\tau\}$, and $Q \xrightarrow{\alpha}$ for all $\alpha \in X_R \cup \{\tau\}$. Take $\sigma_L = X_L\rho_L$ and $\sigma_R = X_R\rho_R$. The fourth rule of Table 3 yields $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$. It remains to show that this decomposition satisfies Conditions (6.2) and (6.3).

Suppose ρ_L has the form $a\eta_L$. Then, by Corollary 8.3, $a \notin X_L$. It follows that X_L is not system-ended in σ_L . In the same manner it follows that X_R is not system-ended in σ_R , and that X is not system-ended in σ .

- Let $\sigma = XX\rho$, with $\sigma \in FT^e(P \parallel_S Q)$ derived from the amended fifth rule of Table 3. Then $P \parallel_S Q \xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$, $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q'$ and $X\rho \in FT^e(P' \parallel_S Q')$. By induction there is a valid decomposition $(X_L\rho_L, X_R\rho_R)$ of $X\rho$ such that $X_L\rho_L \in FT^*(P')$ and $X_R\rho_R \in FT^*(Q')$. Note that $P' \xrightarrow{a}$ for all $a \in X_L \cup \{\tau\}$. Depending on which rule of Table 1 was employed to derive $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q'$,

- (i) either $P \xrightarrow{t} P'$ and $Q' = Q$,
- (ii) or $P' = P$ and $Q \xrightarrow{t} Q'$.

For reasons of symmetry, I may assume the former.

Let $X'_L := \{a \in X_L \mid P \xrightarrow{a}\}$ and $X'_R := (X \setminus S) \cup \{a \in X \cap S \mid Q \xrightarrow{a}\}$. Using the fourth rule of Table 3, $\sigma_L := X'_L X_L\rho_L \in FT^*(P')$, and by the fifth rule $\sigma_L \in FT^*(P)$, also using that $P \xrightarrow{\tau}$. Moreover, by the fourth rule of Table 3, $\sigma_R := X'_R X_R\rho_R \in FT^*(Q)$, using that $Q \xrightarrow{\alpha}$ for all $\alpha \in (X \setminus S) \cup \{\tau\}$. It remains to show the validity of the decomposition of σ that yields σ_L and σ_R . Conditions (6.2) and (6.3) hold trivially. So it remains to check Condition (6.4).

Let $a \in X \setminus S$. Then $a \in X_L \setminus S$ and $P \parallel_S Q \xrightarrow{a}$, so $P \xrightarrow{a}$, and $a \in X'_L \setminus S$.

Conversely, $X'_L \setminus S \subseteq X_L \setminus S = X \setminus S$, and by definition $X'_R \setminus S = X \setminus S$.

Let $a \in X \cap S$. Then $P \parallel_S Q \not\stackrel{a}{\rightarrow}$, so $P \not\stackrel{a}{\rightarrow}$ or $Q \not\stackrel{a}{\rightarrow}$. In case $Q \not\stackrel{a}{\rightarrow}$, by definition $a \in X'_R \cap S$. In case $P \not\stackrel{a}{\rightarrow}$, it is not possible that $a \in X_R$. Hence $a \in X_L$, by Property (6.4) of the given valid decomposition of $X\rho$. Moreover, $P \not\stackrel{a}{\rightarrow}$, so $a \in X'_L \cap S$.

Conversely, by definition $X'_R \cap S \subseteq X \cap S$ and $X'_L \cap S \subseteq X_L \cap S \subseteq X \cap S$.

- Let $\sigma = Xa\rho$, with $\sigma \in FT^e(P \parallel_S Q)$ derived from the sixth rule of Table 3. Then $P \parallel_S Q \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X \cup \{\tau\}$, $a \in X$, $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q'$ and $a\rho \in FT^e(P' \parallel_S Q')$. By induction there is a valid decomposition of $\zeta := a\rho$ such that $\zeta_L \in FT^*(P')$ and $\zeta_R \in FT^*(Q')$. Let $X_L := (X \setminus S) \cup \{b \in X \cap S \mid P \not\stackrel{b}{\rightarrow}\}$ and $X_R := (X \setminus S) \cup \{b \in X \cap S \mid Q \not\stackrel{b}{\rightarrow}\}$. Then $P \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X_L \cup \{\tau\}$, and $Q \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in X_R \cup \{\tau\}$. Take $\sigma_L = X_L \zeta_L$ and $\sigma_R = X_R \zeta_R$. I have to show that this is a valid decomposition of σ , and that $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$.

Considering that, for all $b \in X \cap S$, $P \parallel_S Q \not\stackrel{b}{\rightarrow}$ iff $P \not\stackrel{b}{\rightarrow} \vee Q \not\stackrel{b}{\rightarrow}$, Condition (6.4) is satisfied.

Depending on which rule of Table 1 was employed to derive $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q'$,

- (i) either $P \xrightarrow{t} P'$ and $Q' = Q$,
- (ii) or $P' = P$ and $Q \xrightarrow{t} Q'$.

For reasons of symmetry, I may assume the former. By the fourth rule of Table 3, $\sigma_R \in FT^*(Q)$.

I now show that X_R is not system-ended in σ_R . Namely, if ζ_R is of the form $b\eta_R$, then $b \notin X_R$ by Corollary 8.3. Thus Condition (6.3) holds.

- As a further case distinction, assume $a \notin S$. Since $a \in X$ and thus $a \in X_L$ and $a \in X_R$, by the above it follows that ζ_R is not of the form $a\eta_R$. Hence, ζ_L has the form $a\eta_L$. The sixth rule of Table 3 yields $\sigma_L \in FT^*(P)$. Since X_L is system-ended in σ_L and X is system-ended in σ , Condition (6.2) holds as well.
- Finally assume $a \in S$. Then ζ_L has the form $a\eta_L$ and ζ_R has the form $a\eta_R$. Since X_R is not system-ended in σ_R , one has $a \notin X_R$, that is, $Q \not\stackrel{a}{\rightarrow}$. As $P \parallel_S Q \not\stackrel{a}{\rightarrow}$, it follows that $P \not\stackrel{a}{\rightarrow}$, and thus $a \in X_L$. The sixth rule of Table 3 yields $\sigma_L \in FT^*(P)$. Since X_L is system-ended in σ_L and X is system-ended in σ , Condition (6.2) holds as well. \square

Proposition A.2. $\sigma \in FT^*(\tau_I(P)) \Leftrightarrow \exists \rho \in FT^*(P). \tau_I(\rho) = \sigma \cup I$.

Proof. Since each state R reachable from $\tau_I(P)$ satisfies $R \not\stackrel{\alpha}{\rightarrow}$ for all $\alpha \in I$, a trivial induction shows that

$$\sigma \in FT^*(\tau_I(P)) \Leftrightarrow \sigma \cup I \in FT^*(\tau_I(P)). \quad (*)$$

“ \Leftarrow ”: Let $\rho \in FT^*(P)$ and ρ survives abstraction from I . By Condition (i) in the definition of abstraction survival, $\tau_I(\rho)$ is of the form $\sigma \cup I$. In view of (*), it suffices to show that $\tau_I(\rho) \in FT^*(\tau_I(P))$. I do so with structural induction on the derivation of $\rho \in FT^*(P)$. The case $\rho = \top$ is trivial.

Assume $\rho \in FT^*(P)$ is obtained from the third rule of Table 3. Then $P \xrightarrow{\tau} P'$ and $\rho \in FT^*(P')$. By induction $\tau_I(\rho) \in FT^*(\tau_I(P'))$. By Table 1 $\tau_I(P) \xrightarrow{\tau} \tau_I(P')$. Hence $\tau_I(\rho) \in FT^*(\tau_I(P))$ by the third rule of Table 3.

Assume $\rho = a\eta \in FT^*(P)$ is obtained from the second rule of Table 3. Then $P \xrightarrow{a} P'$ and $\eta \in FT^*(P')$. By induction $\tau_I(\eta) \in FT^*(\tau_I(P'))$. By Table 1 $\tau_I(P) \xrightarrow{a} \tau_I(P')$, with $\alpha = \tau$ if $a \in I$ and $\alpha = a$ otherwise. In the first case $\tau_I(\rho) = \tau_I(\eta) \in FT^*(\tau_I(P))$ by the third rule of Table 3. In the second case $\tau_I(\rho) = a\tau_I(\eta) \in FT^*(\tau_I(P))$ by the second rule of Table 3.

Assume $\rho = X\eta \in FT^*(P)$ is obtained from the fourth rule of Table 3. Then $P \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$ and $\eta \in FT^*(P)$. If η has the form $a\zeta$, then $a \notin X$ by Corollary 8.3. In particular, $a \notin I$, using Condition (i) in the definition of abstraction survival. Consequently, $\tau_I(\rho) = X\tau_I(\eta)$, for any possibly form of η . Furthermore, $\tau_I(P) \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$. Here I use that $I \subseteq X$. By induction $\tau_I(\eta) \in FT^*(\tau_I(P))$. Hence $\tau_I(\rho) \in FT^*(\tau_I(P))$ by the fourth rule of Table 3.

Assume $\rho = X\eta \in FT^*(P)$ is obtained from the fifth rule of Table 3. Then $P \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$, $P \xrightarrow{t} P'$ and $\rho \in FT^*(P')$. So $\tau_I(P) \xrightarrow{t} \tau_I(P')$ and $\tau_I(P) \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$, using that $I \subseteq X$. By induction $\tau_I(\rho) \in FT^*(\tau_I(P'))$. Note that $\tau_I(\rho)$ has the form $X\zeta$, although not necessarily with $\zeta = \tau_I(\eta)$. The fifth rule of Table 3 yields $\tau_I(\rho) \in FT^*(\tau_I(P))$.

Assume $\rho = X\eta \in FT^*(P)$ is obtained from the sixth rule of Table 3. Then $P \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$, $P \xrightarrow{t} P'$ and $\eta \in FT^*(P')$. So $\tau_I(P) \xrightarrow{t} \tau_I(P')$ and $\tau_I(P) \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$. By induction $\tau_I(\eta) \in FT^*(\tau_I(P'))$. Using that ρ survives abstraction, I consider three alternatives for η .

- Let $\eta = c_0 \cdots c_n \top$ with the $c_i \in I$. Then $\tau_I(\rho) = X\top$. The first and fourth rules of Table 3 yield $\tau_I(\rho) \in FT^*(\tau_I(P))$.
- Let $\eta = c_0 \cdots c_n X\zeta$ with the $c_i \in I$. Then $\tau_I(\rho) = \tau_I(\eta) = \tau_I(X\zeta)$, and $\tau_I(\eta)$ has the form $X\zeta'$. The fifth rule of Table 3 yields $\tau_I(\rho) \in FT^*(\tau_I(P))$.
- Let $\eta = c_1 \cdots c_n a\zeta$ with the $c_i \in I$ and $a \notin I$. Then $a \in X$ (using Condition (iii) of abstraction survival if $n > 0$, and the side condition of the sixth rule of Table 3 if $n = 0$) and $\tau_I(\rho) = X\tau_I(\eta)$ with $\tau_I(\eta) = a\tau_I(\zeta)$. The sixth rule of Table 3 yields $\tau_I(\rho) \in FT^*(\tau_I(P))$.

“ \Rightarrow ”: Let $\sigma \in FT^*(\tau_I(P))$; by (*) I may assume that $\sigma = \sigma \cup I$. I have to find a $\rho \in FT^*(P)$ such that $\tau_I(\rho) = \sigma$. I do so with structural induction on the derivation of $\sigma \in FT^*(\tau_I(P))$.

Assume $\sigma \in FT^*(\tau_I(P))$ is obtained from the first rule of Table 3. Take $\rho = \top$, so that $\tau_I(\rho) = \top = \sigma$. The first rule of Table 3 yields $\rho \in FT^*(P)$.

Assume $\sigma \in FT^*(\tau_I(P))$ is obtained from the second rule of Table 3. Then $\sigma = a\zeta$, $\tau_I(P) \xrightarrow{a} \tau_I(P')$ and $\zeta \in FT^*(\tau_I(P'))$. So $P \xrightarrow{a} P'$ and $a \notin I$. By induction, there is a $\rho \in FT^*(P')$ such that $\tau_I(\rho) = \zeta$. The second rule of Table 3 yields $a\rho \in FT^*(P)$. Moreover, $\tau_I(a\rho) = a\zeta = \sigma$.

Assume $\sigma \in FT^*(\tau_I(P))$ is obtained from the third rule of Table 3. Then $\tau_I(P) \xrightarrow{\tau} \tau_I(P')$ and $\sigma \in FT^*(\tau_I(P'))$. So $P \xrightarrow{\alpha} P'$ with $\alpha \in I \cup \{\tau\}$. By induction, there is a $\rho \in FT^*(P')$ such that $\tau_I(\rho) = \sigma$. In case $\alpha = \tau$, the third rule of Table 3 yields $\rho \in FT^*(P)$. In case $\alpha \neq \tau$, the second rule of Table 3 yields $\alpha\rho \in FT^*(P)$; moreover, $\tau_I(\alpha\rho) = \tau_I(\rho) = \sigma$.

Assume $\sigma \in FT^*(\tau_I(P))$ is obtained from the fourth rule of Table 3. Then $\sigma = X\zeta$, $\zeta \in FT^*(\tau_I(P))$ and $\tau_I(P) \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$. So $P \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$. By induction, there is a $\rho \in FT^*(P)$ such that $\tau_I(\rho) = \zeta$. Hence $X\rho \in FT^*(P)$ by the fourth rule of Table 3. In case ρ has the form \top or $Y\eta$, one has $\tau_I(X\rho) = X\tau_I(\rho) = X\zeta = \sigma$, which needed to be shown. So suppose ρ has the form $a\eta$. Then $a \notin X$ by Corollary 8.3. In particular, $a \notin I$, using Condition (i) in the definition of abstraction survival. Consequently, $\tau_I(X\rho) = X\tau_I(\rho) = X\zeta = \sigma$.

Assume $\sigma \in FT^*(\tau_I(P))$ is obtained from the fifth rule of Table 3. Then $\sigma = X\zeta$, $\tau_I(P) \xrightarrow{t} \tau_I(P')$, $\sigma \in FT^*(\tau_I(P'))$ and $\tau_I(P) \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$. So $P \xrightarrow{t} P'$ and $P \not\rightarrow^\alpha$ for all $\alpha \in X \cup \{\tau\}$. By induction, there is a $\rho \in FT^*(P')$ such that $\tau_I(\rho) = \sigma$. This ρ must have the form $X\eta$. Hence $\rho \in FT^*(P)$ by the fifth rule of Table 3.

Finally, assume $\sigma \in FT^*(\tau_I(P))$ is obtained from the sixth rule of Table 3. Then $\sigma = Xa\zeta$, $a \in X$, $\tau_I(P) \xrightarrow{t} \tau_I(P')$, $a\zeta \in FT^*(\tau_I(P))$ and $\tau_I(P) \not\xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$. So $P \xrightarrow{t} P'$ and $P \not\xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$. By induction, there is a $\rho \in FT^*(P')$ such that $\tau_I(\rho) = a\zeta$. So $a \notin I$ and ρ has the form $c_1 \dots c_n a \eta$ with $c_i \in I \subseteq X$ for $i = 1, \dots, n$ and $\tau_I(\eta) = \zeta$. The sixth rule of Table 3 yields $X\rho \in FT^*(P)$. Moreover, $\tau_I(X\rho) = Xa\tau_I(\eta) = Xa\zeta = \sigma$. \square

Proposition A.3. $\sigma \in FT^*(\mathcal{R}(P)) \Leftrightarrow \exists \rho \in FT^*(P). \rho \in \mathcal{R}^{-1}(\sigma)$.

Proof. “ \Leftarrow ”: Let $\rho \in FT^*(P)$. With structural induction on the derivation of $\rho \in FT^*(P)$ I show that, for any σ with $\rho \in \mathcal{R}^{-1}(\sigma)$, one has $\sigma \in FT^*(\mathcal{R}(P))$.

If $\rho \in FT^*(P)$ is obtained from the first rule of Table 3, then $\sigma = \rho = \top$ and $\sigma \in FT^*(\mathcal{R}(P))$.

Assume $\rho = a\eta \in FT^*(P)$ is obtained from the second rule. Then $P \xrightarrow{a} P'$ and $\eta \in FT^*(P')$. Moreover, $\sigma = b\zeta$ with $(a, b) \in \mathcal{R}$ and $\eta \in \mathcal{R}^{-1}(\zeta)$. So $\mathcal{R}(P) \xrightarrow{b} \mathcal{R}(P')$ by Table 1, and $\zeta \in FT^*(\mathcal{R}(P'))$ by induction. It follows that $\sigma \in FT^*(\mathcal{R}(P))$.

Assume $\rho \in FT^*(P)$ is obtained from the third rule of Table 3. Then $P \xrightarrow{\tau} P'$ and $\rho \in FT^*(P')$. So $\mathcal{R}(P) \xrightarrow{\tau} \mathcal{R}(P')$ by Table 1, and $\sigma \in FT^*(\mathcal{R}(P'))$ by induction. It follows that $\sigma \in FT^*(\mathcal{R}(P))$.

Assume $\rho = Y\eta \in FT^*(P)$ is obtained from the fourth rule. Then $P \not\xrightarrow{\alpha}$ for all $\alpha \in Y \cup \{\tau\}$ and $\eta \in FT^*(P)$. Moreover, $\sigma = X\zeta$ with $Y = \mathcal{R}^{-1}(X)$ and $\eta \in \mathcal{R}^{-1}(\zeta)$. So $\mathcal{R}(P) \not\xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$, and $\zeta \in FT^*(\mathcal{R}(P))$ by induction. It follows that $\sigma \in FT^*(\mathcal{R}(P))$.

Assume $\rho = Y\eta \in FT^*(P)$ is obtained from the fifth rule. Then $P \not\xrightarrow{\alpha}$ for all $\alpha \in Y \cup \{\tau\}$, $P \xrightarrow{t} P'$ and $\rho \in FT^*(P')$. Moreover, $\sigma = X\zeta$ with $Y = \mathcal{R}^{-1}(X)$. So $\mathcal{R}(P) \not\xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$, $\mathcal{R}(P) \xrightarrow{t} \mathcal{R}(P')$ and $\sigma \in FT^*(\mathcal{R}(P'))$ by induction. It follows that $\sigma \in FT^*(\mathcal{R}(P))$.

Assume $\rho = Y a \eta \in FT^*(P)$ is obtained from the sixth rule. Then $P \not\xrightarrow{\alpha}$ for all $\alpha \in Y \cup \{\tau\}$, $P \xrightarrow{t} P'$, $a \in Y$ and $a\eta \in FT^*(P)$. Moreover, $\sigma = X b \zeta$ with $Y = \mathcal{R}^{-1}(X)$, $(a, b) \in \mathcal{R}$, $b \in X$ and $a\eta \in \mathcal{R}^{-1}(b\zeta)$. So $\mathcal{R}(P) \not\xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$, $\mathcal{R}(P) \xrightarrow{t} \mathcal{R}(P')$ and $b\zeta \in FT^*(\mathcal{R}(P'))$ by induction. It follows that $\sigma \in FT^*(\mathcal{R}(P))$.

“ \Rightarrow ”: Let $\sigma \in FT^*(\mathcal{R}(P))$. With structural induction on the derivation of $\sigma \in FT^*(\mathcal{R}(P))$ I show that there exists a $\rho \in FT^*(P)$ with $\rho \in \mathcal{R}^{-1}(\sigma)$.

If $\sigma = \top \in FT^*(\mathcal{R}(P))$ is obtained from the first rule of Table 3, take $\rho = \top$. Now $\rho \in FT^*(P)$.

Assume $\sigma = b\zeta \in FT^*(\mathcal{R}(P))$ is obtained from the second rule of Table 3. Then $\mathcal{R}(P) \xrightarrow{b} \mathcal{R}(P')$ and $\zeta \in FT^*(\mathcal{R}(P'))$. So $P \xrightarrow{a} P'$ for some a with $(a, b) \in \mathcal{R}$, and by induction there is an $\eta \in FT^*(P')$ with $\eta \in \mathcal{R}^{-1}(\zeta)$. Take $\rho = a\eta$. Then $\rho \in FT^*(P)$ and $\rho \in \mathcal{R}^{-1}(\sigma)$.

Assume $\sigma \in FT^*(\mathcal{R}(P))$ is obtained from the third rule of Table 3. Then $\mathcal{R}(P) \xrightarrow{\tau} \mathcal{R}(P')$ and $\sigma \in FT^*(\mathcal{R}(P'))$. So $P \xrightarrow{\tau} P'$, and by induction there is a $\rho \in FT^*(P')$ with $\rho \in \mathcal{R}^{-1}(\sigma)$. Hence $\rho \in FT^*(P)$.

Assume $\sigma = X\zeta \in FT^*(\mathcal{R}(P))$ is obtained from the fourth rule of Table 3. Then $\mathcal{R}(P) \not\xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$ and $\zeta \in FT^*(\mathcal{R}(P))$. So $P \not\xrightarrow{\alpha}$ for all $\alpha \in \mathcal{R}^{-1}(X) \cup \{\tau\}$. By induction there is an $\eta \in FT^*(P)$ with $\eta \in \mathcal{R}^{-1}(\zeta)$. Take $\rho = \mathcal{R}^{-1}(X)\eta$. Then $\rho \in FT^*(P)$ by the fourth rule of Table 3. In case ζ has the form $b\zeta'$, then η has the form $a\eta'$ with $(a, b) \in \mathcal{R}$; moreover, Corollary 8.3 yields $b \notin X$ as well as $a \notin \mathcal{R}^{-1}(X)$. Consequently, $\rho \in \mathcal{R}^{-1}(\sigma)$.

Assume $\sigma = X\zeta \in FT^*(\mathcal{R}(P))$ is obtained from the fifth rule of Table 3. Then $\mathcal{R}(P) \xrightarrow{\alpha} \mathcal{R}(P')$ for all $\alpha \in X \cup \{\tau\}$, $\mathcal{R}(P) \xrightarrow{t} \mathcal{R}(P')$ and $\sigma \in FT^*(\mathcal{R}(P'))$. So $P \xrightarrow{\alpha} P'$ for all $\alpha \in \mathcal{R}^{-1}(X) \cup \{\tau\}$, $P \xrightarrow{t} P'$, and by induction there is a $\rho \in FT^*(P')$ with $\rho \in \mathcal{R}^{-1}(\sigma)$. Of course ρ must have the form $\mathcal{R}^{-1}(X)\eta$. So $\rho \in FT^*(P)$ by the fifth rule of Table 3.

Finally assume $\sigma = Xb\zeta \in FT^*(\mathcal{R}(P))$ is obtained from the sixth rule of Table 3. Then $\mathcal{R}(P) \xrightarrow{\alpha} \mathcal{R}(P')$ for all $\alpha \in X \cup \{\tau\}$, $b \in X$, $\mathcal{R}(P) \xrightarrow{t} \mathcal{R}(P')$ and $b\zeta \in FT^*(\mathcal{R}(P'))$. So $P \xrightarrow{\alpha} P'$ for all $\alpha \in \mathcal{R}^{-1}(X) \cup \{\tau\}$, $P \xrightarrow{t} P'$, and by induction there is an $a\eta \in FT^*(P')$ with $(a, b) \in \mathcal{R}$ and $\eta \in \mathcal{R}^{-1}(\zeta)$. So $a \in \mathcal{R}^{-1}(X)$. Take $\rho = \mathcal{R}^{-1}(X)a\eta$. Then $\rho \in FT^*(P)$ by the sixth rule of Table 3. Moreover, $\rho \in \mathcal{R}^{-1}(\sigma)$. \square

Proposition A.4. $FT^{r*}(P \parallel_S Q) = col(FT^{r*}(P) \parallel_S FT^{r*}(Q))$.

Proof. “ \supseteq ”: Let $\sigma \in FT^{r*}(P) \parallel_S FT^{r*}(Q)$, and fix a valid decomposition of σ as a witness. With the clearly valid generalisation of Observation 5.3 to $FT^{r*}(P)$ it suffices to show that $\sigma \in FT^{r*}(P \parallel_S Q)$.

- Let $\sigma = \text{STAB}$. Then $\text{STAB} \in FT^{r*}(P)$ and $\text{STAB} \in FT^{r*}(Q)$. So $\text{stable}(P)$ and $\text{stable}(Q)$. Indeed, this implies $\text{stable}(P \parallel_S Q)$, and thus $\sigma \in FT^{r*}(P \parallel_S Q)$.
- Let $\sigma = \top$, $\sigma = a\rho$ or $\sigma = X\rho$. Then $\sigma_L \in FT^*(P)$ and $\sigma_R \in FT^*(Q)$, so from (the proof of) Proposition A.1 one obtains $\sigma \in FT^*(P \parallel_S Q) \subseteq FT^{r*}(P \parallel_S Q)$.
- Let $\sigma = tX\rho$. Assume that $\sigma_L = tX_L\rho_L \in FT^{r*}(P)$ and $\sigma_R = X_R\rho_R \in FT^{r*}(Q)$; the other case follows by symmetry. Then $\sigma_R \in FT^*(Q)$. Moreover, $\text{STAB} \in FT^{r*}(Q)$, so Q is stable. By Definition 6.11, $P \xrightarrow{t} P'$, $X_L\rho_L \in FT^*(P')$ and $P \xrightarrow{\alpha} P'$ for $\alpha \in X_L \cup \{\tau\}$. Thus $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q$ by Table 1, and $X\rho \in FT^*(P' \parallel_S Q)$ by (the proof of) Proposition A.1. Furthermore, $X_R\rho_R \in FT^*(Q)$ must be derived by the fourth to sixth rule of Table 3, so $Q \xrightarrow{\alpha} P'$ for all $\alpha \in X_R \cup \{\tau\}$. Hence $P \parallel_S Q \xrightarrow{\alpha} P'$ for all $\alpha \in X \cup \{\tau\}$, using (6.4). Therefore, $\sigma \in FT^{r*}(P \parallel_S Q)$.
- Let $\sigma = \text{POSTSTAB}$. Then either $\text{POSTSTAB} \in FT^{r*}(P) \wedge \text{POSTSTAB} \in FT^{r*}(Q)$, or $\text{POSTSTAB} \in FT^{r*}(P) \wedge \text{STAB} \in FT^{r*}(Q)$, or $\text{STAB} \in FT^{r*}(P) \wedge \text{POSTSTAB} \in FT^{r*}(Q)$. Either way, P and Q both have paths of τ -transitions to a stable state, and at least one of them is nonempty. It follows that $P \parallel_S Q$ has a nonempty path of τ -transitions to a stable state, so $\sigma \in FT^{r*}(P \parallel_S Q)$.

“ \subseteq ”: Let $FT^{re}(P)$ be defined exactly as $FT^{r*}(P)$ —see Definition 6.11—but writing $tXX\sigma$ instead of $tX\sigma$, and employing $FT^e(P)$ and $FT^e(P')$, as defined in the proof of Proposition A.1, instead of $FT^*(P)$ and $FT^*(P')$. Obviously, $FT^{r*}(P \parallel_S Q) \subseteq col(FT^{re}(P \parallel_S Q))$, so, using that col is monotonous, it suffices to show that $FT^{re}(P \parallel_S Q) \subseteq FT^{r*}(P) \parallel_S FT^{r*}(Q)$.

So let $\sigma \in FT^{re}(P \parallel_S Q)$. With structural induction on the derivation of $\sigma \in FT^{re}(P \parallel_S Q)$ from the (amended) rules of Tables 1 and 3 I show that $\sigma \in FT^{r*}(P) \parallel_S FT^{r*}(Q)$. This means that in case $\sigma \neq \text{STAB}, \text{POSTSTAB}$ I have to give a valid decomposition of σ such that $\sigma_L \in FT^{r*}(P)$ and $\sigma_R \in FT^{r*}(Q)$; moreover, I have to show that $\text{STAB} \in FT^{r*}(P)$ and $\text{STAB} \in FT^{r*}(Q)$ in case σ has the form $tX\rho$.

- Let $\sigma = \top$, $\sigma = a\rho$ or $\sigma = X\rho$. Then $\sigma \in FT^e(P \parallel_S Q)$. So by the proof of Proposition A.1 there is a valid decomposition of σ such that $\sigma_L \in FT^*(P) \subseteq FT^{r*}(P)$ and $\sigma_R \in FT^*(Q) \subseteq FT^{r*}(Q)$.
- Let $\sigma = \text{STAB}$. Then $P \parallel_S Q \xrightarrow{\tau} P'$, so $P \xrightarrow{\tau} P'$ and $Q \xrightarrow{\tau} P'$. So $\text{STAB} \in FT^{r*}(P)$ and $\text{STAB} \in FT^{r*}(Q)$.
- Let $\sigma = tXX\rho$. By Definition 6.11, $P \parallel_S Q \xrightarrow{\alpha} P'$ for all $\alpha \in X \cup \{\tau\}$, $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q'$ and $X\rho \in FT^e(P' \parallel_S Q')$. It follows that $P \xrightarrow{\tau} P'$ and $Q \xrightarrow{\tau} P'$, so $\text{STAB} \in FT^{r*}(P)$ and $\text{STAB} \in FT^{r*}(Q)$. By the proof of Proposition A.1 there is a valid decomposition of $X\rho$

such that $X_L\rho_L \in FT^*(P')$ and $X_R\rho_R \in FT^*(Q')$. Depending on which rule of Table 1 derived $P \parallel_S Q \xrightarrow{t} P' \parallel_S Q'$,

- (i) either $P \xrightarrow{t} P'$ and $Q' = Q$,
- (ii) or $P' = P$ and $Q \xrightarrow{t} Q'$.

For reasons of symmetry, I may assume the former.

Let $X'_L := \{a \in X_L \mid P \xrightarrow{a} \}$ and $X'_R := (X \setminus S) \cup \{a \in X \cap S \mid Q \xrightarrow{a} \}$. Using the fourth rule of Table 3, $X'_L X_L \rho_L \in FT^*(P')$, so by Definition 6.11 $\sigma_L := tX'_L X_L \rho_L \in FT^{r*}(P)$. Moreover, by the fourth rule of Table 3, $\sigma_R := X'_R X_R \rho_R \in FT^*(Q) \subseteq FT^{r*}(Q)$, using that $Q \xrightarrow{\alpha} \}$ for all $\alpha \in (X \setminus S) \cup \{\tau\}$. It remains to show the validity of the decomposition of σ into σ_L and σ_R . This proceeds exactly as in the corresponding case in the proof of Proposition A.1 (the case $\sigma = XX\rho$ in direction “ \subseteq ”).

- Let $\sigma = \text{POSTSTAB}$. Then $P \parallel_S Q$ has a nonempty path of τ -transitions to a stable state. This path projects to paths of τ -transitions from P and from Q to stable states, and at least one of them must be nonempty. It follows that either $\text{POSTSTAB} \in FT^{r*}(P) \wedge \text{POSTSTAB} \in FT^{r*}(Q)$, OR $\text{POSTSTAB} \in FT^{r*}(P) \wedge \text{STAB} \in FT^{r*}(Q)$, OR $\text{STAB} \in FT^{r*}(P) \wedge \text{POSTSTAB} \in FT^{r*}(Q)$. \square

Proposition A.5. Let $\text{STAB} \neq \sigma \neq \text{POSTSTAB}$. Then

$$\sigma \in FT^{r*}(\tau_I(P)) \Leftrightarrow \exists \rho \in FT^{r*}(P). \tau_I(\rho) = \sigma \cup I.$$

Moreover, $\text{STAB} \in FT^{r*}(\tau_I(P)) \Leftrightarrow \text{STAB} \in FT^{r*}(P) \wedge \mathcal{I}(P) \cap I = \emptyset$,

and $\text{POSTSTAB} \in FT^{r*}(\tau_I(P)) \Leftrightarrow (\exists c_0 c_1 \dots c_n I \top \in FT^{r*}(P) \text{ where } n \geq 0 \text{ and all } c_i \in I) \vee (\text{POSTSTAB} \in FT^{r*}(P) \wedge I \top \in FT^{r*}(P))$.

Proof. Let $\text{STAB} \neq \sigma \neq \text{POSTSTAB}$. Since each state R reachable from $\tau_I(P)$ satisfies $R \xrightarrow{\alpha} \}$ for all $\alpha \in I$, a trivial induction shows that

$$\sigma \in FT^{r*}(\tau_I(P)) \Leftrightarrow \sigma \cup I \in FT^{r*}(\tau_I(P)). \quad (*)$$

“ \Leftarrow ”: Let $\rho \in FT^{r*}(P)$ and ρ survives abstraction from I . By Condition (i) in the definition of abstraction survival, $\tau_I(\rho)$ is of the form $\sigma \cup I$. In view of (*), it suffices to show that $\tau_I(\rho) \in FT^{r*}(\tau_I(P))$.

Suppose $\rho = \top$, $\rho = a\eta$ or $\rho = X\eta$. Then $\rho \in FT^*(P)$. So $\tau_I(\rho) \in FT^*(\tau_I(P)) \subseteq FT^{r*}(\tau_I(P))$ by the proof of Proposition A.2.

Suppose $\rho = tX\eta$. Then $P \xrightarrow{\alpha} \}$ for all $\alpha \in X \cup \{\tau\}$, $P \xrightarrow{t} P'$ and $X\eta \in FT^*(P')$, by Definition 6.11. So $\tau_I(X\eta) \in FT^*(\tau_I(P'))$ by the proof of Proposition A.2. Moreover, $\tau_I(P) \xrightarrow{t} \tau_I(P')$ and $\tau_I(P) \xrightarrow{\alpha} \}$ for all $\alpha \in X \cup \{\tau\}$, using that $I \subseteq X$. Note that $\tau_I(X\eta)$ has the form $X\zeta$, although not necessarily with $\zeta = \tau_I(\eta)$. Definition 6.11 yields $\tau_I(\rho) = t\tau_I(X\eta) \in FT^{r*}(\tau_I(P))$.

“ \Rightarrow ”: Let $\sigma \in FT^{r*}(\tau_I(P))$; by (*) I may assume that $\sigma = \sigma \cup I$. I have to find a $\rho \in FT^{r*}(P)$ such that $\tau_I(\rho) = \sigma$.

Suppose $\sigma = \top$, $\sigma = a\eta$ or $\sigma = X\eta$. Then $\sigma \in FT^*(\tau_I(P))$. So by the proof of Proposition A.2 there is a $\rho \in FT^*(P) \subseteq FT^{r*}(P)$ such that $\tau_I(\rho) = \sigma$.

Suppose $\sigma = tX\zeta$. Then $\tau_I(P) \xrightarrow{\alpha} \}$ for all $\alpha \in X \cup \{\tau\}$, $\tau_I(P) \xrightarrow{t} \tau_I(P')$ and $X\zeta \in FT^*(\tau_I(P'))$, by Definition 6.11. Hence $P \xrightarrow{\alpha} \}$ for all $\alpha \in X \cup \{\tau\}$, and $P \xrightarrow{t} P'$. By the proof of Proposition A.2 there is a $X\eta \in FT^*(P')$ such that $\tau_I(X\eta) = X\zeta$. So $\rho := tX\eta \in FT^{r*}(P)$ by Definition 6.11, and $\tau_I(\rho) = \sigma$.

The second statement of Proposition A.5 is trivial. Now consider the third.

“ \Leftarrow ”: Suppose $\exists c_0 c_1 \dots c_n I \top \in FT^{r*}(P)$ where $n \geq 0$ and all $c_i \in I$. Then P has a

nonempty path, all of which transitions are labelled τ or $c_i \in I$, ending in a state P' satisfying $P' \xrightarrow{\alpha}$ for all $\alpha \in I \cup \{\tau\}$. Consequently, $\tau_I(P)$ has a nonempty part, all of which transitions are labelled τ , ending in a state $\tau(P')$ satisfying $\tau_I(P') \xrightarrow{\tau}$. Consequently, $\tau_I(P) \xrightarrow{\tau}$ and $\emptyset \top \in FT^*(\tau_I(P))$. Thus $\text{POSTSTAB} \in FT^{r*}(\tau_I(P))$.

Now suppose $\text{POSTSTAB} \in FT^{r*}(P) \wedge I \top \in FT^{r*}(P)$. Then $P \xrightarrow{\tau}$ and P has a path, all of which transitions are labelled τ , ending in a state P' satisfying $P' \xrightarrow{\alpha}$ for all $\alpha \in I \cup \{\tau\}$. This path must be nonempty. Again it follows that $\text{POSTSTAB} \in FT^{r*}(\tau_I(P))$.

“ \Leftarrow ”: Suppose $\text{POSTSTAB} \in FT^{r*}(\tau_I(P))$. Then $\tau_I(P)$ has a nonempty part, all of which transitions are labelled τ , ending in a state $\tau(P')$ satisfying $\tau_I(P') \xrightarrow{\tau}$. Consequently, P has a nonempty path, all of which transitions are labelled τ or $c \in I$, ending in a state P' satisfying $P' \xrightarrow{\alpha}$ for all $\alpha \in I \cup \{\tau\}$. In case on this path some transitions are labelled $c \in I$, one obtains $\exists c_0 c_1 \dots c_n I \top \in FT^{r*}(P)$ where $n \geq 0$ and all $c_i \in I$. In case there are no such transitions, $\text{POSTSTAB} \in FT^{r*}(P)$ and $I \top \in FT^{r*}(P)$. \square

Proposition A.6. $\sigma \in FT^{r*}(\mathcal{R}(P)) \Leftrightarrow \exists \rho \in FT^{r*}(P). \rho \in \mathcal{R}^{-1}(\sigma)$.

Proof. “ \Leftarrow ”: Let $\rho \in FT^{r*}(P)$, and let σ satisfy $\rho \in \mathcal{R}^{-1}(\sigma)$. I have to show that $\sigma \in FT^{r*}(\mathcal{R}(P))$.

Let $\rho = \top$, $\rho = a\eta$ or $\rho = X\eta$. Then $\rho \in FT^*(P)$, so by Proposition A.3 $\sigma \in FT^*(\mathcal{R}(P)) \subseteq FT^{r*}(\mathcal{R}(P))$.

Let $\rho = tY\eta$. Then $P \xrightarrow{\alpha}$ for all $\alpha \in Y \cup \{\tau\}$, $P \xrightarrow{t} P'$ and $Y\eta \in FT^*(P')$. Moreover, $\sigma = tX\zeta$ with $Y = \mathcal{R}^{-1}(X)$ and $Y\eta \in \mathcal{R}^{-1}(X\zeta)$. Consequently $\mathcal{R}(P) \xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$, $\mathcal{R}(P) \xrightarrow{t} \mathcal{R}(P')$ and $X\zeta \in FT^*(\mathcal{R}(P'))$ by Proposition A.3. It follows that $\sigma \in FT^{r*}(\mathcal{R}(P))$.

Let $\rho = \text{STAB}$. Then $\text{stable}(P)$, so $\text{stable}(\mathcal{R}(P))$ and $\sigma = \text{STAB} \in FT^{r*}(\mathcal{R}(P))$.

Let $\rho = \text{POSTSTAB}$. Then $P \xrightarrow{\tau}$ and $\emptyset \top \in FT^*(P)$. So $\mathcal{R}(P) \xrightarrow{\tau}$ and $\emptyset \top \in FT^*(\mathcal{R}(P))$. Therefore $\sigma = \text{POSTSTAB} \in FT^{r*}(\mathcal{R}(P))$.

“ \Rightarrow ”: Let $\sigma \in FT^{r*}(\mathcal{R}(P))$. I have to find a $\rho \in FT^{r*}(P)$ with $\rho \in \mathcal{R}^{-1}(\sigma)$.

Let $\sigma = \top$, $\sigma = a\eta$ or $\sigma = X\eta$. Then $\sigma \in FT^*(\mathcal{R}(P))$, so by Proposition A.3 there is a $\rho \in FT^*(P) \subseteq FT^{r*}(P)$ with $\rho \in \mathcal{R}^{-1}(\sigma)$.

Let $\sigma = tX\zeta$. Then $\mathcal{R}(P) \xrightarrow{\alpha}$ for all $\alpha \in X \cup \{\tau\}$, $\mathcal{R}(P) \xrightarrow{t} \mathcal{R}(P')$ and $X\zeta \in FT^*(\mathcal{R}(P'))$. By Proposition A.3 there is a $Y\eta \in FT^*(P')$ with $Y\eta \in \mathcal{R}^{-1}(X\zeta)$. In particular $Y = \mathcal{R}^{-1}(X)$. Therefore, $P \xrightarrow{\alpha}$ for all $\alpha \in Y \cup \{\tau\}$. Moreover, $P \xrightarrow{t} P'$. Take $\rho := tY\eta$. Then $\rho \in FT^{r*}(P)$ and $\rho \in \mathcal{R}^{-1}(\sigma)$.

Let $\sigma = \text{STAB}$. Then $\text{stable}(\mathcal{R}(P))$, so $\text{stable}(P)$ and $\rho = \text{STAB} \in FT^{r*}(P)$.

Let $\sigma = \text{POSTSTAB}$. Then $\mathcal{R}(P) \xrightarrow{\tau}$ and $\emptyset \top \in FT^*(\mathcal{R}(P))$. So $P \xrightarrow{\tau}$ and $\emptyset \top \in FT^*(P)$. Therefore $\rho = \text{POSTSTAB} \in FT^{r*}(P)$. \square