Gambling for Leadership: Verification of Root Contention in IEEE 1394

M.I.A. Stoelinga

Computing Science Institute/

CSI-R9904 March 1999

Computing Science Institute Nijmegen
Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Gambling for Leadership: Verification of Root Contention in IEEE 1394

Mariëlle Stoelinga
Computing Science Institute
University of Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
`marielle@cs.kun.nl`

Abstract

This report presents a formal verification of root contention in IEEE 1394. Root contention is a simple yet realistic protocol that solves leader election for two processes, using coin flips.

The verification has been carried out in the probabilistic automaton model of [Seg95]. Correctness of an implementation automaton w.r.t. a specification automaton is defined as so-called fair trace distribution inclusion.

Similarly to the non-probabilistic setting, probabilistic simulation relations provide a technique for proving trace distribution inclusion. Then additional reasoning proves fair trace distribution inclusion. Our verification follows this strategy. We introduce two simulation relations, viz. probabilistic step refinements and probabilistic hyperstep refinements. These notions are not very complicated and are sufficient in our verification. They are special cases of the simulation relations from [Seg95].

The verification also involves non-probabilistic techniques. For instance, the invariants have been checked with the model checker SMV.

Key words and phases: IEEE 1394, leader election algorithms, communication protocols, probabilistic and distributed algorithms, formal verification, SMV, probabilistic automata.
AMS Subject Classification: 68Q10, 68Q22, 68Q60, 68Q75.
CR Subject Classification: C.2.2, C.3, F.1.2, K.1.

1 Introduction

Recently, the analysis of probabilistic, distributed algorithms and protocols has gained new attention. Various methods and formalisms have been extended with probabilism and quite some case studies have been carried out using these formalisms, c.f. [Agg94, PSL97].

This report verifies a small sub-protocol of the IEEE 1394 protocol, called root contention. The IEEE 1394 high performance serial bus [IEE96, IEE98], also called FireWire, has been developed for interconnecting computer and consumer equipment, such as PCs, VCRs and CD players. The bus is "hot pluggable," which means that equipment can be added and removed at any time, and allows quick, reliable and inexpensive high-bandwidth transfer. This protocol probably becomes the new standard for connecting digital multimedia equipment. Various parts of IEEE have been verified formally, see [DGRV97, Sha98, Lut97]. However, as far as we know, root contention has not.

Root contention in IEEE 1394 is a simple but realistic protocol that involves both real-time and probabilistic aspects. As a first approach we abstract from the real-time aspects and model time passage by discrete actions. The verification in this report is carried out in the probabilistic automaton model from [Seg95]. Following the tradition, the correctness of the protocol is proven by establishing a probabilistic simulation relation between the implementation and the specification, both probabilistic automata.

The probabilistic simulation relations from [Seg95] are rather complex. In order to simplify the simulation proofs, this report introduces the notions of probabilistic step refinement and of probabilistic hyperstep refinement. These are rather obvious extensions of non-probabilistic simulations. However, it has taken some effort to find a formulation which makes the definition fit into the standard pattern of simulation relations.

The strategy followed in the simulation proof is the following. Given the protocol specification P and the abstract specification S, we define two intermediate automata, I and A. The automaton I abstracts from the message passing in P but keeps all probabilistic choices. The automaton A combines the probabilistic choices in I. Then we can separate our concerns. The simulation between P and I, a probabilistic step refinement, is easy from probabilistic point of view and mainly deals with traditional, non-probabilistic techniques like proving invariants. Therefore, existing tools for the analysis of non-probabilistic systems can be used. We have checked the invariants with the tool SMV. The probabilistic analysis is concentrated in the simulation relations between between I and A and between A and S. Since these automata are small, this is not so difficult any more.

Furthermore, we have proven inclusion of fair behaviour – technically: fair trace distribution inclusion – between the protocol specification P and the high level specification S. Therefore, we have developed a result that reduces reasoning about fair probabilistic executions to reasoning about fairness of (non-probabilistic) executions.

This report is organized as follows. Section 2 recalls some probability theory and Section 3 introduces the probabilistic automaton model. Section 4 provides automaton models of the implementation and the specifications of the root contention protocol. Then Sections 6 and 7 proves the simulation relations. Finally section 8 presents the conclusions and some topics for future research.

2 Probability theory

This section recalls some probability theory. We denote the set $\mathbb{R} \cup \{\infty\}$ by $[0, \infty]$.

Summation over index sets

Definition 2.1 Let $\mathcal{I}$ be a(n index) set and $x_i \in [0, \infty]$ for all $i \in \mathcal{I}$. Define $\sum_{i\in\mathcal{I}} x_i$ by

1. $\sum_{i\in\emptyset} x_i \stackrel{\mathrm{d}}{=} 0$;
2. $\sum_{i\in\mathcal{I}} x_i \stackrel{\mathrm{d}}{=} x_{i_1} + x_{i_2} + x_{i_3} + \cdots + x_{i_n}$, if $\mathcal{I}$ is finite, nonempty and $\mathcal{I} = \{i_1, i_2, i_3, \ldots, i_n\}$;
3. $\sum_{i\in\mathcal{I}} x_i \stackrel{\mathrm{d}}{=} \mathrm{supr}\{\sum_{i\in\mathcal{J}} x_i \mid \mathcal{J} \subseteq \mathcal{I} \text{ is finite }\}$, if $\mathcal{I}$ is infinite.

Here $\mathrm{supr}\, X$ denotes the supremum, i.e. the smallest upper bound, of the set X. Notice that that $\sum_{i\in\mathbb{N}} x_i = \sum_{i=0}^{\infty} x_i$ because the summation order is irrelevant, due to the fact that $x_i \geq 0$.

Fact 2.2 *Let $\mathcal{I}, \mathcal{I}_1, \mathcal{I}_2, \ldots$ be sets, $\mathcal{I} = \cup_{n\in\mathbb{N}} \mathcal{I}_n$ and $x_i \in [0, \infty]$ for all $i \in \mathcal{I}$.*

1. *If* $\mathcal{I}_1 \subseteq \mathcal{I}_2$, *then* $\sum_{i\in\mathcal{I}_1} x_i \leq \sum_{i\in\mathcal{I}_2} x_i$;
2. *If* $\mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \mathcal{I}_3, \dots$, *then* $\sum_{i\in\mathcal{I}} x_i = \lim_{n\to\infty} \sum_{i\in\mathcal{I}_n} x_i$;
3. *If* $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, \dots$ *are pairwise disjoint, then* $\sum_{i\in\mathcal{I}} x_i = \sum_{n\in\mathbb{N}} \sum_{i\in\mathcal{I}_n} x_i$.

Fact 2.3 *Let* $\mathcal{I}$ *be a set,* $x_i \in [0,\infty]$ *for all* $i \in \mathcal{I}$ *and* $\sum_{i\in\mathcal{I}} x_i < \infty$. *Then the set* $\{x_i \in [0,\infty] \mid x_i > \varepsilon, i \in \mathcal{I}\}$ *is finite for every* $\varepsilon > 0$ *and* $\{x_i \in [0,\infty] \mid x_i > 0, i \in \mathcal{I}\}$ *is countable.*

Probability spaces and distributions

Definition 2.4 A *probability space* is a triple $(\Omega, \mathcal{F}, \mathbf{P})$, where

- Ω is a set, called the *sample space*,
- $\mathcal{F} \subseteq 2^\Omega$ is *σ-field*, i.e. a collection of subsets of Ω which is closed under countable[1] union and complement and which contains Ω,
- $\mathbf{P} : \mathcal{F} \to [0,1]$ is a *probability measure* on $\mathcal{F}$, which means that $\mathbf{P}[\Omega] = 1$ and for any countable collection $\{C_i\}_i$ of pairwise disjoint subsets in $\mathcal{F}$ we have $\mathbf{P}[\cup_i C_i] = \sum_i \mathbf{P}[C_i]$.

If $\{x\} \in \mathcal{F}$ then we write $\mathbf{P}[x]$ for $\mathbf{P}[\{x\}]$. If $\mathcal{P}$ is a probability space, we denote its sample space by $\Omega_\mathcal{P}$, its σ-field by $\mathcal{F}_\mathcal{P}$ and its probability measure by $\mathbf{P}_\mathcal{P}$.

Definition 2.5 A *probability distribution (function)* over a set X is a function $\mu : X \to [0,1]$ such that

$$\sum_{x\in X} \mu(x) = 1.$$

Define the *support of* μ by $\mathrm{supp}(\mu) \overset{\mathrm{d}}{=} \{\mathrm{x} \in \mathrm{X} \mid \mu(\mathrm{x}) > 0\}$. By fact 2.3 this is a countable set. We denote the set of all probability distributions over X by $\Pi(X)$.

Remark 2.6 A probability space $(\Omega, \mathcal{F}, \mathbf{P})$ is *discrete* if $\mathcal{F} = 2^\Omega$ and $\mathbf{P}[C] = \sum_{x\in C} \mathbf{P}[x]$, for all $C \subseteq \Omega$. A discrete probability space $(\Omega, \mathcal{F}, \mathbf{P})$ determines a unique probability distribution $f : \Omega \to [0,1]$ by $f(x) = \mathbf{P}[x]$. Conversely, a probability distribution f determines a unique discrete probability space: take $\Omega = X$, $\mathcal{F} = 2^\Omega$ and $\mathbf{P}[C] = \sum_{x\in C} f(x)$ for all $C \subseteq \Omega$.

We denote a (probability distribution) function f on a countable domain by enumerating it as a set of pairs. So, if $\mathrm{Dom}(f) = \{x_1, x_2 \dots\}$ then denote f by $\{x_1 \mapsto f(x_1), x_2 \mapsto f(x_2) \dots\}$. If the domain of f is known, then we often leave out elements of probability zero. For instance, the probability distribution assigning probability one to an element $x \in X$ is denoted by $\{x \mapsto 1\}$, irrespective of X. Such distribution is called the *Dirac distribution* over x. The *uniform distribution* over a finite set, say $\{x_1, \dots, x_n\}$, is given by $\{x_1 \mapsto \frac{1}{n}, \dots, x_n \mapsto \frac{1}{n}\}$.

Definition 2.7 Let X and Y be sets. If $\mu \in \Pi(X)$ and $\nu \in \Pi(Y)$ then the *product* $\mu \times \nu : X \times Y \to [0,1]$ defined by

$$(\mu \times \nu)(x,y) \overset{\mathrm{d}}{=} \mu(x) \cdot \nu(y)$$

is a probability distribution over $X \times Y$.

[1] In our terminology, countable objects include finite ones.

The operation above can also be defined for general probability spaces by a more complex definition, see [Seg95].

Definition 2.8 Let $\mathcal{P} = (\Omega, \mathcal{F}, \mathbf{P})$ be a probability distribution, Ω' a set and $f : \Omega \to \Omega'$ be a function. Define the *image space of* $\mathcal{P}$ *under* f, notation $f_*(\mathcal{P})$, as the triple consisting of

1. $f(\Omega)$;
2. $\{X \in 2^{f(\Omega)} \mid f^{-1}(X) \in \Omega_{\mathcal{P}}\}$;
3. $\mathbf{P} \circ f^{-1}$.

It is not difficult to show that $f_*(\mathcal{P})$ is a probability space indeed. If $\mathcal{P}$ is discrete, then so is $f_*(\mathcal{P})$. We write $f_*(\mu)$ to indicate the distribution function associated to $f_*(\mathcal{P}_\mu)$, where $\mathcal{P}_\mu$ is the distribution function associated to μ.

Lemma 2.9 *The operations* $_*$ *and* $\circ$ *commute, that is* $(g \circ f)_* = g_* \circ f_*$ *for all functions* $f : X \to Y$ *and* $g : Y \to Z$.

The below definition gives two formulations for choosing a probability space with a certain probability.

Definition 2.10

1. If $\{p_i\}_{i \in \mathcal{I}}$ is a collection of reals in $[0, 1]$ such that $\sum_i p_i = 1$ and $\mu_i \in \Pi(X_i)$ for all $i \in \mathcal{I}$, then $x \mapsto \sum_i p_i \cdot \mu_i(x)$ is a probability space over $\bigcup_i X_i$.
2. Let X be a set and $\mu \in \Pi(\Pi(X))$. Then μ induces a probability distribution μ_F on X, given by
$$\mu_F(x) = \sum_{\nu \in \Pi(X)} \mu(\nu) \cdot \nu(x).$$
for all $x \in X$. We call μ_F *the fold of* μ. The idea is that μ_F chooses between all probability distributions the distribution ν with probability $\mu(\nu)$ and then elements x of X with probability $\nu(x)$.

Example 2.11 Let x, y, z denote different elements.

1. If $f : \{x, y\} \to Y$ for some set Y, then
$$f_*(\{x \mapsto \tfrac{1}{3}, y \mapsto \tfrac{2}{3}\}) = \begin{cases} \{f(x) \mapsto 1\} & \text{if } f(x) = f(y), \\ \{f(x) \mapsto \tfrac{1}{3}, f(y) \mapsto \tfrac{2}{3}\} & \text{otherwise;} \end{cases}$$
2. $\{x \mapsto \frac{1}{3}, y \mapsto \frac{2}{3}\} \times \{x \mapsto \frac{1}{2}, z \mapsto \frac{1}{2}\} = \{(x,x) \mapsto \frac{1}{6}, (x,z) \mapsto \frac{1}{6}, (y,x) \mapsto \frac{1}{3}, (y,z) \mapsto \frac{1}{3}\}$;
3. If $\mu\{x \mapsto \frac{1}{3}, y \mapsto \frac{2}{3}\} = \frac{1}{2}$ and $\mu\{x \mapsto \frac{1}{2}, z \mapsto \frac{1}{2}\} = \frac{1}{2}$, then
$$\begin{aligned} \mu_F =& \tfrac{1}{2} \cdot \{x \mapsto \tfrac{1}{3}, y \mapsto \tfrac{2}{3}\} + \tfrac{1}{2} \cdot \{x \mapsto \tfrac{1}{2}, z \mapsto \tfrac{1}{2}\} \\ =& \{x \mapsto \tfrac{5}{12}, y \mapsto \tfrac{1}{3}, z \mapsto \tfrac{1}{4}\}. \end{aligned}$$

3 Probabilistic probabilistic automata

This section introduces probabilistic automata and their behaviour. Most of the concepts in 3.1 and 3.2 have been taken over from [Seg95]. However, we have reformulated their definitions to a form that we believe is more readable.

3.1 The model

Definition 3.1 A *probabilistic automaton* A consists of five components:

1. a set $states_A$ of *states*;
2. a nonempty set $start_A \subseteq states_A$ of *start states*;
3. an *action signature* $sig_A = (ext_A, int_A)$, consisting of *external* and *internal actions* respectively. Then define the set of *actions* as $act_A \overset{\mathrm{d}}{=} ext_A \cup int_A$;
4. a *transition relation* $trans_A \subseteq states_A \times act_A \times \Pi(states_A)$. We write $s \xrightarrow{a}_A \mu$ if $(s, a, \mu) \in trans_A$;
5. a *task partition* $tasks_A$, which is a partial equivalence relation over act_A with countably many equivalence classes.

Sometimes, a more general definition of probabilistic automata is given by having $trans_A \subseteq states_A \times \Pi(act_A \times states_A)$. In this context the probabilistic automata from the definition are called *simple* probabilistic automata.

Definition 3.2 Let A be a probabilistic automaton. The automaton A^-, the non-probabilistic variant of A, which behaves like A but discards all probabilistic information, is defined by

1. $states_{A^-} \overset{\mathrm{d}}{=} states_A$;
2. $start_{A^-} \overset{\mathrm{d}}{=} start_A$;
3. $sig_{A^-} \overset{\mathrm{d}}{=} sig_A$;
4. $trans_{A^-} \overset{\mathrm{d}}{=} \{s \xrightarrow{a}_{A^-} s' \mid \exists \mu \in \Pi(states_A)[s \xrightarrow{a}_A \mu \wedge \mu(s') > 0\}$;
5. $tasks_{A^-} \overset{\mathrm{d}}{=} tasks_A$.

Definition 3.3 For a probabilistic automaton A define $reach_A$, the set of *reachable states of* A, by $reach_A \overset{\mathrm{d}}{=} reach_{A^-}$.

Definition 3.4 Let A be a probabilistic automaton and $X \subseteq ext_A$. The *restriction of* A *to* X, notation $A\lceil_X$, is defined by

1. $states_{A\lceil_X} \overset{\mathrm{d}}{=} states_A$;
2. $start_{A\lceil_X} \overset{\mathrm{d}}{=} start_A$;
3. $sig_{A\lceil_X} \overset{\mathrm{d}}{=} (X, act_A \setminus X)$;
4. $trans_{A\lceil_X} \overset{\mathrm{d}}{=} trans_A$;
5. $tasks_{A\lceil_X} \overset{\mathrm{d}}{=} tasks_A$.

Fact 3.5 *For all probabilistic automata A and $X \subseteq ext_A$ we have $A^-\lceil_X = (A\lceil_X)^-$.*

Definition 3.6 Let A_1 and A_2 be two probabilistic automata.

1. Then A_1 and A_2 are *compatible* if

   (a) $int_{A_1} \cap act_{A_2} = \emptyset$, $act_{A_1} \cap int_{A_2} = \emptyset$ and

   (b) for all $C_1 \in \mathit{tasks}_{A_1}$, $C_2 \in \mathit{tasks}_{A_2}$, either $C_1 \cap C_2 = \emptyset$ or $C_1 = C_2$.

2. For A_1 and A_2 compatible define the *parallel composition* of A_1 and A_2, notation $A_1 \parallel A_2$, by

   (a) $\mathit{states}_{A_1 \parallel A_2} = \mathit{states}_{A_1} \times \mathit{states}_{A_2}$;

   (b) $\mathit{start}_{A_1 \parallel A_2} = \mathit{start}_{A_1} \times \mathit{start}_{A_2}$;

   (c) $\mathit{sig}_{A_1 \parallel A_2} = (\mathit{ext}_{A_1} \cup \mathit{ext}_{A_2}, \mathit{int}_{A_1} \cup \mathit{int}_{A_2})$;

   (d) $\mathit{trans}_{A_1 \parallel A_2}$ is the set of triples $((s_1, s_2), a, \mu_1 \times \mu_2)$ such that for $i = 1, 2$, if $a \in \mathit{act}_{A_i}$ then $(s_i, a, \mu_i) \in \mathit{trans}_{A_i}$, otherwise $\mu_i = \{s_i \mapsto 1\}$;

   (e) $\mathit{tasks}_{A_1 \parallel A_2} = \mathit{tasks}_{A_1} \cup \mathit{tasks}_{A_2}$.

Informally, two probabilistic automata synchronize on their common actions and evolve independently on others. Whenever synchronization occurs, the state reached is obtained by choosing a state independently for both probabilistic automata. See Remark 3.24 for a comment on the treatment of task partitions.

3.2 The behaviour of probabilistic automata

Definition 3.7 An *execution (execution fragment, trace)* of a probabilistic automaton A is an execution (execution fragment, trace) of A^-. The set of executions (execution fragments, traces) and finite executions (execution fragments, traces) of A are respectively denoted by $\mathit{execs}(A)(\mathit{frags}(A), \mathit{traces}(A))$ and by $\mathit{execs}^*(A)$ $(\mathit{frags}^*(A), \mathit{traces}^*(A))$.

An execution fragment is the result of resolving both probabilistic and nondeterministic choices. A *probabilistic* execution fragment only resolves the nondeterministic choices. In any state of the probabilistic execution fragment, we choose probabilistically between the enabled transitions. We choose δ to obtain a finite execution (with some probability). Each of these executions lead to a Markov chain, which allows us to compute the probabilities on the behaviour obtained by the specific choice of resolving the nondeterminism.

Definition 3.8 A *probabilistic execution fragment* E of a probabilistic automaton A consists of three components.

1. A state space $\mathit{states}_E \subseteq \mathit{frags}^*(A) \cup \mathit{frags}^*(A) \cdot \delta$;

2. A start state $\mathit{start}_E \in \mathit{states}_E \cap \mathit{frags}^*(A)$;

3. A transition function $\mathit{trans}_E : \mathit{states}_E \to \Pi(\mathit{states}_E)$ such that

   (a) for all $\alpha \in \mathit{states}_E \cap \mathit{frags}^*(A)$ there is a $\mu_\alpha \in \Pi(\mathit{trans}_A(\mathit{last}(\alpha)) \cup \{\delta\})$ satisfying

$$\mathit{trans}_E(\alpha)(\alpha\delta) = \mu_\alpha(\delta),$$
$$\mathit{trans}_E(\alpha)(\alpha a s) = \sum_{\nu : (\mathit{last}(\alpha), a, \nu) \in \mathit{trans}_A} \mu_\alpha(a, \nu) \cdot \nu(s)$$

   (b) and for all $\alpha \in \mathit{states}_E \cap \mathit{frags}^*(A) \cdot \delta$

$$\mathit{trans}_E(\alpha)(\alpha) = 1.$$

Here $trans_A(t)$ denotes $\{(a,\nu) \mid (t,a,\nu) \in trans_A\}$. Notice that $trans_E(\alpha)$ indeed is a probability distribution over $states_E$. The idea is that μ_α chooses probabilistically between all transitions starting from $last(\alpha)$ and δ. Furthermore, we require that every state in $states_E$ is reachable from $start_E$ via the relation $\{(\alpha,\alpha') \mid trans_E(\alpha)(\alpha') > 0\}$.

A *probabilistic execution* E is a probabilistic execution fragment such that $start_E \in start_A$. The set of probabilistic execution (fragments) is denoted by $pexecs(A)$ ($pfrags(A)$).

Fact 3.9 *1. A probabilistic execution fragment E is a Markov chain over $states_E$.*

2. The state space $states_E$ is countable, due to the reachability condition.

A Markov chain allows us to compute the probability of reaching a state. Hence, we can compute the probability on (certain) sets of execution fragments, given a probabilistic execution. The following definition associates a probability space to a probabilistic execution fragment.

Definition 3.10 The *associated probability space* $E = (\Omega_E, \mathcal{F}_E, \mathbf{P}_E)$ of a probabilistic execution fragment E is defined by

1. Ω_E is the limit closure of $states_E$ under the prefix order $\sqsubseteq$;
2. $\mathcal{F}_E$ is the smallest σ-field that subsumes the set $\{C_\alpha \mid \alpha \in states_E\}$. Here C_α is the *cone on* α, defined by $C_\alpha \overset{\mathrm{d}}{=} \{\beta \in \Omega_E \mid \alpha \sqsubseteq \beta\}$;
3. $\mathbf{P}_E$ is the unique measure on $\mathcal{F}_E$ such that $\mathbf{P}_E[C_\alpha]$ is the product of the probabilities on the (shortest) path starting in the start state of E and leading to α. So $\mathbf{P}_E[C_{s_0 a_1 s_1 a_2 \ldots a_n s_n}] = trans_E(s_0) \cdot trans_E(s_0 a_1 s_1) \cdots trans_E(s_0 a_1 s_1 \ldots a_n s_n)$.

The fact that $(\Omega_E, \mathcal{F}_E, \mathbf{P}_E)$ is well-defined follows from standard measure theory arguments, see in [Seg95] for a similar case and [Hal50] for a more general treatment. When no confusion arises, we denote both the probabilistic execution fragment and its associated probability space by E.

The following definition allows us to compute the probability on (certain) sets of traces given a probabilistic execution.

Definition 3.11 The *trace distribution* H of a probabilistic execution fragment E of A is the probability space given by

1. $\Omega_H = ext_A{}^* \cup ext_A{}^\infty$;
2. $\mathcal{F}_H$ is the smallest σ-field that subsumes the set $\{C_\alpha \mid \alpha \in ext^*(A)\}$. Now $C_\alpha \overset{\mathrm{d}}{=} \{\beta \in \Omega_H \mid \alpha \sqsubseteq \beta\}$;
3. $\mathbf{P}_H = \mathbf{P}_E \circ trace^{-1}$, that is $\mathbf{P}_H[X] = \mathbf{P}_E[trace^{-1}(X)]$ for all $X \in \mathcal{F}_H$.

The fact that $(\Omega_H, \mathcal{F}_H, \mathbf{P}_H)$ is well-defined follows from the fact that $X \in \mathcal{F}_H \implies trace^{-1}(X) \in \mathcal{F}_E$, see [PSL97] and [Hal50]. Notice that the probability space above is quite similar to the image space $trace_*(E)$, but that it may contain some more elements, which all have probability zero.

Notation 3.12 The set of trace distributions of A is denoted by $trdistr(A)$. If $trdistr(A) \subseteq trdistr(B)$ then we write $A \sqsubseteq_{\mathrm{TD}} B$. It is obvious that $\sqsubseteq_{\mathrm{TD}}$ is a partial order.

3.3 Step refinements and hyperstep refinements

This section introduces two new probabilistic simulation relations, viz. probabilistic step refinements and probabilistic hyperstep refinements. We show that these are sound for trace distribution inclusion.[2]

Probabilistic step refinements

Definition 3.13 Let A, B be two probabilistic automata with the same external actions. A *probabilistic step refinement* is a function $r : \mathit{states}_A \to \mathit{states}_B$ such that:

1. for all $s \in \mathit{start}_A, r(s) \in \mathit{start}_B$;
2. for all $s \in \mathit{reach}_A, a \in \mathit{act}_A, \mu \in \Pi(\mathit{states}_A)$, if $s \xrightarrow{a}_A \mu$ then either of the following conditions is met:
   (a) $r(s) \xrightarrow{a}_B r_*(\mu)$ or
   (b) $a \in \mathit{int}_A \wedge r(s) \xrightarrow{a'}_B r_*(\mu)$ for some $a' \in \mathit{int}_B$ or
   (c) $a \in \mathit{int}_A \wedge r_*(\mu) = \{r(s) \mapsto 1\}$.

We write $A \sqsubseteq_{\mathrm{PSR}} B$ if there is a probabilistic step refinement between A and B. Remark that the third condition is equivalent to $a \in \mathit{int}_A \wedge \forall s' \in \mathrm{supp}(\mu)[\mathrm{r}(\mathrm{s}') = \mathrm{r}(\mathrm{s})]$.

Example 3.14 The following diagrams illustrate three typical situations that may occur if r is a probabilistic step refinement from A to B. The transitions on the left are steps of the probabilistic automaton A, those on the right of B.

Condition 2a. and $r(t_1) \neq r(t_2) \neq r(t_3) \neq r(t_1)$:

$$\begin{tikzcd} & s \arrow[dl, "a"', "p"] \arrow[d, "q"] \arrow[dr, "1-p-q"] & \\ t_1 & t_2 & t_3 \end{tikzcd} \qquad \begin{tikzcd} & r(s) \arrow[dl, "a"', "p"] \arrow[d, "q"] \arrow[dr, "1-p-q"] & \\ r(t_1) & r(t_2) & r(t_3) \end{tikzcd}$$

Condition 2b., $a \in \mathit{int}_A$, $a' \in \mathit{int}_B$ and $r(t_1) = r(t_2) \neq r(t_3)$:

$$\begin{tikzcd} & s \arrow[dl, "a"', "p"] \arrow[d, "q"] \arrow[dr, "1-p-q"] & \\ t_1 & t_2 & t_3 \end{tikzcd} \qquad \begin{tikzcd} r(s) \arrow[d, "a'"', "p+q"] \arrow[dr, "1-p-q"] & \\ r(t_1) = r(t_2) & r(t_3) \end{tikzcd}$$

Condition 2c. and $a \in \mathit{int}_A$:

$$\begin{tikzcd} & s \arrow[dl, "a"', "p"] \arrow[d, "q"] \arrow[dr, "1-p-q"] & \\ t_1 & t_2 & t_3 \end{tikzcd} \qquad r(s) = r(t_1) = r(t_2) = r(t_3)$$

[2]Concerning the terminology, we use the prefix *hyper* to refer to a probabilistic simulation of type $\mathit{states}_A \times \Pi(\mathit{states}_B)$. Our simulation relations abstract from internal computation, so one could could add the adjective "weak" to their names.

Fact 3.15 *If r is a probabilistic step refinement from A to B then it is a step refinement from A^- to B^- and a probabilistic step refinement from $A\lceil_X$ to $B\lceil_X$ for all $X \subseteq ext_A$.*

Theorem 3.16 *The relation $\sqsubseteq_{\text{PSR}}$ is a preorder, i.e. it is reflexive and transitive.*

PROOF: Reflexivity is obvious. For transitivity, suppose r is a probabilistic refinement from A to B and u is one from B to C. We claim that $u \circ r$ is a probabilistic step refinement from A to C. In fact the proof, which is spelled out below, is similar to the proof of transitivity for non-probabilistic step refinements. We only need that $u_* \circ r_* = (u \circ r)_*$.
condition 1 Obviously, if $s \in start_A$, then $(u \circ r)(s) \in start_C$.
condition 2 Assume $s \xrightarrow{a}_A \mu$. As r is a probabilistic step refinement from A to B we have the following cases.

1. $a \in int_A$ and $r_*(\mu) = \{r(s) \mapsto 1\}$. Then $(u_* \circ r_*)(\mu) = u_*(\{r(s) \mapsto 1\}) = u_*(\{r(s) \mapsto 1\}) = \{u(r(s)) \mapsto 1\}$;

2. $r(s) \xrightarrow{a}_B r_*(\mu)$. As u is a probabilistic step refinement we are in either of the following cases:

   (a) $a \in int_B$ and $u_*(r_*(\mu)) = \{u(r(s)) \mapsto 1\}$. Then $a \in int_A$ by $ext_A = ext_B$ and $(u \circ r)_*(\mu) = \{u(r(s)) \mapsto 1\}$ by $u_* \circ r_* = u \circ r$. Therefore, $u \circ r$ is a probabilistic step refinement from A to C;.

   (b) $u(r(s)) \xrightarrow{a}_C u_*(r_*(\mu))$. Then $(u \circ r)(s) \xrightarrow{a}_C (u \circ r)_*(\mu))$ by $(u \circ r)_* = u_* \circ r_*$ and therefore $u \circ r$ is a probabilistic step refinement from A to C;

   (c) $u(r(s)) \xrightarrow{a''}_C u_*(r_*(\mu))$ and $a' \in int_B$ and $a'' \in int_C$: similarly to the case above;

3. $r(s) \xrightarrow{a'}_B r_*(\mu)$, $a \in int_A$ and $a' \in int_B$: similarly to the above case.

Therefore, $u \circ r$ is a probabilistic step refinement from A to C. □

Probabilistic hyperstep refinements

Definition 3.17 Let X, Y be sets and $R \subseteq X \times \Pi(Y)$. Define the *lifting of R* to $\Pi(X) \times \Pi(Y)$, notation R_{**}, as the set of pairs $(\mu, \nu) \in \Pi(X) \times \Pi(Y)$ such that there exists a choice function $r : \text{supp}(\mu) \to \Pi(\text{Y})$ for R, (i.e. a function such that $(x, r(x)) \in R$ for all $x \in \text{supp}(\mu)$) satisfying

$$\sum_{x \in \text{supp}(\mu)} \mu(x) \cdot r(x) = \nu.$$

The idea is that we can obtain ν by choosing the probability distribution $r(x)$, which should be related to x, with probability $\mu(x)$. The above sum is a probability distribution by Definition 2.10.

Fact 3.18 *If R is a function – considered as a set of pairs – then so is R_{**} and for all $\mu \in \Pi(X)$*

$$R_{**}(\mu) = \sum_{x \in X} \mu(x) \cdot R(x) = \{R(x) \mapsto \mu(x) \mid x \in X\}_F.$$

Example 3.19 Given a probabilistic automaton A and an action $a \in act_A$, we can lift the relation $\xrightarrow{a}$ over $states_A \times \Pi(states_A)$ to the relation $\xrightarrow{a}_{**}$ over $\Pi(states_A) \times \Pi(states_A)$. For instance, if $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$ and $s_1 \neq s_2$, then

$$\{s_1 \mapsto \tfrac{1}{3}, s_2 \mapsto \tfrac{2}{3}\} \xrightarrow{a}_{**} \tfrac{1}{3} \cdot \mu_1 + \tfrac{2}{3} \cdot \mu_2.$$

Intuitively, if $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$ and the probability on being in s_1 is $\frac{1}{3}$ and on being in s_2 is $\frac{2}{3}$, then we choose the next state according to μ_1 with probability $\frac{1}{3}$ and according to μ_2 with probability $\frac{2}{3}$. If there is another a–transition, say $s_2 \xrightarrow{a} \nu$, then we can also choose the next state according to s_1 with probability $\frac{1}{3}$ and according to ν with probability $\frac{2}{3}$. Hence

$$\{s_1 \mapsto \tfrac{1}{3}, s_2 \mapsto \tfrac{2}{3}\} \xrightarrow{a}_{**} \tfrac{1}{3} \cdot \mu_1 + \tfrac{2}{3} \cdot \nu.$$

We do *not* have

$$\{s_1 \mapsto \tfrac{1}{3}, s_2 \mapsto \tfrac{2}{3}\} \xrightarrow{a}_{**} \tfrac{1}{3} \cdot \mu_1 + \tfrac{1}{3} \cdot \mu_2 + \tfrac{1}{3} \cdot \nu.$$

Furthermore, if the sequence $s_1, s_2, \ldots$ is such that $s_i \xrightarrow{a}_{**} \mu_i$ and $s_i \neq s_j$ for $i \neq j$, then we have that

$$\{s_i \mapsto \tfrac{1}{2^i} \mid i \in \mathbb{N}\} \xrightarrow{a}_{**} \sum_{i \in \mathbb{N}} \tfrac{1}{2^i} \cdot \mu_i.$$

Notice that in fact the tuple $(\Pi(states_A), \{\{s \mapsto 1\} \mid s \in startA\}, sig_A, \rightarrow_{A^{**}})$ is in fact a non-probabilistic automaton. However, the moves of this automaton do not coincide with the lifting of the moves of the automaton A.

Definition 3.20 Let A, B be two probabilistic automata with the same external actions. A *probabilistic hyperstep refinement* is a function $R : states_A \to \Pi(states_B)$ such that:

1. for all $s \in start_A$, $R(s) = \{s' \mapsto 1\}$ for some $s' \in start_B$;
2. for all $s \in reach_A$, $a \in act_A$ and $\mu \in \Pi(states_A)$, if $s \xrightarrow{a}_A \mu$ then either of the following conditions hold
   (a) $R(s) \xrightarrow{a}_{B^{**}} R_{**}(\mu)$ or
   (b) $a \in int_A \wedge R(s) \xrightarrow{a'}_{B^{**}} R_{**}(\mu)$ for some $a' \in int_B$ or
   (c) $a \in int_A \wedge R(s) = R_{**}(\mu)$.

We write $A \sqsubseteq_{\text{PHSR}} B$ if there is a probabilistic hyperstep refinement between A and B.

The idea of the above notion is the following. If $s \xrightarrow{a}_A \mu$ and $R(s) = \nu$, then the state $s \in states_A$ is simulated by the state $t \in states_B$ with probability $\nu(t)$. Consider case 2a. We require that, for t with $\nu(t) > 0$, there is a ν'_t such that $t \xrightarrow{a}_B \nu'_t$ and this transition is simulated with probability $\nu(t)$ too: the requirement implies $\nu \xrightarrow{a}_{B^{**}} \sum_t \nu(t) \cdot \nu'_t = \vartheta$ by definition of $\rightarrow_{**}$. As R is a simulation relation, μ should be related to ϑ, which means $R_{**}(\mu) = \vartheta$.

The following theorem establishes some relations between the preorders defined in this section. It implies that the simulations $\sqsubseteq_{\text{PSR}}$ and $\sqsubseteq_{\text{PHSR}}$ are sound for trace distribution inclusion. The proof is based on the soundness result for the simulation relation $\sqsubseteq_{\text{PFS}}$ from [Seg95], where $A \sqsubseteq_{\text{PFS}} B$ means that there is a probabilistic forward simulation from A to B.

Theorem 3.21 *Let A, B be probabilistic automata with the same external actions.*

1. *If $A \sqsubseteq_{\mathrm{PSR}} B$ then $A \sqsubseteq_{\mathrm{PHSR}} B$;*
2. *if $A \sqsubseteq_{\mathrm{PHSR}} B$ then $A \sqsubseteq_{\mathrm{PFS}} B$;*
3. *if $A \sqsubseteq_{\mathrm{PFS}} B$ then $A \sqsubseteq_{\mathrm{TD}} B$.*

3.4 Fairness in probabilistic automata

This section introduces notions of fairness for executions, traces, probabilistic executions and trace distributions. We state a result that concludes inclusion of fair trace distributions from fair trace inclusion via a probabilistic step refinement.

Definition 3.22 Let A be a (probabilistic) automaton.

1. An execution of A is called *(weakly) fair* if the following conditions hold for each class C of $tasks_A$:
   (a) if α is finite then none of the actions in C is enabled in the final state of α;
   (b) if α is infinite, then α contains either infinitely many actions from C or infinitely many occurrences of states in which no action in C is enabled;
2. a trace of A is *fair* in A if it is the trace of a fair execution.

The sets of fair executions and fair traces of A are denoted by $fexecs(A)$ and $ftraces(A)$ respectively.

Lemma 3.23 *Let A_1, A_2 be (probabilistic) automata and let α be an execution of $A_1 \parallel A_2$. If the projections of α on A and B are both fair, then α is fair.*

PROOF: Let $\alpha = s_0 a_1 s_1 \ldots = (s_{0,1}, s_{0,2}) a_1 (s_{1,1}, s_{1,2}) a_2 (s_{2,1}, s_{2,2}) a_3 \ldots$ be an execution of $A_1 \parallel A_2$. Let C be a task class from $tasks_{A_1 \parallel A_2}$. Then $C \in tasks_{A_i}$, for $i = 1$ or $i = 2$. We prove that α is fair with respect to C by distinguishing between the following cases.

1. If α is finite, then the projection of α on A_i is finite. Then C is not enabled in the last state of the projection, so neither is C is enabled in the last state of α. Therefore, α is fair with respect to C.
2. If α is infinite, then distinguish between two more cases: If the projection of α to A_i is finite, then let $s_{k,i}$ be its last state. Then C is not enabled in $s_{k,i}$. The execution α does not contain actions from A_i after s_k. More precisely, $a_n \notin act_A$ for $n \geq k$. But then $s_{k,i} = s_{n,i}$ for all $n \geq k$, which implies that C is disabled in s_n for all $n \geq k$. So, C is infinitely often disabled. Therefore, α is fair with respect to C.

   If the projection of α to A_i is infinite, then either the projection contains infinitely many actions in C, in which case α contains infinitely many infinitely many actions in C, or C is infinitely often disabled in the projection, in which case it is infinitely often disabled in α. Therefore, α is fair with respect to C.

□

Remark 3.24 The definitions of compatible and of task partition we have given here induce a notion of fairness which is different from that of literature, see for example [Lyn96]. In particular, our notion of fairness does not satisfy the properties with respect to compositionality from literature, viz. the converse of 3.23. However, Proposition 7.6 shows that the automaton model P of the protocol does meet this property. The same trivially holds for the other automata defined in this report.

Lemma 3.25 *Let $E = (\Omega_E, \mathcal{F}_E, \mathbf{P}_E)$ be a probabilistic execution of a probabilistic automaton A. Then $fexecs(A) \cap \Omega_E \in \mathcal{F}_E$.*

PROOF: Similar to the proof of the corresponding result in [PSL97]. □

Definition 3.26 1. A probabilistic execution $E = (\Omega_E, \mathcal{F}_E, \mathbf{P}_E)$ is called *fair* if $\mathbf{P}_E[fexecs(A) \cap \Omega_E] = 1$

2. A trace distribution H is called *fair* if it is the distribution of a fair execution.

Notation 3.27 The set of fair probabilistic executions of A is denoted by *fpexecs*(A) and the set of fair trace distributions by *ftrdistr*(A). If *ftrdistr*$(A) \subseteq$*ftrdistr*(B) then we write $A \sqsubseteq_{\text{FTD}} B$. It is obvious that $\sqsubseteq_{\text{FTD}}$ is a partial order.

Fairness and probabilistic step refinements

Lemma 3.28 *Let A and B be probabilistic automata with the same external actions. Let $r : states_A \to states_B$ be a probabilistic step refinement Then r induces a relation $\tilde{r} \subseteq frags(A) \times frags(B)$ as follows: if $\alpha = s_0 a_1 s_1 \ldots \in fragsA$,* index$(\alpha)$ *is the index set of α, $\beta = t_0 b_1 t_1 b_2 \ldots \in frags(B)$ and* index(β) *is the index set of β, then $\alpha \tilde{r} \beta$ holds if and only if there exists a surjective, nondecreasing index mapping $m :$* index$(\alpha) \to$ index(β), *such that for all $i \in$* index(α), $j \in$ index(β),

1. *$m(0) = 0$;*
2. *$r(s_i) = t_{m(i)}$;*
3. *if $i > 0$ then either of the following conditions holds*
   (a) *$a_i = b_{m(i)} \wedge m(i) = m(i-1) + 1$ or*
   (b) *$a_i \in int_A \wedge b_{m(i)} \in int_B \wedge m(i) = m(i-1) + 1$ or*
   (c) *$a_i \in int_A \wedge m(i) = m(i-1)$.*

Fact 3.29 *For each probabilistic step refinement r, the relation $\tilde{r}$ has the following properties: For all $\alpha \in frags(A)$ and $\beta \in frags(B)$*

1. *$\alpha \tilde{r} \beta \implies \operatorname{lng} \alpha \geq \operatorname{lng} \beta$;*
2. *$\alpha \tilde{r} \beta \implies trace(\alpha) = trace(\beta)$.*

Claim 3.30 *([SV99a]) Let A and B be probabilistic automata with the same external actions and let $r : states_A \to states_B$ be a probabilistic step refinement. Then r induces a function $\tilde{r} : pfrags(A) \to pfrags(B)$ with the following properties:*

1. *For all $E \in frags(A)$, E and $\tilde{r}(E)$ have the same trace distribution;*
2. *Assume that $\forall \alpha \in frags(A), \beta \in frags(B)[\alpha$ is fair $\wedge \alpha \tilde{r} \beta \implies \beta$ is fair$]$. Then $\forall E \in pfrags(A)[E$ is fair $\implies \tilde{r}(E)$ is fair$]$.*

Claim 3.31 *([SV99a]) Let A and B be probabilistic automata with the same external actions and let $r : states_A \to states_B$ be a probabilistic step refinement.*

1. *$A \sqsubseteq_{\mathrm{TD}} B$.*
2. *If $\forall \alpha \in frags(A), \beta \in frags(B)[\alpha$ is fair $\wedge \alpha \tilde{r} \beta \implies \beta$ is fair$]$, then $A \sqsubseteq_{\mathrm{FTD}} B$.*

4 Root contention in IEEE 1394

The IEEE 1394 protocol is a high performance serial bus protocol, developed for interconnecting consumer equipment such as PCs, VCRs and CD players. The lowest level of this protocol runs a phase, called the Tree Identify Phase, which elects a leader among the components. The leader serves as a bus manager in subsequent phases of the protocol.

The components and their connections are represented respectively as nodes and edges in an undirected graph. In the Tree Identify Phase, the protocol checks whether the graph is a(n undirected) tree and, if so, it identifies a root c.q. leader. The root is elected by directing all edges of the tree. The nodes, starting with the leaves, send requests for being a child, that is to direct the edge from the sender (c.q. child) to the receiver (c.q. parent). Then they wait for acknowledgments.

If all edges but one have been directed, a situation called *root contention* may occur. Two nodes are in root contention if they have both sent a request to each other. In this situation, it is not clear which of them should be the root. At this moment a part of the protocol is executed which resolves this root contention. The aim of this paper is to verify this part of IEEE 1394 formally.

Informally, the part of IEEE 1394 [IEE96] that solves root contention is described as follows. A process in root contention first flips a coin. If head comes up, then it waits a short time between 0.76 and $0.80 \mu s$ (micro seconds). If tail comes up, then it waits a long time; between 1.60 and $1.64 \mu s$ [3] Then the processes, regardless the values of their coins, check whether input has arrived. If so – then this can only be a request and the process has not yet sent one – the process sends an acknowledgment and declares itself root. If no input has arrived, then the process sends a request itself and waits for reply. If it receives an acknowledgment then it declares itself child. If it receives a request, – then both processes have sent requests and are in root contention again – the procedure starts all over again. The delay on the communication is 0.22 nano seconds per meter and the communication wires are at most 16 meter.

Informally, the processes in root contention behave as follows. The protocol elects the slowest process as root. This is so because the slower process receives a request before it checks its input and then becomes the root. Therefore, if one of the processes' coin flips yields head and the other yields tail, the tail process is elected as root. If the outcomes of both coin flips are the same then the the processes flip check input and send requests almost simultaneously. Then they both send and receive requests, so root contention occurs again. However, it can be the case that one of the processes is somewhat faster that the other, e.g. if one process takes a delay of $0.76 \mu s$ and the other of 0.80. The the first process receives a request when checking its input. As it has not sent a request, it becomes the root. Which of the two scenario's is actually carried out depends on physical circumstances such as temperature and processor speed.

The key idea of the protocol is that, eventually the coin flips in both processes will yield a different result with probability one: the probability that the first n

[3]These figures have been taken from the IEEE 1394a standard [IEE98]; the IEEE 1394 standard [IEE96] provides respectively $0.26, 0.30, 0.60$ and $0.64 \mu s$.

coin flips are the same is $\frac{1}{2^n}$. Therefore, the probability that eventually a root is elected equals one.

4.1 The probabilistic automaton specification of root contention

This section presents an automaton specification of the root contention protocol in IEEE 1394. We firstly specify the automaton in precondition-effect style and then we explain the model informally.

Choices in the model

As a first approach to IEEE 1394, we use an untimed model, in which the passage of a time unit time is modeled by a discrete action tick. We observe that the long delay, between 1.60 and $1.64\mu s$, is almost twice as long as the short delay, between 0.76 and $0.80\mu s$. Therefore, we model the short delay as the passage of one time unit and the long delay as the passage of two time units.

As time is a global notion, which elapses with the same speed for both processes, the processes should synchronize on tick actions. This approach is similar to the treatment of time in the timed automaton model from [LV96].

The delay on the communication channels, 0.22 nano seconds per meter is relatively small when compared to the delays mentioned above. We model the communication between the processes by instantaneous actions.

The model

As root contention only occurs in two (adjacent) nodes of the network the specification discards all other nodes and only describes the two processes P_1 and P_2 in root contention. The protocol specification P is defined by

$$P \stackrel{\mathrm{d}}{=} (P_1 \parallel P_2) \upharpoonright_{\mathsf{root}_1,\mathsf{root}_2} .$$

Below the specification of P_i is given in precondition-effect style. Let P_j be the other process, i.e. $i, j \in \{1,2\}, i \neq j$. A more intuitive explanation follows.

We define an auxiliary automaton P_{flip} for technical reasons in the correctness proof.

$$P_{flip} \stackrel{\mathrm{d}}{=} (P_1 \parallel P_2) \upharpoonright_{\mathsf{flip}_1,\mathsf{flip}_2,\mathsf{root}_1,\mathsf{root}_2} .$$

variables	**range**	**initially**
status_i	$init, waiting, checked, root, child$	$init$
coin_i	$head, tail$	
clock_i	$0, 1, 2, \infty$	0
delay_i	$1, 2$	
in_i	$empty, ack, req$	$empty$
port_i	$empty, ack, req$	$empty$
sent_i	$empty, ack, req$	$empty$

The **external actions** of P_i are flip_i, tick, root_i, $\mathsf{send}_i(req)$, $\mathsf{send}_i(ack)$, $\mathsf{send}_j(req)$ and $\mathsf{send}_j(ack)$ and the **internal actions** $\mathsf{poll\_once}_i$, $\mathsf{poll\_many}_i$, child_i and retry_i. The **task partition** consists of two classes, {tick} and {flip_i, $\mathsf{poll\_once}_i$, $\mathsf{poll\_many}_i$, $\mathsf{send}_i(req)$, $\mathsf{send}_i(ack)$, root_i, child_i, retry_i}.

action	precondition	effect
flip_i	$\text{status}_i = \mathit{init}$	$\text{status}_i := \mathit{waiting}$ $\text{coin}_i := \begin{cases} \mathit{head} & \frac{1}{2} \\ \mathit{tail} & \frac{1}{2} \end{cases}$ `if` $\text{coin}_i = \mathit{head}$ `then` $\text{delay}_i := 1$ `else` $\text{delay}_i := 2$ `fi`
tick	$\text{status}_i = \mathit{waiting} \wedge$ $\text{clock}_i < \text{delay}_i$	$\text{clock}_i := \text{clock}_i + 1$
tick	$\text{status}_i = \mathit{checked} \wedge$ $\text{in}_i = \mathit{empty} \wedge$ $\text{port}_i = \mathit{empty} \wedge$ $\text{sent}_i = \mathit{req}$	$\text{clock}_i := \text{clock}_i + 1$
tick	$\text{status}_i = \mathit{root} \vee$ $\text{status}_i = \mathit{child}$	`if` $\text{clock}_i < 2$ `then` $\text{clock}_i := \text{clock}_i + 1$ `else` $\text{clock}_i := \infty$ `fi`
$\mathsf{poll\_once}_i$	$\text{status}_i = \mathit{waiting} \wedge$ $\text{clock}_i = \text{delay}_i \wedge$ $\text{in}_i = \mathit{empty} \wedge$ $\text{sent}_i = \mathit{empty}$	$\text{status}_i := \mathit{checked}$ $\text{in}_i := \text{port}_i$ $\text{port}_i := \mathit{empty}$
$\mathsf{poll\_many}_i$	$\text{status}_i = \mathit{checked} \wedge$ $\text{in}_i = \mathit{empty} \wedge$ $\text{port}_i \neq \mathit{empty} \wedge$ $\text{sent}_i = \mathit{req}$	$\text{in}_i := \text{port}_i$ $\text{port}_i := \mathit{empty}$
$\mathsf{send}_i(\mathit{req})$	$\text{status}_i = \mathit{checked} \wedge$ $\text{in}_i = \mathit{empty} \wedge$ $\text{sent}_i = \mathit{empty}$	$\text{sent}_i := \mathit{req}$
$\mathsf{send}_i(\mathit{ack})$	$\text{status}_i = \mathit{checked} \wedge$ $\text{in}_i = \mathit{req} \wedge$ $\text{sent}_i = \mathit{empty}$	$\text{sent}_i := \mathit{ack}$
root_i	$\text{status}_i = \mathit{checked} \wedge$ $\text{sent}_i = \mathit{ack}$	$\text{status}_i := \mathit{root}$
child_i	$\text{status}_i = \mathit{checked} \wedge$ $\text{in}_i = \mathit{ack}$	$\text{status}_i := \mathit{child}$
retry_i	$\text{status}_i = \mathit{checked} \wedge$ $\text{in}_i = \mathit{req} \wedge$ $\text{sent}_i = \mathit{req}$	$\text{status}_i := \mathit{init}$ $\text{clock}_i := 0$ $\text{in}_i := \mathit{empty}$ $\text{sent}_i := \mathit{empty}$
$\mathsf{send}_j(m)$	true	$\text{port}_i := m$

The intuition behind the variables of process P_i is the following. The variable status_i reflects the phase in the protocol. It subsequently gets the values *init*, *waiting*, *checked*, *init*, *waiting*, ..., until the process is elected as the root or the child. This leads to $\text{status}_i = \mathit{root}$ or $\text{status}_i = \mathit{child}$ respectively. The value *waiting* indicates the first phase of the protocol, in which the process waits and checks input; the value *checked* indicates the second phase of the protocol, in which the process reacts on the input. The variable coin_i contains the outcome of the last coin flip. The variable clock_i records the number of time units passed since the last initiation of the protocol (i.e. since the last time that status_i was *init*). The variable delay_i

determines the number of time units, 1 or 2, the process waits after its coin flip, before checking input. Input messages from P_j arrive at P_i's variable port_i. We may think of the variable in_i as a internal memory cell in which input is stored. The variable sent_i contains the last message sent since the last initiation of the protocol.

The process P_i in root contention starts with coin flipping, taking flip_i. This action probabilistically chooses the value of coin_i, determines the delay corresponding to the outcome of this coin flip and moves to the next phase ($\mathrm{status}_i := \mathit{waiting}$.) If $\mathrm{coin}_i = \mathit{head}$, then the process waits a short time unit. Indeed, if $\mathrm{coin}_i = \mathit{head}$, then $\mathrm{delay}_i = 1$, so the process takes only one action tick. If $\mathrm{coin}_i = \mathit{tail}$, then $\mathrm{delay}_i = 2$, so the process waits two time units, i.e. it takes two actions tick.

Now the process has waited the required amount of time, which means that $\mathrm{clock}_i = \mathrm{delay}_i$, the process checks whether input has arrived: by taking the action $\mathsf{poll\_once}_i$, it copies the value of the port to the variable in_i. Furthermore, the process moves to the phase *checked*.

Now two situations may occur: either the process has received a request , or it has received no message at all. (An acknowledgment can only arrive if the process has send a request, which is not the case.) So either $\mathrm{in}_i = \mathit{req}$, or $\mathrm{in}_i = \mathit{empty}$.

Case 1: $\mathrm{in}_i = \mathit{empty}$. As the process has not sent a request itself, it sends an acknowledgment, by taking $\mathsf{send}_i(\mathit{ack})$. Then it declares itself root, by performing root_i.

Case 2: $\mathrm{in}_i = \mathit{req}$. As the process has not sent a request yet, it sends one now, taking $\mathsf{send}_i(\mathit{req})$. Now the process waits for reply. The process polls input until a message arrive. The polling on input and the arrival of a message are combined in the action $\mathsf{poll\_many}$. However, while waiting for input it may synchronize on an action tick if no input has come. This only happens if the process' coin is head an the other coin is tail.

If the input arrived is a request, then the process has both sent and received a request. This means that it is in root contention again. The process reruns the protocol and takes the action retry_i. This action resets the variables. If the input arrived is an acknowledgment, then P_i declares itself child performing child_i. When root or child the process only performs tick actions. In order to keep the model finite, the variable clock moves to the value ∞ if more than 2 time units have passed.

5 Stepwise abstraction of P

This section defines the automata I, A and S. Sections 6 and 7 prove that these automata are stepwise abstractions of P: We prove $P \sqsubseteq_{\mathrm{TD}} I \sqsubseteq_{\mathrm{TD}} A \sqsubseteq_{\mathrm{TD}} S$ and $P \sqsubseteq_{\mathrm{FTD}} I \sqsubseteq_{\mathrm{FTD}} A \sqsubseteq_{\mathrm{FTD}} S$.

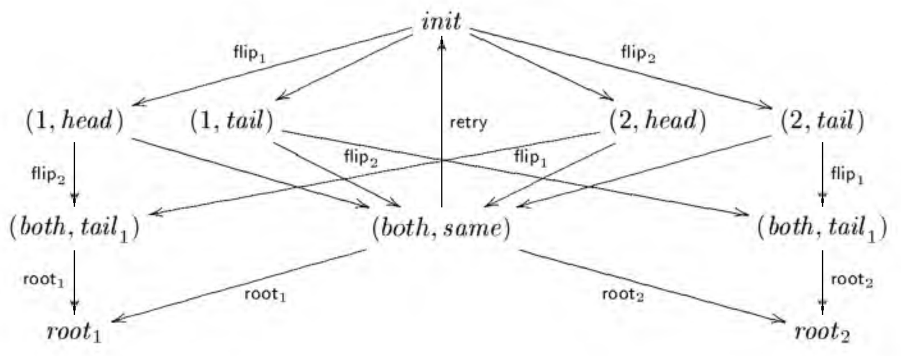
The specification automaton S – the S stands specification – is the most abstract specification. It expresses leader election for two processes.

The abstract automaton A is derived from S by including a minimal amount of probabilistic information: it contains one flip action that combines the two flip actions from P. The automaton I has been derived from A by separating the flip action from A into two flip actions: one for each process.

5.1 The intermediate specification

We define an intermediate specification I and an auxiliary automaton I_{flip}. The state names in I (and I_{flip}) stand for the following. The state *init* is the initial state and root_i means that process i has been elected as the root. The state $(i, \mathit{outcome})$ reflects that process i has flipped its coin but that the other process has not yet flipped it. The field *outcome* records the result *head* or *tail*. The state $(\mathit{both}, \mathit{outcome})$ means that both processes have flipped their coins. The second field *outcome* records again the result which can be *same*, $(both, tail_1)$ or $(both, tail_2)$. The value *same* means that the results are the same and The value $(both, tail_i)$ that the results are different and that P_i's coin is tail.

A diagram of the automaton I and a formal definition are given below. All transitions lead to a uniform distribution over the next states, that is, the transitions labeled by an action flip_i choose between the next states with probability $\frac{1}{2}$. The other transitions choose with probability one.



Definition 5.1 Define the probabilistic automaton I_{flip} by:

$$
\begin{aligned}
states_{I_{flip}} &\overset{\mathrm{d}}{=} \{init, (1, head), (1, tail), (2, head), (2, tail),\\
&\qquad (both, same), (both, tail_1), (both, tail_2), root_1, root_2\};\\
start_{I_{flip}} &\overset{\mathrm{d}}{=} \{init\};\\
act_{I_{flip}} &\overset{\mathrm{d}}{=} \{\mathsf{flip}_1, \mathsf{flip}_2, \mathsf{retry}, \mathsf{root}_1, \mathsf{root}_2\};\\
int_{I_{flip}} &\overset{\mathrm{d}}{=} \{\mathsf{retry}\};\\
ext_{I_{flip}} &\overset{\mathrm{d}}{=} \{\mathsf{flip}_1, \mathsf{flip}_2, \mathsf{root}_1, \mathsf{root}_2\};\\
trans_{I_{flip}} &\overset{\mathrm{d}}{=} \{\ \big(init, \mathsf{flip}_1, \{(1, head) \mapsto \tfrac{1}{2}, (1, tail) \mapsto \tfrac{1}{2}\}\big)\ ,\\
&\quad \big(init, \mathsf{flip}_2, \{(2, head) \mapsto \tfrac{1}{2}, (2, tail) \mapsto \tfrac{1}{2}\}\big)\ ,\\
&\quad \big((1, tail), \mathsf{flip}_2, \{(both, same) \mapsto \tfrac{1}{2}, (both, tail_1) \mapsto \tfrac{1}{2}\}\big)\ ,\\
&\quad \big((1, head), \mathsf{flip}_2, \{(both, same) \mapsto \tfrac{1}{2}, (both, tail_2) \mapsto \tfrac{1}{2}\}\big)\ ,\\
&\quad \big((2, tail), \mathsf{flip}_1, \{(both, same) \mapsto \tfrac{1}{2}, (both, tail_2) \mapsto \tfrac{1}{2}\}\big)\ ,\\
&\quad \big((2, head), \mathsf{flip}_1, \{(both, same) \mapsto \tfrac{1}{2}, (both, tail_1) \mapsto \tfrac{1}{2}\}\big)\ ,\\
&\quad ((both, tail_1), \mathsf{root}_1, \{root_1 \mapsto 1\})\ ,\\
&\quad ((both, tail_2), \mathsf{root}_2, \{root_2 \mapsto 1\})\ ,\\
&\quad ((both, same), \mathsf{root}_1, \{root_1 \mapsto 1\})\ ,\\
&\quad ((both, same), \mathsf{root}_2, \{root_2 \mapsto 1\})\ ,\\
&\quad ((both, same), \mathsf{retry}, \{init \mapsto 1\})\};\\
tasks_{I_{flip}} &\overset{\mathrm{d}}{=} \{\{\mathsf{flip}_1, \mathsf{flip}_2, \mathsf{root}_1, \mathsf{root}_2, \mathsf{retry}\}\}.
\end{aligned}
$$

Now define

$$I \overset{\mathrm{d}}{=} I_{flip} \lceil_{\mathsf{root}_1, \mathsf{root}_2}.$$

5.2 The abstract specification

The informal meaning of the state names in automaton A is the following. The state *init* is the initial state and the state $root_i$ indicates that P_i has been elected as the root. The other states *same*, $tail_1$, $tail_2$ reflect that both processes have flipped their coins. In the state *same* both coin flips have yielded the same result. The value $tail_i$ means that the results are different and that process i's coin equals tail.

The following diagram gives a graphical representation of A.

$$\begin{tikzcd}
 & init \arrow[dl, "\mathsf{flip}"', "\frac{1}{4}"] \arrow[d, bend left, "\frac{1}{2}"] \arrow[dr, "\frac{1}{4}"] & \\
tail_1 \arrow[d, "\mathsf{root}_1"'] & same \arrow[u, bend left, "\mathsf{retry}"] \arrow[dl, "\mathsf{root}_1"] \arrow[dr, "\mathsf{root}_2"'] & tail_2 \arrow[d, "\mathsf{root}_2"] \\
root_1 & & root_2
\end{tikzcd}$$

Definition 5.2 Define the probabilistic automaton A by:

$$\begin{aligned}
states_A &\overset{\mathrm{d}}{=} \{init, same, tail_1, tail_2, root_1, root_2\};\\
start_A &\overset{\mathrm{d}}{=} \{init\};\\
act_A &\overset{\mathrm{d}}{=} \{\mathsf{flip}, \mathsf{retry}, \mathsf{root}_1, \mathsf{root}_2\};\\
in_A &\overset{\mathrm{d}}{=} \{\mathsf{flip}, \mathsf{retry}\};\\
ext_A &\overset{\mathrm{d}}{=} \{\mathsf{root}_1, \mathsf{root}_2\};\\
trans_A &\overset{\mathrm{d}}{=} \{\, (init, \mathsf{flip}, \{tail_1 \mapsto \tfrac{1}{4}, tail_2 \mapsto \tfrac{1}{4}, same \mapsto \tfrac{1}{2}\})\,,\\
&\qquad (tail_1, \mathsf{root}_1, \{root_1 \mapsto 1\})\,,\\
&\qquad (tail_2, \mathsf{root}_2, \{root_2 \mapsto 1\})\,,\\
&\qquad (same, \mathsf{root}_1, \{root_1 \mapsto 1\})\,,\\
&\qquad (same, \mathsf{root}_2, \{root_2 \mapsto 1\})\,,\\
&\qquad (same, \mathsf{retry}, \{init \mapsto 1\})\};\\
tasks_A &= \{\{\mathsf{flip}, \mathsf{root}_1, \mathsf{root}_2, \mathsf{retry}\}\}.
\end{aligned}$$

5.3 The specification

The specification of a leader election protocol simply requires that one of the processes is elected as the leader, here named root. A graphical representation of S is given by:

$$\begin{tikzcd}
 & init \arrow[dl, "\mathsf{root}_1"'] \arrow[dr, "\mathsf{root}_2"] & \\
root_1 & & root_2
\end{tikzcd}$$

Definition 5.3 Define the probabilistic automaton S by:

$$\begin{aligned}
states_S &\overset{\mathrm{d}}{=} \{init, root_1, root_2\};\\
start_S &\overset{\mathrm{d}}{=} \{init\};\\
acts_S &\overset{\mathrm{d}}{=} \{\mathsf{root}_1, \mathsf{root}_2\};\\
ints_S &\overset{\mathrm{d}}{=} \emptyset;\\
exts_S &\overset{\mathrm{d}}{=} \{\mathsf{root}_1, \mathsf{root}_2\};\\
trans_S &\overset{\mathrm{d}}{=} \{\,(init, \mathsf{root}_1, \{root_1 \mapsto 1\})\,,\\
&\qquad (init, \mathsf{root}_2, \{root_2 \mapsto 1\})\};\\
tasks_S &\overset{\mathrm{d}}{=} \{\{\mathsf{root}_1, \mathsf{root}_2\}\}.
\end{aligned}$$

6 Trace distribution inclusion

This section proves the correctness P with respect to S as far as safety properties concern. We therefore establish the following relations.

1a. $P_{flip}{}^{-} \sqsubseteq_{\mathrm{SR}} I_{flip}{}^{-}$, meaning that there is a (non-probabilistic) step refinement from $P_{flip}{}^{-}$ to $I_{flip}{}^{-}$;

1b. $P_{flip} \sqsubseteq_{\mathrm{PSR}} I_{flip}$.

Then Fact 3.15 implies $P \sqsubseteq_{\mathrm{PSR}} I$;

2. $I \sqsubseteq_{\mathrm{PHSR}} A$;

3. $A \sqsubseteq_{\mathrm{PSR}} S$.

Then $P \sqsubseteq_{\mathrm{TD}} S$ follows from soundness of $\sqsubseteq_{\mathrm{PSR}}$ and $\sqsubseteq_{\mathrm{PHSR}}$, stated in Theorem 3.21, and from transitivity of $\sqsubseteq_{\mathrm{TD}}$.

6.1 The probabilistic step refinement between P_{flip} and I_{flip}

This section proves that $P \sqsubseteq_{\mathrm{TD}} I$ by establishing a probabilistic step refinement r between P_{flip} and I_{flip}. We firstly present the invariants and we point out how we checked them with the model checker SMV, then we define the refinement mapping r.

Invariants

Definition 6.1 Define the following abbreviations for $i, j \in \{1, 2\}, i \neq j$.

$$\begin{aligned}
contention_i &\overset{\mathrm{d}}{=} (\mathrm{status}_i = checked \wedge \mathrm{sent}_i = req \wedge (\mathrm{in}_i = req \vee \mathrm{port}_i = req))\\
&\qquad \vee \mathrm{status}_i = init;\\
waiting_i &\overset{\mathrm{d}}{=} \mathrm{status}_i = waiting;\\
midphase_i &\overset{\mathrm{d}}{=} \mathrm{status}_i \in \{checked, child\} \wedge \neg contention_i;\\
\rho(c_1, c_2) &\overset{\mathrm{d}}{=} \begin{cases} same & \text{if } c_1 = c_2,\\ tail_1 & \text{if } c_1 \neq c_2 \wedge c_1 = tail,\\ tail_2 & \text{otherwise.}\end{cases}
\end{aligned}$$

The following lemma formulates the invariants used in the refinement proof.

Lemma 6.2 *For all $s \in reach_{P_{flip}}, i, j \in \{1,2\}, i \neq j$*

$$
\begin{aligned}
s \models & \text{status}_i = \mathit{init} \implies (\mathit{waiting}_j \vee \mathit{contention}_j);\\
s \models & \text{sent}_i = \mathit{ack} \implies (\text{coin}_i = \mathit{tail} \vee \text{coin}_1 = \text{coin}_2);\\
s \models & \text{sent}_i = \mathit{ack} \implies \mathit{midphase}_j;\\
s \models & \text{in}_i = \mathit{ack} \implies \text{port}_i = \mathit{empty};\\
s \models & (\text{status}_i = \mathit{checked} \wedge \text{in}_i = \mathit{req} \wedge \text{sent}_i = \mathit{empty}) \implies\\
& (\text{port}_j = \mathit{empty} \wedge \text{in}_j = \mathit{empty} \wedge \text{status}_j = \mathit{checked});\\
s \models & (\text{status}_i = \mathit{checked} \wedge \text{sent}_i = \mathit{empty} \wedge \text{in}_i = \mathit{empty}) \implies\\
& \text{port}_i = \text{sent}_j \wedge \text{port}_j = \mathit{empty} \wedge \text{in}_j = \mathit{empty} \wedge\\
& (\text{status}_j = \mathit{checked} \vee \text{status}_j = \mathit{waiting});\\
s \models & \text{sent}_1 = \mathit{req} \wedge \text{sent}_2 = \mathit{req} \implies \text{coin}_1 = \text{coin}_2;\\
s \models & \text{sent}_i = \mathit{req} \wedge \text{port}_i = \mathit{req} \implies \text{status}_i = \mathit{checked}.
\end{aligned}
$$

Checking the invariants with SMV

The SMV system is a tool for checking finite state systems against specifications in CTL. The SMV language describes an unlabeled transition system. Basically, SMV allows one to declare a variable of a certain type and to specify its initial value and a next-value. The assignments of all variables are performed in parallel. For an introduction to SMV, the reader is referred to [McM92].

There are a few differences between SMV and automata in precondition-effect style, which prevents immediate encoding of automata in SMV. In order to minimize the risk of introducing errors, we have followed a structured method of encoding automata and invariants into SMV. We have takem the following three steps.

wegThe translation in SMV of the automaton P^- can be found in the appendix. We have taken the following steps.

1. Unlike a state in an automaton, needs a state in SMV to have a successor. We therefore add an action skip to the automaton, which leaves the variables unchanged and which is taken when no other action is enabled. Then these automata have exactly the same reachable states. The automaton P^- however enables at least one action in each reachable state. This is not obvious and can be checked in SMV, by showing that the disjunction of all preconditions holds in every reachable state. Although the action skip is superfluous, we leave it in the SMV encoding, for sake of a systematic translation.

2. For every variable in the automaton, we declare a variable with the same name in the SMV encoding. In order to simulate the action labels, we operate in two alternating phases. (This is crucial since an action may specify the next values of several variables at the time, whereas the next-function in SMV concerns one variable at the time. We must prevent the translation to update variable x according to action a and the variable y according to b.) Phase one nondeterministically chooses one of the enabled actions and phase two updates the variables according to the chosen action. In order to encode the set of enabled actions efficiently, we add, for each action a, this action to a list if it is enabled and if not we add a default action which is enabled. The latter is necessary to ensure that the list constructed is not empty.

3. The translation of the invariants is obvious. The invariants have the form $\forall s \in reach_P \forall i, j \in \{1,2\}, i \neq j[s \models \Phi(i,j)]$. This leads to the SMV equivalent

$AG(\Phi(1,2) \wedge \Phi(2,1))$. Notice that $AG\Psi$ means that Ψ holds for all paths, for all states on the path, which is indeed for all reachable states.

The refinement mapping

Definition 6.3 Define the function $r : \mathit{states}_{P_{\mathit{flip}}^-} \to \mathit{states}_{I_{\mathit{flip}}^-}$ by

$$r(s) \overset{\mathrm{d}}{=} \begin{cases} \mathit{init} & \text{if } s \models \mathit{contention}_1 \wedge \mathit{contention}_2; \\ (1, s.\mathrm{coin}_1) & \text{if } s \models \mathit{waiting}_1 \wedge \mathit{contention}_2; \\ (2, s.\mathrm{coin}_2) & \text{if } s \models \mathit{waiting}_2 \wedge \mathit{contention}_1; \\ \rho(s.\mathrm{coin}_1, s.\mathrm{coin}_2) & \text{if } s \models (\mathit{waiting}_1 \vee \mathit{midphase}_1) \wedge \\ & \qquad (\mathit{waiting}_2 \vee \mathit{midphase}_2); \\ \mathit{root}_1 & \text{if } s \models \mathrm{status}_1 = \mathit{root}; \\ \mathit{root}_2 & \text{otherwise.} \end{cases}$$

Notice that the conditions in the above case distinction are disjoint, so the refinement r is well-defined.

Lemma 6.4 *The function r is a step refinement from the automaton P_{flip}^- to I_{flip}^-.*

PROOF: Standard. □

Lemma 6.5 *For all reachable states $s, t_1, t_2 \in \mathit{reach}_{P^-}$, for all $i \in \{1,2\}$,*

$$s \xrightarrow{\mathsf{flip}_i}_{P^-} t_1 \wedge s \xrightarrow{\mathsf{flip}_i}_{P^-} t_2 \wedge t_1 \neq t_2 \implies r(t_1) \neq r(t_2).$$

(Notice that $t_1 \neq t_2$ in this case means $t_1.\mathrm{coin}_i \neq t_2.\mathrm{coin}_i$.)

Lemma 6.6 *The function r is a probabilistic step refinement from P_{flip} to I_{flip}.*

PROOF: Let $s \in \mathit{states}_{P_{\mathit{flip}}}$.

1. The action flip_i: Let $i \in \{1,2\}$ and $s \xrightarrow{a}_{P_{flip}} \mu \in \mathit{trans}_{P_{\mathit{flip}}}$. Then there are $t_1, t_2 \in \mathit{states}_{P_{\mathit{flip}}}$ such that $\mu = \{t_1 \mapsto \frac{1}{2}, t_2 \mapsto \frac{1}{2}\}$ and $t_1.\mathrm{coin}_i \neq t_2.\mathrm{coin}_i$. Therefore, we have $r(t_1) \neq r(t_2)$. Then $s \xrightarrow{a}_{P_{flip}^-} t_1 \in \mathit{trans}_{P_{\mathit{flip}}^-}$, and $s \xrightarrow{a}_{P_{flip}^-} t_2 \in \mathit{trans}_{P_{\mathit{flip}}^-}$ by the definition of P_{flip}^-. As flip_i is an output action and r a step refinement, we have $r(s) \xrightarrow{\mathsf{flip}_i}_{I_{flip}^-} r(t_1) \in \mathit{trans}_{I_{\mathit{flip}}^-}$ and $r(s) \xrightarrow{\mathsf{flip}_i}_{I_{flip}^-} r(t_2) \in \mathit{trans}_{I_{\mathit{flip}}^-}$. Then there exist ν_1 and ν_2 such that $r(s) \xrightarrow{\mathsf{flip}_i}_{I_{flip}} \nu_1 \in \mathit{trans}_{I_{\mathit{flip}}^-}$, $r(t_1) \in \mathrm{supp}(\nu_1)$ and $r(s) \xrightarrow{\mathsf{flip}_i}_{I_{flip}} \nu_2 \in \mathit{trans}_{I_{\mathit{flip}}}$, $r(t_2) \in \mathrm{supp}(\nu_2)$. However, I_{flip} enables at most one transition labeled by flip_i from each state and this transition leads to a uniform distribution over two states. Therefore, $\nu_1 = \nu_2 = \{r(t_1) \mapsto \frac{1}{2}, r(t_2) \mapsto \frac{1}{2}\} = r_*(\mu)$. Then $r(s) \xrightarrow{a}_{I_{flip}} r_*(\mu)$;

2. The other transitions. Let $s \xrightarrow{a}_{P_{flip}} \mu \in \mathit{trans}_{P_{\mathit{flip}}}$, $a \neq \mathsf{flip}_1, \mathsf{flip}_2$. Then μ is a Dirac distribution, say that $\mu = \{t \mapsto 1\}$. Then we have $s \xrightarrow{a}_{P_{flip}^-} t$. As r is a step refinement we have three cases:

(a) $s \xrightarrow{a}_{P_{flip}} \mu$. Then $\mu = \{t \mapsto 1\}$ and $r(s) \xrightarrow{a}_{I_{flip}} r(t)$. Then there is a ν such that $r(s) \xrightarrow{a}_{I_{flip}} \nu$ and $r(t) \in \text{supp}(\nu)$. As transitions in I_{flip} unequal to flip lead to Dirac distributions, we have $\nu = \{r(t) \mapsto 1\} = r_*(\mu)$. So, $r(s) \xrightarrow{a}_{I_{flip}} r_*(\mu) \in trans_{I_{flip}}$;

(b) $a \in int_{P_{flip}}$ and $s \xrightarrow{a'}_{P_{flip}} \mu$ for some $a' \in int_{I_{flip}}$. Then $a' \neq \mathsf{flip}_i$, so we are in a simular case to the one above;

(c) $a \in int_{P_{flip}}$ and $r(s) = r(t)$. Then $r_*(\mu) = r_*(\{t \mapsto 1\}) = \{r(t) \mapsto 1\} = \{r(s) \mapsto 1\}$.

□

Corollary 6.7 $P_{flip} \sqsubseteq_{\text{PSR}} I_{flip}$.

Corollary 6.8 $P \sqsubseteq_{\text{PSR}} I$.

Corollary 6.9 $P \sqsubseteq_{\text{TD}} I$.

6.2 The probabilistic hyperstep refinement between I and A

This section proves $I \sqsubseteq_{\text{TD}} A$ by estabilishing a probabilistic hyperstep refinement R between I and A.

Definition 6.10 Define a function $R : states_I \to \Pi(states_A)$ by

$$\begin{aligned}
R(init) &= \{init \mapsto 1\};\\
R(1, head) &= \{tail_2 \mapsto \tfrac{1}{2}, same \mapsto \tfrac{1}{2}\};\\
R(2, head) &= \{tail_1 \mapsto \tfrac{1}{2}, same \mapsto \tfrac{1}{2}\};\\
R(1, tail) &= \{tail_1 \mapsto \tfrac{1}{2}, same \mapsto \tfrac{1}{2}\};\\
R(2, tail) &= \{tail_2 \mapsto \tfrac{1}{2}, same \mapsto \tfrac{1}{2}\};\\
R(both, tail_1) &= \{tail_1 \mapsto 1\};\\
R(both, tail_2) &= \{tail_2 \mapsto 1\};\\
R(same) &= \{same \mapsto 1\};\\
R(root_1) &= \{root_1 \mapsto 1\};\\
R(root_2) &= \{root_2 \mapsto 1\}.
\end{aligned}$$

Lemma 6.11 *The function R is a probabilistic hyperstep refinement from I to A.*

PROOF: We prove that R meets the conditions in Definition 3.20.

condition 1. $R(init) = \{init \mapsto 1\}$ and $init \in start_A$.

condition 2. Let $s \in states_I$, $a \in act_I, \mu \in \Pi(states_I)$ such that $s \xrightarrow{a}_I \mu$. Let $i, j \in \{1, 2\}, i \neq j$.

1. Consider the transition $init \xrightarrow{\mathsf{flip}_i}_I \{(i, head) \mapsto \frac{1}{2}, (i, tail) \mapsto \frac{1}{2}\}$, We have that $R(init) = \{init \mapsto 1\}$ and

$$\begin{aligned}
&R_{**}\{(i, head) \mapsto \tfrac{1}{2}, (i, tail) \mapsto \tfrac{1}{2}\} =\\
&\tfrac{1}{2} \cdot R(i, head) + \tfrac{1}{2} \cdot R(i, tail) =\\
&\tfrac{1}{2} \cdot \{tail_j \mapsto \tfrac{1}{2}, same \mapsto \tfrac{1}{2}\} + \tfrac{1}{2} \cdot \{tail_i \mapsto \tfrac{1}{2}, same \mapsto \tfrac{1}{2}\} =\\
&\{tail_1 \mapsto \tfrac{1}{4}, tail_2 \mapsto \tfrac{1}{4}, same \mapsto \tfrac{1}{2}\}.
\end{aligned}$$

And indeed, $R(init) = \{init \mapsto 1\} \xrightarrow{\mathsf{flip}_i}_{A^{**}} \{tail_1 \mapsto \frac{1}{4}, tail_2 \mapsto \frac{1}{4}, same \mapsto \frac{1}{2}\}$.

2. Consider the transition $(i, tail) \xrightarrow{\mathsf{flip}_j}_I \{(both, same) \mapsto \frac{1}{2}, (both, tail_i) \mapsto \frac{1}{2}\}$, for $i \in \{1, 2\}$. We have $R(i, tail) = \{tail_i \mapsto \frac{1}{2}, same \mapsto \frac{1}{2}\}$ and

   $R_{**}\{(both, same) \mapsto \frac{1}{2}, (i, tail) \mapsto \frac{1}{2}\} =$
   $\frac{1}{2} \cdot R(both, same) + \frac{1}{2} \cdot R(both, tail_i) =$
   $\{same \mapsto \frac{1}{2}, tail_i \mapsto \frac{1}{2}\} =$
   $R(i, tail)$.

3. Similarly, one can prove that for the transition $(i, head) \xrightarrow{\mathsf{flip}_j}_I \{(both, same) \mapsto \frac{1}{2}, (both, tail_i) \mapsto \frac{1}{2}\}$, $R_{**}\{(both, same) \mapsto \frac{1}{2}, (both, tail_i) \mapsto \frac{1}{2}\} = R(i, head)$.

4. The transition $(both, same) \xrightarrow{\mathsf{retry}}_I \{init \mapsto 1\}$. We have that $R(both, same) = \{same \mapsto 1\} \xrightarrow{\mathsf{retry}}_{A^{**}} \{init \mapsto 1\} = 1 \cdot R(init) = R_{**}\{init \mapsto 1\}$.

5. The transitions $s \xrightarrow{\mathsf{root}_i}_I \{root_i \mapsto 1\}$, for $s \in \{(both, tail_i), (both, same)\}$ and $i \in \{1, 2\}$. We have that $R(s) = \{s \mapsto 1\} \xrightarrow{\mathsf{root}_i}_{A^{**}} \{root_i \mapsto 1\} = 1 \cdot R(root_i) = R_{**}\{root_i \mapsto 1\}$.

□

Corollary 6.12 *$I \sqsubseteq_{\text{TD}} A$.*

6.3 The probabilistic step refinement between A and S

This section proves $A \sqsubseteq_{\text{TD}} S$ by establishing a probabilistic step refinement u between A and S.

Definition 6.13 Define a function $u : states_A \to states_S$ by

$$\begin{aligned}
u(init) &= init;\\
u(same) &= init;\\
u(tail_1) &= init;\\
u(tail_2) &= init;\\
u(root_1) &= root_1;\\
u(root_2) &= root_2.
\end{aligned}$$

Lemma 6.14 *The function u is a probabilistic step refinement from A to S.*

Proof:

condition 1 $u(init) = init \in start_S$.

condition 2

1. Consider the transition $init \xrightarrow{\mathsf{flip}}_A \{tail_1 \mapsto \frac{1}{4}, tail_2 \mapsto \frac{1}{4}, same \mapsto \frac{1}{2}\}$. Then $u(init) = init = u(same) = u(tail_1) = u(tail_2)$, so condition 2b of Definition 3.13 is met.

2. Consider the transition $init \xrightarrow{\mathsf{retry}}_A \{init \mapsto 1\}$.Then $u(same) = init = u(init)$, so condition 2b of Definition 3.13 is met.

3. Consider the transition $s \xrightarrow{\mathsf{root}_i}_A \{root_i \mapsto 1\}$, for $s \in \{same, tail_i\}$. Then $u(s) \xrightarrow{\mathsf{root}_i}_S u_*\{root_i \mapsto 1\} = \{u(root_i) \mapsto 1\} = \{root_i \mapsto 1\}$.

□

Corollary 6.15 *$A \sqsubseteq_{\text{TD}} S$.*

Corollary 6.16 *$P \sqsubseteq_{\text{TD}} S$.*

7 Fair trace distribution inclusion

This section proves that P implements S, i.e. that $P \sqsubseteq_{\mathrm{FTD}} S$. We therefore establish the following relations.

1. $P_{flip} \sqsubseteq_{\mathrm{FTD}} I_{flip}$, using that the refinement r from ${P_{flip}}^{-}$ to ${I_{flip}}^{-}$ preserves fairness of executions. Then $P \sqsubseteq_{\mathrm{FTD}} I$ follows because P and P_{flip} as well as I and I_{flip} have the same task partitions;
2. $I \sqsubseteq_{\mathrm{FTD}} A$;
3. $A \sqsubseteq_{\mathrm{FTD}} S$.

Then $P \sqsubseteq_{\mathrm{FTD}} S$ follows from transitivity of $\sqsubseteq_{\mathrm{FTD}}$.

Fair trace distribution inclusion for P and I

This section proves that $P \sqsubseteq_{\mathrm{FTD}} I$ by showing that the refinement mapping r defined in Definition 6.3 from P_{flip} to I_{flip} preserves fairness of executions. Our first aim is to show that the fair traces of P satisfy the properties desired with respect to modularity, c.f. Remark 3.24.

Fair and diverging executions in P and P_i

This section proves that the fair executions and the so-called diverging executions of P and P_i are all strongly related. This implies that no "time deadlocks" can appear in P and P_i and that the specification P has the desired properties with respect to the parallel composition.

Notation 7.1 For $i \in \{1,2\}$, let $C_i \stackrel{\mathrm{d}}{=} \{\mathsf{flip}_i, \mathsf{poll\_once}_i, \mathsf{poll\_many}_i, \mathsf{send}_i(req), \mathsf{root}_i, \mathsf{child}_i, \mathsf{send}_i(ack), \mathsf{retry}_i\}$. Then $tasks_{P_i} = \{\{\mathsf{tick}\}, C_i\}$ and $tasks_P = \{\{\mathsf{tick}\}, C_1, C_2\}$. Notice that every action of P is contained in a task class.

An execution of a timed automaton is called diverging if time elapses to infinity. As time is modeled by the action tick in P and P_i, we have the following analogy.

Definition 7.2 An execution of P, P_1 or P_2 is called *diverging* if it contains infinitely many tick actions.

Proposition 7.3 *1. Every reachable state $s \in P^{-}$ enables at least one transition;*

2. For all infinite executions $\alpha = s_0 a_1 s_1 \ldots$ of P^{-} there are infinitely many k's such that

$$\begin{aligned} s_k \models & (\mathrm{status}_1 = \mathit{waiting} \wedge \mathrm{clock}_1 = 0 \wedge \mathrm{status}_2 = \mathit{waiting} \wedge \mathrm{clock}_2 = 0) \vee \\ & (\mathrm{status}_2 = \mathit{root} \wedge \mathrm{status}_2 = \mathit{child}) \vee \\ & (\mathrm{status}_1 = \mathit{root} \wedge \mathrm{status}_1 = \mathit{child}); \end{aligned}$$

3. For all infinite executions $\alpha = s_0 a_1 s_1 \ldots$ of P^{-} there are infinitely many k's such that

$$\begin{aligned} s_k \models & (\mathrm{status}_1 = \mathit{init} \vee \mathrm{status}_2 = \mathit{init} \vee \\ & \mathrm{status}_1 = \mathit{root} \vee \mathrm{status}_2 = \mathit{root}). \end{aligned}$$

PROOF:

1. It has been checked in SMV that the disjunction of the preconditions of all actions is true in every reachable state of P^-;
2. We have checked with SMV that the formula $AGAF((\mathrm{status}_1 = \mathit{waiting} \wedge \mathrm{clock}_1 = 0 \wedge \mathrm{status}_2 = \mathit{waiting} \wedge \mathrm{clock}_2 = 0) \vee (\mathrm{status}_2 = \mathit{root} \wedge \mathrm{clock}_2 = \mathit{child}) \vee (\mathrm{status}_1 = \mathit{root} \wedge \mathrm{clock}_1 = \mathit{child}))$ holds for the SMV model of P^-.
3. We have checked that the formula $AGAF(\mathrm{status}_1 = \mathit{init} \vee \mathrm{status}_2 = \mathit{init} \vee \mathrm{status}_1 = \mathit{root} \vee \mathrm{status}_2 = \mathit{root})$ holds for the SMV model of P^-.

□

Proposition 7.4 *Every fair execution of P^- is infinite and diverging.*

PROOF: Let α be a fair execution of P^-. Assume that α is finite. Then the last state of α enables an action by Proposition 7.3. As this action is in one of the task classes, α is not fair. Therefore, α is infinite. Then Proposition 7.3-2 implies that the action tick is infinitely often enabled. Then tick is infinitely often taken because if tick is enabled, then no other action is enabled. (It is easy to see that the disjunction of the preconditions of tick and any other actions P^- is false.) Therefore, α is diverging. □

Proposition 7.5 *Every diverging execution of P_i^- is fair.*

PROOF: It is easy to see that that the disjunction of the preconditions of tick and any action of C_i is false. This means that if tick is enabled, then no action in C_i is enabled. As a diverging execution contains infinitely many ticks, the actions from C_i are infinitely often disabled. Hence, every diverging execution of P is fair. □

Corollary 7.6 *Let $\alpha \in execs(P)$. Then the following statements are equivalent.*

1. *α is fair;*
2. *α is diverging;*
3. *The projections of α on P_1 and P_2 are both diverging;*
4. *The projections of α on P_1 and P_2 are both fair.*

PROOF: Proposition 7.4 states that $1 \Longrightarrow 2$. The definition of diverging immediately yields $2 \Longrightarrow 3$ and $3 \Longrightarrow 4$ follows from Proposition 7.5. Proposition 3.23 finally implies $4 \Longrightarrow 1$. □

Fair trace distribution inclusion between P and I

Recall that $r : states_{P_{flip}} \to states_{I_{flip}}$ is the probabilistic step refinement defined in 6.3 and that a probabilistic step refinement induces a relation $\tilde{r}$ according to Lemma 3.28. Now we prove fair trace distribution inclusion between P and I by showing that the refinement r preserves fairness of executions.

Lemma 7.7 *The relation $\tilde{r} \subseteq frags(P_{flip}) \times frags(I_{flip})$ induced by r is a function.*

PROOF: For $s, t \in states_{I^-}$ there is at most one action a such that $s \xrightarrow{a}_{I^-} t$. □

Notation 7.8 In the sequel we write $\tilde{r}(\alpha) = \beta$ for $(\alpha, \beta) \in \tilde{r}$.

Lemma 7.9 *The induced function $\tilde{r} : states_{P_{flip}^-} \to states_{I_{flip}^-}$ preserves fairness, i.e. $\alpha \in fexecs(P_{flip}) \implies \tilde{r}(\alpha) \in fexecs(I_{flip})$.*

PROOF: It is clear that in the automaton $I_{flip}{}^-$

- all infinite executions are fair and
- a finite execution is fair if and only if its last state is either $root_1$ or $root_2$.

Let $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ be a fair execution of $P_{flip}{}^-$. Lemma 7.3 yields that α is infinite. Consider the following cases.

If there are k, i such that $s_k \models \mathrm{status}_i = root$, then $r(s_k) = root_i$ by Definition 6.3. As the state $root_i$ is quiescent in $I_{flip}{}^-$, $\tilde{r}(\alpha)$ is finite and ends in a quiescent state, so $\tilde{r}(\alpha)$ is a fair execution.

If, on the contrary, there are no k,i such that $s_k \models \mathrm{status}_i = root]$, then Lemma 7.3-3 implies that there are infinitely many states $s_{n_1}, s_{n_2}, s_{n_3}, \ldots$ such that

$$s_{n_k} \models \mathrm{status}_1 = init \vee \mathrm{status}_2 = init.$$

Let $k \in \mathbb{N}$. As s_{n_k} only enables the actions flip_1 and flip_2, we have $a_{n_k+1} = \mathsf{flip}_i$, which means that the action taken by α in the state s_{n_k} is either flip_1 or flip_2. As flip_i is an external action of $P_{flip}{}^-$, we have $r(s_{n_k}) \xrightarrow{\mathsf{flip}_i}_{I_{flip}^-} r(s_{n_k+1})$ and therefore $\tilde{r}(s_{n_k} a_{n_k+1} s_{n_k+1}) = r(s_{n_k}) a_{n_k+1} r(s_{n_k+1}) = r(s_{n_k}) \mathsf{flip}_i r(s_{n_k+1})$. Therefore, $\tilde{r}(\alpha)$ contains infinitely many flip actions and is therefore infinite, c.q. fair. $\Box$

Corollary 7.10 $P_{flip} \sqsubseteq_{\mathrm{FTD}} I_{flip}$.

Corollary 7.11 $P \sqsubseteq_{\mathrm{FTD}} I$.

Fair trace distribution inclusion for I and A

This section proves that $I \sqsubseteq_{\mathrm{FTD}} A$ by an ad hoc probabilistic argument. It is left as a topic for future research to find a proof similar to that of Section 7. The main part of this research is to define a notion of "preserving fairness of executions" for probabilistic hyperstep refinements.

Lemma 7.12 *If H is a fair trace distribution of I, then $\mathbf{P}_H[\{root_1\}] \geq \frac{1}{4}$ and $\mathbf{P}_H[\{root_2\}] \geq \frac{1}{4}$.*

PROOF: Consider a fair probabilistic execution E of I of a fair trace distribution H. Then there is a unique μ such that $(init, \mu) \in trans_E$. As $init$ is not a fair trace distribution of I, $\mathbf{P}_E[init] = 0$, so $\delta \notin \mathbf{P}_\mu$. Then there is a $p \in [0,1]$ such that

$$\mu = p \cdot \{init\mathsf{flip}_1(1, head) \mapsto \tfrac{1}{2}, init\mathsf{flip}_1(1, tail) \mapsto \tfrac{1}{2}\} +$$
$$(1-p) \cdot \{init\mathsf{flip}_2(2, head) \mapsto \tfrac{1}{2}, init\mathsf{flip}_2(2, tail) \mapsto \tfrac{1}{2}\}.$$

case 1: $p \neq 0, 1$. Then there is a unique ν_1 such that $(init\mathsf{flip}_1(1, tail), \nu_1) \in trans_E$. By fairness of E, $\mathbf{P}_E[init\mathsf{flip}_1(1, tail)] = 0$, so $\delta \notin \mathrm{supp}(\nu_1)$. Then

$$\nu_1 = \{init\mathsf{flip}_1(1, tail)\mathsf{flip}_2(both, tail_1) \mapsto \tfrac{1}{2},$$
$$init\mathsf{flip}_1(1, tail)\mathsf{flip}_2(both, same) \mapsto \tfrac{1}{2}\}.$$

Again by fairness, the set $trans_E$ contains the tuple $(init\mathsf{flip}_1(1, tail)\mathsf{flip}_2(both, tail_1),$ $\{init\mathsf{flip}_1(1, tail)\mathsf{flip}_2(both, tail_1)\mathsf{root}_1 root_1 \mapsto 1\})$. Similarly, we have that

$$\begin{aligned}&(init\mathsf{flip}_2(1, head), \nu_2) \in trans_E,\\ &\qquad \text{where } \nu_2 = \{init\mathsf{flip}_2(1, head)\mathsf{flip}_1(both, tail_1) \mapsto \tfrac{1}{2},\\ &\qquad\qquad init\mathsf{flip}_2(1, head)\mathsf{flip}_1(both, same) \mapsto \tfrac{1}{2}\}\text{and}\\ &(init\mathsf{flip}_2(2, head)\mathsf{flip}_1(both, tail_1),\\ &\qquad \{init\mathsf{flip}_2(2, head)\mathsf{flip}_1(both, tail_1)\mathsf{root}_1 root_1 \mapsto 1\}) \in trans_E.\end{aligned}$$

Then

$$\mathbf{P}_E[C_{init\mathsf{flip}_1(1,tail)\mathsf{flip}_2(both,tail_1)\mathsf{root}_1 root_1}] = p \cdot \tfrac{1}{2} \cdot \tfrac{1}{2}$$
$$\mathbf{P}_E[C_{init\mathsf{flip}_2(2,head)\mathsf{flip}_1(both,tail_1)\mathsf{root}_1 root_1}] = (1-p) \cdot \tfrac{1}{2} \cdot \tfrac{1}{2}$$

case 2: $p = 1$.Then there is a unique ν such that $(init\mathsf{flip}_1(1, tail), \nu) \in trans_E$. By fairness of E, $\delta \notin \mathrm{supp}(\nu)$ so $\nu = \{\mathsf{flip}_2(both, tail_1) \mapsto \frac{1}{2}, \mathsf{flip}_2(both, same) \mapsto \frac{1}{2}\}$. Then

$$\mathbf{P}_E[C_{init\mathsf{flip}_1(1,tail)\mathsf{flip}_2(both,tail_1)\mathsf{root}_1 root_1}] = \tfrac{1}{2} \cdot \tfrac{1}{2} = p \cdot \tfrac{1}{2} \cdot \tfrac{1}{2}$$
$$\mathbf{P}_E[C_{init\mathsf{flip}_2(2,head)\mathsf{flip}_1(both,tail_1)\mathsf{root}_1 root_1}] = 0 = (1-p) \cdot \tfrac{1}{2} \cdot \tfrac{1}{2}$$

case 3: $p = 0$. Similarly to the case above one proves

$$\mathbf{P}_E[C_{init\mathsf{flip}_1(1,tail)\mathsf{flip}_2(both,tail_1)\mathsf{root}_1 root_1}] = p \cdot \tfrac{1}{2} \cdot \tfrac{1}{2}$$
$$\mathbf{P}_E[C_{init\mathsf{flip}_2(2,head)\mathsf{flip}_1(both,tail_1)\mathsf{root}_1 root_1}] = (1-p) \cdot \tfrac{1}{2} \cdot \tfrac{1}{2}$$

Now, let H be the trace distribution of E. Then

$$\begin{aligned}&\mathbf{P}_H[root_1] =\\ &\mathbf{P}_E[traces^{-1}\{root_1\}] \geq\\ &\mathbf{P}_E[C_{init\mathsf{flip}_1(1,tail)\mathsf{flip}_2(both,tail_1)\mathsf{root}_1 root_1} \cup\\ &\qquad C_{init\mathsf{flip}_2(2,head)\mathsf{flip}_1(both,tail_1)\mathsf{root}_1 root_1}] =\\ &\mathbf{P}_E[C_{init\mathsf{flip}_1(1,tail)\mathsf{flip}_2(both,tail_1)\mathsf{root}_1 root_1}] +\\ &\qquad \mathbf{P}_E[C_{init\mathsf{flip}_2(2,head)\mathsf{flip}_1(both,tail_1)\mathsf{root}_1 root_1}] =\\ &p \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} + (1-p) \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} =\\ &\tfrac{1}{4}.\end{aligned}$$

Similarly one proves $\mathbf{P}_H[root_2] \geq \frac{1}{4}$. □

Lemma 7.13 *If H is a fair trace distribution of I, then $\mathbf{P}_H[\{root_1\}] \geq \frac{1}{4}$ and $\mathbf{P}_H[\{root_2\}] \geq \frac{1}{4}$.*

PROOF: Consider a fair probabilistic execution E of I of a fair trace distribution H. Given a state $\alpha \in states_E$, let μ_α be a probability distribution in $\Pi(trans_A(last(\alpha)) \cup \{\delta\})$ as in Definition 3.8. We have $init \in states_E$. Consider $trans_E(init)$. As $init$ is not a fair execution of I, we have $\mathbf{P}_E[init] = 0$ and therefore $\mu_{init}(\delta) = 0$. Write

$$p = \mu_{init}\left(init \xrightarrow{\mathsf{flip}_1} \{init\mathsf{flip}_1(1, head) \mapsto \tfrac{1}{2}, init\mathsf{flip}_1(1, tail) \mapsto \tfrac{1}{2}\}\right)$$

Assume $p \neq 0, 1$. Then

$$\begin{aligned}
&\mathit{trans}_E(\mathit{init})(\mathit{init}\mathsf{flip}_1(1, \mathit{head})) = p \cdot \tfrac{1}{2}\\
&\mathit{trans}_E(\mathit{init})(\mathit{init}\mathsf{flip}_1(1, \mathit{tail})) = p \cdot \tfrac{1}{2}\\
&\mathit{trans}_E(\mathit{init})(\mathit{init}\mathsf{flip}_2(2, \mathit{head})) = (1-p) \cdot \tfrac{1}{2}\\
&\mathit{trans}_E(\mathit{init})(\mathit{init}\mathsf{flip}_2(2, \mathit{tail})) = (1-p) \cdot \tfrac{1}{2}
\end{aligned}$$

Let $\alpha = \mathit{init}\mathsf{flip}_1(1, \mathit{tail}), \mathit{init}\mathsf{flip}_2(2, \mathit{head})$. Then $\alpha \in \mathit{states}_E$ and $\mu_\alpha(\delta) = 0$ by fairness. As $(1, \mathit{tail})$ and $(2, \mathit{head})$ enable one transition, viz. $(1, \mathit{tail}) \xrightarrow{\mathsf{flip}} \{(\mathit{both}, \mathit{tail}_1) \mapsto \frac{1}{2}, (\mathit{both}, \mathit{same}) \mapsto \frac{1}{2}\}$ and $(1, \mathit{tail}) \xrightarrow{\mathsf{flip}} \{(\mathit{both}, \mathit{tail}_1) \mapsto \frac{1}{2}, (\mathit{both}, \mapsto)\frac{1}{2}\}$, we have

$$\begin{aligned}
&\mathit{trans}_E(\mathit{init}\mathsf{flip}_1(1, \mathit{tail}))(\mathit{init}\mathsf{flip}_1(1, \mathit{tail})\mathsf{flip}_2(\mathit{both}, \mathit{tail}_1)) = \tfrac{1}{2}\\
&\mathit{trans}_E(\mathit{init}\mathsf{flip}_2(2, \mathit{head}))(\mathit{init}\mathsf{flip}_2(2, \mathit{head})\mathsf{flip}_1(\mathit{both}, \mathit{tail}_1)) = \tfrac{1}{2}.
\end{aligned}$$

Again fairness and the fact that the state $(\mathit{both}, \mathit{tail}_1)$ enables only one transition yield

$$\begin{aligned}
&\mathit{trans}_E(\mathit{init}\mathsf{flip}_1(1, \mathit{tail})\mathsf{flip}_2(\mathit{both}, \mathit{tail}_1))\\
&\qquad\qquad(\mathit{init}\mathsf{flip}_1(1, \mathit{tail})\mathsf{flip}_2(\mathit{both}, \mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1) = 1\\
&\mathit{trans}_E(\mathit{init}\mathsf{flip}_2(2, \mathit{head})\mathsf{flip}_1(\mathit{both}, \mathit{tail}_1))\\
&\qquad\qquad(\mathit{init}\mathsf{flip}_2(2, \mathit{head})\mathsf{flip}_1(\mathit{both}, \mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1) = 1.
\end{aligned}$$

From the above equations it follows immediately that

$$\begin{aligned}
&\mathbf{P}_E[C_{\mathit{init}\mathsf{flip}_1(1,\mathit{tail})\mathsf{flip}_2(\mathit{both},\mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1}] = p \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot 1\\
&\mathbf{P}_E[C_{\mathit{init}\mathsf{flip}_2(2,\mathit{head})\mathsf{flip}_1(\mathit{both},\mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1}] = (1-p) \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot 1.
\end{aligned}$$

For $p = 0, 1$ an even simpler calculation proves the above equation. Now, as H is the trace distribution of E, we have

$$\begin{aligned}
&\mathbf{P}_H[\mathit{root}_1] =\\
&\mathbf{P}_E[\mathit{traces}^{-1}\{\mathit{root}_1\}] \geq\\
&\mathbf{P}_E[C_{\mathit{init}\mathsf{flip}_1(1,\mathit{tail})\mathsf{flip}_2(\mathit{both},\mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1} \cup\\
&\qquad\qquad C_{\mathit{init}\mathsf{flip}_2(2,\mathit{head})\mathsf{flip}_1(\mathit{both},\mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1}] =\\
&\mathbf{P}_E[C_{\mathit{init}\mathsf{flip}_1(1,\mathit{tail})\mathsf{flip}_2(\mathit{both},\mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1}] +\\
&\qquad\qquad \mathbf{P}_E[C_{\mathit{init}\mathsf{flip}_2(2,\mathit{head})\mathsf{flip}_1(\mathit{both},\mathit{tail}_1)\mathsf{root}_1\,\mathit{root}_1}] =\\
&p \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} + (1-p) \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} =\\
&\tfrac{1}{4}.
\end{aligned}$$

Similarly one proves $\mathbf{P}_H[\mathit{root}_2] \geq \frac{1}{4}$. □

Lemma 7.14 *If H is a fair trace distribution of I, then $\mathbf{P}_H[\{\mathit{root}_1, \mathit{root}_2\}] = 1$.*

PROOF:

Let L and R be the sets of execution fragments of I defined by:

$$\begin{aligned}
L \overset{\mathrm{d}}{=} \{&\mathit{init}\mathsf{flip}_1(1, \mathit{head})\mathsf{flip}_2(\mathit{both}, \mathit{same})\mathsf{retry},\\
&\mathit{init}\mathsf{flip}_1(1, \mathit{tail})\mathsf{flip}_2(\mathit{both}, \mathit{same})\mathsf{retry},\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{head})\mathsf{flip}_1(\mathit{both}, \mathit{same})\mathsf{retry},\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{tail})\mathsf{flip}_1(\mathit{both}, \mathit{same})\mathsf{retry}\}\\
R \overset{\mathrm{d}}{=} \{&\mathit{init}\mathsf{flip}_1(1, \mathit{head})\mathsf{flip}_2(\mathit{both}, \mathit{tail}_2)\mathsf{root}_2\mathit{root}_2,\\
&\mathit{init}\mathsf{flip}_1(1, \mathit{head})\mathsf{flip}_2\mathit{same}\mathsf{root}_2\mathit{root}_2,\\
&\mathit{init}\mathsf{flip}_1(1, \mathit{head})\mathsf{flip}_2\mathit{same}\mathsf{root}_1\mathit{root}_1,\\
&\mathit{init}\mathsf{flip}_1(1, \mathit{tail})\mathsf{flip}_2(\mathit{both}, \mathit{tail}_1)\mathsf{root}_1\mathit{root}_1,\\
&\mathit{init}\mathsf{flip}_1(1, \mathit{tail})\mathsf{flip}_2\mathit{same}\mathsf{root}_2\mathit{root}_2,\\
&\mathit{init}\mathsf{flip}_1(1, \mathit{tail})\mathsf{flip}_2\mathit{same}\mathsf{root}_1\mathit{root}_1,\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{head})\mathsf{flip}_1(\mathit{both}, \mathit{tail}_1)\mathsf{root}_1\mathit{root}_1,\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{head})\mathsf{flip}_1\mathit{same}\mathsf{root}_1\mathit{root}_1,\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{head})\mathsf{flip}_1\mathit{same}\mathsf{root}_2\mathit{root}_2,\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{tail})\mathsf{flip}_1(\mathit{both}, \mathit{tail}_2)\mathsf{root}_2\mathit{root}_2,\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{tail})\mathsf{flip}_1\mathit{same}\mathsf{root}_2\mathit{root}_2,\\
&\mathit{init}\mathsf{flip}_2(2, \mathit{tail})\mathsf{flip}_1\mathit{same}\mathsf{root}_1\mathit{root}_1\}
\end{aligned}$$

Then it is easy to see that the fair executions of I^- are given by

$$\mathit{fexecs}(I^-) = L^* \cdot R \cup L^\infty.$$

Now, let E be a probabilistic execution of I with trace distribution H. It follows by induction that $0 \leq \mathbf{P}_E[\cup_{\beta \in L^n} C_\beta] \leq \frac{1}{2^n}$ for all $n \in \mathbb{N}$. Then

$$\mathbf{P}_E[L^\infty] = \mathbf{P}_E[\cap_{n\in\mathbb{N}} \cup_{\beta\in L^n} C_\beta] = \lim_{n\to\infty} \mathbf{P}_E[\cup_{\beta\in L^n} C_\beta] = 0$$

and therefore,

$$\mathbf{P}_E[\{\mathsf{root}_1, \mathsf{root}_2\}] \geq \mathbf{P}_E[L^* \cdot R] = \mathbf{P}_E[L^* \cdot R \cup L^\infty] = \mathbf{P}_E[\mathit{fexecs}(I^-)] = 1.$$

□

Lemma 7.15 $I \sqsubseteq_{\mathrm{FTD}} A$.

Proof: Let H be a fair trace distribution of I. Then $\mathbf{P}_H[\{\mathit{root}_1, \mathit{root}_2\}] = 1$ and $\mathbf{P}_H[\{\mathit{root}_1\}] \geq \frac{1}{4}$, say $p = \mathbf{P}_H[\{\mathit{root}_1\}] - \frac{1}{4}$. The diagram below represents a fair probabilistic execution of A with trace distribution H.

$$\begin{tikzcd}
 & \mathit{init} \arrow[dl, "\mathsf{flip}"', "\frac{1}{4}"] \arrow[d, "\mathsf{flip}"', "\frac{1}{2}"] \arrow[dr, "\mathsf{flip}", "\frac{1}{4}"'] & \\
\mathit{tail}_1 \arrow[d, "\mathsf{root}_1"'] & \mathit{same} \arrow[dl, "\mathsf{root}_1"', "p"] \arrow[dr, "\mathsf{root}_2", "1-p"'] & \mathit{tail}_2 \arrow[d, "\mathsf{root}_2"] \\
\mathit{root}_1 & & \mathit{root}_2
\end{tikzcd}$$

So $H \in \mathit{ftrdistr}(A)$. □

Fair trace distribution inclusion for A and S

This section proves that A implements S by an ad hoc probabilistic argument. Notice that $A^- \not\sqsubseteq_{\mathrm{FTD}} S^-$, which means that the probabilistic nature of the protocol is crucial for fairness (c.q. liveness).

Lemma 7.16 $A \sqsubseteq_{\mathrm{FTD}} S$.

Proof: Firstly, we prove that $\mathbf{P}_H[\{root_1, root_2\}] = 1$ for all fair trace distributions H of A. Let H be a fair trace distribution in A and let E be a fair probabilistic execution with trace distribution H. We show that $\mathbf{P}_E[fexecs(A^-)] = 1$. Firstly, define

$$L \stackrel{\mathrm{d}}{=} init\,\mathsf{flip}\,same\,\mathsf{retry};$$

$$\begin{aligned} R \stackrel{\mathrm{d}}{=} \{ & init\,\mathsf{flip}\,tail_1\,\mathsf{root}_1\,root_1,\\ \{ & init\,\mathsf{flip}\,same\,\mathsf{root}_1\,root_1,\\ \{ & init\,\mathsf{flip}\,tail_2\,\mathsf{root}_2\,root_2,\\ \{ & init\,\mathsf{flip}\,same\,\mathsf{root}_2\,root_2\}. \end{aligned}$$

Then, similarly to the proof of Lemma 7.14, we have

$$fexecs(A^-) = L^*{\cdot}R \cup L^\infty.$$

Furthermore,

$$\mathbf{P}_E[L^\infty] = \mathbf{P}_E[\textstyle\bigcap_{n\in\mathbb{N}} C_{L^n}] = \lim_{n\to\infty} \mathbf{P}_E[C_{L^n}] = 0,$$

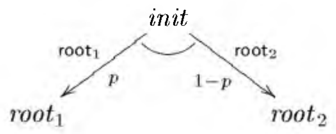
because $0 \le \mathbf{P}_E[C_{L^n}] \le \frac{1}{2^n}$ for all n. So,

$$\mathbf{P}_E[L^*{\cdot}R] = 1$$

and as all these probabilistic executions have trace $root_1$ or $root_2$,

$$\mathbf{P}_H[\{root_1, root_2\}] = \mathbf{P}_E[trace^{-1}\{root_1, root_2\}] \ge \mathbf{P}_E[L^*{\cdot}R] = 1.$$

Therefore, $\mathbf{P}_H[\{root_1, root_2\}] = 1$. Now, let $\mathbf{P}_H[root_1] = p$. Then



represents a fair probabilistic execution of S with trace distribution H. So $H \in ftrdistr(S)$. □

Corollary 7.17 $P \sqsubseteq_{\mathrm{FTD}} S$.

8 Conclusions and further research

The main result established in this paper is that the root contention subprotocol in IEEE 1394 is correct in a model that abstracts from real–time aspects. The formal verification carried out in the probabilistic automaton model model from [Seg95] was not too complex. Actually, most of the work concerned the analysis of non-probabilistic aspect of the protocol. A similar conclusion was drawn in [PSL97] for the verification of the Aspnes and Herlidy protocol.

As a first approach to this protocol, we have verified the protocol in an discrete–time model. A follow-up of this study, dealing with real–time is reported in [SV99b]. In fact, we have modeled a discrete time model, where time can have integer values, into a untimed model using an action tick. As tick is neither an input not an output action, we dropped distinction between input and output actions and only distinguish between internal and external actions. The notion of parallel composition therefore becomes more general but also somewhat artificial. However, the absence of time allowed us to use tools for untimed systems.

We have introduced several techniques and strategies to make the verification easier, which can be useful in the analysis of other realistic protocols and algorithms. Some of these techniques give rise to new questions and topics for future research.

1. Within our verification, we introduced two intermediate automata between the specification and the implementation. This allowed a separation of concern and the use of existing tools for the analysis of non-probabilistic systems. The actual probabilistic verification was carried out on an automaton with less than ten states. Furthermore, we introduced two probabilistic simulation relations. These are less general than the simulation relations from [SL95] but they simplified the verification to a large extend.

2. We have proven a result that reduces reasoning about fair probabilistic executions to reasoning about fairness of (non-probabilistic) executions. We are aiming at a more general result that relates probabilistic execution properties (such as fairness) to execution properties via probabilistic step refinements and probabilistic hyperstep refinements.

3. We have described a method for encoding finite I/O automata in precondition effect-style and invariant properties into the SMV input language. It would be desirable to describe this relation more formally and to prove the claims made in this report more formally. Moreover, such a rigorous description would be a first step to an automatic translation from the IOA language [GLV97], which is a formal syntax for I/O automata in precondition-effect style, to SMV. Although IOA-to-PVS, IOA-to-Larch and IOA-to-Spin compilers are currently being constructed, an IOA-to-SMV compiler is still useful because SMV is easy–to–use model checker, whereas PVS and Larch are interactive proof tools.

Another interesting topic for future research would be to use tools in the verification of the probabilistic aspects of the protocol as well. Furthermore, it should not be to difficult to do some performance analysis, for instance one could calculate upper and lower bounds on the expected number of ticks before a leader is elected.

Acknowledgments

Thanks to Judi Romijn and to David Griffioen for their useful comments on earlier versions of the specification and to Marco Devillers for an introduction to SMV.

References

[Agg94] S. Aggarwal. Time optimal self-stabilizing spanning tree algorithms. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994. Available as Technical Report MIT/LCS/TR-632.

[DGRV97] M.C.A. Devillers, W.O.D. Griffioen, J.M.T Romijn, and F.W. Vaandrager. Verification of a leader election protocol — formal methods applied to IEEE 1394. Technical Report CSI-R9728, Computing Science Institute, University of Nijmegen, December 1997. Submitted.

[GLV97] S.J. Garland, N.A. Lynch, and M. Vaziri. IOA: A language for specifiying, programming, and validating distributed systems, September 1997. Available through URL `http://larch.lcs.mit.edu:8001/~garland/ioaLanguage.html`.

[Hal50] P.R. Halmos. *Measure Theory.* Van Nostrand Reinhold Company Inc, New York, 1950.

[IEE96] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, August 1996.

[IEE98] IEEE Computer Society. P1394a Draft Standard for a High Performance Serial Bus (Supplement). Draft 2.0, March 1998.

[Lut97] S.P. Luttik. Description and formal specification of the Link layer of P1394. In I. Lovrek, editor, *Proceedings of the 2nd International Workshop on Applied Formal Methods in System Design,* Zagreb, pages 43–56, 1997. Also available as Report SEN-R9706, CWI, Amsterdam. See URL `http://www.cwi.nl/~luttik/`.

[LV96] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[Lyn96] N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Fransisco, California, 1996.

[McM92] K.L. McMillan. The smv system, draft, February 1992. Available through URL `http://www.cs.cmu.edu/~modelcheck/smv.html`.

[PSL97] A. Pogosyants, R. Segala, and N.A. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. In M. Mavronicolas and Ph. Tsigas, editors, *Proceedings of 11th International Workshop on Distributed Algorithms (WDAG'97),* Saarbrucken, Germany, September 1997, volume 1320 of *Lecture Notes in Computer Science*, pages 111–125. Springer-Verlag, 1997. Also, Technical Memo MIT/LCS/TM-555, Laboratory for Computer Science, Massachusetts Institute of Technology.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.

[Sha98] C. Shankland. *The Tree Identify Protocol of IEEE 1394*, pages 299–319. Stichting Mathematisch Centrum, 1998.

[SL95] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[SV99a] M.I.A. Stoelinga and F.W. Vaandrager. Gambling together in Monte Carlo: Step refinements for probabilistic automata. Report CSI-R99xx, Computing Science Institute, University of Nijmegen, Nijmegen, 1999. To appear.

[SV99b] M.I.A. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. Report CSI-R9905, Computing Science Institute, University of Nijmegen, Nijmegen, 1999.