A Probabilistic Model for Parametric Fairness in Isabelle/HOL ⋆

Jinshuang Wang, Xingyuan Zhang, Yusen Zhang, and Huabing Yang

PLA University of Science and Technology, Nanjing 210007, China
wangjinshuang@gmail.com, xingyuanz@gmail.com, zys49_nj@126.com, yanghuabing@gmail.com

Abstract. In paper [1], a liveness proof method suitable for inductive protocol verification is proposed. The utility of this method has been confirmed by several machine checked formal verifications[2–4]. One remaining question about [1] is the meaning of *Parametric Fairness*, a new fairness notion adapted from Pnueli's *Extreme Fairness*[5] to suit the setting of higher-order logic. This paper tries to answer this question. As a standard practice in establishing a fairness notion, this paper constructs a probabilistic model for parametric fairness in Isabelle/HOL. Using this model, it is shown that most infinite executions of a concurrent system are parametrically fair. Therefore the definition of parametric fairness in paper [1] is reasonable. This work gives a firmer basis for existing and forthcoming formal verifications based on the method of paper [1].

Keywords : Liveness Proof, Inductive Protocol Verification, Probabilistic Model, Parametric Fairness.

1 Introction

Paulson's inductive approach for protocol verification[6] has been used to verify fairly complex security protocols [7, 8]. The success gives incentives to extend this approach to a general approach for concurrent system verification. To achieve this goal, a method for the verification of liveness properties is needed.

In paper [1], a liveness proof method suitable for inductive protocol verification is proposed. In [1], proof rules for liveness properties (both response and reactivity) are derived. The proof rules are used to reduce the proof of liveness properties to the proof of safety properties, so that the original inductive approach's advantages in the proof of safety properties can be fully exploited. The utility of this method has been confirmed by several machine checked formal verifications[2–4].

Paper [1] uses a new notion of fairness – *Parametric Fairness*, which is an adaption of Pnueli's *Extreme Fairness*[5] to suit the setting of higher-order logic. As a standard practice in establishing a fairness notion, this

⋆ This research was funded by National Natural Science Foundation of China , under grant 60373068 'Machine-assisted correctness proof of complex programs' paper constructs a probabilistic model for *Parametric Fairness*. Using this model, it is shown that most infinite executions of a concurrent system are parametrically fair. Therefore the definition of parametric fairness given in [1] is reasonable. The purpose of this paper is to give a firmer basis for existing and forthcoming formal verifications based on the method of paper [1].

As pointed out in [5], standard fairness notions like *weak fairness* and *strong fairness* are not adequate for showing some obvious liveness properties. Consider the concurrent system in Figure 1. It is intuitively obvious that once the system is in initial state 3, it will eventually get into the final state 0, supposing all actions in the system are fairly treated. However, both *weak fairness* and *strong fairness* fail to capture such an intuitive notion. For example, the infinite execution $(e_2.\ e_1.\ e_4.\ e_0)^\omega$ is fair according to both *weak fairness* and *strong fairness*. The problem with *weak fairness* and *strong fairness* is that they only stress the fair treatment of actions, without considering the state under which these actions are taken. In execution $(e_2.\ e_1.\ e_4.\ e_0)^\omega$, even all actions in the system, i.e. e_0, e_1, e_2, e_4, are fairly treated, but it is still not strong enough to force the system into state 0, therefore the intuitively obvious liveness property mentioned above does not hold under this standardly fair execution.

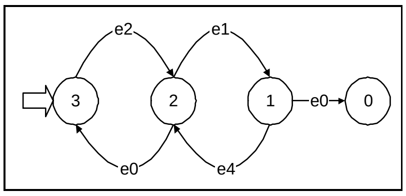

Fig. 1. The diagram of a concurrent system

To solve this problem, paper [5] proposes the notion of extreme fairness which requires all combinations of states and actions be fairly treated. In Figure 1, this amounts to requiring all pairs in $\{\mathit{0},\ \mathit{1},\ \mathit{2},\ \mathit{3}\}\times\{e_0,\ e_1,\ e_2,\ e_4\}$ be fairly treated. The above standardly fair execution $(e_2.\ e_1.\ e_4.\ e_0)^\omega$ is not extremely fair, because the pair $(\mathit{1},\ e_0)$ is not fairly treated. To specify that every state-action pair be fairly treated in higher-order logic, we have to quantify universally over the type of state predicates. This universal quantification turned out to be problematic, because for any infinite execution σ, we can always construct a higher-order state predicate φ_σ which is not fairly treated in σ. By instantiating the universally quantified variable to φ_σ, it can be shown that execution σ is not fair. Therefore the universal quantification over the type of state predicates makes the straightforward formulation of extreme fairness so restrictive that almost no execution can satisfy it. The details of this construction can be found in paper [1].

As a solution, paper [1] proposes *Parametric Fairness* as a refinement of *extreme fairness*. In parametric fairness, the list of state-action pairs which need to be fairly treated is explicitly given as a parameter to the fairness definition. Because in any concrete liveness proof, the state-action pairs which need to be fairly treated can always be given explicitly, the notion of parametric fairness achieves the same proof theoretical power as extreme fairness without being too restrictive semantically.

One remaining question about parametric fairness is whether the notion is liberal enough to accommodate most infinite executions of the underlying concurrent system. This paper answers the question by showing that if the nondeterminacy in concurrent executions is resolved by random choice, a probability space can be constructed with the execution set as basis, and that the set of parametrically fair executions is measurable and has probability 1. The result means that almost all concurrent executions are parametrically fair.

Similar works have already been done for extreme fairness and other notions of fairness[5, 9, 10], but none is in the setting of higher-order logic. Our contribution is to show that the formulation of parametric fairness in higher-order logic still has the desired property. Additionally, our work is more reliable because all preceding works are pencil and paper ones done outside object logic, while ours is inside the object logic Isabelle/HOL, machine-checked and coexisting with concrete verifications. Parametric fairness has the same motivation as extreme fairness, i.e. to reflect the probabilistic nature of concurrent executions. This paper shows that the goal is indeed achieved. Therefore the liveness proof method proposed in [1] is now on a firmer basis.

The rest of this paper is organized as follows: Section 2 introduces the notion of parametric fairness. Section 3 introduces the notion of probabilistic execution and constructs a probability space on sets of infinite executions. Section 4 assigns a probabilistic meaning to parametric fairness by showing that the probability of the set of fair executions equals to 1. Section 5 summarizes related works. Section 6 concludes.

2 The notion of parametric fairness

2.1 Concurrent systems

In inductive approach, system states are identified with finite executions, which are represented as lists of events. These events are arranged in reverse order of happening, the decision of which event to happen next is decided according to current system state. Formal definitions for concurrent system is given in Fig. 2, where the type of events is a polymorphic type $'a$, and the type of system states is $'a\ list$. We identify system states with finite execution traces. The terms *system state* and *finite execution* are used interchangeably. Both of them are written as τ, τ', τ_1, τ_2, etc.

```
constdefs i-th :: (nat ⇒ 'a) ⇒ nat ⇒ 'a (-- [64, 64] 1000)
  σ_i ≡ σ i

consts prefix :: (nat ⇒ 'a) ⇒ nat ⇒ 'a list ([[-]]- [64, 64] 1000)
primrec [[σ]]_0 = []
          [[σ]]_(Suc i) = σ_i # [[σ]]_i

constdefs may-happen ::
  'a list ⇒ ('a list × 'a) set ⇒ 'a ⇒ bool (- [-> - [64, 64, 64] 50)
  τ [cs> e ≡ (τ, e) ∈ cs

consts vt :: ('a list × 'a) set ⇒ 'a list set
inductive vt cs
  intros
  vt-nil [intro] : [] ∈ vt cs
  vt-cons [intro] : [[τ ∈ vt cs; τ [cs> e]] ⟹ (e # τ) ∈ vt cs

consts derivable :: 'a ⇒ 'b ⇒ bool (- ⊢ - [64, 64] 50)
defs (overloaded)
  fnt-valid-def:  cs ⊢ τ ≡ τ ∈ vt cs
  inf-valid-def:  cs ⊢ σ ≡ ∀i. [[σ]]_i [cs> σ_i
```

Fig. 2. The definitions of concurrent system

The type of infinite executions is $nat \Rightarrow 'a$, which is often abbreviated as $'a\ seq$. For an infinite execution σ, the event happens at the $i{-}th$ step is $\sigma\ i$, which is also written as σ_i. The first i events of an infinite execution σ can be packed into a list, which exactly forms a finite execution. Such a packing is written as $[\![\sigma]\!]_i$, which is also called a *prefix* of σ.

A concurrent system is often written as cs, and its type is $('a\ list \times 'a)\ set$. The expression $(\tau,\ e) \in cs$ means that the event e is legitimate to happen under state τ according to cs. The notation $(\tau,\ e) \in cs$ is also written as $\tau\ [cs{>}\ e$. The set of *valid* finite executions of cs is written as $vt\ cs$. The expression $\tau \in vt\ cs$ is also written as $cs \vdash \tau$. The operator $\vdash$ is overloaded, so that σ is a valid infinite execution of cs can be written as $cs \vdash \sigma$. An infinite execution σ is valid under cs iff all of its prefixes are valid.

2.2 Embedding LTL

LTL (Linear Temporal Logic) is widely used for the specification and verification of concurrent systems. A shallow embedding of LTL is given in Fig.3. In this paper, LTL is used to express various temporal properties of infinite executions, including liveness properties.

```
types ′a tlf = (nat ⇒ ′a) ⇒ nat ⇒ bool

consts valid-under :: ′a ⇒ ′b ⇒ bool (- ⊨ -  [64, 64] 50)
defs (overloaded)  pr ⊨ φ ≡ let (σ, i) = pr in φ σ i
defs (overloaded)  σ ⊨ φ ≡ (σ::nat ⇒ ′a, (0::nat)) ⊨ φ

constdefs always :: ′a tlf ⇒ ′a tlf  (□- [64] 65)
  □φ ≡ λ σ i. ∀ j. i ≤ j ⟶ (σ, j) ⊨ φ

constdefs eventually :: ′a tlf ⇒ ′a tlf (◇- [64] 65)
  ◇φ ≡ λ σ i. ∃ j. i ≤ j ∧ (σ, j) ⊨ φ
```

Fig. 3. A shallow embedding of LTL

The type of LTL formulae is defined as *′a tlf*. The expression $(\sigma,\ i) \models \varphi$ means that LTL formula φ is valid at the $i{-}th$ step of σ. The operators *always* $\Box$ and *eventual* $\Diamond$ are defined literally.

2.3 Introduction of parametric fairness

In paper [1], two liveness proof rules are derived. The one for response properties is:

$$[\![RESP\ cs\ F\ E\ N\ P\ Q;\ cs \vdash \sigma;\ PF\ cs\ \{\!|F,\ E,\ N|\!\}\ \sigma]\!] \Longrightarrow \sigma \models \Box\langle\psi\rangle \hookrightarrow \Diamond\langle\varphi\rangle$$

and the one for reactivity properties is:

$$[\![REACT\ cs\ F\ E\ N\ P\ Q;\ cs \vdash \sigma;\ PF\ cs\ \{\!|F,\ E,\ N|\!\}\ \sigma]\!] \Longrightarrow \sigma \models \Box\Diamond\langle\psi\rangle \hookrightarrow \Box\Diamond\langle\varphi\rangle$$

To prove a liveness result, an execution path must be found, which goes from the starting state characterized by ψ to the ending state characterized by φ. Such a path is represented by a list of (state predicate, event function)-pairs, which needs to be fairly treated, so that the execution of the concurrent system will eventually go along the path. The path is abstract in the sense that instead of concrete (state, event)-pairs, it consists of (state predicate, event function)-pairs, where the state predicate specifies the condition under which the event yielded by the event function is to happen. The functions F and E are given by verification staff to generate the state predicates and event functions respectively. The N is the length of the path. The expression $\{\!|F,\ E,\ N|\!\}$ represents the generated (state predicate, event function)-pair list. The pair list should be properly designed to form a chain leading from ψ to φ. Such a requirement is expressed in the definitions of both $RESP$ and $REACT$[1].

The purpose of this paper is to give a probabilistic meaning to the remaining *PF cs* $\{\!|F,\ E,\ N|\!\}$ σ, which requires that the infinite execution σ is parametrically fair with respect to the parameter $\{\!|F,\ E,\ N|\!\}$. The formal definition of *PF* is given in Fig. 4.

```
  EFα :: ('a list × 'a) set ⇒ ('a list ⇒ bool) ⇒ ('a list ⇒ 'a) ⇒ (nat ⇒ 'a) ⇒ bool
  EFα cs P E σ ≡
  σ ⊨ □◇(λ σ i. P ⟦σ⟧i ∧ ⟦σ⟧i [cs> E ⟦σ⟧i) ⟶ σ ⊨ □◇(λ σ i. P ⟦σ⟧i ∧ σi = E ⟦σ⟧i)

  EF :: ('a list × 'a) set ⇒ (nat ⇒ 'a) ⇒ bool
  EF cs σ ≡ ∀ P E. EFα cs P E σ

types 'a pe = ('a list ⇒ bool) × ('a list ⇒ 'a)

  PF :: ('a list × 'a) set ⇒ 'a pe list ⇒ (nat ⇒ 'a) ⇒ bool
  PF cs pel σ ≡ list-all (λ (P, E). EFα  cs P E σ) pel
```

Fig. 4. Different notions of fairness

In Fig. 4, expression $EF_\alpha\ cs\ P\ E\ \sigma$ specifies that state predicate P and event function E are fairly treated in execution σ, expression $EF\ cs\ \sigma$ is the literal translation of extreme fairness, which requires *every* state predicate and event function being fairly treated. This definition is shown in [1] to be too restrictive. The definition of *PF cs pel* σ represents our remedy, which only requires the (state predicate, event function)-pairs appearing in parameter *pel* being fairly treated. In the following, we are going to construct a measure space over the power set of all infinite executions, and show that *almost all* infinite executions satisfy the requirement of *PF*. However, since our approach works at meta level, the premise *PF cs* $\{\!|F,\ E,\ N|\!\}$ σ can not be removed from the final results. It is there to provide people with information that the liveness result is achieved through the enabling execution path $\{\!|F,\ E,\ N|\!\}$.

When a complex concurrent system is verified, people usually need to derive many liveness results, each with its own enabling path. The result in this paper also means that the requirement of *PF cs pel* σ, using the conjunction of all these enabling paths as the parameter *pel*, can still be satisfied by *almost all* infinite executions.

3 Probability space construction

3.1 Formalizing probabilistic execution

To model random choice, we introduce a function R, where $R(\tau,\ e)$ is the probability of event e being chosen to happen under system state τ. Therefore, $R(\tau,\ e) = 0$ means event e is not eligible to happen under

state τ, while $R(\tau,\ e) > 0$ means e is eligible to happen with $R(\tau,\ e)$ as the happening probability. Every R represents an execution strategy of a concurrent system. The underlying concurrent system can be defined in terms of R as $CS \equiv \{(\tau,\ e).\ 0 < R\ (\tau,\ e)\}$. The set of events eligible to happen under state τ is given as $N\ \tau \equiv \{e.\ 0 < R\ (\tau,\ e)\}$.
It is natural to assert the following axioms:

1. $0 \leq R\ (\tau,\ e) \wedge R\ (\tau,\ e) \leq 1$, which is a routine requirement of probability theory.
2. $CS \vdash \tau \longrightarrow (\sum e{\in}N\ \tau.\ R\ (\tau,\ e)) = 1$. The purpose of this axiom is to fulfill the standard probability theory requirement that the summation of all possible outcomes must equal to 1. This axiom also entails that every valid finite execution can always be extended by at least one event. By introducing a *Tick* event, which represents the ticks of a system wide clock, this requirement can be satisfied easily, since nothing can prevent time from advancing. This axiom also entails *finite* $(N\ \tau)$, which means that the underlying concurrent system CS is finitely branching.
3. $\forall \tau\ e.\ 0 < R\ (\tau,\ e) \longrightarrow bnd \leq R\ (\tau,\ e)$, where bnd is the lower bound of all nonzero R-probabilities. It is natural to require that $0 < bnd \wedge bnd < 1$.

Funcion R induces a measure function π on finite executions:

$$\pi\ [] = 1$$
$$\pi\ (e\#\tau) = R(\tau,\ e) * \pi\ \tau$$

For any valid system state τ, we have $\pi\ \tau > 0$. For any valid infinite execution σ, we have $\forall\ i.\ \pi\ [\![\sigma]\!]_i > 0$. The base set(or sample space), on which we are going to construct the probability space, is defined as:

$$Path \equiv \{\sigma.\ (\forall\ i.\ \pi\ [\![\sigma]\!]_i > 0)\}$$

It can be shown that $(CS \vdash \sigma) = (\sigma \in Path)$, i.e. $Path$ coincides with the set of all valid infinite executions of the concurrent system CS.

3.2 Outline of the construction

The definition of probability space is given in Fig. 5. A probability space is defined to be a measure space $(U,\ F,\ Pr)$, where U is the base set, F the family of measurable sets (also called measurables), Pr the measure function. For a probability space $(U,\ F,\ Pr)$, the measure of the base set must be 1. The definition of measure space uses standard notions such as σ-algebra, positivity and countable additivity. In the definition of countable additivity, we use Isabelle library function *sums*, where '*f sums c*' stands for $\sum_{n=0}^{\infty} f(n) = c$. The rest of the definitions are self-explaining.

```
consts algebra :: ('a set × 'a set set) ⇒ bool
  algebra (U, F) = (F ⊆ Pow(U) ∧ {} ∈ F ∧ (∀a∈F. (U − a) ∈ F) ∧
                   (∀a b. a ∈ F ∧ b ∈ F ⟶ a ∪ b ∈ F))

consts sigma-algebra ::('a set × 'a set set) ⇒ bool
  sigma-algebra(U, F) = (F ⊆ Pow(U) ∧ U ∈ F ∧ (∀a ∈ F. U − a ∈ F) ∧
                        (∀a. (∀ i::nat. a(i) ∈ F) ⟶ (⋃i. a(i)) ∈ F))

consts positive:: ('a set set × ('a set ⇒ real)) ⇒ bool
  positive(F, Pr) = (Pr {} = 0 ∧ (∀A. A ∈ F ⟶ 0 ≤ Pr A))

consts countably-additive:: ('a set set × ('a set ⇒ real)) ⇒ bool
  countably-additive(F, Pr) =
    (∀f::(nat ⇒ 'a set). range(f) ⊆ F ∧
                        (∀m n. m ≠ n ⟶ f(m) ∩ f(n) = {}) ∧
                        (⋃i. f(i)) ∈ F
                    ⟶ (λn. Pr(f(n))) sums Pr (⋃i. f(i)))

consts measure-space:: ('a set × 'a set set × ('a set ⇒ real)) ⇒ bool
  measure-space (U, F, Pr) =
     (sigma-algebra (U, F) ∧ positive (F, Pr) ∧ countably-additive (F, Pr))

consts prob-space:: ('a set × 'a set set × ('a set ⇒ real)) ⇒ bool
  prob-space (U, F, Pr) = (measure-space (U, F, Pr) ∧ Pr U = 1)
```

Fig. 5. Definition of probability space

The notion of σ-algebra is a generalization of algebra. As shown in Fig. 5, the only difference is that algebra is closed under finite union, while σ-algebra is closed under countable union. A standard way to obtain σ-algebra is to construct an algebra $(U,\ F)$ first, then use operator *sigma* to generate a σ-algebra $(U,\ \textit{sigma}(U,F))$. The definition of *sigma* is:

consts *sigma* :: $('a\ set \times 'a\ set\ set) \Rightarrow 'a\ set\ set$
inductive *sigma M* **intros**
  basic: $(\textit{let}\ (U,\ A) = M\ \textit{in}\ (a \in A)) \Longrightarrow a \in \textit{sigma}\ M$
  empty: $\{\} \in \textit{sigma}\ M$
  complement: $a \in \textit{sigma}\ M \Longrightarrow (\textit{let}\ (U,\ A) = M\ \textit{in}\ U - a) \in \textit{sigma}\ M$
  union: $(\bigwedge i{::}nat.\ a\ i \in \textit{sigma}\ M) \Longrightarrow (\bigcup i.\ a\ i) \in \textit{sigma}\ M$

According to *Carathéodory's extension theorem*[11, 12], a measure Pr on F can be generalized naturally to a measure Pr' on $\textit{sigma}(U,F)$, so that $(U,\ \textit{sigma}(U,F),\ Pr')$ forms a measure space. The presentation of this extension theorem in Isabelle/HOL is as follows:

$[\![\textit{algebra}\ (U,\ F);\ \textit{positive}\ (F,\ Pr);\ \textit{countably-additive}\ (F,\ Pr)]\!]$
$\Longrightarrow \exists P.\ (\forall A.\ A \in F \longrightarrow P\ A = Pr\ A) \wedge \textit{measure-space}\ (U,\ \textit{sigma}\ (U,\ F),\ P)$

This theorem suggests the way we are going to construct the probability space for infinite executions. The base set is *Path*, on top of which an algebra *PA* will be defined. A measure μ on *PA* will be given with $\mu(Path)=1$. Using extension theorem, we can then get the desirable probability space.

3.3 An algebra of infinite execution sets with measure μ

The set of infinite executions with finite execution τ as prefix is defined as follows:

$palg\text{-}embed\ \tau \equiv \{\sigma \in Path.\ [\![\sigma]\!]_{|\tau|} = \tau\}$

The $|\tau|$ in this definition is the length of τ.

A set of infinite executions is said to be supported by a list of finite executions $[\tau_0, \tau_1, \tau_2, \ldots, \tau_n]$, if every infinite execution in the set is prefixed by some τ_i among $\tau_0, \tau_1, \tau_2, \ldots, \tau_n$. It is easy to see that the set supported by $[\tau_0, \tau_1, \tau_2, \ldots, \tau_n]$ is $\bigcup_{i=0}^{n} (palg\text{-}embed\ \tau_i)$. But we write the supported set in a slightly different way as $palgebra\text{-}embed([\tau_0, \tau_1, \tau_2, \ldots, \tau_n])$, which is defined as the following:

consts $palgebra\text{-}embed :: 'a\ list\ list \Rightarrow 'a\ seq\ set$
primrec
  $palgebra\text{-}embed\ [] = \{\}$
  $palgebra\text{-}embed\ (\tau \# l) = (palg\text{-}embed\ \tau) \cup palgebra\text{-}embed\ l$

The subset family *PA* mentioned earlier is defined as:

$PA \equiv \{S.\ \exists\, l.\ palgebra\text{-}embed\ l = S \wedge S \subseteq Path\}$

The set family *PA* can be understood as consisting of only those sets of valid infinite executions which are supported by some list of finite executions.

We intend to base the measure of a set $S \in PA$ on the measures of its supporting lists. For this purpose, the measure μ_0 is defined as:

$\mu_0\ l \equiv (\sum \tau \in set\ l.\ \pi\ \tau)$

However, a set S may be supported by many different lists with different μ_0-values. It is natural to define the measure of S to be the lower limit of these μ_0-values. The following measure μ_1 is a refinement of μ_0 to serve this intention:

$\mu_1\ l \equiv inf\ (\lambda r.\ \exists\, l'.\ palgebra\text{-}embed\ l = palgebra\text{-}embed\ l' \wedge \mu_0\ l' = r)$

Measure μ_1 will always return the measure of S properly, no matter which l among the supporting lists of S is used as argument.

Due to the restriction of Isabelle/HOL, we can not give a measure on *PA* explicitly. To solve this problem, the following measure μ is defined on the type of all infinite execution sets, not just those in *PA*:

constdefs μ:: *'a seq set* $\Rightarrow$ *real*
  $\mu\ S \equiv sup\ (\lambda r.\ \exists\, b.\ \mu_1\ b = r \wedge (palgebra\text{-}embed\ b) \subseteq S)$

It can be shown that $[\![S \in PA;\ S = palgebra\text{-}embed(l)]\!] \Longrightarrow \mu(S) = \mu_1(l)$, which means the formal definition of μ is in accordance with our informal intentions discussed above. The definitions of μ and μ_1 are copied from [11].

We proved that *algebra* $(Path,\ PA)$, *positive* $(PA,\ \mu)$ and *countably-additive* $(PA,\ \mu)$. These are precisely the conditions required by *Carathéodory's extension theorem.* By applying the extension theorem, we get the probability space $(Path,\ sigma(Path,\ PA),\ P)$, on which we are going to prove that the set of parametrically fair executions is measurable and has probability 1.

The proof of *algebra* $(Path,\ PA)$ is straightforward and skipped. The proof of *countably-additive* $(PA,\ \mu)$ is based on the fact that:

$[\![range\ f \subseteq PA;\ \bigwedge m\ n.\ m \neq n \Longrightarrow f\ m \cap f\ n = \{\};\ (\bigcup n\ f\ n) \in PA]\!]$
$\Longrightarrow \exists\, N.\ \forall\, n.\ N \leq n \longrightarrow f\ n = \{\}$

The proof of this lemma is done using an argument similar to the proof of *König's Lemma*[12]. This lemma means that for any family f of subsets with $(\bigcup i.\ f\ i) \in PA$, there are only finite many $f(i)$s that are nonempty. Therefore, countable additivity is reduced to finite additivity, which is proved easily by induction.

Some routine properties of probability space are listed as below:

- *Monotonicity:* If A and B are measurable and $A \subseteq B$, then $P(A) \leq P(B)$.
- *Subadditivity:* If family A_i is measurable, then $P(\bigcup_{i=0}^{\infty} A_i) \leq \sum_{i=0}^{\infty} P(A_i)$.
- *Lower-Continuity:* If $A_0 \subseteq A_1 \subseteq \ldots \subseteq A_n \ldots$ is a chain of measurables, then $\lim_{i\to\infty} P(A_i) = P(\bigcup_{i=0}^{\infty} A_i)$.
- If $\forall\, i.\ P(f(i)) = 1$, and $n \neq 0$, then $P(\bigcap i \in \{0..<n\}.\ f(i)) = 1$.

4 The probabilistic meaning of parametric fairness

The purpose of this section is to show that:

$$set\ pel \neq \{\} \Longrightarrow P\ \{\sigma \in Path.\ PF\ CS\ pel\ \sigma\} = 1 \qquad (1)$$

which means the probability of the set of valid parametrically fair executions is 1. This result gives a probabilistic meaning to parametric fairness by showing that almost all valid probabilistic executions are parametrically fair. All the sets, which appear as arguments to P in this section, have been proved to be measurable in the probability space constructed in Section 3.

For this purpose, the following lemma is proved:

$$\mathit{set\ pel} \neq \{\} \Longrightarrow \{\sigma \in \mathit{Path}.\ \mathit{PF\ CS\ pel}\ \sigma\} = (\bigcap_{\iota \in \mathit{set\ pel}} \mathit{Fair}\iota\ \iota\ \mathit{CS\ lf})$$

to reduce the calculation of $P\ \{\sigma \in \mathit{Path}.\ \mathit{PF\ CS\ pel}\ \sigma\}$ to the calculation of $P(\ \bigcap_{\iota \in \mathit{set\ pel}} \mathit{Fair}\iota\ \iota\ \mathit{CS\ lf})$. The *Fairι* represents a generalized way to present fairness[10], where fairness is defined relative to *label*s. The definition of *Fairι* is given in Fig. 6.

enabled ι cs γ $\tau \equiv \exists$e. $(\iota \in \gamma\ (\tau, \text{e}) \wedge (\tau, \text{e}) \in \text{cs})$

taken ι cs γ $\tau \equiv \iota \in \gamma\ (\text{tl}(\tau), \text{hd}(\tau))$

fairι ι cs γ $\sigma \equiv$
$\sigma \models \Box\Diamond(\lambda\sigma\ \text{i. enabled}\ \iota\ \text{cs}\ \gamma\ [\![\sigma]\!]_i) \longrightarrow \sigma \models \Box\Diamond(\lambda\sigma\ \text{i. taken}\ \iota\ \text{cs}\ \gamma\ [\![\sigma]\!]_{Suc(i)})$

fair L cs γ $\sigma \equiv (\forall \iota \in \text{L. fair}\iota\ \iota\ \text{cs}\ \gamma\ \sigma)$

Fairι ι cs $\gamma \equiv \{\sigma.\ \text{cs} \vdash \sigma \wedge \text{fair}\iota\ \iota\ \text{cs}\ \gamma\ \sigma\}$

lf $(\tau, \text{e}) = \{(\text{Q, E}).\ \text{Q}\ \tau \wedge \text{e} = (\text{E}\ \tau)\}$

Fig. 6. A general fairness notion

The definition of *Fairι* uses *fairι*, where *fairι ι cs* γ σ means the particular execution σ is fair to label ι. An infinite execution is said to be fair to a label ι if ι is taken infinite many times whenever it is enabled infinite many times in σ. The parameter γ in both *Fairι ι cs* γ and *fairι ι cs* γ σ is used to assign label sets to events so that the notions *enabled* and *taken* can be made precise. For any event e, expression $\gamma(\tau,e)$ represents the set of labels assigned to e under system state τ. A label ι is said to be enabled in state τ (written as *enabled ι cs* γ τ) if there exists some event e eligible to happen under state τ and ι belongs to $\gamma(\tau,e)$. A label ι is said to be taken in state τ if ι is assigned to the last execution step of τ. The last step of τ is denoted by $hd(\tau)$ and the state before the last step is denoted by $tl(\tau)$.

The expression *Fairι ι cs* γ yields the set of infinite executions of concurrent system *cs*, which are fair to label ι. The expression $\bigcap_{\iota \in \mathit{set\ pel}} \mathit{Fair}\iota\ \iota$ *CS lf* represents executions of *CS* which are fair to every label ι in the set $set(pel)$. In the latter expression, labels are the $(Q,\ E)$-pairs contained in the list *pel*. A $(Q,\ E)$-pair is said to be assigned to (τ,e) if $Q(\tau) \wedge e{=}E(\tau)$ holds. This notion is reflected in the function *lf*, which is used as the parameter γ.

The next step is to show that $P(\ \bigcap_{\iota \in \mathit{set\ pel}} \mathit{Fair}\iota\ \iota\ \mathit{CS\ lf}) = 1$. Since we have $\forall\, i.\ P\ (f_i) = 1 \wedge n \neq 0 \Longrightarrow P\ (\bigcap i \in \{0..{<}n\}.\ f_i) = 1$ for any set family *f*, it suffices to show $P\ (\mathit{Fair}\iota\ \iota\ \mathit{CS\ lf}) = 1$ for any one label ι. Equivalently, it is to show that $P\ (\mathit{Path}{-}(\mathit{Fair}\iota\ \iota\ \mathit{CS\ lf})) = 0$.

The next lemma we have proved is:

$$Path-(Fair\iota\ \iota\ CS\ lf) = (\bigcup_{\tau\in\{xs.\ CS\ \vdash\ xs\}} \Gamma\ \iota\ CS\ lf\ \tau)$$

Now, we only need to show:

$$P(\bigcup_{\tau\in\{xs.\ CS\ \vdash\ xs\}} \Gamma\ \iota\ CS\ lf\ \tau) = 0$$

According to the *subadditivity* of probability measure mentioned in Section 3.3, it is sufficient to show that $P\ (\Gamma\ \iota\ CS\ lf\ \tau) = 0$ for each finite execution τ. The definition of Γ is:

$\Gamma\ \iota\ cs\ \gamma\ \tau \equiv$
$\{\sigma \in palg\text{-}embed\ \tau.$
$(\forall\, i.\ \exists j.\ i \leq j \wedge enabled\ \iota\ cs\ \gamma\ [\![\sigma]\!]_{j\ +\ (|\tau|)}) \wedge$
$(\forall\, j{\geq}|\tau|.\ \neg\ taken\ \iota\ cs\ \gamma\ [\![\sigma]\!]_{Suc\ j})\}$

An infinite execution σ is said to be *unfair to* ι *and indexed by* τ if τ is a prefix of σ and starting from the end of τ, ι is not taken on σ while being enabled infinitely often at the same time. The expression $\Gamma\ \iota\ cs\ \gamma\ \tau$ represents the set of such infinite executions. Suppose σ is in $\Gamma\ \iota\ cs\ \gamma\ \tau$, if we count the number of times ι is enabled after τ, this number will eventually transcend any natural number i. If $UF\ \iota\ cs\ \gamma\ \tau\ i$ is the set of infinite executions (the definition is given in Fig.7), in which the number of times ι is enabled after τ is no less than $i{+}1$, then we have:

$\Gamma\ \iota\ CS\ \gamma\ \tau \subseteq UF\ \iota\ CS\ \gamma\ \tau\ i$

from this, we have $P(\Gamma\ \iota\ CS\ lf\ \tau) \leq P(UF\ \iota\ CS\ lf\ \tau\ i)$. If we can futher prove that $lim_{i\to\infty}P(UF\ \iota\ CS\ lf\ \tau\ i) = 0$, then we can have $P\ (\Gamma\ \iota\ CS\ lf\ \tau){=}0$.

Since all UF-sets are in $sigma(Path,PA)$, and are not supported by finite sets of finite executions. This makes UF-sets difficult to deal with. The set function BUF is defined in Fig. 7 to solve this problem, where the definition limits the value of k to the parameter up, so that every $BUF\ \iota\ CS\ lf\ \tau\ up\ i$ is supported by a corresponding finite set $SUF\ \iota\ CS\ lf\ \tau\ up\ i$ which consists of finite executions. Therefore, we have

$$P\ (BUF\ \iota\ CS\ lf\ \tau\ up\ i) = (\textstyle\sum \tau \in SUF\ \iota\ CS\ lf\ \tau\ up\ i.\ \pi(\tau)) \tag{2}$$

Every finite execution in $SUF\ \iota\ CS\ lf\ \tau\ up\ (i{+}1)$ is extended from some $\tau_1{\in}SUF\ \iota\ CS\ lf\ \tau\ up\ i$ by avoiding all the events to which ι is assigned. Due to the avoidance of these events and (2), we managed to prove that:

$$P\ (BUF\ \iota\ CS\ lf\ \tau\ up\ (i\ +\ 1)) \leq (1\ -\ bnd) * P\ (BUF\ \iota\ CS\ lf\ \tau\ up\ i) \tag{3}$$

$$\text{e-at } \iota \text{ cs } \gamma\ \sigma \text{ j k} \equiv (\text{if enabled } \iota \text{ cs } \gamma\ [\![\sigma]\!]_{j+k} \text{ then 1 else 0})$$

$$\text{t-at } \iota \text{ cs } \gamma\ \sigma \text{ j k} \equiv (\text{if taken } \iota \text{ cs } \gamma\ [\![\sigma]\!]_{Suc(j+k)} \text{ then 1 else 0})$$

$$\begin{aligned}
&\text{cnt-e } \iota \text{ cs } \gamma\ \sigma \text{ j } 0 = \text{e-at } \iota \text{ cs } \gamma\ \sigma \text{ j } 0\\
&\text{cnt-e } \iota \text{ cs } \gamma\ \sigma \text{ j (Suc i)} = \text{e-at } \iota \text{ cs } \gamma\ \sigma \text{ j (Suc i)} + \text{cnt-e } \iota \text{ cs } \gamma\ \sigma \text{ j i}
\end{aligned}$$

$$\begin{aligned}
&\text{cnt-t } \iota \text{ cs } \gamma\ \sigma \text{ j } 0 = \text{t-at } \iota \text{ cs } \gamma\ \sigma \text{ j } 0\\
&\text{cnt-t } \iota \text{ cs } \gamma\ \sigma \text{ j (Suc i)} = \text{t-at } \iota \text{ cs } \gamma\ \sigma \text{ j (Suc i)} + \text{cnt-t } \iota \text{ cs } \gamma\ \sigma \text{ j i}
\end{aligned}$$

$$\begin{aligned}
\text{UF } \iota \text{ cs } \gamma\ \tau \text{ i} \equiv \{&\sigma \in \text{palg-embed } \tau.\ \exists \text{k. enabled } \iota \text{ cs } \gamma\ [\![\sigma]\!]_{((\ |\tau|\ )+\ k)} \wedge\\
&\text{cnt-e } \iota \text{ cs } \gamma\ \sigma\ (\ |\tau|\ ) \text{ k} = \text{Suc i} \wedge \text{cnt-t } \iota \text{ cs } \gamma\ \sigma\ (\ |\tau|\ ) \text{ k} = 0\}
\end{aligned}$$

$$\begin{aligned}
\text{BUF } \iota \text{ cs } \gamma\ \tau \text{ up i} \equiv \{&\sigma \in \text{palg-embed } \tau.\ \exists \text{k<up. enabled } \iota \text{ cs } \gamma\ [\![\sigma]\!]_{((\ |\tau|\ )+\ k)} \wedge\\
&\text{cnt-e } \iota \text{ cs } \gamma\ \sigma\ (\ |\tau|\ ) \text{ k} = \text{Suc i} \wedge \text{cnt-t } \iota \text{ cs } \gamma\ \sigma\ (\ |\tau|\ ) \text{ k} = 0\}
\end{aligned}$$

$$\begin{aligned}
\text{SUF } \iota \text{ cs } \gamma\ \tau \text{ up i} \equiv \{&[\![\sigma]\!]_{((\ |\tau|\ )\ +\ Suc\ k)}\ |\sigma \text{ k. } \sigma \in \text{palg-embed } \tau \wedge \text{k} < \text{up} \wedge\\
&\text{enabled } \iota \text{ cs } \gamma\ [\![\sigma]\!]_{((\ |\tau|\ )+\ k)} \wedge\\
&\text{cnt-e } \iota \text{ cs } \gamma\ \sigma\ (\ |\tau|\ ) \text{ k} = \text{Suc i} \wedge \text{cnt-t } \iota \text{ cs } \gamma\ \sigma\ (\ |\tau|\ ) \text{ k} = 0\}
\end{aligned}$$

Fig. 7. Definitions of sets of unfair executions

This together with the lemma $lim_{up\to\infty}P\ (BUF\ \iota\ cs\ \gamma\ \tau\ up\ i) = P\ (UF\ \iota\ cs\ \gamma\ \tau\ i)$ finally leads to the proof of

$$P\ (UF\ \iota\ CS\ lf\ \tau\ (i\ +\ 1)) \leq (1\ -\ bnd) * P\ (UF\ \iota\ CS\ lf\ \tau\ i),$$

which entails that $lim_{i\to\infty}P(UF\ \iota\ CS\ lf\ \tau\ i) = 0$.
Packing all the above up, the final result in equation (1) is obtained.

5 Related works

Paulson's inductive approach for protocol verification[13] has been used to verify fairly complex security protocols [7, 8]. We have proposed an extension[1] to the inductive approach, which has been applied to several applications[2–4]. This paper aims for providing a probabilistic basis for our extension.

Probability space constructions for I/O automata can be found in[12, 14]. However, the execution sequence for I/O automata is an alternation of states and events, which is different from the pure event sequence used in our approach. This difference makes such works unsuitable for our purpose.

The probability space construction in this paper is heavily inspired by Hurd's construction of probability space over sets of 0-1 sequences[11], but our construction is more general in the following senses:

– Our execution sequence contains events, the type of which could be very rich, while Hurd's sequence contains only 0 and 1. Because of this difference, the *canonical form inductive principle* in [11] does not hold anymore, this makes the proofs of canonical properties significantly more complicated.
– In our construction, the probability of the event e happening under system state τ is given by $R(\tau,\ e)$, which is varying from state to state. Because of this, the measure on finite execution $\tau_1 @ \tau_2$ is no longer distributive, i.e. $\pi(\tau_1 @ \tau_2) = \pi(\tau_1) \times \pi(\tau_2)$ no longer holds. We have to find new proofs for lemmas which depend on this distributive property and these new proofs tend to be more complicated.

Generally, to deal with these differences, we have to use different techniques. Although many lemmas look very similar, their proofs are actually very different and much more complicated in our work. Another difference is that we define measure space as a triple $(U,\ F,\ Pr)$, where the U is the explicitly given base set. We do not follow Hurd's treatment to represent U as the *UNIV*-set of the execution sequence type, because in our setting the base set contains only valid infinite executions, therefore, we need the additional U to make this clear. Additionally, our treatment seems more standard than Hurd's, because standard mathematics treatment of probability theory is typeless.

Stefan Richter [15] ported some of Hurd's work to Isablle/HOL for the purpose of probabilistic algorithm verification. The definition of *sigma* and ten of our lemmas are copy-and-modified from Richter's work, the modification is due to the above mentioned difference in the definition of measure space. This overlapped part consists only of a very small part of our formalization. Additionally, we have given a full Isabelle proof of the *Carathéodory's extension theorem*, which is absent in Stefan Richter's work.

As discussed in [16], there are many fairness notions with *weak fairness* and *strong fairness* as the standard ones. Nonstandard fairness notions, such as extreme fairness[5] and α-fairness[9], are introduced to reflect more adequately the underlying probabilistic execution, so that liveness proofs can be simplified. Baier proposed a general notion of fairness[10] which subsumes all existing fairness notions. This paper shows that parametric fairness is just another instance of this general fairness notion.

While standard fairness notions can be expressed directly using LTL, extreme fairness and α-fairness and our parametric fairness can not. Therefore, existing formulation of both extreme fairness and α-fairness are extra-LTL, at meta level, not mechanized. In this paper, LTL is embedded in Isabelle/HOL, therefore HOL serves as a mechanized meta language, in which nonstandard fairness notions can be expressed as well as its probabilistic model. We work under the belief that a mechanized meta theory may yield more reliability, maintainability and flexibility, as argued in Müller's thesis[17].

Compared with [17], embeddings of temporal logic are very similar. However, our method uses a much simpler system model, and we believe this simpler model is adequate and more convenient to use in practice, as shown in the works[13, 7, 8, 2–4]. Work [17] deals with standard fairness notions, while this paper deals with nonstandard fairness. Therefore, the liveness proof method is inherently different.

6 Conclusion

Inductive protocol verification is a method worth further extending. This paper together with [1–4] makes such an extension sound and practical. Liveness results usually are obtained using model-checking techniques [18]. Compared with the usual model-checking approach, our liveness verification method is based on theorem proving. It has the drawback of being less automatic, but it also has the advantage of not suffering from state explosion. Theorem proving based methods can verify systems of arbitrary large size, whereas model-checking based methods usually can only verify systems up to a limited size. By further automating, our method for liveness proof could become a promising alternative to model-checking.

In this paper, a probability space is established on the set of valid infinite executions of a concurrent system. It is then proved that the set of parametrically fair executions has probability 1 in this probability space. This result assigns a meaning to the conception of *Parametric Fairness* we proposed in [1], i.e. almost all valid infinite executions are *Parametrically Fair*. Therefore, Parametric Fairness is a reasonable fairness notion which captures the non-deterministic nature of concurrent execution.

Acknowledgments: We have benefited greatly from many E-mail discussions with Joe Hurd and Stefan Richter.

References

1. Zhang, X., Yang, H., Wang, Y.: Liveness reasoning for inductive protocol verification. In: The 'Emerging Trend' of TPHOLs 2005. Oxford University Computing Lab. PRG-RR-05-02 (2005) 221–235

2. Yang, H., Zhang, X., Wang, Y.: Liveness proof of an elevator control system. In: The 'Emerging Trend' of TPHOLs 2005. Oxford University Computing Lab. PRG-RR-05-02 (2005) 190–204
3. Yang, H., Zhang, X., Wang, Y.: A correctness proof of the srp protocol. In: SSN2006: The 2nd international workshop on security in systems and networks. (2006)
4. Yang, H., Zhang, X., Wang, Y.: A correctness proof of the dsr protocol. In Cao, J., Stojmenovic, I., Jia, X., Das, S.K., eds.: The Second International Conference on Mobile Ad-hoc and Sensor Networks (MSN2006). Number 4325 in LNCS, Berlin Heidelberg, Springer (2006) 72–83
5. Pnueli, A.: On the extremely fair treatment of probabilistic algorithms. In: STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1983) 278–290
6. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Computer Security **6** (1998) 85–128
7. Paulson, L.C.: Verifying the set protocol: Overview. In: FASec. (2002) 4–14
8. Paulson, L.C.: Inductive analysis of the internet protocol tls. ACM Transactions on Computer and System Security **2**(3) (1999) 332–351
9. Pnueli, A., Zuck, L.D.: Probabilistic verification. Information and Computation **103**(1) (1993) 1–29
10. Baier, C., Kwiatkowska, M.Z.: On the verification of qualitative properties of probabilistic processes under fairness constraints. Inf. Process. Lett. **66**(2) (1998) 71–79
11. Hurd, J.: Formal Verification of Probabilistic Algorithms. PhD thesis, Univ. of Cambridge (2002)
12. Wu, S.H., Smolka, S.A., Stark, E.W.: Composition and behaviors of probabilistic I/O automata. In: Fifth International Conference on Concurrence Theory. Volume 836 of LNCS. (1994)
13. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Comput. Secur. **6**(1-2) (1998) 85–128
14. Segala, R.: Modeling and verification of randomized distributed real-time systems. PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science (1995) Also appears as technical report MIT/LCS/TR-676.
15. Richter, S.: Formalizing integration theory with an application to probabilistic algorithms. In Slind, K., Bunker, A., Gepalakrishnan, G., eds.: Proceedings of TPHOLs 2004. Number 3223 in LNCS, Pack City, Springer (2004) 271–286
16. Francez, N.: Fairness. Texts and Monographs in Computer Science. Springer-Verlag (1986)
17. Müller, O.: A verification environment for I/O automata based on formalized meta-Theory. PhD thesis, Technische Universität München (1998)
18. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5) (1994) 1512–1542