Liveness Reasoning with Isabelle/HOL*

Jinshuang Wang[1,2], Huabing Yang[1], and Xingyuan Zhang[1]

[1] PLA University of Science and Technology, Nanjing 210007, China
[2] State Key Lab for Novel Software Technology, Nanjing University, Nanjing 210093, China
{wangjinshuang,xingyuanz}@gmail.com

Abstract. This paper describes an extension of Paulson's inductive protocol verification approach for liveness reasoning. The extension requires no change of the system model underlying the original inductive approach. Therefore, all the advantages, which makes Paulson's approach successful for safety reasoning are kept, while liveness reasoning becomes possible. To simplify liveness reasoning, a new fairness notion, named *Parametric Fairness* is used instead of the standard ones. A probabilistic model is established to support this new fairness notion. Experiments with small examples as well as real world communication protocols confirm the practicality of the extension. All the work has been formalized with Isabelle/HOL using Isar.

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

1 Introduction

Paulson's inductive approach has been used to verify safety properties of many security protocols [1, 2]. The success gives incentives to extend this approach to a general approach for protocol verification. To achieve this goal, a method for the verification of liveness properties is needed. According to Manna and Pnueli [3], temporal properties are classified into three classes: safety properties, response properties and reactivity properties. The original inductive approach only deals with safety properties. In this paper, proof rules for liveness properties (both response and reactivity) are derived under the same execution model as the original inductive approach. These liveness proof rules can be used to reduce the proof of liveness properties to the proof of safety properties, a task well solved by the original approach.

The proof rules are derived based on a new notion of fairness, *parametric fairness*, which is an adaption of the α-fairness [4,5,6] to the setting of HOL. *Parametric fairness* is properly stronger than standard fairness notions such as *weak fairness* and *strong fairness*. We will explain why the use of *parametric fairness* can deliver more liveness results through simpler proofs.

A probabilistic model is established to show the soundness of our new fairness notion. It is proved that the set of all parametrically fair execution traces is measurable and has probability 1. Accordingly, the definition of parametric fairness is reasonable.

* This research was funded by 863 Program(2007AA01Z409) and NNSFC(60373068) of China.

S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 485–499, 2009.

The practicability of this liveness reasoning approach has been confirmed by experiments of various sizes [7, 8, 9]. All the work has been formalized with Isabelle/HOL using Isar [10, 11], although the general approach is not necessarily confined to this particular system.

The paper is organized as the following: section 2 presents *concurrent system*, the execution model of inductive approach; section 3 gives a shallow embedding of LTL (Linear Temporal Logic); section 4 gives an informal explanation of parametric fairness; section 5 explains the liveness proof rules; section 6 establishes a probabilistic model for parametric fairness; section 7 describes liveness verification examples. section 8 discusses related works; section 9 concludes.

2 Concurrent Systems

In the inductive approach, system state only changes with the happening of events. Accordingly, it is natural to represent a system state with the list of events happening so far, arranged in reverse order. A system is concurrent because its states are nondeterministic, where a state is *nondeterministic*, if there are more than one event eligible to happen under that state. The specification of a *concurrent system* is just a specification of this eligible relation. Based on this view, formal definition of concurrent system is given in Fig. 1.

$\sigma_i \equiv \sigma$ i

primrec $[\![\sigma]\!]_0 = []$
$[\![\sigma]\!]_{(Suc\ i)} = \sigma_i \# [\![\sigma]\!]_i$

τ [cs> e $\equiv (\tau, e) \in$ cs

inductive-set vt :: ('a list × 'a) set ⇒ 'a list set
for cs ::('a list × 'a) set **where**
vt-nil [intro] : [] ∈ vt cs |
vt-cons [intro]: $[\![\tau \in$ vt cs; τ [cs> e$]\!] \Longrightarrow (e \# \tau) \in$ vt cs

consts derivable :: 'a ⇒ 'b ⇒ bool (- ⊢ - [64, 64] 50)
fnt-valid-def: cs ⊢ $\tau \equiv \tau \in$ vt cs
inf-valid-def: cs ⊢ $\sigma \equiv \forall$ i. $[\![\sigma]\!]_i$ [cs> σ_i

Fig. 1. Definition of Concurrent System

The type of concurrent systems is $('a\ list \times 'a)\ set$, where $'a$ is the type of events. Concurrent systems are written as *cs*. The expression $(\tau, e) \in cs$ means that event e is eligible to happen under state τ in concurrent system *cs*. The notation $(\tau, e) \in cs$ is abbreviated as $\tau\ [cs> e$. The set of reachable states is written as *vt cs* and $\tau \in vt\ cs$ is abbreviated as $cs \vdash \tau$. An execution of a concurrent system is an infinite sequence of events, represented as a function with type $nat \Rightarrow 'a$. The *i*-th event in execution σ is abbreviated as σ_i. The prefix consisting of the first *i* events is abbreviated as $[\![\sigma]\!]_i$. For σ to be a valid execution of *cs* (written as $cs \vdash \sigma$), σ_i must be eligible to happen under $[\![\sigma]\!]_i$.

Infinite execution σ is called *nondeterministic* if it has infinitely many nondeterministic prefixes. It is obvious from the definition of $cs \vdash \sigma$ that σ represents the choices on which event to happen next at its nondeterministic prefixes.

3 Embedding LTL

LTL (Linear Temporal Logic) used to represent liveness properties in this paper is defined in Fig. 2. LTL formulae are written as φ, ψ, κ etc. The type of LTL formulae is defined as *'a tlf*. The expression $(\sigma, i) \models \varphi$ means that LTL formula φ is valid at moment i of the infinite execution σ. The operator $\models$ is overloaded, so that $\sigma \models \varphi$ can be defined as the abbreviation of $(\sigma, 0) \models \varphi$. The *always* operator $\Box$, *eventual* operator $\Diamond$, *next* operator $\odot$, *until* operator $\rhd$ are defined literally. An operator $\langle$-$\rangle$ is defined to lift a predicate on finite executions up to a LTL formula. The temporal operator $\hookrightarrow$ is the lift of logical implication $\longrightarrow$ up to LTL level. For an event e, the term $(\!|e|\!)$ is a predicate on finite executions stating that the last happened event is e. Therefore, the expression $\langle(\!|e|\!)\rangle$ is an LTL formula saying that event e happens at the current moment.

```
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::nat⇒'a) ⊨ (φ::'a tlf) ≡ (σ::nat ⇒ 'a, (0::nat)) ⊨ φ

 □φ ≡ λ σ i. ∀ j. i ≤ j ⟶ (σ, j) ⊨ φ
 ◇φ ≡ λ σ i. ∃ j. i ≤ j ∧ (σ, j) ⊨ φ

constdefs lift-pred :: ('a list ⇒ bool) ⇒ 'a tlf   (⟨-⟩ [65] 65)
 ⟨P⟩ ≡ λ σ i. P ⟦σ⟧_i

constdefs lift-imply :: 'a tlf ⇒ 'a tlf ⇒ 'a tlf (- ↪ - [65, 65] 65)
 φ ↪ ψ ≡ λ σ i. φ σ i ⟶ ψ σ i

constdefs last-is :: 'a ⇒ 'a list ⇒ bool
 last-is e τ ≡ (case τ of [] ⇒ False | (e' # τ') ⇒ e' = e)

syntax -is-last :: 'a ⇒ ('a list ⇒ bool) ((|-|) [64] 1000)
translations (|e|) ⇌ last-is e
```

Fig. 2. The Embedding of Linear Temporal Logic in Isabelle/HOL

4 The Notion of Parametric Fairness

In this section, *parametric fairness* is introduced as a natural extension of standard fairness notions. To show this point, the definition of parametric fairness *PF* is given in Fig. 3 as the end of a spectrum of fairness notions starting from standard ones.

A study of Fig. 3 may reveal how the definition of *PF* is obtained through incremental modifications, from the standard *weak fairness WF* to the standard *strong fairness SF* and finally through the less standard extreme fairness *EF*. Each fairness notion *?F* has a

constdefs WF_α :: ('a list × 'a) set ⇒ 'a ⇒ (nat ⇒ 'a) ⇒ bool
WF_α cs e $\sigma \equiv \sigma \models \Box(\lambda\, \sigma\ \mathrm{i}.\ [\![\sigma]\!]_i\ [\mathrm{cs}\!>\ \mathrm{e}) \longrightarrow \sigma \models \Box\Diamond(\lambda\, \sigma\ \mathrm{i}.\ \sigma_i = \mathrm{e})$

constdefs WF :: ('a list × 'a) set ⇒ (nat ⇒ 'a) ⇒ bool
WF cs $\sigma \equiv \forall$ e. WF_α cs e σ

constdefs SF_α :: ('a list × 'a) set ⇒ 'a ⇒ (nat ⇒ 'a) ⇒ bool
SF_α cs e $\sigma \equiv \sigma \models \Box\Diamond(\lambda\, \sigma\ \mathrm{i}.\ [\![\sigma]\!]_i\ [\mathrm{cs}\!>\ \mathrm{e}) \longrightarrow \sigma \models \Box\Diamond(\lambda\, \sigma\ \mathrm{i}.\ \sigma_i = \mathrm{e})$

constdefs SF :: ('a list × 'a) set ⇒ (nat ⇒ 'a) ⇒ bool
SF cs $\sigma \equiv \forall$ e. SF_α cs e σ

constdefs EF_α :: ('a list × 'a) set ⇒ ('a list ⇒ bool) ⇒ ('a list ⇒ 'a) ⇒ 'a seq ⇒ bool
EF_α cs P E $\sigma \equiv\ \sigma \models \Box\Diamond(\lambda\, \sigma\ \mathrm{i}.\ \mathrm{P}\ [\![\sigma]\!]_i \wedge [\![\sigma]\!]_i\ [\mathrm{cs}\!>\ \mathrm{E}\ [\![\sigma]\!]_i)$
$\longrightarrow \sigma \models \Box\Diamond(\lambda\, \sigma\ \mathrm{i}.\ \mathrm{P}\ [\![\sigma]\!]_i \wedge \sigma_i = \mathrm{E}\ [\![\sigma]\!]_i)$

constdefs EF :: ('a list × 'a) set ⇒ (nat ⇒ 'a) ⇒ bool
EF cs $\sigma \equiv \forall$ P E. EF_α cs P E σ

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

constdefs PF :: ('a list × 'a) set ⇒ 'a pe list ⇒ (nat ⇒ 'a) ⇒ bool
PF cs pel $\sigma \equiv$ list-all (λ (P, E). EF_α cs P E σ) pel

Fig. 3. Definition of Fairness Notions

corresponding pre-version *?*F_α. For example, *WF* is obtained from WF_α by quantifying over *e*.

For any nondeterministic execution σ, its progress towards desirable states depends on the choice of helpful events under the corresponding helpful states (or helpful prefixes). The association between helpful states and helpful events can be specified using (P, E)-pairs, where P is a predicate on states used to identify helpful states and E is the *choice function* used to choose the corresponding helpful events under P-states. A pair (P, E) is called *enabled* under state τ if $P\ \tau$ is true, and it is called *executed* under τ if $E\ \tau$ is chosen to happen.

An infinite execution σ is said to treat the pair (P, E) fairly, if (P, E) is executed infinitely often in σ, unless (P, E) is enabled for only finitely many times. In an unified view, every fairness notion is about how fair (P, E)-pairs should be treated by infinite executions. For example, *EF* requires that any expressible (P, E)-pair be fairly treated while *PF* only requires that (P, E)-pairs in the parameter *pel* be fairly treated, and this explains why it is called *parametric fairness*.

Common sense tells us that a fair execution σ should make nondeterministic choices randomly. However, standard fairness notions such as *WF* and *SF* fail to capture this intuition. Consider the concurrent system defined in Fig. 4 and Fig. 5.

Fig. 4 is a diagram of the concurrent system cs_2 formally defined in Fig. 5, where diagram states are numbered with the value of function F_2. Any *intuitively fair* execution starting from state *2* should finally get into state *0*. If a fairness notion *?F* correctly captures our intuition, the following property should be valid:

$$[\![cs_2 \vdash \sigma;\ ?F\ cs_2\ \sigma]\!] \Longrightarrow \sigma \models \Box(\langle\lambda\, \tau.\ F\ \tau = 2\rangle \hookrightarrow \Diamond\ \langle\lambda\, \tau.\ F\ \tau = 0\rangle) \tag{1}$$

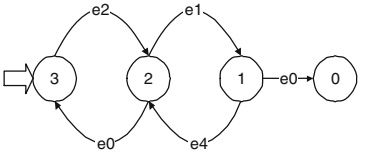

Fig. 4. The diagram of cs_2

datatype Evt = e_0 | e_1 | e_2 | e_3 | e_4

fun F_2 :: Evt list ⇒ nat
where
$F_2\ [] = 3$ |
$F_2\ (e_0 \# \tau) = (\text{if } (F_2\ \tau = 1) \text{ then } 0 \text{ else}$
$\quad \text{if } (F_2\ \tau = 2) \text{ then } 3 \text{ else } F_2\ \tau)$ |
$F_2\ (e_1 \# \tau) = (\text{if } (F_2\ \tau = 2) \text{ then } 1 \text{ else } F_2\ \tau)$ |
$F_2\ (e_2 \# \tau) = (\text{if } (F_2\ \tau = 3) \text{ then } 2 \text{ else } F_2\ \tau)$ |
$F_2\ (e_4 \# \tau) = (\text{if } (F_2\ \tau = 1) \text{ then } 2 \text{ else } F_2\ \tau)$

inductive-set cs_2 :: (Evt list × Evt) set
where
$r_0 : F_2\ \tau = 1 \Longrightarrow (\tau, e_0) \in cs_2$ |
$r_1 : F_2\ \tau = 2 \Longrightarrow (\tau, e_1) \in cs_2$ |
$r_2 : F_2\ \tau = 3 \Longrightarrow (\tau, e_2) \in cs_2$ |
$r_3 : F_2\ \tau = 2 \Longrightarrow (\tau, e_0) \in cs_2$ |
$r_4 : F_2\ \tau = 1 \Longrightarrow (\tau, e_4) \in cs_2$

Fig. 5. The definition of cs_2

Unfortunately, neither *WF* nor *SF* serves this purpose. Consider the execution $(e_2.\ e_1.\ e_4.\ e_0)^\omega$, which satisfies both *WF* cs_2 and *SF* cs_2 while violating the conclusion of (1).

The deficiency of standard fairness notions such as *SF* and *WF* is their failure to specify the association between helpful states and helpful events explicitly. For example, *SF* only requires any infinitely enabled event be executed infinitely often. Even though execution $(e_2.\ e_1.\ e_4.\ e_0)^\omega$ satisfies *SF*, it is not intuitively random in that it is still biased towards avoiding the helpful event e_0 under the corresponding helpful state *1*. Therefore, (1) is not valid if *?F* is instantiated either to *SF* or *WF*. *Extreme fairness* was proposed by Pnueli [12] to solve this problem. The *EF* is a direct expression of extreme fairness in HOL, which requires that *all* expressible (*P*,*E*)-pairs be fairly treated. Execution $(e_2.\ e_1.\ e_4.\ e_0)^\omega$ does not satisfy *EF*, because the (P, E)-pair $(\lambda\tau.\ F_2\ \tau = 1,\ \lambda\tau.\ e_1)$ is not fairly treated. In fact, (1) is valid if *?F* is instantiated to *EF*.

Unfortunately, a direct translation of *extreme fairness* in HOL is problematic, because the universal quantification over *P*, *E* may accept any well-formed expression of the right type. Given any nondeterministic execution σ, it is possible to construct a pair (P_σ, E_σ) which is not fairly treated by σ. Accordingly, any nondeterministic execution σ is not *EF*, as confirmed by the following lemma:

$$\sigma \models \Box\Diamond(\lambda\sigma\ i.\ \{e.\ [\![\sigma]\!]_i\ [cs\!>\ e \wedge e \neq \sigma_i\} \neq \{\}) \Longrightarrow \neg\ EF\ cs\ \sigma \tag{2}$$

The premise of (2) formally expresses that σ is nondeterministic. The construction of (P_σ, E_σ) is a diagonal one which makes a choice different from the one made by σ on every nondeterministic prefix. Detailed proof of (2) can be found in [13].

Now, since most infinite executions of a concurrent system are nondeterministic, *EF* will rule out almost all executions. If *EF* is used as the fairness premises of a liveness property, the overall statement is practically meaningless. Parametric fairness *PF* is proposed to solve the problem of *EF*. Instead of requiring all (P, E)-pairs be fairly

treated, *PF* only requires (P, E)-pairs appearing in its parameter *pel* be fairly treated. Since there are only finitely many (P, E)-pairs in *pel*, most executions of a concurrent system are kept, even if every (P, E)-pair in *pel* rules out some measurement of them. Section 6 will make this argument precise by establishing a probabilistic model for *PF*.

5 Liveness Rules

According to Manna [3], response properties are of the form $\sigma \models \Box(\langle P\rangle \hookrightarrow \Diamond\langle Q\rangle)$, where $\langle P\rangle$ and $\langle Q\rangle$ are *past formulae* (in [3]'s term), obtained by lifting predicates on finite traces. The conclusion of (1) is of this form. Reactivity properties are of the form $\sigma \models (\Box\Diamond\langle P\rangle) \hookrightarrow (\Box\Diamond\langle Q\rangle)$ meaning: *if P holds infinitely often in σ, then Q holds infinitely often in σ as well.*

The proof rule for response property is the theorem *resp-rule*:

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

and the proof rule for reactivity property is the theorem *react-rule*:

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

The symbols used in these two theorems are given in Fig. 6.

```
consts pel-of :: ('a list ⇒ nat) ⇒ ('a list ⇒ 'a) ⇒ nat ⇒
                   (('a list ⇒ bool) × ('a list ⇒ 'a)) list ({|-, -, -|} [64, 64, 64] 1000)
primrec {|F, E, 0|} = []
        {|F, E, (Suc n)|} = (λ τ. F τ = Suc n, E) # {|F, E, n|}

syntax -drop :: 'a list ⇒ nat ⇒ 'a list (⌈-⌉- [64, 64] 1000)
translations ⌈l⌉n ⇌ drop n l

 ⌈P ⟼ ¬Q ∗⌉ ≡ λ τ. (∃ i ≤ |τ|. P ⌈τ⌉i ∧ (∀ k. 0 < k ∧ k ≤ i ⟶ ¬ Q ⌈τ⌉k))

locale RESP =
  fixes cs :: ('a list × 'a) set and F :: 'a list ⇒ nat  and E :: 'a list ⇒ 'a
  and   N :: nat and P :: 'a list ⇒ bool and Q :: 'a list ⇒ bool
  assumes mid: [[cs ⊢ τ; ⌈P ⟼ ¬Q ∗⌉ τ; ¬ Q  τ]] ⟹  0 < F τ ∧ F τ < N
  assumes fd:   [[cs ⊢ τ; 0 < F τ]] ⟹ τ [cs> E τ ∧ F (E τ # τ) < F τ

locale REACT =
  fixes cs :: ('a list × 'a) set and F :: 'a list ⇒ nat and E :: 'a list ⇒ 'a
  and N :: nat  and P :: 'a list ⇒ bool and Q :: 'a list ⇒ bool
  assumes init: [[cs ⊢ τ; P τ]] ⟹ F τ < N
  assumes mid: [[cs ⊢ τ; F τ < N; ¬ Q τ]] ⟹ τ [cs> E τ ∧ F (E τ # τ) < F τ
```

Fig. 6. Premises of Liveness Rules

Let's explain the *resp-rule* first. Premise *RESP cs F E N P Q* expresses constraints on *cs*'s state transition diagram. *RESP* is defined as a locale predicate, where assumption *mid* requires: for any state τ, after reaching P before reaching Q (characterized by

$\lceil P \longmapsto \neg Q * \rceil\ \tau$), the value of $F\ \tau$ is between *0* and *N*; assumption *fd* requires that if $F\ \tau$ is between *0* and *N*, event $E\ \tau$ must be eligible to happen under τ and the happening of it will decrease the value of function *F*. For any state satisfying $\lceil P \longmapsto \neg Q * \rceil\ \tau$, by repeatedly applying *fd*, a path leading from τ to *Q* can be constructed. If every pair in list $[(\lambda\ \tau\ .\ F\ \tau = n, E) \mid n \in \{1 \ldots N\}]$ is fairly treated by σ, σ will follow this path and eventually get into a *Q* state. This fairness requirement is expressed by the premise *PF cs* $\{\!|F, E, N|\!\}\ \sigma$, where $\{\!|F, E, N|\!\}$ evaluates to $[(\lambda\ \tau\ .\ F\ \tau = n, E) \mid n \in \{1 \ldots N\}]$.

As an example, when *resp-rule* is used to prove statement (1), the *P* is instantiated to $(\lambda\ \tau.\ F_2 = 2)$, and *Q* to $(\lambda\ \tau.\ F_2 = 0)$. The *F* is instantiated to F_2, and *E* to the following E_2:

constdefs E_2 :: *Evt list* ⇒ *Evt*
$E_2\ \tau \equiv$ (*if* $(F_2\ \tau = 3)$ *then* e_2 *else if* $(F_2\ \tau = 2)$ *then* e_1 *else if* $(F_2\ \tau = 1)$ *then* e_0 *else* e_0)

The *N* is instantiated to *3*, so that $\{\!|F_2, E_2, 3|\!\}$ evaluates to $[(\lambda\ \tau.\ F_2\ \tau = 3, E), (\lambda\ \tau.\ F_2\ \tau = 2, E), (\lambda\ \tau.\ F_2\ \tau = 1, E)]$.

Now, let's explain rule *react-rule*. Premise *PF cs* $\{\!|F, E, N|\!\}\ \sigma$ still has the same meaning. Since now *P*-states are reached infinitely often, a condition much stronger than in *resp-rule*, premise *REACT* can be weaker, which only requires there exists a path leading to *Q* from every *P* state, while *RESP* requires the existence of such a path on every $\lceil P \longmapsto \neg Q * \rceil$ state.

6 Probabilistic Model for PF

6.1 Some General Measure Theory

The definition of probability space given in Fig. 7 is rather standard, where *U* is the base set, *F* the measurables, *Pr* the measure function. 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 $\sum_{n=0}^{\omega} f(n) = c$.

Carathéodory's extension theorem [14,15] is the standard way to construct probability space. The theorem is proved using Isabelle/HOL:

$$\begin{array}{l}[\![\mathit{algebra}\ (U, F); \mathit{positive}\ (F, Pr); \mathit{countably\text{-}additive}\ (F, Pr)]\!] \\ \Longrightarrow \exists P.\ (\forall A.\ A \in F \longrightarrow P\ A = Pr\ A) \wedge \mathit{measure\text{-}space}\ (U, \mathit{sigma}\ (U, F), P)\end{array} \tag{3}$$

where the definition of *sigma* is in Fig. 8.

6.2 Probability Space on Infinite Executions

Elements used to construct a measure space for infinite executions are defined in Fig. 7 as locale *RCS*, which accepts a parameter *R*, where $R(\tau, e)$ is the probability of choosing event *e* to happen under state τ. The definition of *RCS* is given in Fig. 9. The purpose of *RCS* is to define a measure space $(PA, Path, \mu)$ on infinite executions in terms of parameter *R*. The underlying concurrent system is given by *CS*. Set $N\ \tau$ contains all events eligible to happen under state τ. Function π is a measure function on finite executions. *Path* is the set of valid infinite executions of *CS*.

```
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. 7. Definition of Probability Space

Set *palgebra-embed*($[\tau_0,\ldots,\tau_n]$) contains all infinite executions prefixed by some τ in $[\tau_0, \tau_1,\ldots,\tau_n]$, and *palgebra-embed*($[\tau_0,\ldots,\tau_n]$) is said to be supported by $[\tau_0,\ldots,\tau_n]$. The measure μ *S* is defined in terms of measures on supporting sets for *S*.

It is proved that *algebra* (*Path*, *PA*), *positive* (*PA*, μ) and *countably-additive* (*PA*, μ). Applying *Carathéodory's extension theorem* to these gives (*PA*, *Path*, μ) as a measure space on infinite executions of *CS*. It can be further derived that (*PA*, *Path*, μ) is a probability space.

```
inductive-set sigma :: ('a set × 'a set set) ⇒ 'a set set for M::('a set × 'a set set) where
 basic: (let (U, A) = M in (a ∈ A)) ⟹ a ∈ sigma M|
 empty: {} ∈ sigma M|
 complement: a ∈ sigma M ⟹ (let (U, A) = M in U − a) ∈ sigma M|
 union: (⋀i::nat. a i ∈ sigma M) ⟹ (⋃i. a i) ∈ sigma M
```

Fig. 8. Definition of the sigma Operator

6.3 The Probabilistic Meaning of PF

In Fig. 10, locale *BTS* is proposed as a refinement of *RCS*. The motivation of *BTS* is to provide a low bound *bnd* for *R*. Function *P* is the probability function on execution sets. Based on *BTS*, the following theorem can be proved and it gives a meaning to *PF*:

Theorem 1. $[\![BTS\ R\ bnd;\ set\ pel \neq \{\}]\!]$
$\Longrightarrow BTS.P\ R\ \{\sigma \in RCS.Path\ R.\ PF\ \{(\tau, e).\ 0 < R\ (\tau, e)\}\ pel\ \sigma\} = 1$

```
locale RCS =
  fixes R :: ('a list × 'a) ⇒ real
  assumes Rrange: 0 ≤ R(τ, e) ∧ R(τ, e) ≤ 1
  fixes CS :: ('a list × 'a) set
  defines CS-def: CS ≡ {(τ, e) . 0 < R(τ, e)}
  fixes N :: 'a list ⇒ 'a set
  defines N-def: N τ ≡ {e. 0 < R(τ, e)}
  assumes Rsum1: CS ⊢ τ ⟹ (∑ e ∈ (N τ). R(τ,e)) = 1
begin
fun π :: 'a list ⇒ real where
  π [] = 1 |
  π (e#τ) = R(τ, e) * π τ

definition Path::'a seq set where
  Path ≡ {σ. (∀ i. π ⟦σ⟧_i > 0)}

definition palg-embed :: 'a list ⇒ 'a seq set where
  palg-embed τ ≡ {σ ∈ Path. ⟦σ⟧_|τ| = τ}

fun palgebra-embed :: 'a list list ⇒ 'a seq set where
  palgebra-embed [] = {} |
  palgebra-embed (τ#l) = (palg-embed τ) ∪ palgebra-embed l

definition PA :: 'a seq set set where
  PA ≡ {S. ∃l. palgebra-embed l = S ∧ S ⊆ Path}

definition palg-measure :: 'a list list ⇒ real (μ_0) where
  μ_0 l ≡ (∑ τ ∈ set l. π τ)

definition palgebra-measure::'a list list ⇒ real (μ_1) where
  μ_1 l ≡ inf (λr. ∃l'. palgebra-embed l = palgebra-embed l' ∧ μ_0 l' = r)

definition μ:: 'a seq set ⇒ real where
  μ S ≡ sup (λr. ∃b. μ_1 b = r ∧ (palgebra-embed b) ⊆ S)
end
```

Fig. 9. Formalization of Probabilistic Execution

Theorem 1 shows *almost all* executions are fair in the sense of *PF*. It says that the set of valid executions satisfying *PF* has probability *1*. The underlying concurrent system is $\{(\tau,e).0<R(\tau,e)\}$, which is the expansion of *CS* in *RCS*. The probability function *BTS.P R* is the one derived from *R* using *BTS*. The set *RCS.Path R* contains all valid infinite executions of the underlying *CS*.

6.4 The Proof of Theorem 1

For the proof of theorem 1, a generalized notion of fairness *GF* is introduced in Fig. 11. *GF* is defined as a locale accepting three parameters, *cs* a concurrent system, *L* a countable set of labels, and *l* is a labeling function. A label ι is said to be *enabled* in state τ (written as *enabled* ι τ) if there exists some event *e* eligible to happen under state τ and ι belongs to $l(\tau,e)$. A label ι is said to be *taken* in state τ if ι is assigned

```
locale BTS = RCS +
  fixes bnd :: real
  assumes system-bound: 0 < bnd ∧ bnd ≤ 1
  and bound-imply: R(τ, e) > 0 ⟹ R(τ, e) ≥ bnd
definition P::('a seq set ⇒ real) where
  P ≡ (SOME P'::('a seq set ⇒ real).
        (∀ A. A ∈ PA ⟶ P' A = μ A) ∧ measure-space (Path, sigma (Path, PA), P'))
```

Fig. 10. The definition of locale BTS

to the last execution step of τ, which is denoted by $hd(\tau)$, and the state before the last step is denoted by $tl(\tau)$. Label ι is said to be *fairly* treated by an infinite execution, if ι is taken infinite many times whenever it is enabled infinite many times in σ. An infinite execution is fair in the sense of *GF*, if all labels in set *L* are fairly treated. By instantiating label set *L* with the set of elements on *pel*(denoted by *set pel*), and label function *l* with function *lf* where $lf\ (\tau, e) = \{(Q, E)|\ (Q, E) \in (set\ pel) \wedge Q\ \tau \wedge e = (E\ \tau)\}$, it can be shown *PF* is just an instance of *GF*:

$$PF\ cs\ pel\ \sigma = GF.fair\ cs\ (set\ pel)\ (lf\ pel)\ \sigma \tag{4}$$

It is proved that the probability of *GF* fair execution equals *1*:

$$BTS.P\ R\ \{\sigma \in RCS.Path\ R.\ GF.fair\ \{(\tau, e).\ 0 < R\ (\tau, e)\}\ L\ l\ \sigma\} = 1 \tag{5}$$

The combination of (4) and (5) gives rise to Theorem 1. The intuition behind (4) it that (P, E)-pairs in *PF* can be seen as labels in *GF*. To prove (5), it is sufficient to prove the probability of unfair executions equals to *0*. In turn, it is sufficient to prove the probability of unfair executions with respect to any one label ι equals to *0*, because

```
locale GF =
  fixes cs:: ('a list × 'a) set
  fixes L :: 'b set
  assumes countable-L: countable L
  and non-empty-L: L ≠ {}
  fixes l :: 'a list × 'a ⇒ 'b set
  assumes subset-l: l(τ,e) ⊆ L
begin
definition enabled:: 'b ⇒ 'a list ⇒ bool where enabled ι τ ≡ (∃ e. (ι ∈ l(τ, e) ∧ (τ, e) ∈ cs))

definition taken :: 'b ⇒ 'a list ⇒ bool where taken ι τ ≡ ι ∈ l(tl(τ), hd(τ))

definition fairι :: 'b ⇒ (nat ⇒ 'a) ⇒ bool where
  fairι ι σ ≡ σ ⊨ □◇(λσ i. enabled ι ⟦σ⟧_i) ⟶ σ ⊨ □◇(λσ i. taken ι ⟦σ⟧_Suc(i))

definition fair :: (nat ⇒ 'a) ⇒ bool where fair σ ≡ ∀ ι ∈ L. fairι ι σ
end
```

Fig. 11. A general fairness definition

the number of labels is countable. If label ι is treated unfairly by execution σ, σ must avoid ι infinite many times, with every avoidance reduce the probability by a certain proportion. This series of avoidances finally reduce the probability to *0*.

7 Liveness Proof Experiments

7.1 Elevator Control System

In [7], Yang developed a formal verification of elevator control system. The events of an elevator control system are: *Arrive p m n*: User *p* arrives at floor *m*, planning to go to floor *n*; *Enter p m n*: User *p* enters elevator at floor *m*, planning to go to floor *n*; *Exit p m n*: User *p* gets off elevator at floor *n*, *m* is the floor, where user *p* entered elevator; *Up n*: A press of the ▲-button on floor *n*; *Down n*: A press of the ▼-button on floor *n*; *To n*: A press of button *n* on elevator's control panel; *StartUp n*: Elevator starts moving upward from floor *n*; *StartDown n*: Elevator starts moving down from floor *n*; *Stop n m*: Elevator stops at floor *m*, before stopping, the elevator is moving from floor *n* to floor *m*; *Pass n m*: Elevator passes floor *n* without stopping, before passing floor *m*, the elevator is moving from floor *n* to floor *m*.

The proved liveness property for the elevator control system is:

$$\begin{array}{l}[\![\mathit{elev\text{-}cs} \vdash \sigma;\\ \ \mathit{PF\ elev\text{-}cs}\\ \ (\{\!|F\ p\ m\ n, E\ p\ m\ n, 4 * H + 3|\!\} \ @\ \{\!|FT\ p\ m\ n, ET\ p\ m\ n, 4 * H + 3|\!\})\ \sigma]\!]\\ \Longrightarrow \sigma \models \Box\langle(\!|\mathit{Arrive\ p\ m\ n}|\!)\rangle \hookrightarrow \Diamond\langle(\!|\mathit{Exit\ p\ m\ n}|\!)\rangle\end{array}$$

The conclusion says: if user *p* arrives at floor *m* and wants to go to floor *n* (represented by the happening of event *Arrive p m n*), then he will eventually get there (represented by the happening of event *Exit p m n*).

The liveness proof splits into two stages. The first is to prove once user *p* arrives, it will eventually get into elevator, the second is to prove once *p* gets into elevator, it will eventually get out of elevator at its destination. Both stages use rule *resp-rule*. The definition of *cs*, *P* and *Q* is obvious. The difficult part is to find a proper definition of *F* and *E* so that premise *RESP cs F E N P Q* can be proved. We explain the finding of *F* and *E* by example of a 5-floor elevator system, the state diagram of which is shown in Fig. 12. Helpful transitions are represented using normal lines while unhelpful transitions with dotted lines.

The arrival of *p* (event *Arrive p 2 4*) brings system from state $\mathit{Arrive\ p\ 2\ 4} \notin \mathit{arr\text{-}set}\ \tau$ to $\mathit{Arrive\ p\ 2\ 4} \in \mathit{arr\text{-}set}\ \tau \wedge 2 \notin \mathit{up\text{-}set}\ \tau$. To call the elevator, *p* will press the ▲-button (event *Up 2*), and this will bring system into macro-state $\mathit{Arrive\ p\ 2\ 4} \in \mathit{arr\text{-}set}\ \tau \wedge 2 \in \mathit{up\text{-}set}\ \tau$, which contains many sub-states characterized by the status of elevator. Status (n, m) means moving from floor *n* to *m*, while $d(n,n)$ means stopped on floor *n* in down passes, $u(n,n)$ in up passes. It can be seen that helpful transitions in Fig. 12 form a spanning tree rooted at the desirable state. Premise *RESP cs F E N P Q* is valid if $F\ \tau$ evaluates to the height of τ in the spanning tree and $E\ \tau$ evaluates to the helpful event leading out of τ. Fig. 12 shows such *F* and *E* can be defined. Details of the definition can be found in [7]. Once *F* and *E* is given, the rest of the proof is straight forward. Statistics in Table 1 may give some idea the amount of work required.

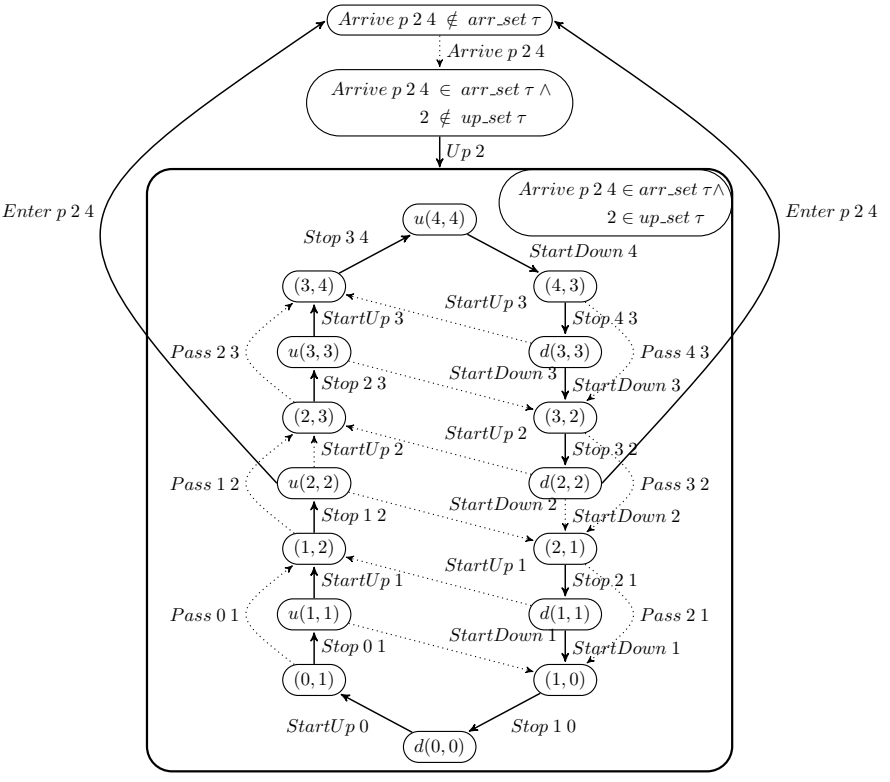

Fig. 12. The state transition diagram of elevator for *Arrive p 2 4*

Table 1. Summary of Isabelle/HOL proof scripts

Contents	**Nr. of lines**	**Nr. of lemmas**	**Nr. of working days**
Definitions of the elevator control system, F, E, FT and ET	≤ 200	–	*
Safety properties	579	12	2
First stage of the proof	2012	10	5
Second stage of the proof	1029	8	3

7.2 Mobile Ad Hoc Secure Routing Protocols

In mobile Ad Hoc networks, nodes keep moving around in a certain area while communicating over wireless channels. The communication is in a peer-to-peer manner, with no central control. This makes the network very susceptible to malicious attacks. Therefore, security is an important issue in Mobile Ad Hoc networks. Secure routing protocol is chosen as our verification target.

To counteract attacks, specialized procedures are used by secure routing protocols together with cryptographic mechanisms such as public key and hashing. The goal of verification is to make sure the procedures combined with cryptographic mechanisms do not collapse even in highly adversary environments.

Attackers are modeled using the same method as Paulson's [1]. The correctness of routing protocol is verified by combining it with an upper transfer layer. It is proved that the transfer protocol can deliver every user packet with the support of the underlying routing protocol. This is a typical liveness property, which says if a user packet gets into the transfer layer at one end it will eventually get out at the other end, just like an elevator passenger.

Because nodes are moving around, existing routes between nodes constantly become obsolete, in which case routing protocols have to initiate route finding procedure, this procedure may be interfered by attacks aimed at slowing down or even collapsing the network.

The verification is like the one for elevator but on a much larger scale. We have managed to find definitions of F and E, which essentially represent spanning trees covering all relevant states like the one in Fig. 12. The proof is laborious but doable, suggesting that specialized tactics may be needed.

We have verified two secure routing protocols, details can be found in [8, 9]. The work in [8,9] confirmed the practicability of our approach on one hand and the resilience of the security protocol on the other hand.

8 Related Works

Approaches for verification of concurrent systems can roughly be divided into theorem proving and model checking. The work in this paper belongs to the theorem proving category, which can deal with infinite state systems.

The purpose of this paper is to extend Paulson's inductive protocol verification approach [1, 2] to deal with general liveness properties, so that it can be used as a general protocol verification approach. The notion of *PF* and the corresponding proof rules were first proposed by Zhang in [13]. At the same time, Yang developed a benchmark verification for elevator control system to show the practicality of the approach [7]. Later, Yang used the approach serious to verify the liveness properties of Mobile Ad Hoc network protocols [8, 9]. These works confirm the practicality of the extension.

The liveness rules in this paper relies on a novel fairness notion *PF*, an adaption of the α-fairness [4, 5, 16] to suit the setting of HOL. The use of *PF* can derive more liveness properties and usually the proofs are simpler than using the standard *WF* and *SF*.

According to Baier's work [5], *PF* should have a sensible probabilistic meaning. To confirm this, Wang established a probabilistic model for *PF*, to show that the measure of *PF* executions equals *1* [17]. The formalization of this probabilistic model is deeply influenced by Hurd and Stefan [14, 15], however, due to different type restrictions from [14], and the need of extension theorem which is absent from [15], most proofs have to be done from scratch. Wang's work established the soundness of *PF*.

9 Conclusion

The advantage of theorem proving over model checking is its ability to deal with infinite state systems. Unfortunately, relatively less work is done to verify liveness properties using theorem proving. This paper improves the situation by proposing an extension of Paulson's inductive approach for liveness verification and showing the soundness, feasibility and practicality of the approach.

The level of automation in our work is still low compared to model checking. One direction for further research is to develop specialized tactics for liveness proof. Additionally, the verification described in this paper is still at abstract model level. Another important direction for further research is to extend it to code level verification.

References

1. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
2. Paulson, L.C.: Inductive analysis of the Internet protocol TLS. ACM Transactions on Computer and System Security 2(3), 332–351 (1999)
3. Manna, Z., Pnueli, A.: Completing the temporal picture. Theor. Comput. Sci. 83(1), 91–130 (1991)
4. Pnueli, A., Zuck, L.D.: Probabilistic verification. Information and Computation 103(1), 1–29 (1993)
5. Baier, C., Kwiatkowska, M.: On the verification of qualitative properties of probabilistic processes under fairness constraints. Information Processing Letters 66(2), 71–79 (1998)
6. Jaeger, M.: Fairness, computable fairness and randomness. In: Proc. 2nd International Workshop on Probabilistic Methods in Verification (1999)
7. Yang, H., Zhang, X., Wang, Y.: Liveness proof of an elevator control system. In: The 'Emerging Trend' of TPHOLs, Oxford University Computing Lab. PRG-RR-05-02, pp. 190–204 (2005)
8. Yang, H., Zhang, X., Wang, Y.: A correctness proof of the srp protocol. In: 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, Rhodes Island, Greece, April 25-29 (2006)
9. Yang, H., Zhang, X., Wang, Y.: A correctness proof of the dsr protocol. In: Cao, J., Stojmenovic, I., Jia, X., Das, S.K. (eds.) MSN 2006. LNCS, vol. 4325, pp. 72–83. Springer, Heidelberg (2006)
10. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
11. Wenzel, M.: Isar - a generic interpretative approach to readable formal proof documents. In: Nipkow, T., Paulson, L.C., Wenzel, M.T. (eds.) Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
12. Pnueli, A.: On the extremely fair treatment of probabilistic algorithms. In: Proceedings of the fifteenth annual ACM symposium on Theory of computing, pp. 278–290. ACM Press, New York (1983)
13. Zhang, X., Yang, H., Wang, Y.: Liveness reasoning for inductive protocol verification. In: The 'Emerging Trend' of TPHOLs, Oxford University Computing Lab. PRG-RR-05-02, pp. 221–235 (2005)
14. Hurd, J.: Formal Verification of Probabilistic Algorithms. Ph.D thesis, University of Cambridge (2002)

15. Richter, S.: Formlizing integration theory with an application to probabilistic algorithms. In: Slind, K., Bunker, A., Gopalakrishnan, G.C. (eds.) TPHOLs 2004. LNCS, vol. 3223, pp. 271–286. Springer, Heidelberg (2004)
16. Pnueli, A.: On the extremely fair treatment of probabilistic algorithms. In: ACM (ed.) Proceedings of the 15th annual ACM Symposium on Theory of Computing, Boston, Massachusetts, April 25–27, pp. 278–290. ACM Press, New York (1983)
17. Wang, J., Zhang, X., Zhang, Y., Yang, H.: A probabilistic model for parametric fairness in isabelle/hol. Technical Report 364/07, Department of Computer Science, University of Kaiserslautern (2007)