Verification of Liveness Properties
The Forgotten and Missing Piece?

Peter Höfner
(Australian National University)

joint work with
Rob van Glabbeek (U Edinburgh)

September 5, 2025

Distributed Systems

- a *distributed system* consists of multiple components that interact (with each other)

Distributed Systems

- a *distributed system* consists of multiple components that interact (with each other)

- examples
  - distributed databases
  - communication networks
  - modern hardware
  - many operating systems, filesystems
  - industrial (control) systems
  - a bank with a network of teller machines
  - a group of people organising a workshop
  - (any) business process
  - airlines
  - ...

  (it's much harder to think of a system that is not distributed)

Verification of Distributed Systems

How to ensure their correct working?

[1] One of many ways, but the one with the highest degree of ensurance

Verification of Distributed Systems

How to ensure their correct working?

Formal methods[1]

[1] One of many ways, but the one with the highest degree of ensurance

Verification of Distributed Systems

How to ensure their correct working?

Formal methods[1]

- **specification formalisms**
  - intended requirements (what should the system accomplish?)
    e.g. temporal logic (LTL, CTL)
  - intended behaviour (how does the system do that?)
    e.g. process algebra, process algebra, Petri nets

[1] One of many ways, but the one with the highest degree of ensurance

Verification of Distributed Systems

How to ensure their correct working?

Formal methods[1]

- **specification formalisms**
  - intended requirements (what should the system accomplish?)
    e.g. temporal logic (LTL, CTL)
  - intended behaviour (how does the system do that?)
    e.g. process algebra, process algebra, Petri nets
- **analysis methods**
  - study and reason about requirements (properties) of the system; mathematically rigorous methods to verify that
    - a specification ensures the required properties
    - an implementation meets the specification

[1] One of many ways, but the one with the highest degree of ensurance

Verification of Distributed Systems

How to ensure their correct working?

Formal methods[1]

- **specification formalisms**
  - intended requirements (what should the system accomplish?)
    e.g. temporal logic (LTL, CTL)
  - intended behaviour (how does the system do that?)
    e.g. process algebra, process algebra, Petri nets
- **analysis methods**
  - study and reason about requirements (properties) of the system; mathematically rigorous methods to verify that
    - a specification ensures the required properties
    - an implementation meets the specification
- **tools** manual, automatic or interactive

[1] One of many ways, but the one with the highest degree of ensurance

(Correctness) Properties of Distributed Systems

- **safety**
  something bad will never happen (Lamport77)
- **liveness**
  something good will happen eventually (Lamport77)

[2] does not consider information flow (hyper properties)

(Correctness) Properties of Distributed Systems

- **safety**
  something bad will never happen (Lamport77)
- **liveness**
  something good will happen eventually (Lamport77)

Theorem (AlpernSchneider85)

any property can be written as the conjunction of a safety property and a liveness property[2]

[2] does not consider information flow (hyper properties)

Why Am I Interested In Liveness

Verified Properties (my contributions)

3

Why Am I Interested In Liveness

Verified Properties (my contributions)

- routing protocols
- communication protocol
- concurrent data structure (incl. mutual exclusion protocols)

3

Why Am I Interested In Liveness

Verified Properties (my contributions)

- routing protocols
- communication protocol
- concurrent data structure (incl. mutual exclusion protocols)

Verified Properties (others, a biased selection)

- operating systems (seL4), (HeiserKleinNorrishEtAl09)[3]
- garbage collection, (HoskingEtAl15)
- . . .

[3] ACM Software System Award 2023

Why Am I Interested In Liveness

Verified Properties (my contributions)

- routing protocols
  (safety & liveness*)
- communication protocol
  (safety & liveness (sort of))
- concurrent data structure (incl. mutual exclusion protocols)
  (safety)

Verified Properties (others, a biased selection)

- operating systems (seL4), (HeiserKleinNorrishEtAl09)[3]
  (safety & functional correctness)
- garbage collection, (HoskingEtAl15)
  (safety)
- ...

[3] ACM Software System Award 2023

Safety is Great, But . . .

Theorem (Safety of Garbage Collectors)

only garbage will be collected

Safety is Great, But . . .

Theorem (Safety of Garbage Collectors)

no non-garbage will be collected

Safety is Great, But . . .

Theorem (Safety of Garbage Collectors)

no non-garbage will be collected

HoskingEtAl: there is always an object at every reference reachable from a mutator root. (HoskingEtAl15)

$$\mathsf{GC} \| \mathsf{M}_1 \| \mathsf{M}_2 \| \ldots \| \mathsf{Sys} \models (\forall r.\ \mathsf{reachable}\ r \rightarrow \mathsf{valid\_ref}\ r)$$

Safety is Great, But . . .

Theorem (unpublished)

the world's fastest, safe garbage collector (verified)

Safety is Great, But . . .

Theorem (unpublished)

the world's fastest, safe garbage collector (verified)

```
skip
```

Why Am I Interested In Liveness

Interesting Systems

- delivery in routing protocols
- collection of garbage
- access to shared resources (mutual exclusion & locks)

Verification of Liveness

Theorem

without a fairness assumption, no system can be live

whether a liveness property holds often depends on underlying *fairness assumptions* one chooses to make

Assumption: Progress

Does $x = 7$ hold (eventually)? (in CTL $\models$ **AF**$(x = 7)$)

$$y := 0;\ x := 7$$

Assumption: Progress

Does $x = 7$ hold (eventually)? (in CTL $\models$ **AF**$(x = 7)$)

$$y := 0;\ x := 7$$

Assumption: Progress

Does $x = 7$ **hold (eventually)?** (in CTL $\models$ **AF**$(x = 7)$)

$$y := 0;\ x := 7$$

Intuition: a system will not stop without a reason

Assumption: Progress

Does $x = 7$ hold (eventually)? (in CTL $\models$ **AF**$(x = 7)$)

$$y := 0;\ x := 7$$

Intuition: a system will not stop without a reason

Definition

a (transition) system in a state that admits a transition will eventually progress, i.e. perform a transition

(progress generally accepted for verification)

Intermediate Note

- the transitions system defines the behaviour as a set of (finite and infinite) traces
- a completeness criterion – e.g. progress – rules out some of these traces

Intermediate Note

- the transitions system defines the behaviour as a set of (finite and infinite) traces
- a completeness criterion – e.g. progress – rules out some of these traces

attn: do not rule out too many traces

Mutual Exclusion

This is the first example of strong fairness in the first paper on fairness. (the intention is that both processes run on the same machine and access the same resource)

$$\textbf{initialise } \mathsf{y} \textbf{ to } 1$$

$$\textbf{while } (true) \begin{cases} \ell_0 & \textbf{noncritical section} \\ \ell_1 & \textbf{await}(\mathsf{y} > 0)\{\mathsf{y} := \mathsf{y}-1\} \\ \ell_2 & \textbf{critical section} \\ \ell_3 & \mathsf{y} := \mathsf{y}+1 \end{cases} \quad \| \quad \textbf{while } (true) \begin{cases} m_0 & \textbf{noncritical section} \\ m_1 & \textbf{await}(\mathsf{y} > 0)\{\mathsf{y} := \mathsf{y}-1\} \\ m_2 & \textbf{critical section} \\ m_3 & \mathsf{y} := \mathsf{y}+1 \end{cases}$$

Mutual Exclusion

$\models \mathbf{AF}(\mathsf{x} = 7)$**?**

initialise x **to** 0; **initialise** y **to** 1

while $(true)$

$$\left\{\begin{array}{ll} \ell_0 & \textbf{noncritical section} \\ \ell_1 & \textbf{await}(\mathsf{y} > 0)\{\mathsf{y} := \mathsf{y}-1\} \\ \ell_2 & \textbf{critical section; x := 7} \\ \ell_3 & \mathsf{y} := \mathsf{y}+1 \end{array}\right.$$

$\|$

while $(true)$

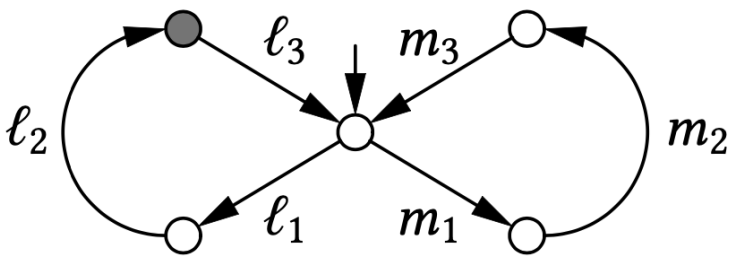
$$\left\{\begin{array}{ll} m_0 & \textbf{noncritical section} \\ m_1 & \textbf{await}(\mathsf{y} > 0)\{\mathsf{y} := \mathsf{y}-1\} \\ m_2 & \textbf{critical section} \\ m_3 & \mathsf{y} := \mathsf{y}+1 \end{array}\right.$$

Mutual Exclusion

$\models$ **AF**(x = 7)**?**

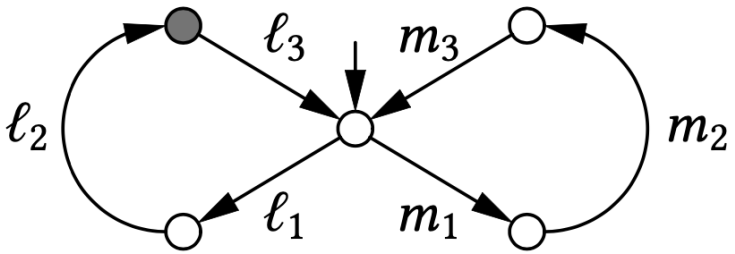
initialise x **to** 0; **initialise** y **to** 1

while $(true)$
$\begin{cases} \ell_0 & \textbf{noncritical section} \\ \ell_1 & \textbf{await}(\mathsf{y} > 0)\{\mathsf{y} := \mathsf{y}-1\} \\ \ell_2 & \textbf{critical section; x := 7} \\ \ell_3 & \mathsf{y} := \mathsf{y}+1 \end{cases}$
$\|$
while $(true)$
$\begin{cases} m_0 & \textbf{noncritical section} \\ m_1 & \textbf{await}(\mathsf{y} > 0)\{\mathsf{y} := \mathsf{y}-1\} \\ m_2 & \textbf{critical section} \\ m_3 & \mathsf{y} := \mathsf{y}+1 \end{cases}$



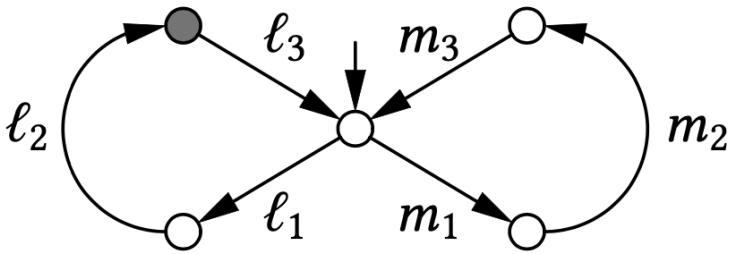
Assumption: Strong Fairness

$\models$ **AF(●)?**



Assumption: Strong Fairness

$\models$ **AF(●)?**



Definition

a *strong fairness assumption* requires that if a task (here a transition) is enabled infinitely often, it will eventually be scheduled

Assumption: Strong Fairness

$\models$ **AF**$(x = 7)$**?**

initialise x **to** 0; **initialise** y **to** 1

while $(true)$

$$\left\{\begin{array}{ll} \ell_0 & \textbf{noncritical section} \\ \ell_1 & \textbf{await}(y > 0)\{y := y-1\} \\ \ell_2 & \textbf{critical section; x := 7} \\ \ell_3 & y := y+1 \end{array}\right.$$

$\|$

while $(true)$

$$\left\{\begin{array}{ll} m_0 & \textbf{noncritical section} \\ m_1 & \textbf{await}(y > 0)\{y := y-1\} \\ m_2 & \textbf{critical section} \\ m_3 & y := y+1 \end{array}\right.$$

Assumption: Strong Fairness

$\models$ **AF**$(x = 7)$**?** yes, under strong fairness

initialise x **to** 0; **initialise** y **to** 1

while $(true)$

$$\left\{\begin{array}{ll} \ell_0 & \textbf{noncritical section} \\ \ell_1 & \textbf{await}(y > 0)\{y := y-1\} \\ \ell_2 & \textbf{critical section; x := 7} \\ \ell_3 & y := y+1 \end{array}\right.$$

$\|$

while $(true)$

$$\left\{\begin{array}{ll} m_0 & \textbf{noncritical section} \\ m_1 & \textbf{await}(y > 0)\{y := y-1\} \\ m_2 & \textbf{critical section} \\ m_3 & y := y+1 \end{array}\right.$$

Assumption: Strong Fairness

$\models$ **AF**$(x = 7)$**?** yes, under strong fairness

initialise x **to** 0; **initialise** y **to** 1

while $(true)$

$$\left\{\begin{array}{ll} \ell_0 & \textbf{noncritical section} \\ \ell_1 & \textbf{await}(y > 0)\{y := y-1\} \\ \ell_2 & \textbf{critical section; x := 7} \\ \ell_3 & y := y+1 \end{array}\right.$$

$\|$

while $(true)$

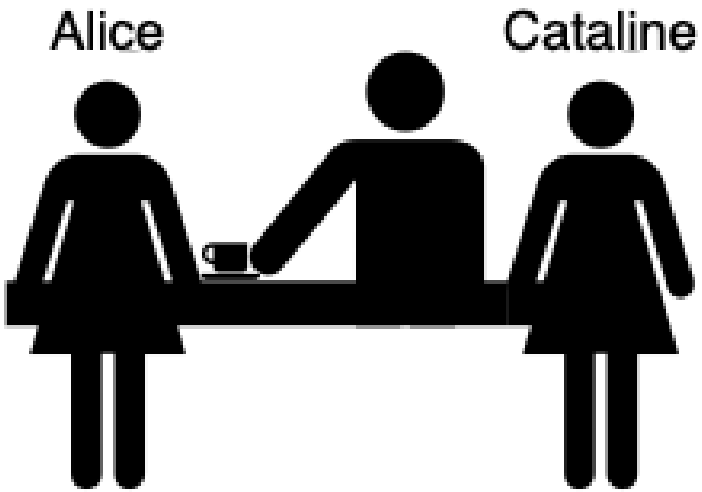
$$\left\{\begin{array}{ll} m_0 & \textbf{noncritical section} \\ m_1 & \textbf{await}(y > 0)\{y := y-1\} \\ m_2 & \textbf{critical section} \\ m_3 & y := y+1 \end{array}\right.$$

this is too strong

A (biased) Bartender

Alice and Cataline order Apple Juice and Coffee, resp.
do they get it?

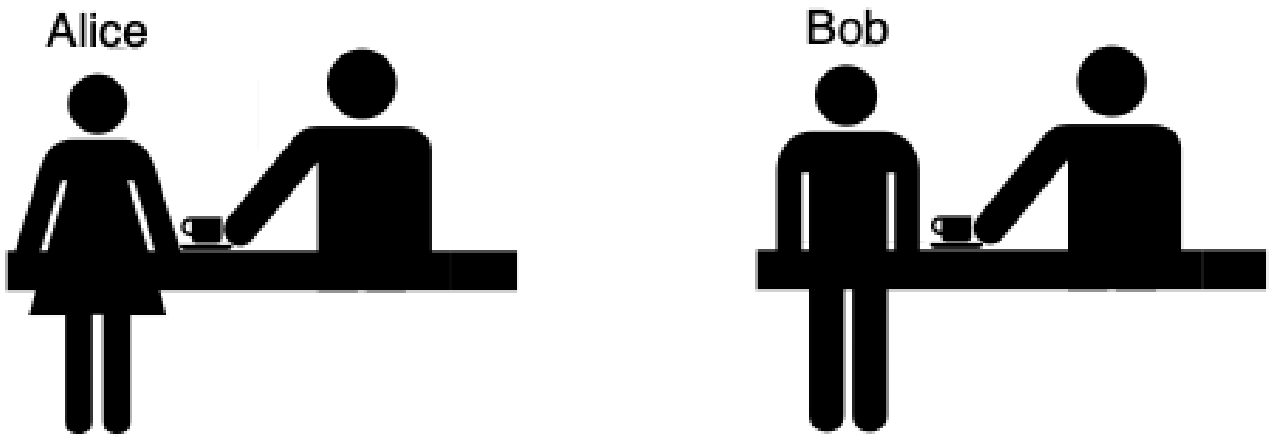
Alice||Bar||Bob



A (biased) Bartender

Alice and Bob order Apple Juice and any Beverage, resp.
do they get it?

$$\text{Alice} \| \text{Bar}_A \| \text{Bar}_B \| \text{Bob}$$



Justness

aim: develop a weaker assumption, that (typically) is warranted, yet still strong enough to allow rigorous mathematical proofs of relevant liveness properties.

Definition (Justness)

once a transition is enabled that stems from a set of *parallel components*, one (or more) of these components will eventually partake in a transition.

Justness

aim: develop a weaker assumption, that (typically) is warranted, yet still strong enough to allow rigorous mathematical proofs of relevant liveness properties.

Definition (Justness)

once a transition is enabled that stems from a set of *parallel components*, **one (or more)** of these components will eventually partake in a transition.

Formal Definition

Definition

A path π in a transition system is *just* if for each transition t with $s := \mathit{source}(t) \in \pi$, a transition u occurs in π past the occurrence of s, such that $t \not\prec\bullet\ u$.

$t \not\prec\bullet\ u$ means that a component affected by the execution of u is a necessary participant in the execution of t

Necessary and Affected Components

idea if a transition u occurs that has no influence on components in $npc(t)$, then the internal state of those components is unchanged any synchronisation between these components that was possible before u occurred, is still possible afterwards

Necessary and Affected Components

Definition

$$t \rightharpoonup u \quad \text{iff} \quad \mathit{npc}(t) \cap \mathit{afc}(u) = \emptyset$$

- set of components $\mathscr{C}$
- necessary components: $\mathit{npc} : \mathit{Tr} \to \mathscr{P}(\mathscr{C})$
- affected components: $\mathit{afc} : \mathit{Tr} \to \mathscr{P}(\mathscr{C})$

- often $\mathit{npc}(t) \subseteq \mathit{afc}(t)$
- if $t, u \in \mathit{Tr}$ with $\mathit{source}(t) = \mathit{source}(u)$ and $\mathit{npc}(t) \cap \mathit{afc}(u) = \emptyset$ then there is a transition $v \in \mathit{Tr}$ with $\mathit{source}(v) = \mathit{target}(u)$ and $\mathit{npc}(v) = \mathit{npc}(t)$.

Necessary and Affected Components (Remark)

Observation

In many formalisms, such as CCS, $\frown\!\bullet$ can be defined in a generic way (the user does not have to do anything)

Justness: Examples

$\models$ **AF**$(\mathrm{x} = 7)$ **under justness?**

$$\textbf{initialise } \mathrm{x} \textbf{ to } 0; \textbf{initialise } \mathrm{y} \textbf{ to } 0$$

$$\begin{array}{rcl} \textbf{while } (true) & \Big\| & x := 7; \\ \mathrm{y} := \mathrm{y}{+}1 & \Big\| & \end{array}$$

Justness: Examples

$\models$ **AF**$(\mathrm{x} = 7)$ **under justness?** yes

$$\textbf{initialise } \mathrm{x} \textbf{ to } 0; \textbf{initialise } \mathrm{y} \textbf{ to } 0$$

$$\begin{array}{rcl} \textbf{while } (true) & \| & x := 7; \\ \mathrm{y} := \mathrm{y}{+}1 & \| & \end{array}$$

Justness: Examples

$\models$ **AF**$(x = 7)$ **under justness?** yes

initialise x **to** 0; **initialise** y **to** 0

while $(true)$ $\quad\|\quad$ $x := 7;$
y := y+1 $\quad\|\quad$

$\models$ **AF**$(x = 7)$ **under justness?**

initialise x **to** 0; **initialise** y **to** 0

while $(true)$
choose
if $true$ **then** y := y+1;
if x = 0 **then** x := 7;
end

Justness: Examples

$\models$ **AF**$(x = 7)$ **under justness?** yes

initialise x **to** 0; **initialise** y **to** 0

while $(true)$ y := y+1 $\;\|\;$ $x := 7;$

$\models$ **AF**$(x = 7)$ **under justness?** no

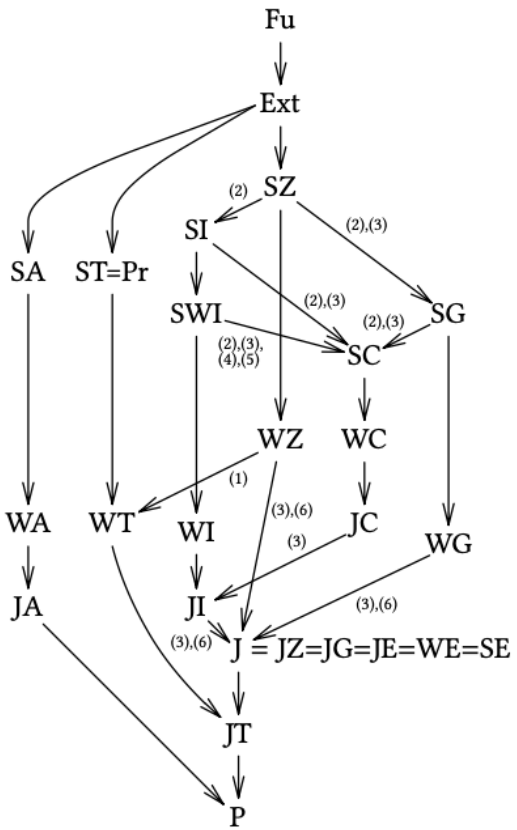
initialise x **to** 0; **initialise** y **to** 0

while $(true)$
  choose
    if $true$ **then** y := y+1;
    if x = 0 **then** x := 7;
  end

Assumption in the Literature (ACMSurvey'20)

Life is not fair, and neither are computers

- classified all notions of fairness in literature
- all are too strong for some scenarios
- except progress, which is too weak
- introduced justness in between



Take Away

- The real world is not fair.
  Most fairness assumptions are often too strong an assumption.
- Justness is a weaker assumption., I believe that it typically is warranted, yet still strong enough to allow rigorous mathematical proofs of relevant liveness properties.