Verifying Liveness Properties: Assumptions and Problems

Peter Höfner
(joint work with R. van Glabbeek and V. Dyseryn)

IFIP 2.3, July 2017, Mooloolaba

Motivation

- Verification of/Reasoning about *Safety* Properties
  - many applications
    - routing protocols
    - mutual exclusion
    - TLBs (Gerwin’s talk)
    - program/functional correctness (David’s talk)

Motivation

- Verification of/Reasoning about *Safety* Properties
  - many applications
    - routing protocols
    - mutual exclusion
    - TLBs (Gerwin’s talk)
    - program/functional correctness (David’s talk)
  - “easy” to achieve
    - at least we know what to do
    - existence of solid theoretical foundations
      - rely guarantee/Owicki-Gries/concurrent separation logic (Thomas)
      - standard techniques relate (labelled) transition systemssimulation, bisimulation, refinement, …

Motivation (2)

- Verification of/Reasoning about *Liveness* Properties
  - many applications
    - routing protocols
    - mutual exclusion
    - TLBs
    - …

Motivation (2)

- Verification of/Reasoning about *Liveness* Properties
  - many applications
    - routing protocols
    - mutual exclusion
    - TLBs
    - …

- “easy” to achieve?

Motivation (2)

- Verification of/Reasoning about *Liveness* Properties
  - many applications
    – routing protocols
    – mutual exclusion
    – TLBs
    – …

- “easy” to achieve? NO
    – do I (we) even know what we do?

An Example:
Peterson's Mutual Exclusion Protocol

$$
\underline{\textbf{Process A}}
$$

$$
\textbf{repeat forever}
\left\{
\begin{array}{ll}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & readyA := true \\
\ell_3 & turn := B \\
\ell_4 & \textbf{await}\ (readyB = false \vee turn = A) \\
\ell_5 & \textbf{critical section} \\
\ell_6 & readyA := false
\end{array}
\right.
$$

- a similar process for B (each line is atomic)
- $readyA$, $readyB$ and $turn$ are shared variables (Booleans)
- initial state: A is in a state before ℓ_1, and $readyA = turn = false$

Peterson's Mutual Exclusion Protocol: Safety

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & readyA := true \\ \ell_3 & turn := B \\ \ell_4 & \textbf{await}\ (readyB = false \vee turn = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & readyA := false \end{array}\right.$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & readyB := true \\ m_3 & turn := A \\ m_4 & \textbf{await}\ (readyA = false \vee turn = B) \\ m_5 & \textbf{critical section} \\ m_6 & readyB := false \end{array}\right.$$

- Safety:
  there is at most one process in the critical section at any time

Peterson's Mutual Exclusion Protocol: Safety

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & \mathit{readyA} := \mathit{true} \\ \ell_3 & \mathit{turn} := B \\ \ell_4 & \textbf{await}\ (\mathit{readyB} = \mathit{false} \vee \mathit{turn} = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & \mathit{readyA} := \mathit{false} \end{array}\right.$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & \mathit{readyB} := \mathit{true} \\ m_3 & \mathit{turn} := A \\ m_4 & \textbf{await}\ (\mathit{readyA} = \mathit{false} \vee \mathit{turn} = B) \\ m_5 & \textbf{critical section} \\ m_6 & \mathit{readyB} := \mathit{false} \end{array}\right.$$

- Safety:
  there is at most one process in the critical section at any time

$$\Box(\neg(\ell_5 \wedge m_5))$$

Peterson's Mutual Exclusion Protocol: Safety

Process A

$$
\textbf{repeat forever}
\left\{
\begin{array}{ll}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & \mathit{readyA} := \mathit{true} \\
\ell_3 & \mathit{turn} := B \\
\ell_4 & \textbf{await}\ (\mathit{readyB} = \mathit{false} \vee \mathit{turn} = A) \\
\ell_5 & \textbf{critical section} \\
\ell_6 & \mathit{readyA} := \mathit{false}
\end{array}
\right.
$$

Process B

$$
\textbf{repeat forever}
\left\{
\begin{array}{ll}
m_1 & \textbf{noncritical section} \\
m_2 & \mathit{readyB} := \mathit{true} \\
m_3 & \mathit{turn} := A \\
m_4 & \textbf{await}\ (\mathit{readyA} = \mathit{false} \vee \mathit{turn} = B) \\
m_5 & \textbf{critical section} \\
m_6 & \mathit{readyB} := \mathit{false}
\end{array}
\right.
$$

- Safety:
  there is at most one process in the critical section at any time

$$\Box(\neg(\ell_5 \wedge m_5))$$

proof: homework

Peterson's Mutual Exclusion Protocol: Liveness

<u>Process A</u>

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & readyA := true \\ \ell_3 & turn := B \\ \ell_4 & \textbf{await}\ (readyB = false \vee turn = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & readyA := false \end{array}\right.$$

<u>Process B</u>

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & readyB := true \\ m_3 & turn := A \\ m_4 & \textbf{await}\ (readyA = false \vee turn = B) \\ m_5 & \textbf{critical section} \\ m_6 & readyB := false \end{array}\right.$$

- Liveness:
  if a process wants to access the critical section,
  it will eventually do so

Peterson's Mutual Exclusion Protocol: Liveness

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & \mathit{readyA} := \mathit{true} \\ \ell_3 & \mathit{turn} := B \\ \ell_4 & \textbf{await}\ (\mathit{readyB} = \mathit{false} \vee \mathit{turn} = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & \mathit{readyA} := \mathit{false} \end{array}\right.$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & \mathit{readyB} := \mathit{true} \\ m_3 & \mathit{turn} := A \\ m_4 & \textbf{await}\ (\mathit{readyA} = \mathit{false} \vee \mathit{turn} = B) \\ m_5 & \textbf{critical section} \\ m_6 & \mathit{readyB} := \mathit{false} \end{array}\right.$$

- Liveness:
  if a process wants to access the critical section,
  it will eventually do so

  formalisation:

Peterson's Mutual Exclusion Protocol: Liveness

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & \mathit{readyA} := \mathit{true} \\ \ell_3 & \mathit{turn} := B \\ \ell_4 & \textbf{await}\ (\mathit{readyB} = \mathit{false} \vee \mathit{turn} = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & \mathit{readyA} := \mathit{false} \end{array}\right.$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & \mathit{readyB} := \mathit{true} \\ m_3 & \mathit{turn} := A \\ m_4 & \textbf{await}\ (\mathit{readyA} = \mathit{false} \vee \mathit{turn} = B) \\ m_5 & \textbf{critical section} \\ m_6 & \mathit{readyB} := \mathit{false} \end{array}\right.$$

- Liveness:
  if a process wants to access the critical section,
  it will eventually do so

  formalisation:
  proof: does the property even hold

Assumption I: Progress

A system in a state that admits an action will eventually perform an action.

Peterson's Mutual Exclusion Protocol: Liveness

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & readyA := true \\ \ell_3 & turn := B \\ \ell_4 & \textbf{await}\ (readyB = false \lor turn = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & readyA := false \end{array}\right.$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & readyB := true \\ m_3 & turn := A \\ m_4 & \textbf{await}\ (readyA = false \lor turn = B) \\ m_5 & \textbf{critical section} \\ m_6 & readyB := false \end{array}\right.$$

- Liveness:
  if a process wants to access the critical section, it will eventually do so

  proof: does the property even hold

Assumption I: Progress

A process in a state that admits a non-blocking action will eventually perform an action.

- *non-blocking action: any action that does not require cooperation*

Peterson's Mutual Exclusion Protocol: Liveness

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & readyA := true \\ \ell_3 & turn := B \\ \ell_4 & \textbf{await}\ (readyB = false \vee turn = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & readyA := false \end{array}\right.$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & readyB := true \\ m_3 & turn := A \\ m_4 & \textbf{await}\ (readyA = false \vee turn = B) \\ m_5 & \textbf{critical section} \\ m_6 & readyB := false \end{array}\right.$$

- Liveness:
  if a process wants to access the critical section, it will eventually do so

  proof: does the property even hold

Peterson's Mutual Exclusion Protocol: Liveness

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & readyA := true \\ \ell_3 & turn := B \\ \ell_4 & \textbf{await}\ (readyB = false \vee turn = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & readyA := false \end{array}\right.$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & readyB := true \\ m_3 & turn := A \\ m_4 & \textbf{await}\ (readyA = false \vee turn = B) \\ m_5 & \textbf{critical section} \\ m_6 & readyB := false \end{array}\right.$$

$$\| \ readyA \ \| \ turn \ \| \ readyB \ \|$$

memory

- Liveness:
  if a process wants to access the critical section, it will eventually do so

  proof: does the property even hold

Standard Assumption: Fairness

If an action is enabled infinitely often/perpetually, the action will be taken

- there are about 25 different versions of fairness in the literature

Standard Assumption: Fairness

If an action is enabled infinitely often/perpetually, the action will be taken

- there are about 25 different versions of fairness in the literature
- all of them imply liveness

Fairness Could be Considered Harmful

$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \ \Big\| \ \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array}$$

- Should $\Diamond(x == 1)$ hold?

.   . .   .

Fairness Could be Considered Harmful

$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \;\Bigg\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array} \;\Bigg\|\; mem_x \;\Bigg\|\; mem_y$$

- Should $\Diamond(x == 1)$ hold?

. . . .

Fairness Could be Considered Harmful

$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \ \Big\| \ \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array} \ \Big\| \ mem_x \ \Big\| \ mem_y$$

- Should $\Diamond(x == 1)$ hold?
  - if the program runs on two machines **YES**
    (if it runs on the same machine the OS hopefully guarantees this)
  - progress cannot guarantee this
  - addition of a fairness assumption seems appropriate

Fairness Could be Considered Harmful

$$\begin{array}{l} \textbf{repeat forever} \\ \quad \textbf{if}(x == 0) \textbf{ then } x := 1 \\ \; [] \; y := y + 1 \end{array} \quad \Bigg\| \; mem_x \; \Bigg\| \; mem_y$$

- Should $\Diamond(x == 1)$ hold?

Fairness Could be Considered Harmful

$$
\begin{array}{l}
\textbf{repeat forever} \\
\quad \textbf{if}(x == 0)\ \textbf{then}\ x := 1 \\
\ [\,]\ y := y + 1
\end{array}
\;\Big\|\; mem_x \;\Big\|\; mem_y
$$

- Should $\Diamond(x == 1)$ hold?
  - **NO**
  - consider the program to be a specification and $\begin{array}{c}\textbf{repeat forever}\\ y := y+1\end{array}$ as implementation

Fairness

- required on top of a specification/implementation
  - rules out particular (infeasible) paths
    (similar to progress)
  - requires deep understanding of the program
    (in contrast to progress)

  - progress and fairness are of different nature
    – progress guarantees continuation, independent of action
    – fairness guarantees particular actions to happen

Fairness

- required on top of a specification/implementation
  - rules out particular (infeasible) paths
    (similar to progress)
  - requires deep understanding of the program
    (in contrast to progress)
    - dangerous since you may enforce properties that do not hold
      (addition of fairness should be considered harmful)

  - progress and fairness are of different nature
    - progress guarantees continuation, independent of action
    - fairness guarantees particular actions to happen

Formalising and Proving Properties

- most formalisms are based on labelled transition systems (LTSs)

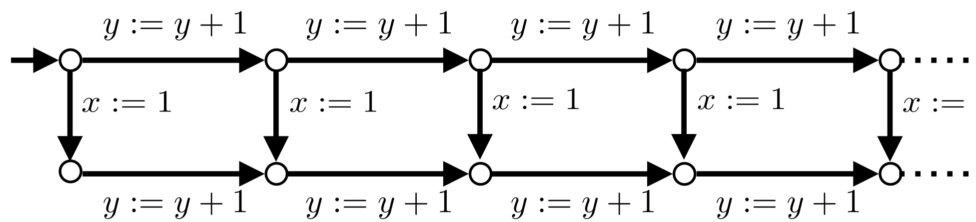
$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \;\Big\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array} \;\Big\|\; mem_x \;\Big\|\; mem_y$$

$$\begin{array}{l}\textbf{repeat forever}\\ \qquad \textbf{if}(x == 0)\ \textbf{then}\ x := 1\\ \quad [\,]\ y := y + 1\end{array} \;\Big\|\; mem_x \;\Big\|\; mem_y$$

Formalising and Proving Properties

- most formalisms are based on labelled transition systems (LTSs)

$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \;\Big\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad y := y+1\end{array} \;\Big\|\; mem_x \;\Big\|\; mem_y$$

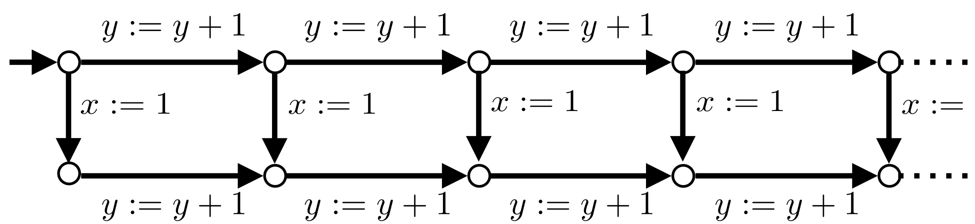


$$\begin{array}{l}\textbf{repeat forever}\\ \quad \textbf{if}(x == 0)\ \textbf{then}\ x := 1\\ [\,]\ y := y+1\end{array} \;\Big\|\; mem_x \;\Big\|\; mem_y$$

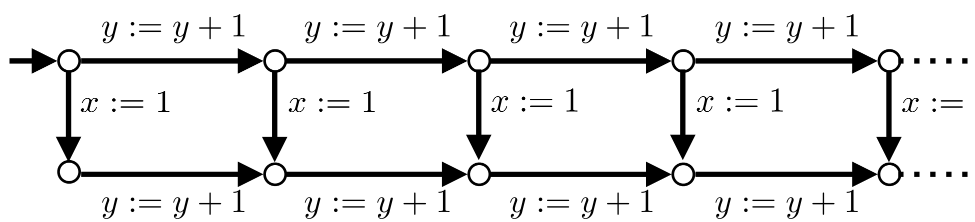
Formalising and Proving Properties

- most formalisms are based on labelled transition systems (LTSs)

$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \ \Big\| \ \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array} \ \Big\| \ mem_x \ \Big\| \ mem_y$$



$$\begin{array}{l}\textbf{repeat forever}\\ \quad \textbf{if}(x == 0)\ \textbf{then}\ x := 1\\ \,[]\ y := y + 1\end{array} \ \Big\| \ mem_x \ \Big\| \ mem_y$$



Summary (intermediate)

- progress not strong enough
- fairness should be considered to be harmful
  - may rule out too many paths
  - may be unrealistic (e.g. implementing a fair scheduler)
- if we find a better solution we loose property preservation under bisimulation (and other relations)

- progress is a property on single processes,
  we should consider interaction
  (in particular when the (shared) memory is modelled)

Justness

If a combination of components in a parallel composition is in a state that admits a non-blocking action, then one (or more) of them will eventually partake in an action

Justness

If a combination of components in a parallel composition is in a state that admits a non-blocking action, then one (or more) of them will eventually partake in an action

- Progress of (combination of) components

- it is a progress property rather than a fairness assumption
- there is a formal definition in CCS

Justness (2)

- justness can distinguish the two programs

$$\mathbf{if}(x == 0)\ \mathbf{then}\ x := 1 \;\left\|\; \begin{array}{l}\mathbf{repeat\ forever}\\ \quad y := y + 1\end{array} \;\right\|\; mem_x \;\|\; mem_y$$

$$\begin{array}{l}\mathbf{repeat\ forever}\\ \quad\quad \mathbf{if}(x == 0)\ \mathbf{then}\ x := 1\\ \quad [\!]\ y := y + 1\end{array} \;\|\; mem_x \;\|\; mem_y$$

Justness (2)

- justness can distinguish the two programs

$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \ \Big\|\ \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array} \ \Big\|\ mem_x \ \Big\|\ mem_y$$

$$\begin{array}{l}\textbf{repeat forever}\\ \quad \textbf{if}(x == 0)\ \textbf{then}\ x := 1\\ \ [\,]\ y := y + 1\end{array} \ \Big\|\ mem_x \ \Big\|\ mem_y$$

- so, are we done?

Coloured Labelled Transition Systems

- idea: label LTS with component performing the action

$$\mathbf{if}(x == 0)\ \mathbf{then}\ x := 1 \;\Bigg\|\; \begin{array}{l}\mathbf{repeat\ forever}\\ \quad y := y + 1\end{array} \;\Bigg\|\; mem_x \;\Bigg\|\; mem_y$$

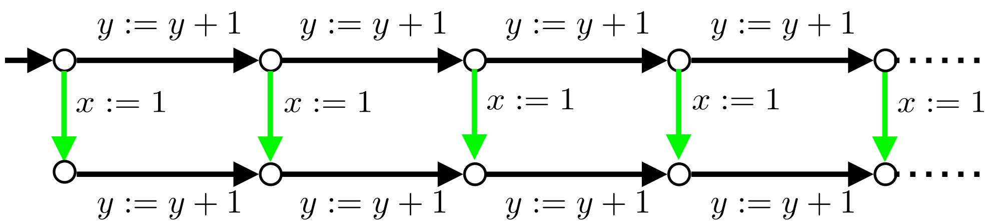
$$\begin{array}{l}\mathbf{repeat\ forever}\\ \quad\ \ \mathbf{if}(x == 0)\ \mathbf{then}\ x := 1\\ \quad [\,]\ y := y + 1\end{array} \;\Bigg\|\; mem_x \;\Bigg\|\; mem_y$$

- (you may want to add multicolors)

Coloured Labelled Transition Systems

- idea: label LTS with component performing the action

$$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \ \Big\|\ \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array} \ \Big\|\ mem_x \ \Big\|\ mem_y$$



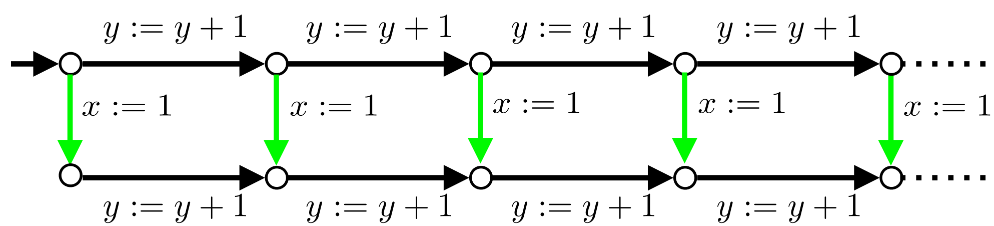
$$\begin{array}{l}\textbf{repeat forever}\\ \quad \textbf{if}(x == 0)\ \textbf{then}\ x := 1\\ \ [\,]\ y := y + 1\end{array} \ \Big\|\ mem_x \ \Big\|\ mem_y$$

- (you may want to add multicolors)

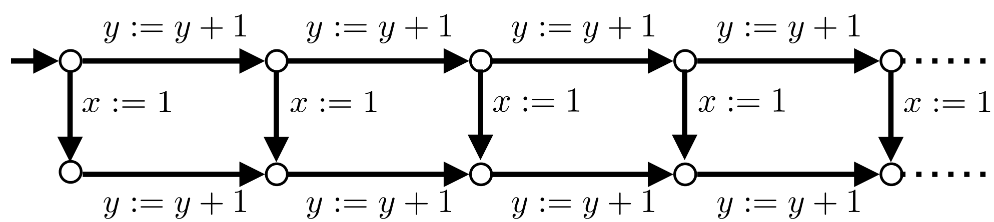
Coloured Labelled Transition Systems

- idea: label LTS with component performing the action

$\textbf{if}(x == 0)\ \textbf{then}\ x := 1 \;\Big\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad y := y + 1\end{array} \;\Big\|\; mem_x \;\Big\|\; mem_y$



$\begin{array}{l}\textbf{repeat forever}\\ \quad \textbf{if}(x == 0)\ \textbf{then}\ x := 1\\ \;[]\; y := y + 1\end{array} \;\Big\|\; mem_x \;\Big\|\; mem_y$



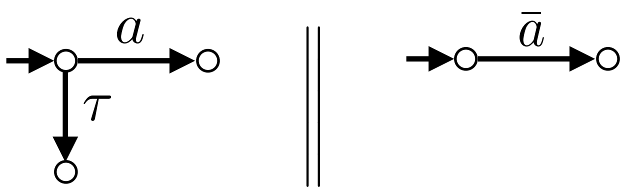
- (you may want to add multicolors)

Justness - Simplification Possible?

*If a combination of components in a parallel composition is in a state that admits a non-blocking action, then **one (or more) of them** will eventually partake in an action*

Justness - Simplification Possible?

If a combination of components in a parallel composition is in a state that admits a non-blocking action, then ***one (or more) of them*** *will eventually partake in an action*



Yet Another Example

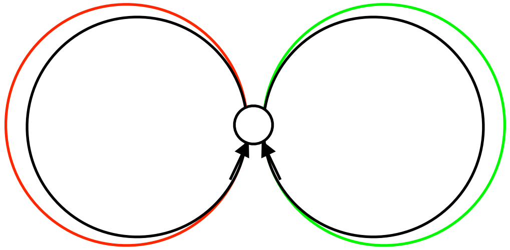
$$\begin{array}{l}\textbf{repeat forever}\\ \quad x := x + 1\end{array} \;\Bigg\|\; mem_x \;\Bigg\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad x := -1\end{array}$$

- under justness, does $\Diamond(x == -1)$ hold?

Yet Another Example

$$\begin{array}{l}\textbf{repeat forever}\\ \quad x := x + 1\end{array} \;\Bigg\|\; mem_x \;\Bigg\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad x := -1\end{array}$$

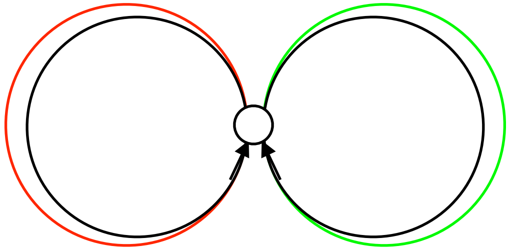
- under justness, does $\Diamond(x == -1)$ hold?



Yet Another Example

$$\begin{array}{l}\textbf{repeat forever}\\ \quad x := x+1\end{array} \;\Bigg\|\; mem_x \;\Bigg\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad x := -1\end{array}$$

- under justness, does $\Diamond(x == -1)$ hold? **NO**



Back to Peterson

<u>**Process A**</u>

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & readyA := true \\ \ell_3 & turn := B \\ \ell_4 & \textbf{await}\ (readyB = false \vee turn = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & readyA := false \end{array}\right.$$

$$\Big\| \; readyA \; \Big\| \; turn \; \Big\| \; readyB \; \Big\|$$

<u>**Process B**</u>

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & readyB := true \\ m_3 & turn := A \\ m_4 & \textbf{await}\ (readyA = false \vee turn = B) \\ m_5 & \textbf{critical section} \\ m_6 & readyB := false \end{array}\right.$$

- under justness, does the liveness property hold?

Back to Peterson

Process A

repeat forever

$$\left\{\begin{array}{ll} \ell_1 & \textbf{noncritical section} \\ \ell_2 & \mathit{readyA} := \mathit{true} \\ \ell_3 & \mathit{turn} := B \\ \ell_4 & \textbf{await}\ (\mathit{readyB} = \mathit{false} \vee \mathit{turn} = A) \\ \ell_5 & \textbf{critical section} \\ \ell_6 & \mathit{readyA} := \mathit{false} \end{array}\right.$$

$$\Big\|\ \mathit{readyA}\ \Big\|\ \mathit{turn}\ \Big\|\ \mathit{readyB}\ \Big\|$$

Process B

repeat forever

$$\left\{\begin{array}{ll} m_1 & \textbf{noncritical section} \\ m_2 & \mathit{readyB} := \mathit{true} \\ m_3 & \mathit{turn} := A \\ m_4 & \textbf{await}\ (\mathit{readyA} = \mathit{false} \vee \mathit{turn} = B) \\ m_5 & \textbf{critical section} \\ m_6 & \mathit{readyB} := \mathit{false} \end{array}\right.$$

- under justness, does the liveness property hold?
  NO (reading can block writing)

Reading blocks Writing

- is this realistic? probably not
- extensions of well-established formalisms avoid this
  - Petri Nets with Read Arcs
  - CCS with signals
  - also avoids reading to block reading (or other actions)

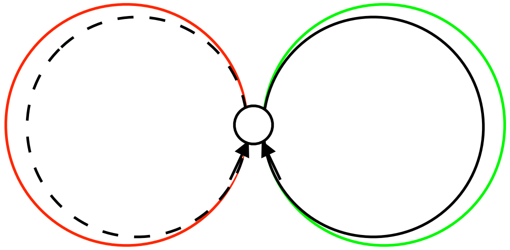
- extensions distinguish “state-changing” and “read” actions

- under these extensions Peterson can be proven to satisfy the liveness property, *under justness only*

Coloured LTSs adapted

$$\begin{array}{l}\textbf{repeat forever}\\ \quad read(x)\end{array} \;\Bigg\|\; mem_x \;\Bigg\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad write(x)\end{array}$$

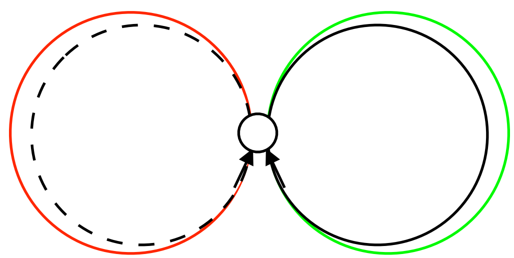
- insert active and passive partners (make reading "asymmetric")



Coloured LTSs adapted

$$\begin{array}{l}\textbf{repeat forever}\\ \quad read(x)\end{array} \;\Bigg\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad \overline{read(x)}\\ \;[]\; write(x)\end{array} \;\Bigg\|\; \begin{array}{l}\textbf{repeat forever}\\ \quad \overline{write(x)}\end{array}$$

- insert active and passive partners (make reading "asymmetric")



Peterson for N Processes

$\underline{\textbf{Process i}}$ $\quad (i \in \{1, \ldots, N\})$

repeat forever

$$\left\{\begin{array}{ll}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & \textbf{for } j \textbf{ in } 1 \ldots N-1 \\
\ell_3 & \quad \mathit{room}[i] := j \\
\ell_4 & \quad \mathit{last}[j] := i \\
\ell_5 & \textbf{await } (\mathit{last}[j] \neq i \vee (\forall k \neq i,\ \mathit{room}[k] < j)) \\
\ell_6 & \textbf{critical section} \\
\ell_7 & \mathit{room}[i] := \mathit{0}
\end{array}\right.$$

- safety:
- liveness:

Peterson for N Processes

$\underline{\textbf{Process i}}$ $\;(i \in \{1, \ldots, N\})$

$$
\textbf{repeat forever}
\left\{
\begin{array}{ll}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & \textbf{for } j \textbf{ in } 1 \ldots N-1 \\
\ell_3 & \quad room[i] := j \\
\ell_4 & \quad last[j] := i \\
\ell_5 & \textbf{await } (last[j] \neq i \vee (\forall k \neq i,\; room[k] < j)) \\
\ell_6 & \textbf{critical section} \\
\ell_7 & room[i] := 0
\end{array}
\right.
$$

- safety: **YES**, but …
- liveness:

Peterson for N Processes

$\underline{\textbf{Process i}}$ $(i \in \{1, \ldots, N\})$

$$
\textbf{repeat forever}\\
\left\{\begin{array}{ll}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & \textbf{for } j \textbf{ in } 1 \ldots N-1 \\
\ell_3 & \quad \mathit{room}[i] := j \\
\ell_4 & \quad \mathit{last}[j] := i \\
\ell_5 & \textbf{await } (\mathit{last}[j] \neq i \vee (\forall k \neq i,\ \mathit{room}[k] < j)) \\
\ell_6 & \textbf{critical section} \\
\ell_7 & \mathit{room}[i] := \mathit{0}
\end{array}\right.
$$

- safety: **YES**, but …
- liveness: progress:

Peterson for N Processes

$\underline{\textbf{Process i}}$ $\;(i \in \{1, \ldots, N\})$

repeat forever

$$
\left\{
\begin{array}{ll}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & \textbf{for } j \textbf{ in } 1 \ldots N-1 \\
\ell_3 & \quad \mathit{room}[i] := j \\
\ell_4 & \quad \mathit{last}[j] := i \\
\ell_5 & \textbf{await } (\mathit{last}[j] \neq i \vee (\forall k \neq i,\; \mathit{room}[k] < j)) \\
\ell_6 & \textbf{critical section} \\
\ell_7 & \mathit{room}[i] := \mathit{0}
\end{array}
\right.
$$

- safety: **YES**, but …
- liveness: progress: **NO,**

Peterson for N Processes

$$\underline{\textbf{Process i}} \quad (i \in \{1, \dots, N\})$$

$$
\begin{array}{l}
\textbf{repeat forever} \\
\left\{
\begin{array}{ll}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & \textbf{for } j \textbf{ in } 1 \dots N-1 \\
\ell_3 & \quad \mathit{room}[i] := j \\
\ell_4 & \quad \mathit{last}[j] := i \\
\ell_5 & \textbf{await } (\mathit{last}[j] \neq i \vee (\forall k \neq i,\ \mathit{room}[k] < j)) \\
\ell_6 & \textbf{critical section} \\
\ell_7 & \mathit{room}[i] := \mathit{0}
\end{array}
\right.
\end{array}
$$

- safety: **YES**, but …
- liveness: progress: **NO**,
  justness:

Peterson for N Processes

$$
\underline{\textbf{Process i}} \quad (i \in \{1, \ldots, N\})
$$

$$
\textbf{repeat forever}
\begin{cases}
\ell_1 & \textbf{noncritical section} \\
\ell_2 & \textbf{for } j \textbf{ in } 1 \ldots N-1 \\
\ell_3 & \quad \mathit{room}[i] := j \\
\ell_4 & \quad \mathit{last}[j] := i \\
\ell_5 & \textbf{await } (\mathit{last}[j] \neq i \vee (\forall k \neq i,\ \mathit{room}[k] < j)) \\
\ell_6 & \textbf{critical section} \\
\ell_7 & \mathit{room}[i] := \mathit{0}
\end{cases}
$$

- safety: **YES**, but …
- liveness: progress: **NO**,
  justness: **NO** (two write actions in parallel)

Write/Write Actions
What about Reality?

- write can block writing
  - Peterson for N processes (PNP) has no liveness property

- write/write can happen in parallel and one action “wins”
  - PNP is safe and live
  - how to model this in (coloured) LTSs
    - adapt active/passive components?
    - parallel writing some kind of broadcast?

- write and write can happen in parallel (potentially producing garbage)
  - PNP is “alive”, but not safe any longer
  - remark: no problem with normal Peterson algorithm
  - remark: no garbage for Boolean (maybe false value, though)

Write/Write Actions
What about Reality?

- write can block writing
  - Peterson for N processes (PNP) has no liveness property

- write/write can happen in parallel and one action “wins”
  - PNP is safe and live
  - how to model this in (coloured) LTSs
    - adapt active/passive components?
    - parallel writing some kind of broadcast?

- write and write can happen in parallel (potentially producing garbage)
  - PNP is “alive”, but not safe any longer
  - remark: no problem with normal Peterson algorithm
  - remark: no garbage for Boolean (maybe false value, though)

Lamport’s Bakery Algorithm is safe and live!!

Conclusion:
Assumptions and Problems with Liveness

- formalisation can be error prone
- assumption I: progress
- assumption II: fairness - **dangerous**
  better justness

- be careful with bisimulation, simulation, refinement, …
  - use coloured extensions

- but what about reality

Conclusion:
Assumptions and Problems with Liveness

- formalisation can be error prone
- assumption I: progress
- assumption II: fairness - **dangerous**
  better justness

- be careful with bisimulation, simulation, refinement, …
  - use coloured extensions

- but what about reality

Did we get the foundations right?

Thank you

Data61
Peter Höfner

t +61 2 9490 5861
e peter.hoefner@data61.csiro.au
w www.data61.csiro.au