Verifying Liveness Properties: Assumptions and Problems

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

IFIP 2.3, July 2017, Mooloolaba

www.data61.csiro.au

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
    - …

- "easy" to achieve? NO
  - do I (we) even know what we do?

An Example: Peterson's Mutual Exclusion Protocol

```
Process A

repeat forever
  ℓ_1   noncritical section
  ℓ_2   $readyA := true$
  ℓ_3   $turn := B$
  ℓ_4   await $(readyB = false \vee turn = A)$
  ℓ_5   critical section
  ℓ_6   $readyA := false$
```

- 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
    ℓ1  noncritical section
    ℓ2  $readyA := true$
    ℓ3  $turn := B$
    ℓ4  await $(readyB = false \lor turn = A)$
    ℓ5  critical section
    ℓ6  $readyA := false$
```

```
Process B
repeat forever
    m1  noncritical section
    m2  $readyB := true$
    m3  $turn := A$
    m4  await $(readyA = false \lor turn = B)$
    m5  critical section
    m6  $readyB := false$
```

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

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

  proof: homework

Peterson's Mutual Exclusion Protocol: Liveness

<u>Process A</u>

```
repeat forever
    ℓ1  noncritical section
    ℓ2  $readyA := true$
    ℓ3  $turn := B$
    ℓ4  await $(readyB = false \lor turn = A)$
    ℓ5  critical section
    ℓ6  $readyA := false$
```

<u>Process B</u>

```
repeat forever
    m1  noncritical section
    m2  $readyB := true$
    m3  $turn := A$
    m4  await $(readyA = false \lor turn = B)$
    m5  critical section
    m6  $readyB := false$
```

- 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
    ℓ1  noncritical section
    ℓ2  $readyA := true$
    ℓ3  $turn := B$
    ℓ4  await $(readyB = false \vee turn = A)$
    ℓ5  critical section
    ℓ6  $readyA := false$
```

Process B

```
repeat forever
    m1  noncritical section
    m2  $readyB := true$
    m3  $turn := A$
    m4  await $(readyA = false \vee turn = B)$
    m5  critical section
    m6  $readyB := false$
```

- 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*

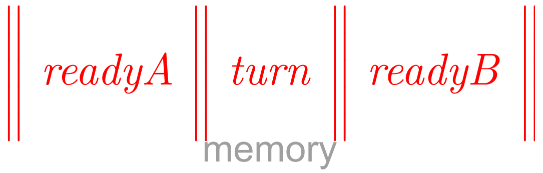
Peterson's Mutual Exclusion Protocol: Liveness

Process A

```
repeat forever
    $\ell_1$  noncritical section
    $\ell_2$  $readyA := true$
    $\ell_3$  $turn := B$
    $\ell_4$  await $(readyB = false \lor turn = A)$
    $\ell_5$  critical section
    $\ell_6$  $readyA := false$
```

Process B

```
repeat forever
    $m_1$  noncritical section
    $m_2$  $readyB := true$
    $m_3$  $turn := A$
    $m_4$  await $(readyA = false \lor turn = B)$
    $m_5$  critical section
    $m_6$  $readyB := false$
```



- 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
- all of them imply liveness

Fairness Could be Considered Harmful

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

- 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

```
$\textbf{repeat forever}$
    $\textbf{if}(x == 0)\ \textbf{then}\ x := 1$
  $[]\ y := y + 1$
```

$$\| \ mem_x \ \| \ mem_y$$

- 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

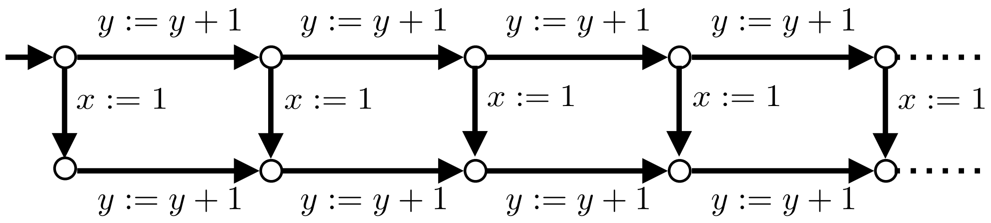
- required on top of a specification/implementation
  - rules out particular (infeasible) paths
    (similar to progress)
  - requires deep understanding of the program
    (in contrast to progress)
    – dangerous since you may enforce properties that do not hold
    (addition of fairness should be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

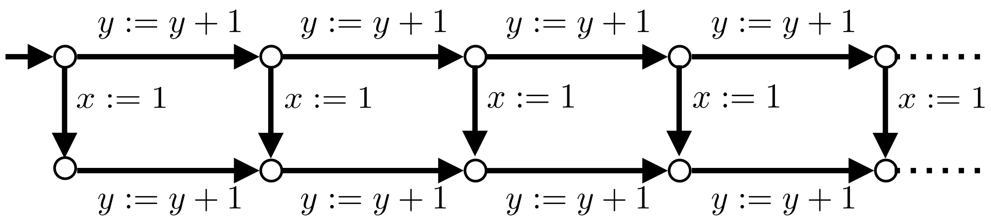
- most formalisms are based on labelled transition systems (LTSs)

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



```
repeat forever
    if$(x == 0)$ then $x := 1$
  [] $y := y + 1$
```

$$\| \; mem_x \; \| \; mem_y$$



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

- 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

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

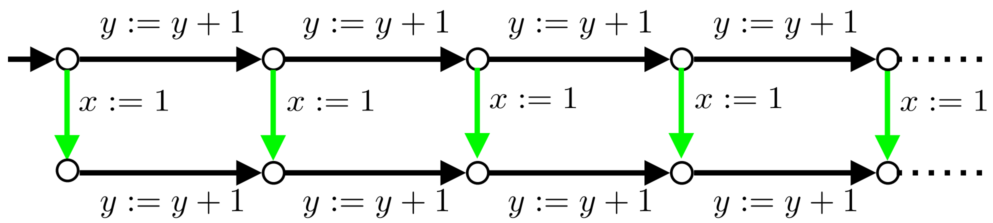
$$\begin{array}{l}\textbf{repeat forever}\\ \qquad \textbf{if}\,(x == 0)\ \textbf{then}\ x := 1\\ \quad [\,]\ y := y + 1\end{array} \;\Big\|\; mem_x \;\Big\|\; mem_y$$

- so, are we done?

Coloured Labelled Transition Systems

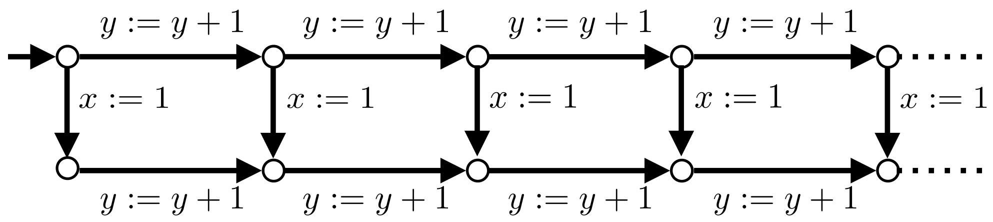
- idea: label LTS with component performing the action

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



```
repeat forever
    if(x == 0) then x := 1
  [] y := y + 1
```

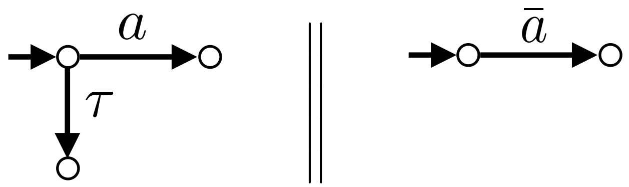
$$\|\ mem_x\ \|\ mem_y$$



- (you may want to add multicolors)

Justness - Simplification Possible?

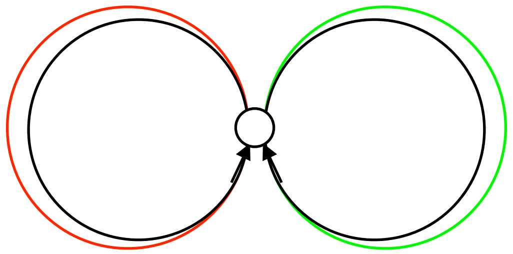
If a combination of components in a parallel composition is in a state that admits a non-blocking action, then ***one (or more) of them*** *will eventually partake in an action*



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

Process A

```
repeat forever
  ℓ_1  noncritical section
  ℓ_2  $readyA := true$
  ℓ_3  $turn := B$
  ℓ_4  await $(readyB = false \lor turn = A)$
  ℓ_5  critical section
  ℓ_6  $readyA := false$
```

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

Process B

```
repeat forever
  m_1  noncritical section
  m_2  $readyB := true$
  m_3  $turn := A$
  m_4  await $(readyA = false \lor turn = B)$
  m_5  critical section
  m_6  $readyB := false$
```

- 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)

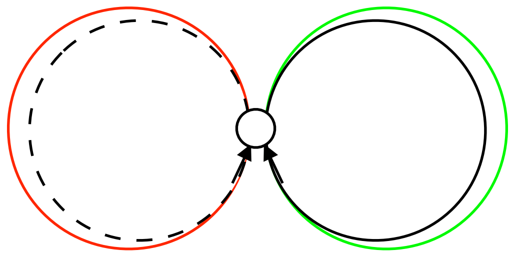
- extensions distinguish “state-changing” and “read” actions

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

Coloured LTSs adapted

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

- insert active and passive partners (make reading “asymmetric”)



Peterson for N Processes

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

```
repeat forever
  ℓ1  noncritical section
  ℓ2  for $j$ in $1 \ldots N-1$
  ℓ3      $room[i] := j$
  ℓ4      $last[j] := i$
  ℓ5  await $(last[j] \neq i \vee (\forall k \neq i,\ room[k] < j))$
  ℓ6  critical section
  ℓ7  $room[i] := 0$
```

- 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)

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?

DATA
61

Thank you

Data61
Peter Höfner

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

CSIRO