Justness: when progress is too weak and fairness it too strong

Peter Höfner
(joint work with R. van Glabbeek)

IFIP 2.3, February 2019, York

Motivation

- verification of/reasoning about *safety* properties
  - many applications
    - routing protocols
    - mutual exclusion
    - garbage collection
  - "easy" to achieve
    - at least we know what to do
    - existence of solid theoretical foundations
      - rely guarantee/Owicki-Gries/(concurrent) separation logic
      - standard techniques relate (labelled) transition systems simulation, bisimulation, refinement, …

Motivation

- examples
  - Garbage Collection
    "No memory is deleted that still used"

    the program **skip** satisfies the safety property

  - Mutual Exclusion Protocols
    "Critical Section is not accessed by more than one process at a time"

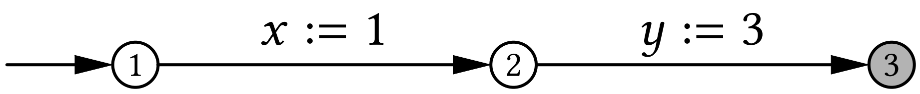
    the program that does not allow any process to enter the critical section satisfies the safety property

  - **Liveness is as important as Safety**

Liveness by Example

- does the following program satisfy $\mathbf{AF}(y = 3)$

$$x := 1;\ y := 3$$



Progress

a (transition) system in a state that admits an outgoing (non-blocking) transition will eventually progress, i.e., perform a transition

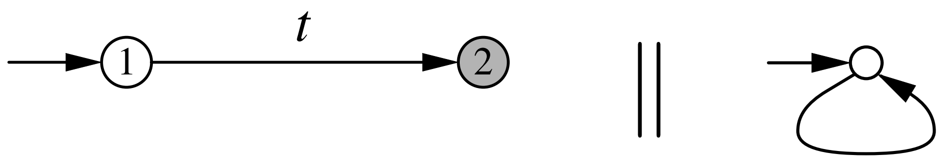
- assumes that no process gets stuck in a state with outgoing transitions
- progress is widely applied (often implicitly, e.g. CCS)
- Misra calls it “minimal progress”

- assumes also that atomic actions always terminate

- generalised to non-blocking actions;
  a non-blocking action cannot be blocked by the environment
  (assignment, I drinking a beer, …)

Progress is Too Weak

- assume the following independent programs



- does $\mathbf{AF}(@2)$ hold (under the assumption of progress)?

- progress is too weak
- progress is not compositional

Completeness Criterion

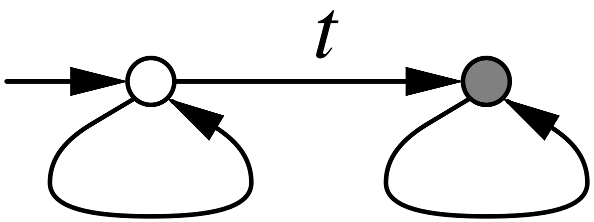
- progress (and other fairness assumptions) rule out paths in a transition system
- progress rules out “incomplete” paths

- completeness criterion F is stronger than H if it rules out at least at least all paths that are ruled out by H

- to verify liveness properties we need something stronger than progress (this is well known)

Weak and Strong Fairness

- $\mathbf{AF}(@2)$ does not hold

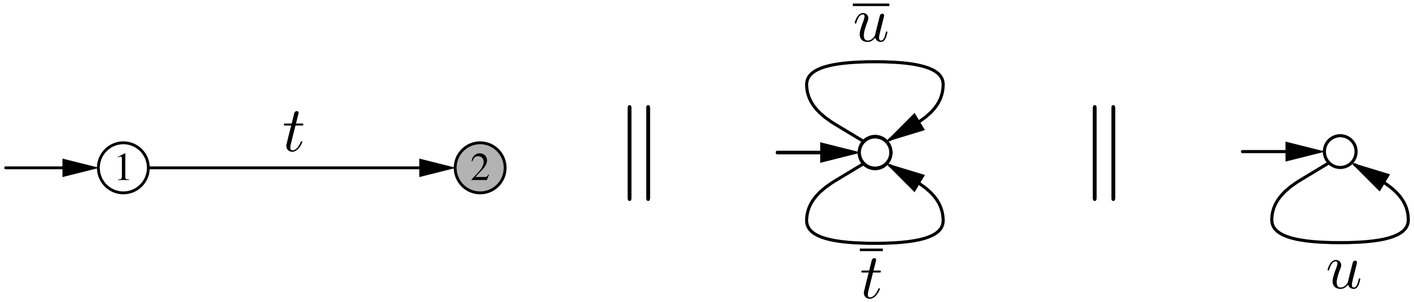


- the standard solution is to add a stronger completion criterion: weak/strong fairness

  weak fairness: If a task, from some point onwards, is perpetually enabled (meaning in each state) it will eventually be scheduled
  strong fairness: If a task is enabled infinitely often, but allowing interruptions during which it is not enabled, it will eventually be scheduled.
- both fairness assumptions guarantee the liveness property

Fairness is Too Strong

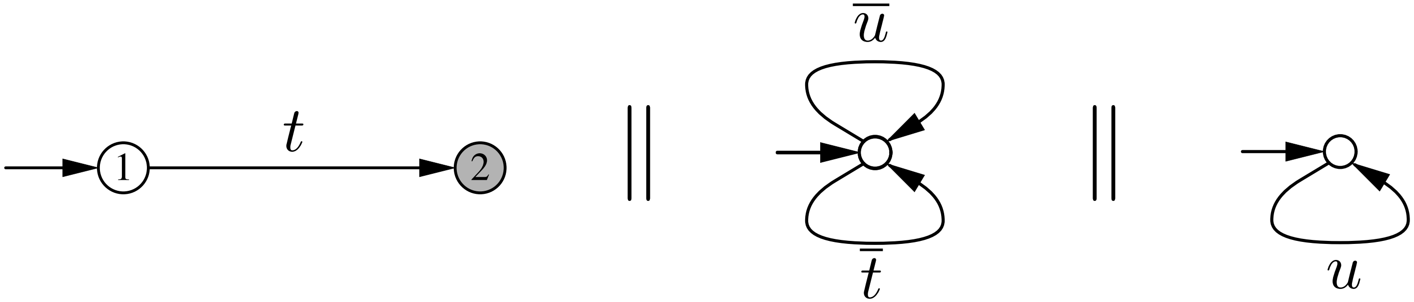
- Another example; adding synchronisation



- you go to a bar, are you guaranteed to get a drink
- weak/strong fairness says "yes",
  but what if the bartender does not like you

Fairness is Too Strong

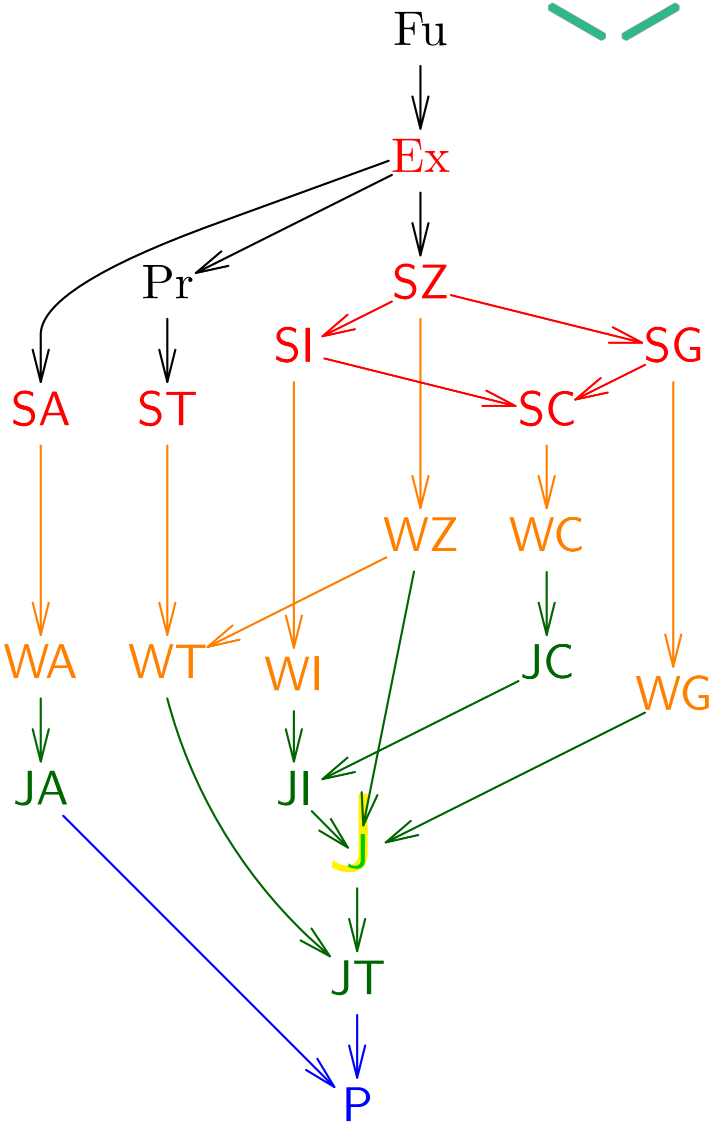
- let the central component be a mutual exclusion protocol



- adding fairness seems counter-intuitive

But there are other notions of fairness

- nearly all notions found in the literature are too strong
- by analysing this we built a taxonomy of fairness notions



Justness

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

- clearly, it is a completeness criterion as well
- it is not entirely compositional, but that is intended

- **Hypothesis/Conjecture:**
  justness can be assumed for all distributed systems
  (in contrast to fairness notions)

How to Formalise Justness

- although the idea is simple, its formalisation is not
  - what is a component
  - how to encode it in transition systems
  - which components partake in an action

- a co-inductive definition for CCS

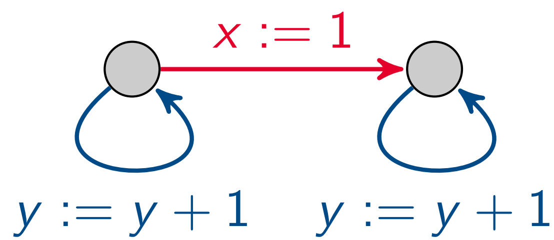
B-justness, for $B \subseteq Act$, is the largest family of predicates on the paths in the LTS of CCS such that

- a finite B-just path ends in a state that admits actions from B only
- a B-just path of a process $P|Q$ can be decomposed into a C-just path of P and a D-just path of Q, for some $C, D \subseteq B$ such that $\tau \in B \vee C \cap \bar{D} = \emptyset$
- a B-just path of $P \backslash L$ can be decomposed into a $B \cup L \cup \bar{L}$-just path of P
- a B-just path of $P[f]$ can be decomposed into an $f^{-1}(B)$-just path of P
- and each suffix of a B-just path is B-just.

How to formalise Justness

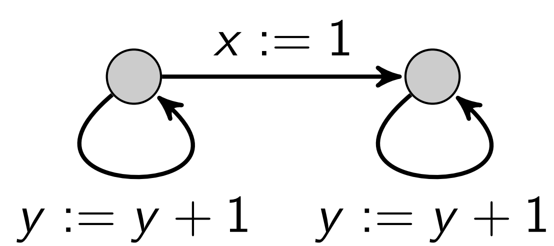
- Component Labelled Transition Systems (CLTS)

$$x := 1 \parallel \textbf{loop } y := y{+}1 \textbf{ forever}$$



- justness allows to distinguish this from

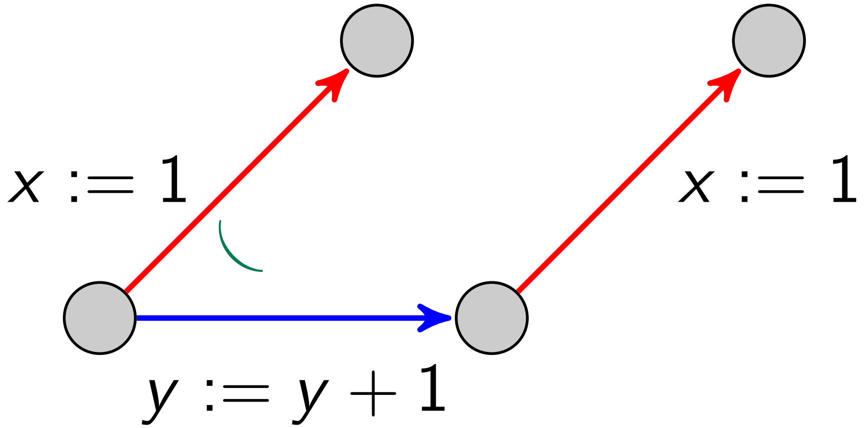
loop
    choose
        if $\texttt{True}$ **then** $y := y + 1$ **fi**
        if $x = 0$ **then** $x := 1$ **fi**
    end
forever



Component Labelled Transition System

A *component-labelled transition system* (CLTS) is a tuple $(S, \mathrm{Tr}, \mathrm{src}, \mathrm{trgt}, \ell, B, comp)$ with S and Tr sets (of *states* and *transitions*), $\mathrm{src}, \mathrm{trgt} : \mathrm{Tr} \to S$, $\ell : \mathrm{Tr} \to Act$ for a set of actions Act, $B \subseteq Act$ a set of *blocking* actions, and $comp : \mathrm{Tr} \to \mathscr{P}(\mathscr{C}) \setminus \emptyset$ for some set of components $\mathscr{C}$, such that:

if $t, v \in \mathrm{Tr}$ with $\mathrm{src}(t) = \mathrm{src}(v)$ and $comp(t) \cap comp(v) = \emptyset$, then there is a $u \in \mathrm{Tr}$ with $\mathrm{src}(u) = \mathrm{trgt}(v)$, $\ell(u) = \ell(t)$ and $comp(u) = comp(t)$.



Justness on CLTSs

Two transitions $t, u \in \mathrm{Tr}$ are *concurrent*, notation $t \smile u$, if

$$comp(t) \cap comp(u) = \emptyset$$

A path π in an CLTS is *just*
if for each transition $t \in \mathrm{Tr}_{\neg B}$ with $s := \mathrm{src}(t) \in \pi$,
a transition u occurs in π past the occurrence of s,
such that $t \not\smile u$.

Results

- both notions of justness coincide (for CCS)
- one colouring is not sufficient as there may be affected and necessary components
  - process algebra with signals / Petri Nets with Read Arcs
  - mechanisms with broadcast mechanisms
  - …
- CLTSs can be “expanded” to two colourings

Examples

- process algebras with signals / Petri Nets with read arcs
  - assume a traffic light,
    - as the light does not change state when a car crosses
    - the traffic light should not be “blocked” while a second car crosses
  - reading values concurrently

  - affected components (car) cannot act without necessary ones
- mechanisms with broadcast
  - broadcast sender is both affected and necessary
  - recipients are

- in general affected and necessary components are independent and can be used to define the concurrency relation which becomes asymmetric

The Good, the Bad, the Ugly

- **The Good**
  - justness seems to be the fundamental “fairness” property that can/should be assumed for any distributed system
  - it probably can be defined for any formalism of concurrency

- **The Bad**
  - although its characterisation is fairly simple, its formal definition is not
  - or at least not yet

- **The ~~Ugly~~ Exciting**
  - new proof theory needs to be developed

Thank you

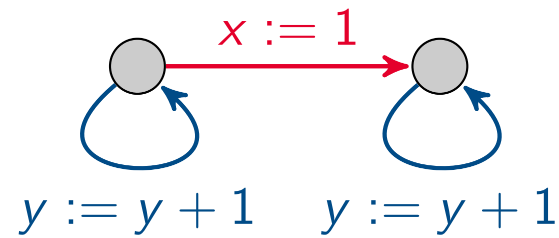
Data61
Peter Höfner

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

Updating Bisimulation

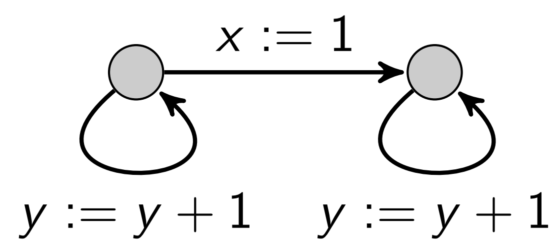
- Component Labelled Transition Systems with Concurrency (CLTS)

$$x := 1 \parallel \textbf{loop } y := y{+}1 \textbf{ forever}$$



- justness allows to distinguish this from

loop
    choose
        if $\texttt{True}$ **then** $y := y + 1$ **fi**
        if $x = 0$ **then** $x := 1$ **fi**
    end
forever



- however, both systems are bisimilar
  a new theory needs to be developed

Bisimulation using Components

Can we build an equivalence $\approx_{cp}$ that meets our needs?