Ensuring Liveness Properties of Distributed Systems with Justness

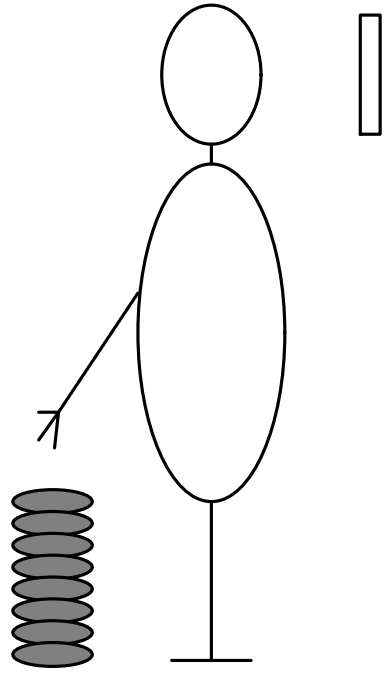
Rob van Glabbeek

University of Edinburgh

June 2023

Liveness properties – an example

Something good will eventually happen.

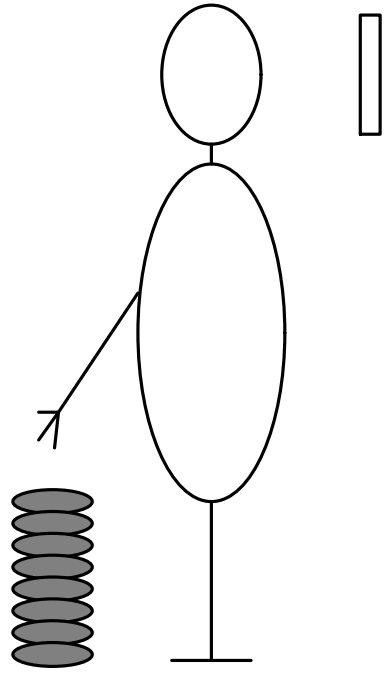


Task: insert an infinite pile of quarters in slot

Liveness property: at least 3 quarters will be inserted.

Liveness properties – an example

Something good will eventually happen.



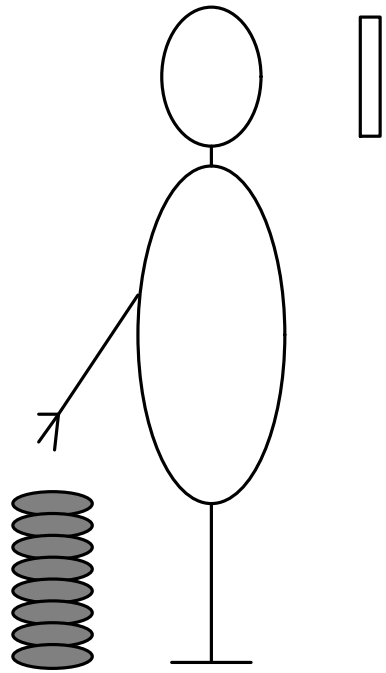
Task: insert an infinite pile of quarters in slot

Liveness property: at least 3 quarters will be inserted.

Intuitively, this property holds, when assuming *progress*.

Liveness properties – an example

Something good will eventually happen.



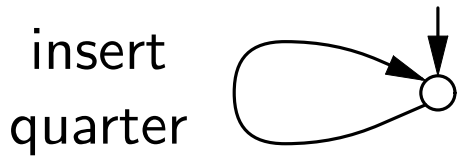
Task: insert an infinite pile of quarters in slot

Liveness property: at least 3 quarters will be inserted.

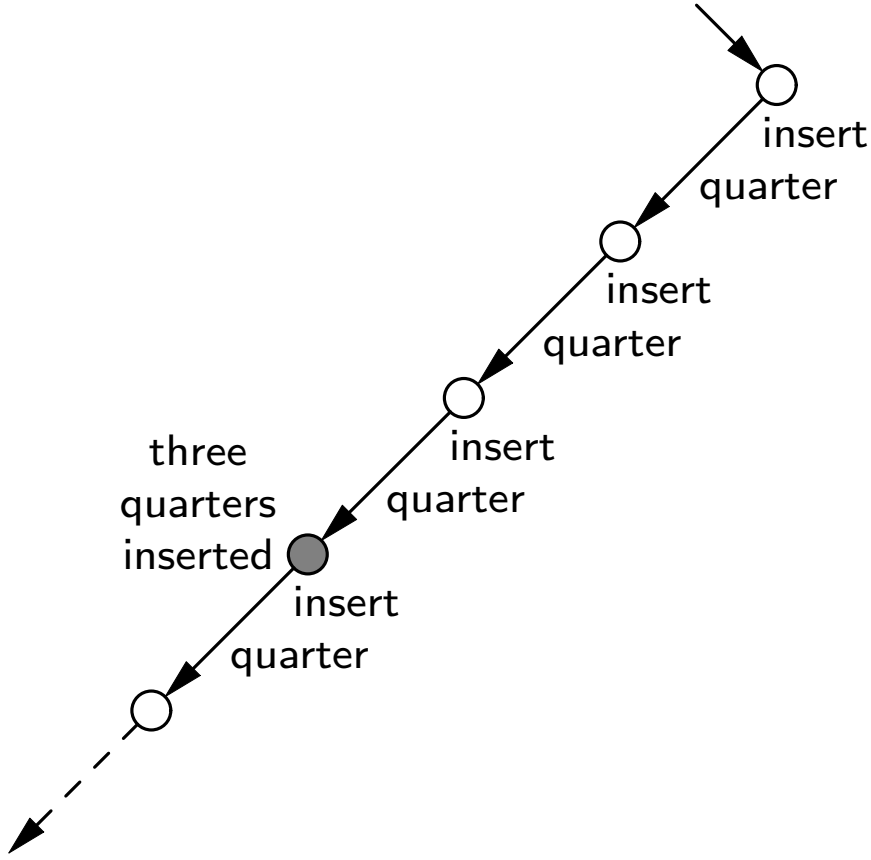
Intuitively, this property holds, when assuming *progress*.

The assumption that a system will not stop without a reason.

Transition system of example



Transition system with success state



Progress, Justness, Fairness and Liveness

Fairness

$\Downarrow$

Justness

$\Downarrow$

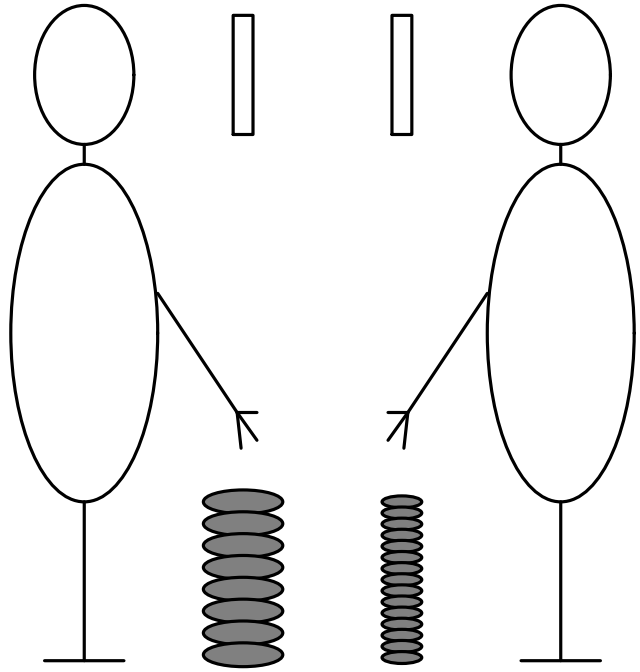
Progress

a hierarchy of assumptions

Liveness properties

somethings one want to obtain,
optionally
when making one such assumption

Liveness properties – a more interesting example

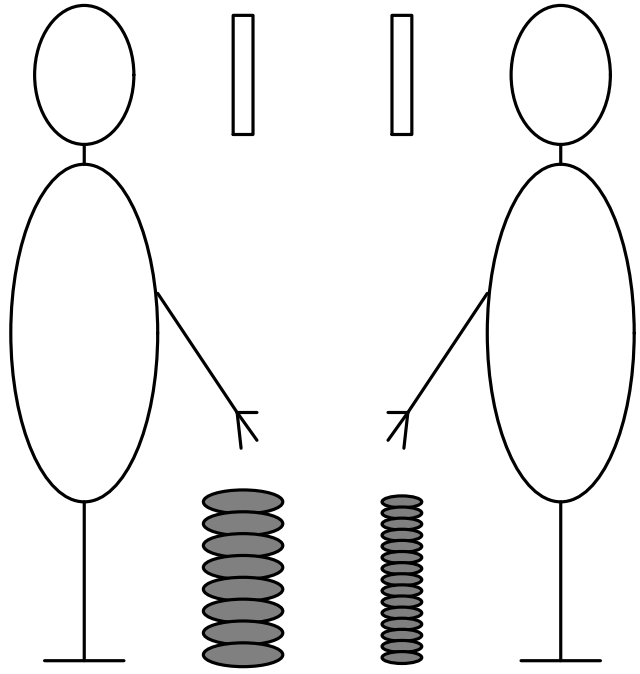


Tasks: insert an infinite pile of quarters in left slot

insert an infinite pile of dimes in right slot

Liveness property: at least 3 quarters will be inserted.

Liveness properties – a more interesting example

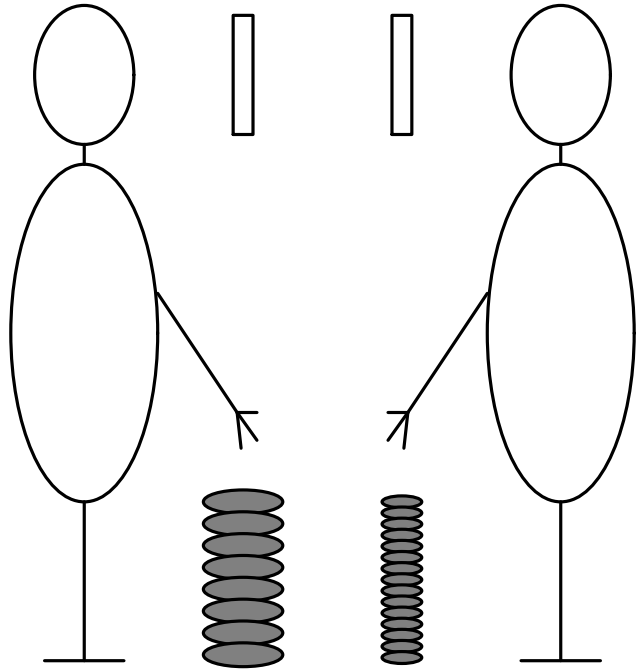


Tasks: insert an infinite pile of quarters in left slot — insert an infinite pile of dimes in right slot

Liveness property: at least 3 quarters will be inserted.

Intuitively, this property holds, when assuming *justness*.

Liveness properties – a more interesting example



Tasks: insert an infinite pile of quarters in left slot — insert an infinite pile of dimes in right slot

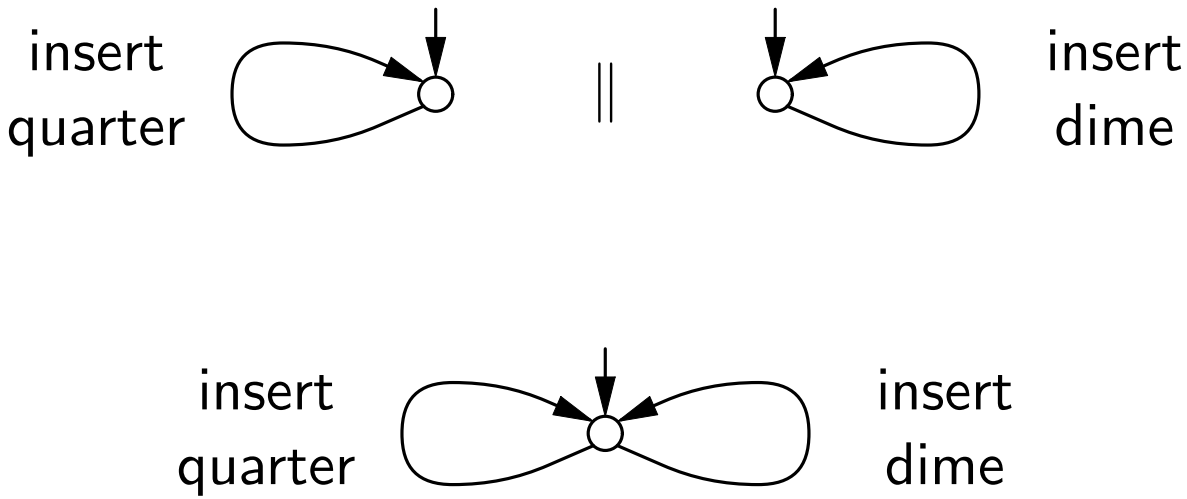
Liveness property: at least 3 quarters will be inserted.

Intuitively, this property holds, when assuming *justness*.

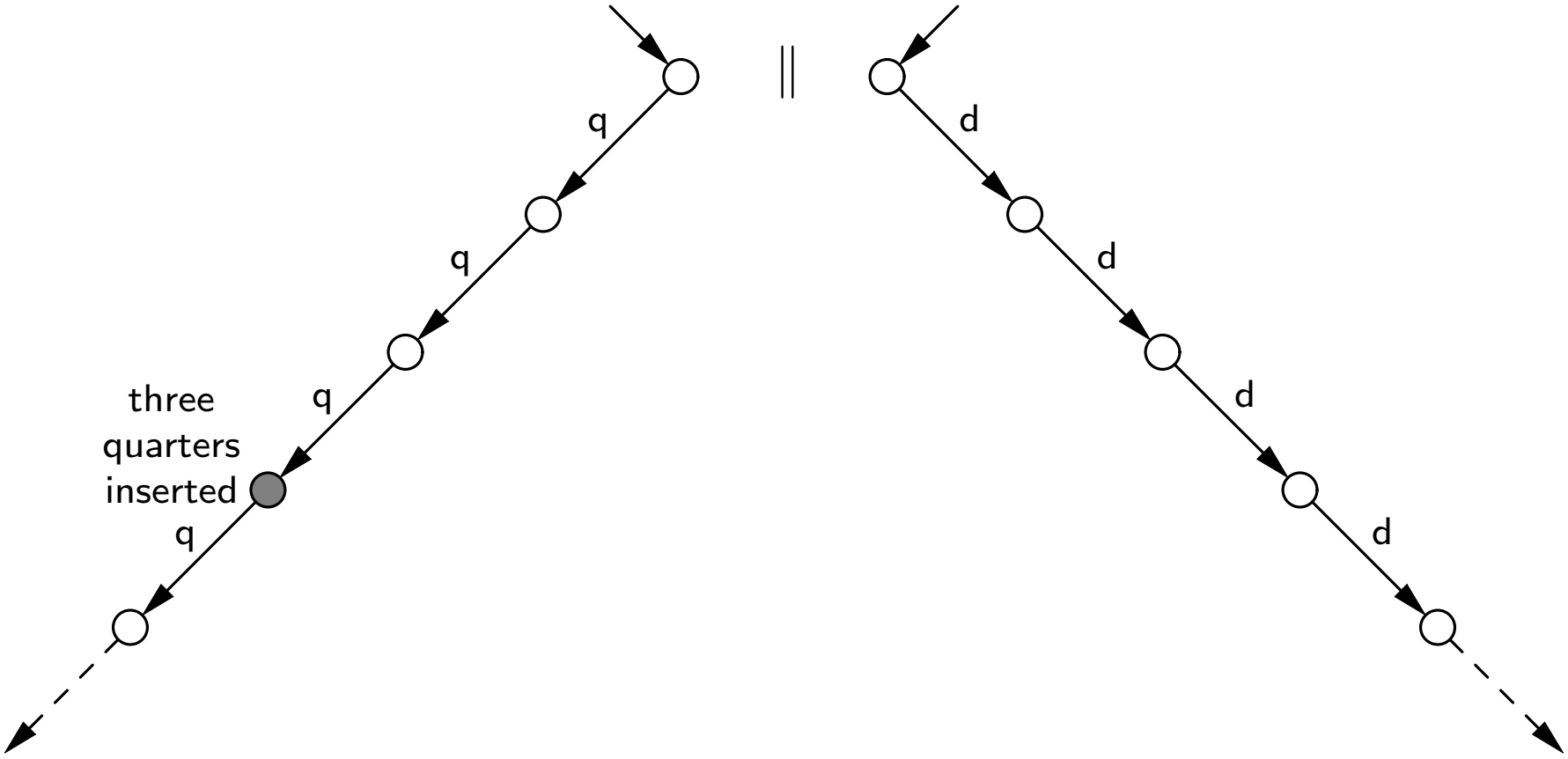
Even a subsystem will not stop without a reason.

Transition system of example

Transition system of example

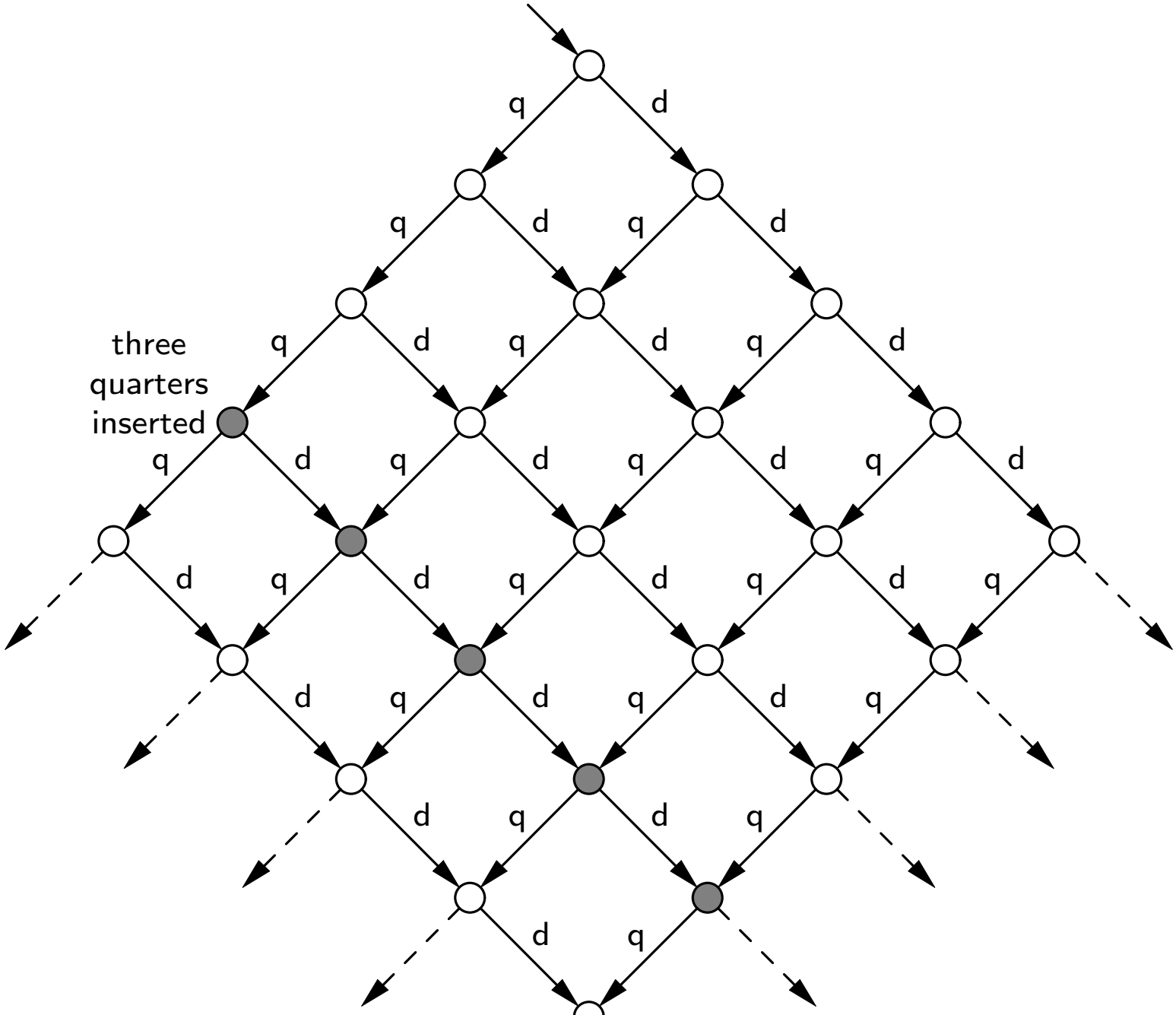


Transition system with success states

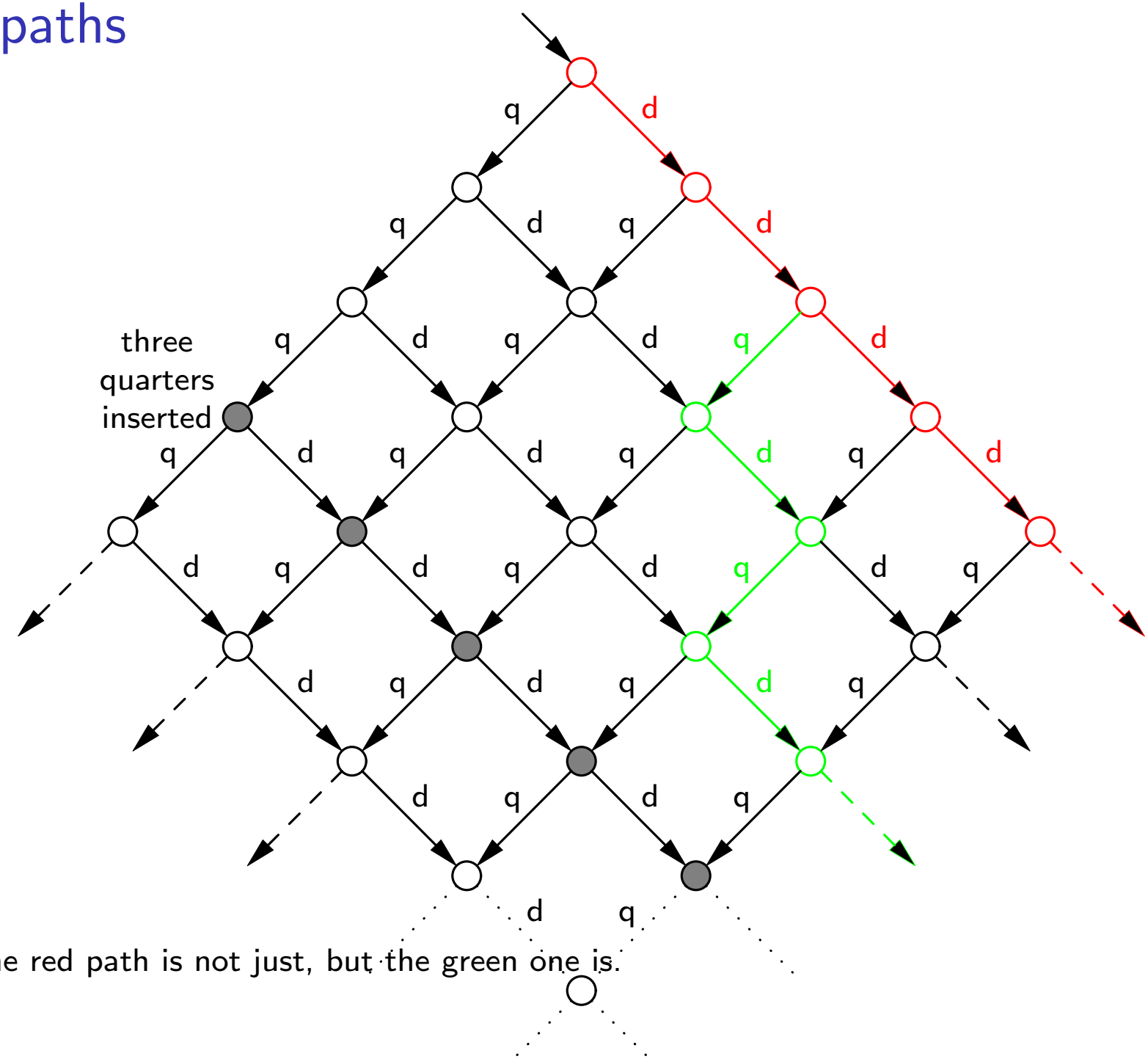


Transition system with success states

$=$



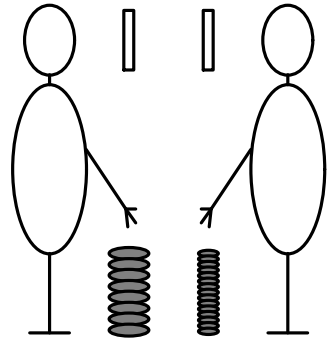
Just paths



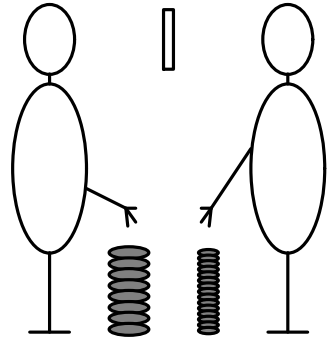
The red path is not just, but the green one is.

Concurrency versus competition

Concurrency:



Competition:

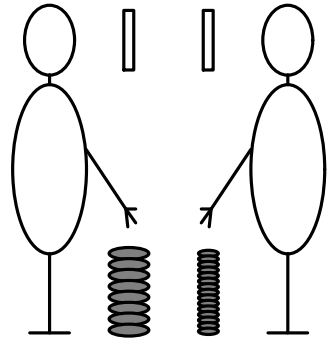


Liveness property: at least 3 quarters will be inserted.

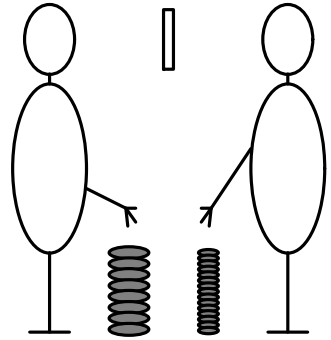
When assuming *justness*
this property holds for the *concurrency* example,
but not for the *competition* example.

Concurrency versus competition

Concurrency:



Competition:

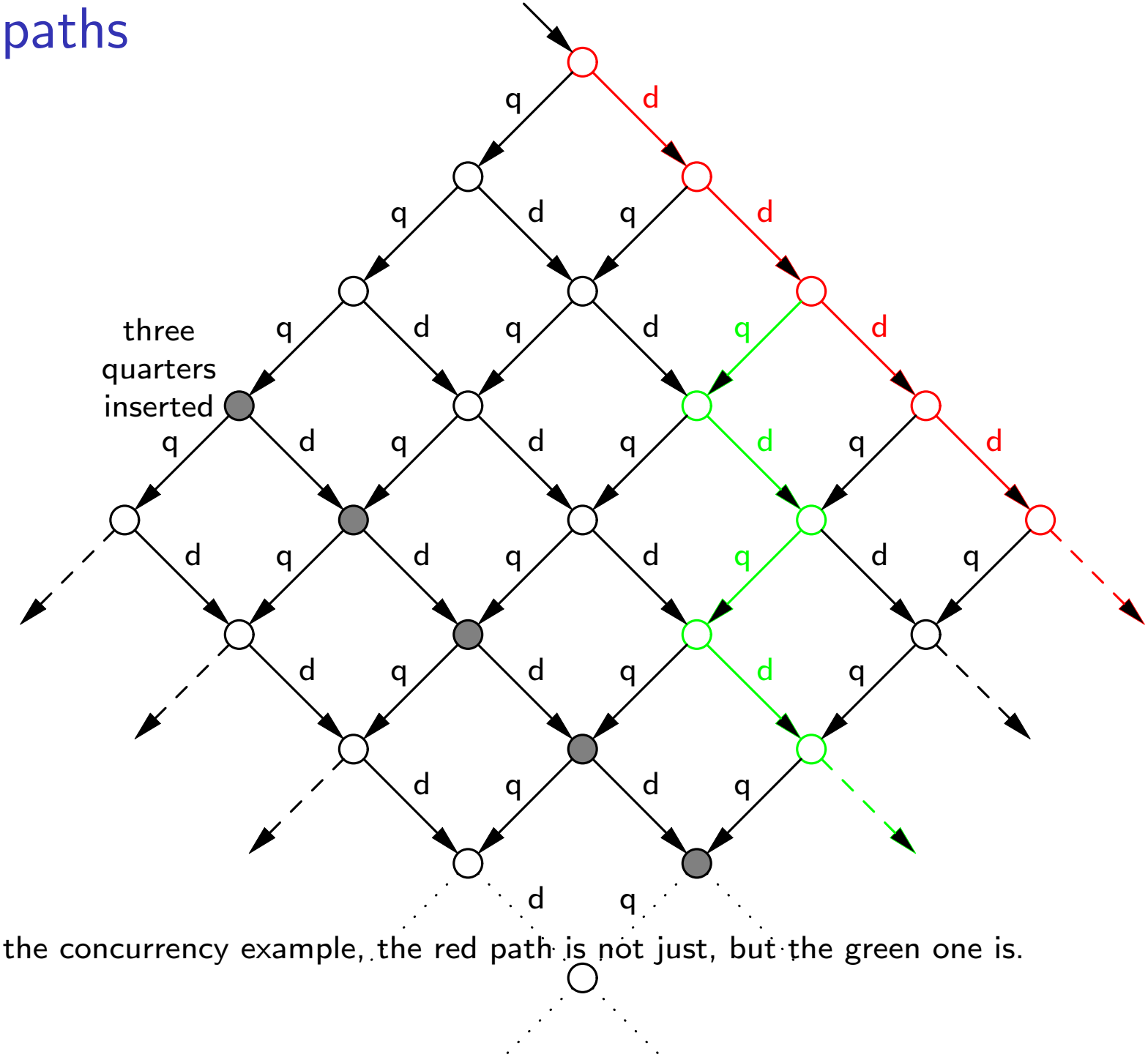


Liveness property: at least 3 quarters will be inserted.

When assuming *justness*
this property holds for the *concurrency* example,
but not for the *competition* example.

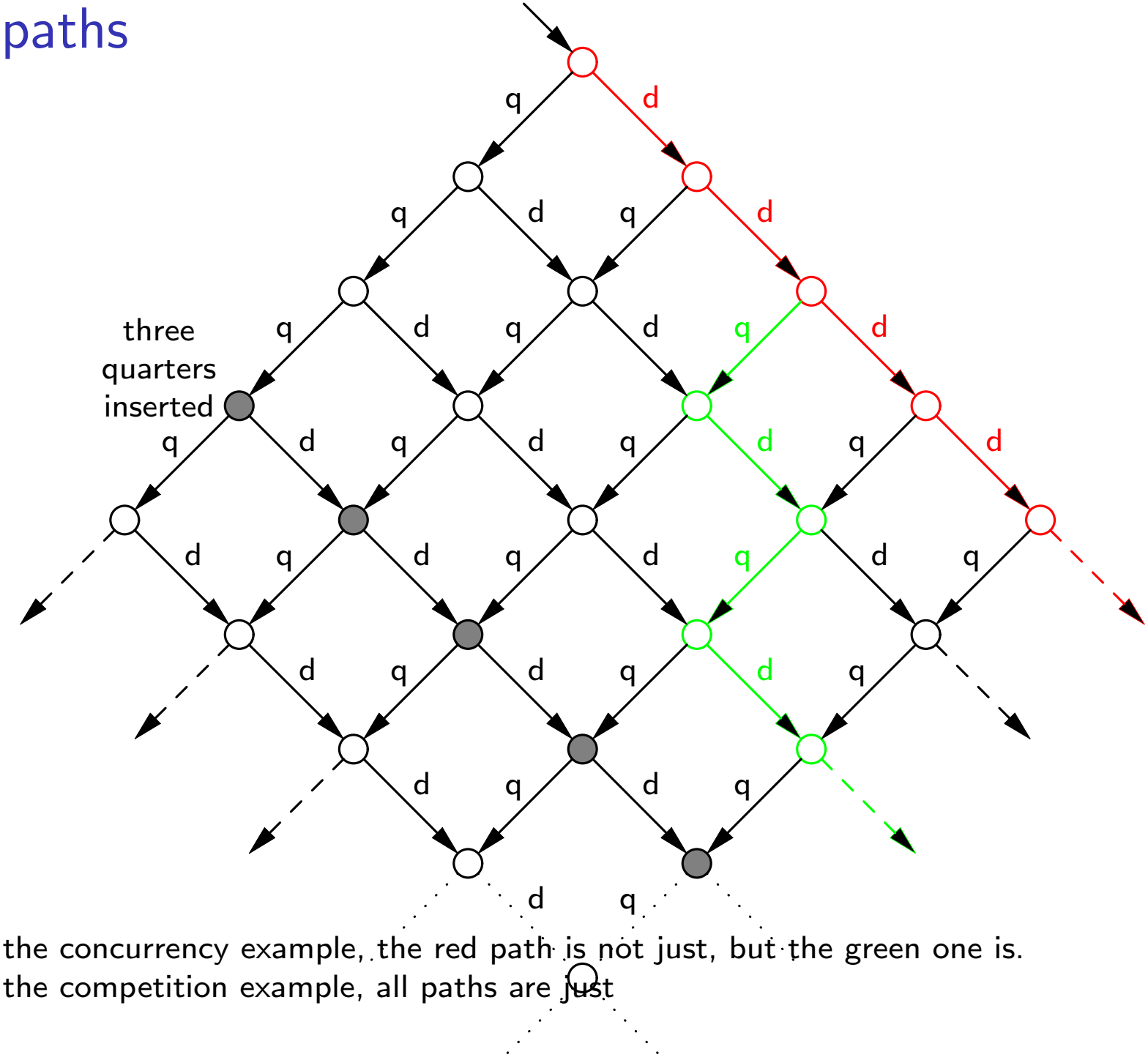
When assuming *fairness* it holds for both examples.

Just paths



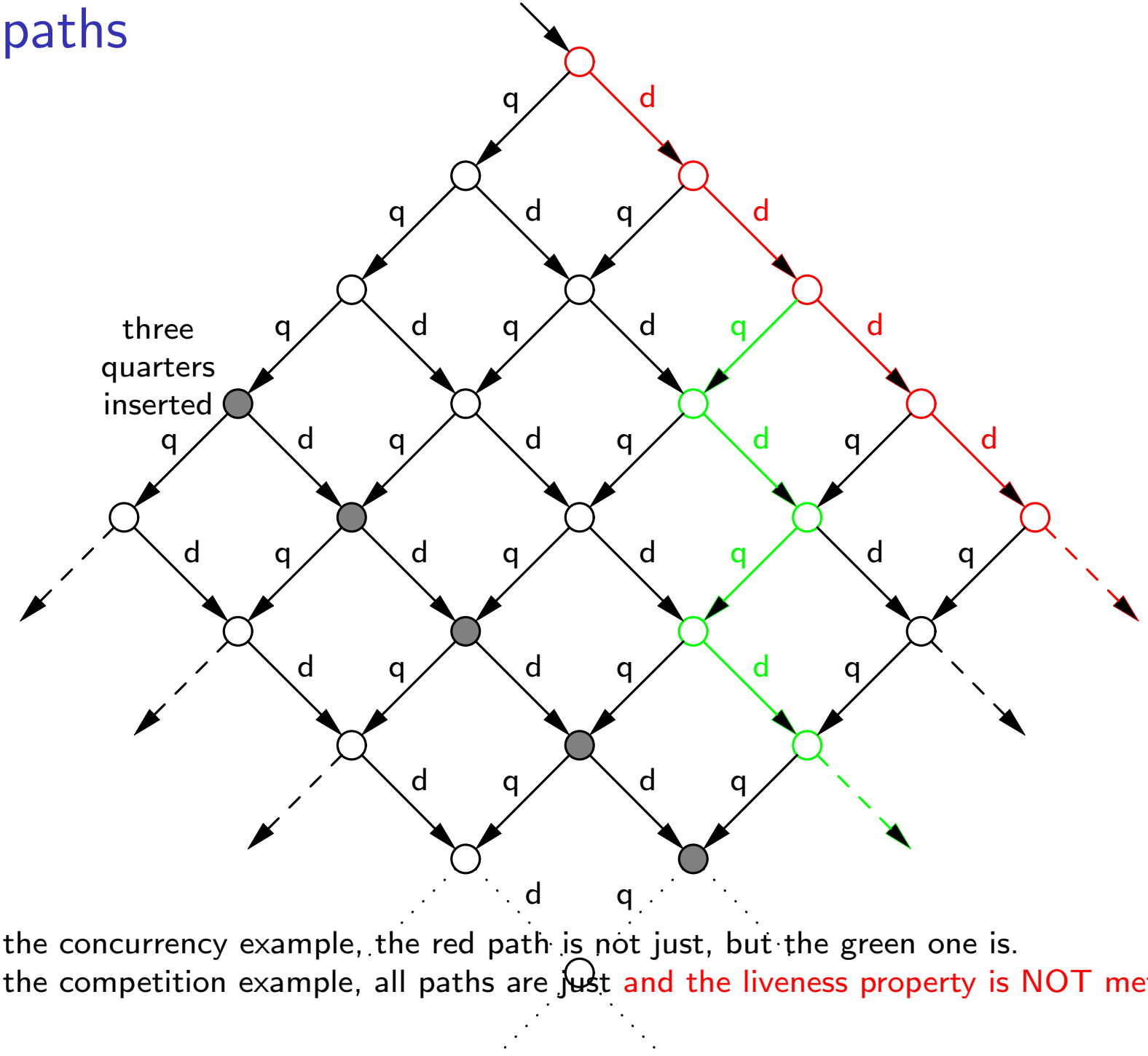
In the concurrency example, the red path is not just, but the green one is.

Just paths



In the concurrency example, the red path is not just, but the green one is.
In the competition example, all paths are just

Just paths



In the concurrency example, the red path is not just, but the green one is.
In the competition example, all paths are just and the liveness property is NOT met.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

When implementing such a protocol, it is *not* reasonable to assume fairness. It is typically not justified.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

When implementing such a protocol, it is *not* reasonable to assume fairness. It is typically not justified.

But assuming justness (no component stops without reason) usually is.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

When implementing such a protocol, it is *not* reasonable to assume fairness. It is typically not justified.

But assuming justness (no component stops without reason) usually is.

Much contemporary research fails to distinguish justness and fairness.
This can lead to unwarranted conclusions and system failures.

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Problem: in standard models of concurrency my concurrency and competition examples have the very same representation!
(e.g. in *labelled transition systems*).

They are *semantically equivalent* (e.g. when using *bisimulation*).

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Problem: in standard models of concurrency my concurrency and competition examples have the very same representation!
(e.g. in *labelled transition systems*).

They are *semantically equivalent* (e.g. when using *bisimulation*).

So we need different models of concurrency, in which these systems have a different representations.
We also need different semantic equivalences.

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Problem: in standard models of concurrency my concurrency and competition examples have the very same representation!
(e.g. in *labelled transition systems*).

They are *semantically equivalent* (e.g. when using *bisimulation*).

So we need different models of concurrency, in which these systems have a different representations.
We also need different semantic equivalences.

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)
- Induction principles $\leftarrow$ requires new ideas
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)
- Induction principles $\leftarrow$ requires new ideas
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.

- Induction principles ← requires new ideas

- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

- Induction principles ← requires new ideas

- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

  Next to abstraction from internal activity, compositionality is the most powerful tool to attack the state-space explosion.

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

  Next to abstraction from internal activity, compositionality is the most powerful tool to attack the state-space explosion.
  A complex system is verified, by verifying its parts, and composing the verified parts in a black-box manner.

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

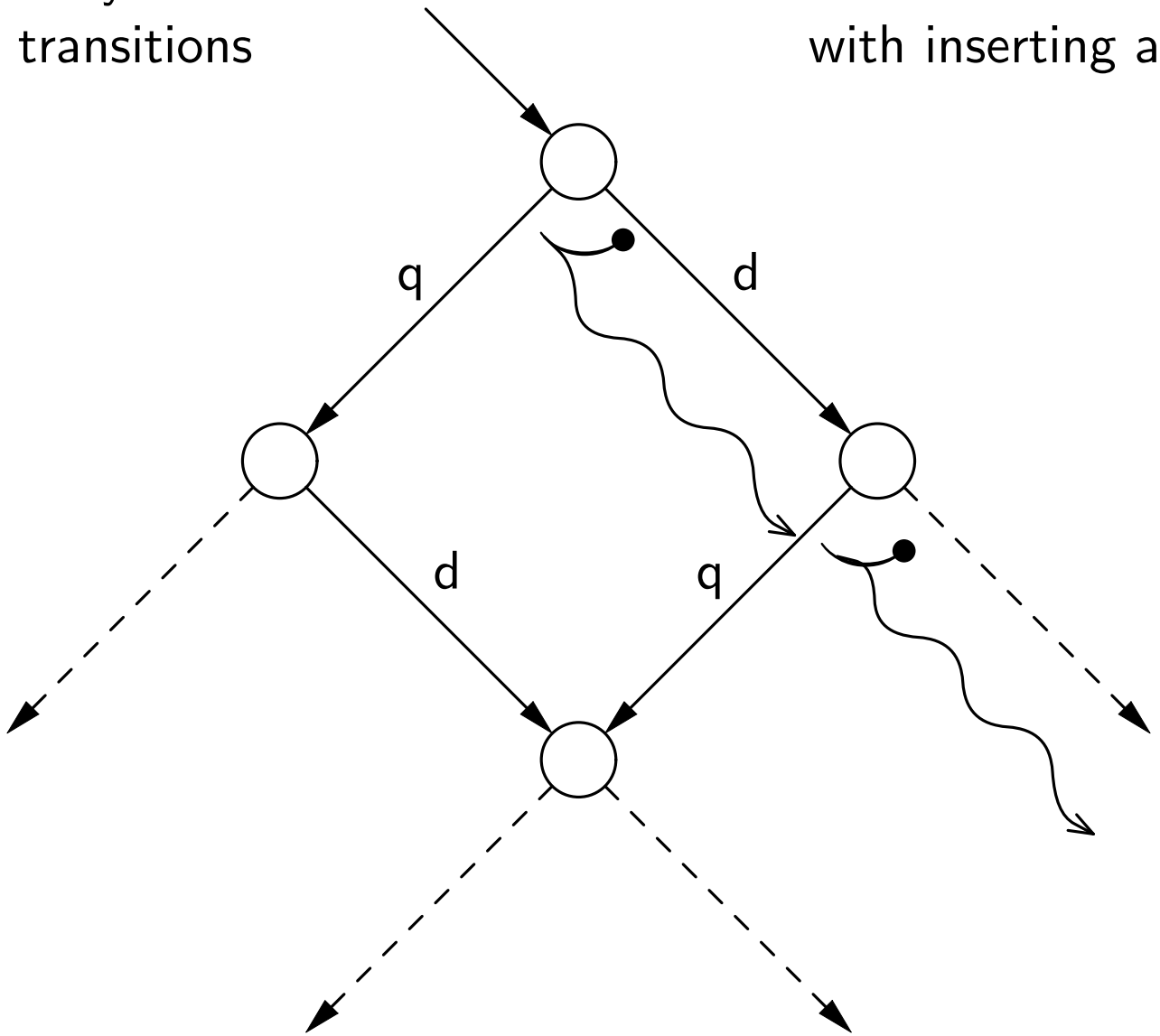
  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

  Next to abstraction from internal activity, compositionality is the most powerful tool to attack the state-space explosion.
  A complex system is verified, by verifying its parts, and composing the verified parts in a black-box manner.
  Syntactic checks on code are known that guarantee that forms of compositional reasoning are warranted. But such work needs to be redone when factoring in justness.

Transition systems with successors

Transition systems
plus a ternary relation
between transitions

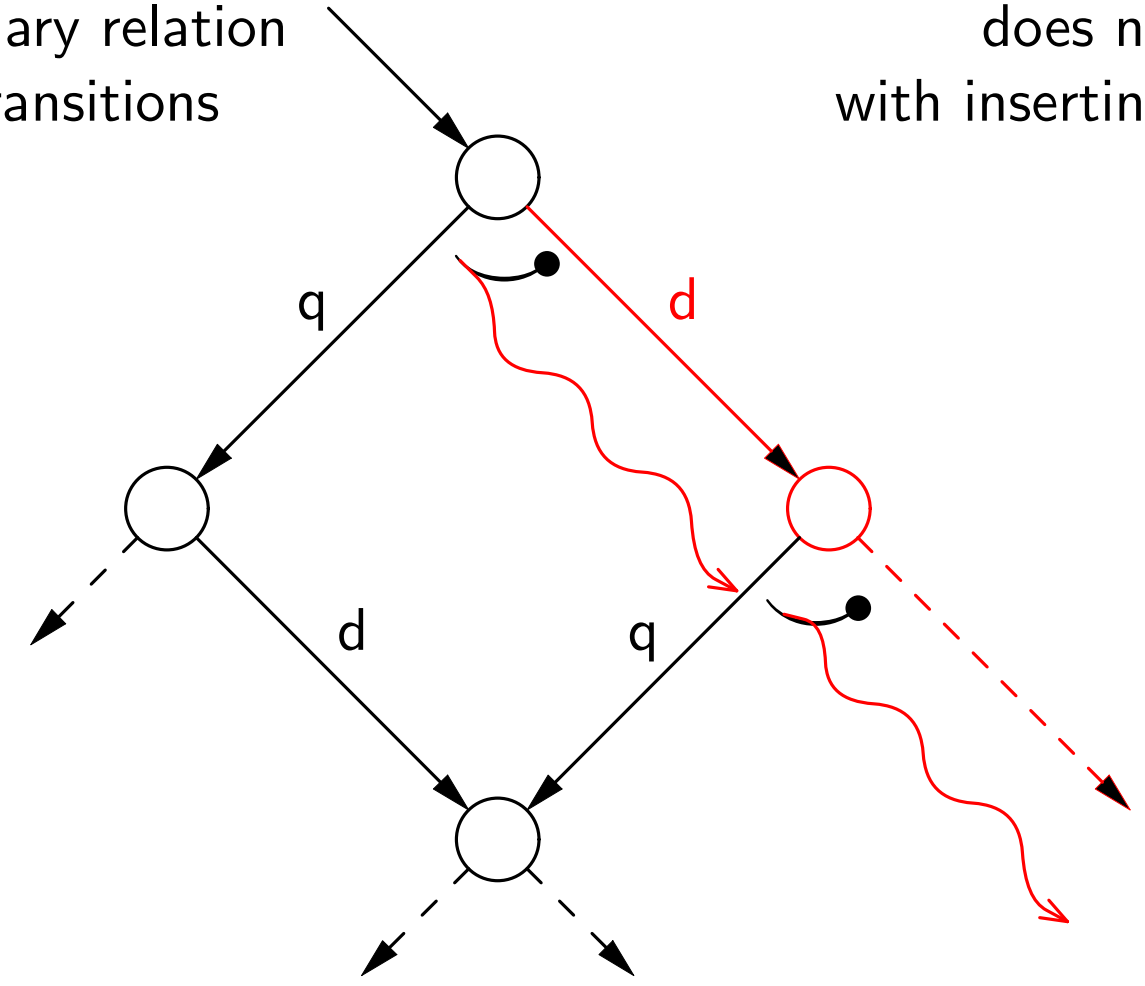
inserting a dime
does not interfere
with inserting a quarter



Formalising Justness

Transition systems
plus a ternary relation
between transitions

inserting a dime
does not interfere
with inserting a quarter



Justness: The system never follows a $\rightarrow$-path
that induces an infinite $\rightsquigarrow$-sequence.

Petri nets

In a Petri net, a run is *just* if for each transition that becomes enabled,
either it occurs in the run further on,
or a token necessary to fire that transition is used by another transition.

Petri nets

In a Petri net, a run is *just* if for each transition that becomes enabled,
either it occurs in the run further on,
or a token necessary to fire that transition is used by another transition.

This is a default assumption in Petri nets
[Reisig 2013: Understanding Petri nets],
not in need of any name like “justness”.

Petri nets

In a Petri net, a run is *just* if for each transition that becomes enabled,
either it occurs in the run further on,
or a token necessary to fire that transition is used by another transition.

This is a default assumption in Petri nets
[Reisig 2013: Understanding Petri nets],
not in need of any name like "justness".

Write $t \smile\!\!\bullet\ u$ if transition t does not need any resource that is consumed by u.

Petri nets

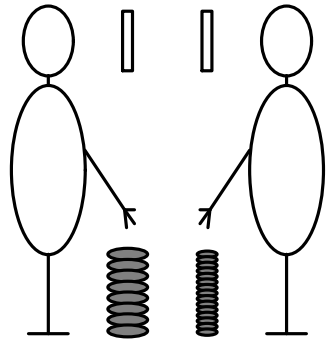
In a Petri net, a run is *just* if for each transition that becomes enabled,
either it occurs in the run further on,
or a token necessary to fire that transition is used by another transition.

This is a default assumption in Petri nets
[Reisig 2013: Understanding Petri nets],
not in need of any name like "justness".

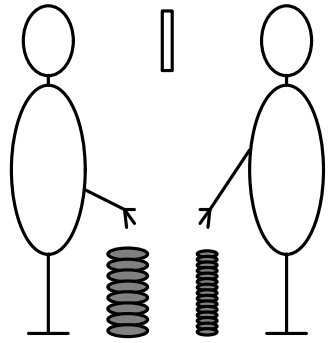
Write $t \smile\!\!\bullet\, u$ if transition t does not need any resource that is consumed by u. Write $t \smile u$ if $t \smile\!\!\bullet\, u$ and $u \smile\!\!\bullet\, t$.

Concurrency versus competition

Concurrency:



Competition:



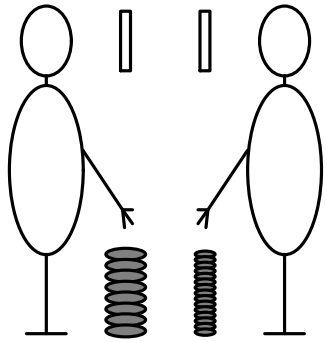
Liveness property: at least 3 quarters will be inserted.

When assuming *justness*
this property holds for the *concurrency* example,
but not for the *competition* example.

When assuming *fairness* it holds for both examples.
When assuming *progress* it holds for neither.

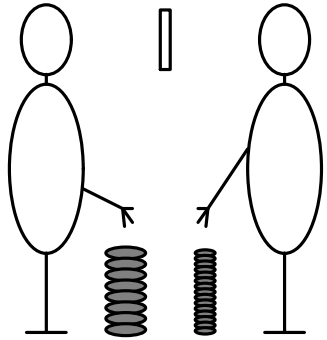
Concurrency versus competition

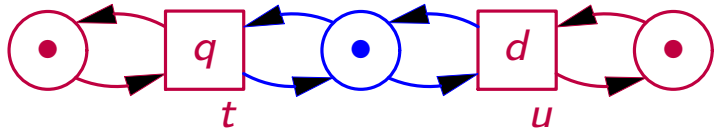
Concurrency:



$t \smile u$

Competition:





$t \not\smile u$

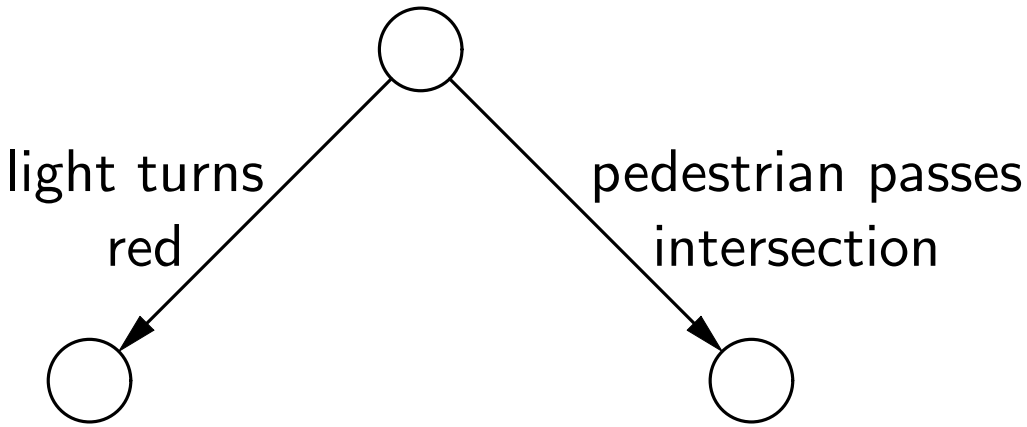
Liveness property: at least 3 quarters will be inserted.

When assuming *justness*
this property holds for the *concurrency* example,
but not for the *competition* example.

When assuming *fairness* it holds for both examples.
When assuming *progress* it holds for neither.

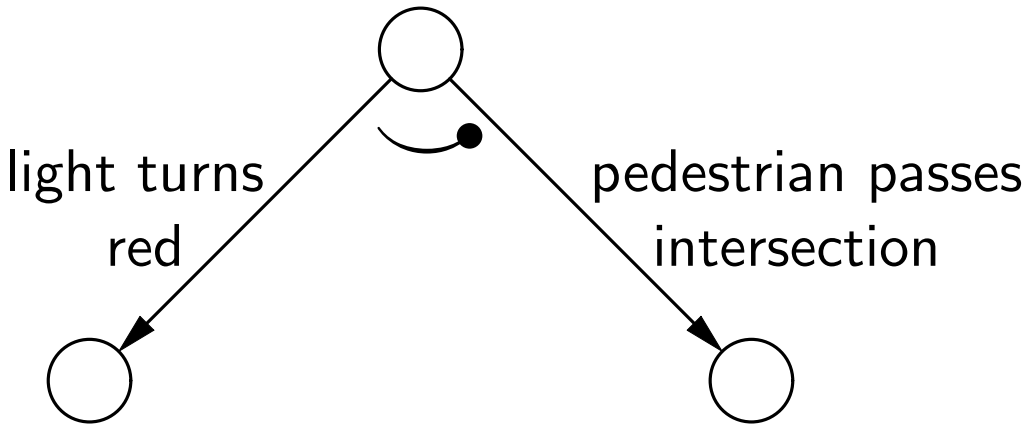
Example of an asymmetric concurrency relation

obedient pedestrian approaching a green traffic light



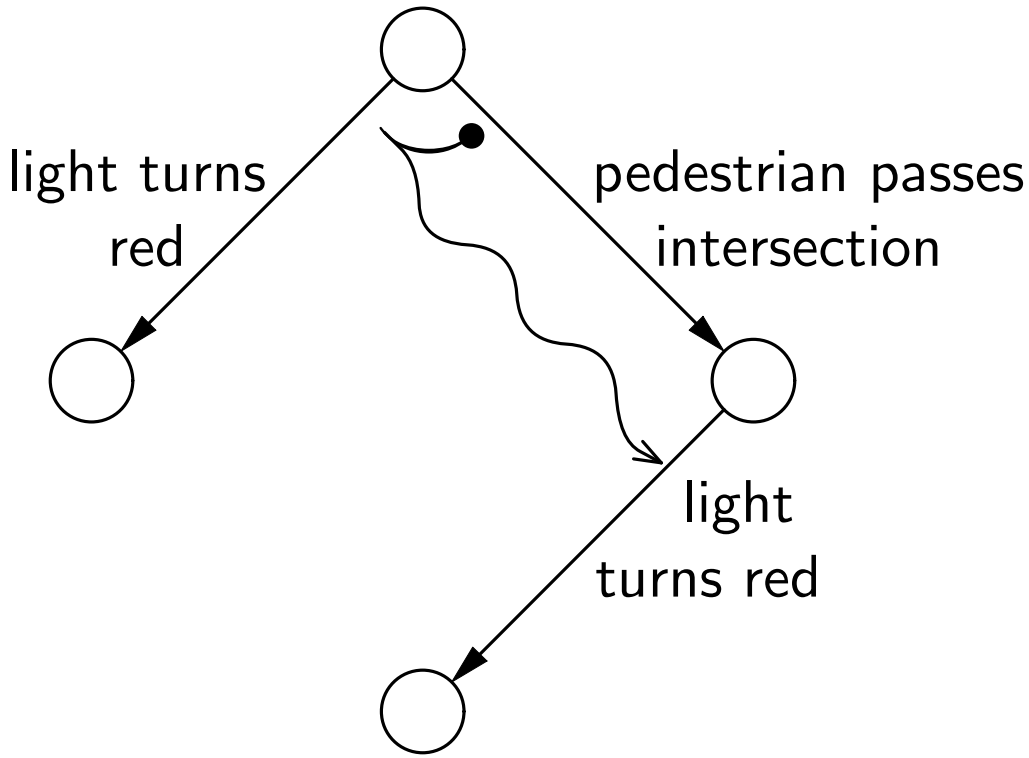
Example of an asymmetric concurrency relation

obedient pedestrian approaching a green traffic light



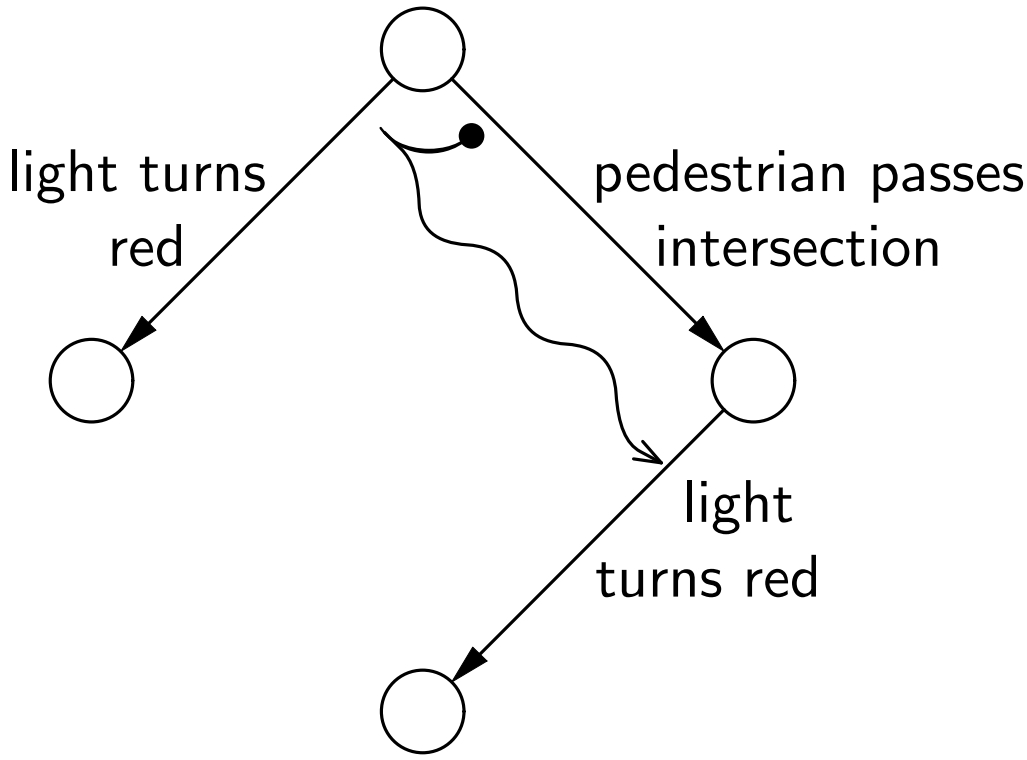
Example of an asymmetric concurrency relation

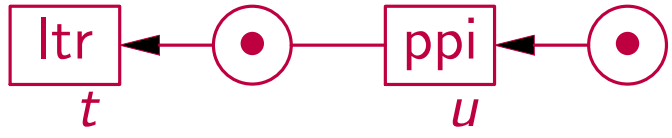
obedient pedestrian approaching a green traffic light



Example of an asymmetric concurrency relation

obedient pedestrian approaching a green traffic light





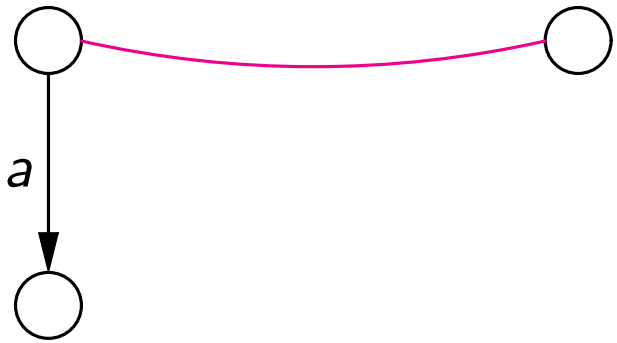
$t \smile u$

Bisimulation equivalence

To show that two systems have the same properties, one traditionally constructs a *bisimulation* between them. This is a relation $\mathcal{R}$ between their states, such that

- The initial states are related:

- The *transfer property* holds:

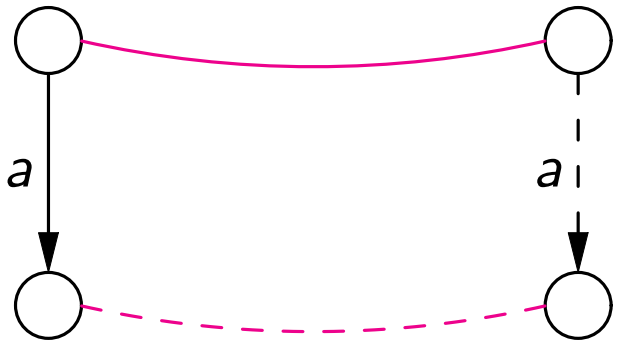


Bisimulation equivalence

To show that two systems have the same properties, one traditionally constructs a *bisimulation* between them. This is a relation $\mathcal{R}$ between their states, such that

- The initial states are related:

- The *transfer property* holds:

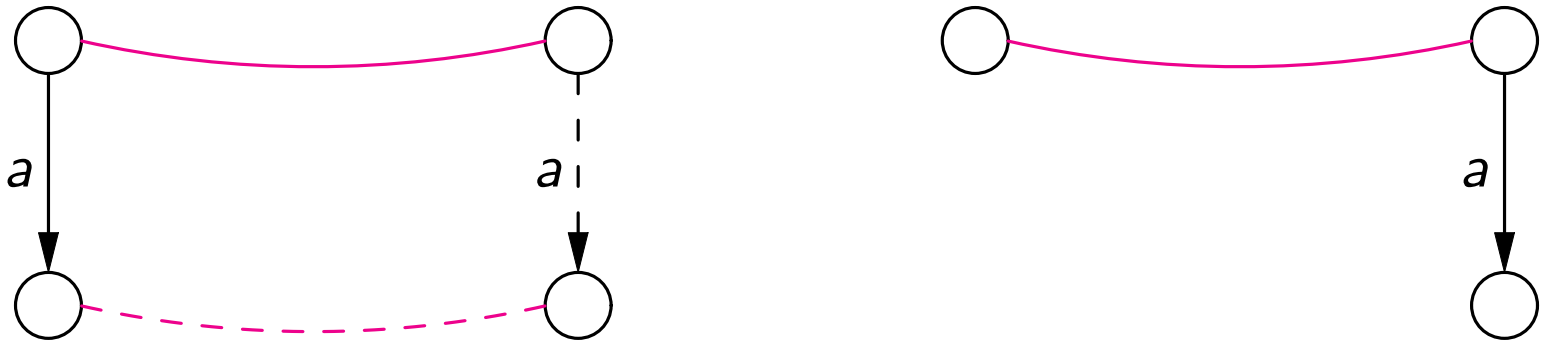


Bisimulation equivalence

To show that two systems have the same properties, one traditionally constructs a *bisimulation* between them. This is a relation $\mathcal{R}$ between their states, such that

- The initial states are related:

- The *transfer property* holds:

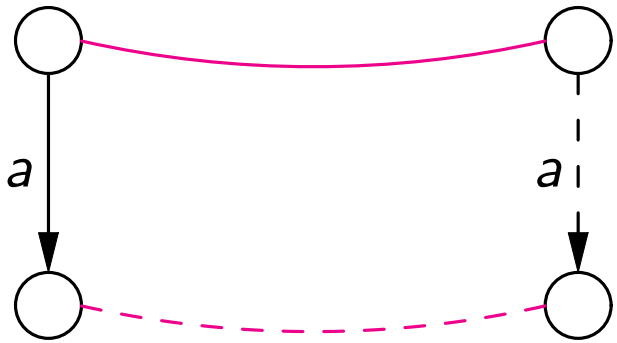


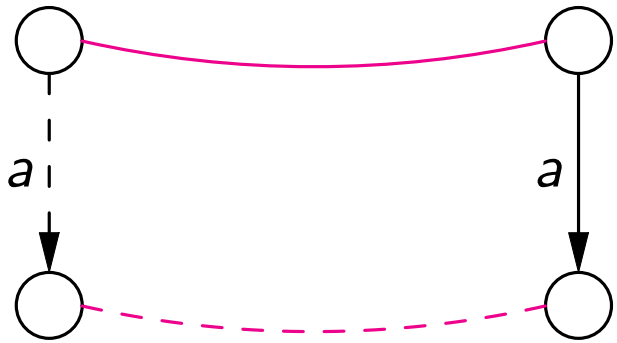
Bisimulation equivalence

To show that two systems have the same properties, one traditionally constructs a *bisimulation* between them. This is a relation $\mathcal{R}$ between their states, such that

- The initial states are related:

- The *transfer property* holds:



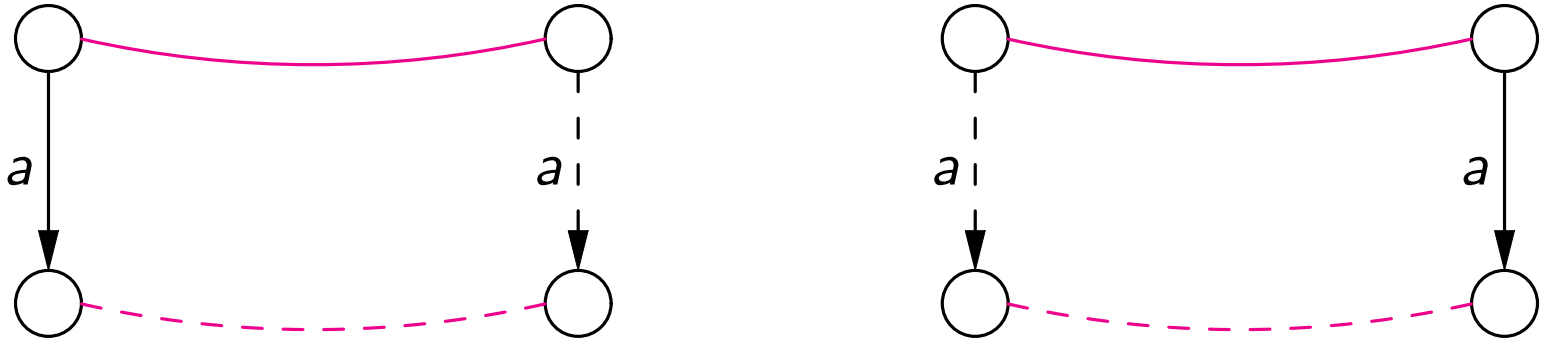


Bisimulation equivalence

To show that two systems have the same properties, one traditionally constructs a *bisimulation* between them. This is a relation $\mathcal{R}$ between their states, such that

- The initial states are related:

- The *transfer property* holds:

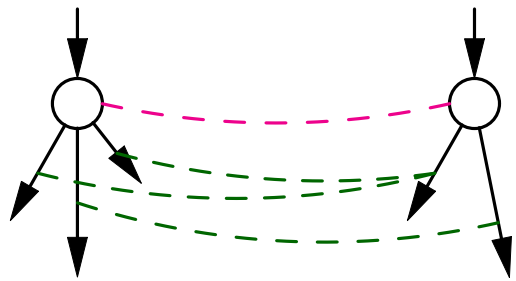


To preserve justness we need a form of bisimulation that also preserves $\rightsquigarrow$.

Enabling preserving bisimulation equivalence

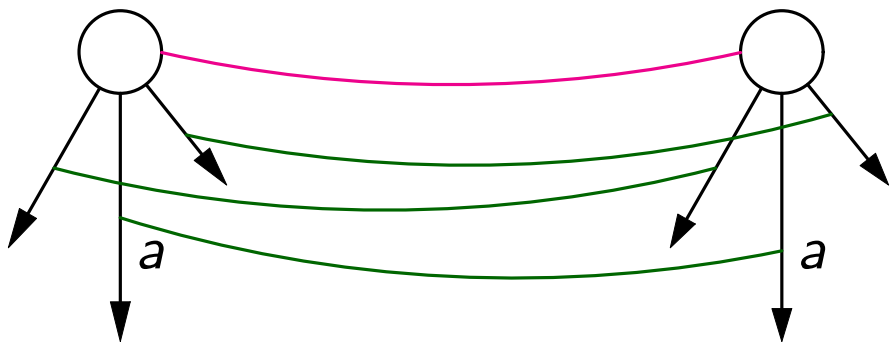
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

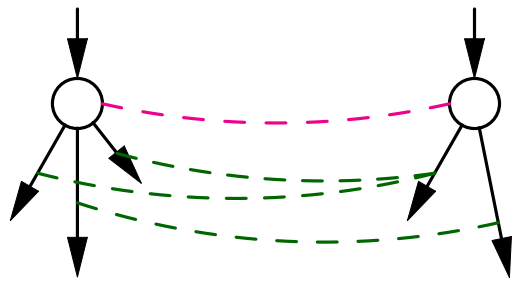
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

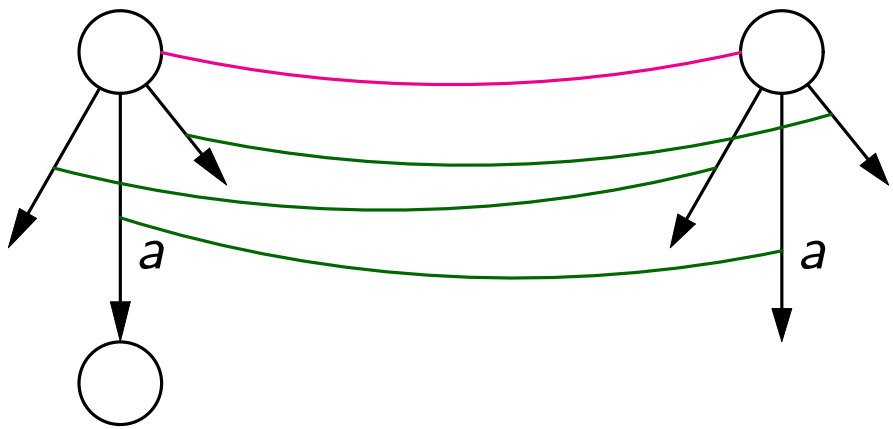
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

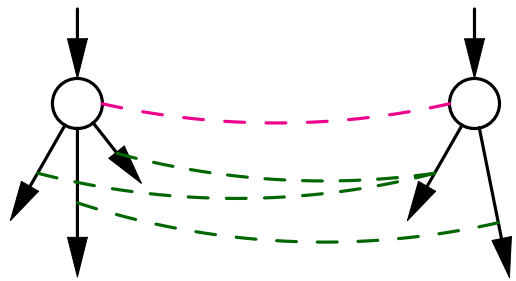
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

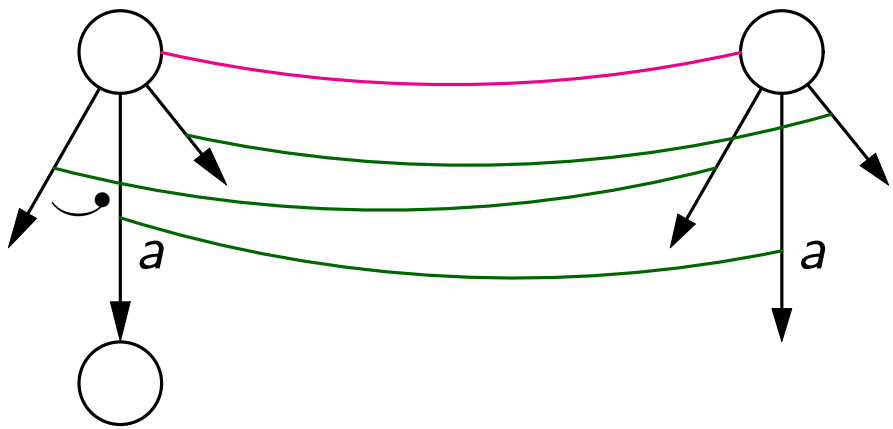
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

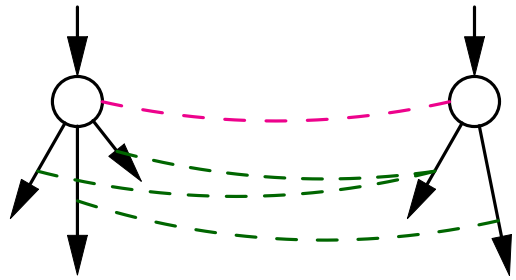
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

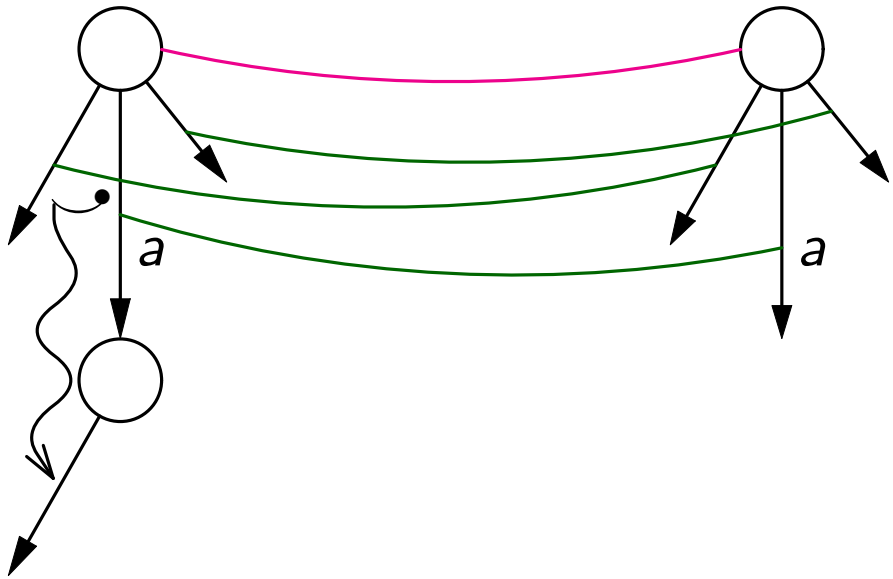
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

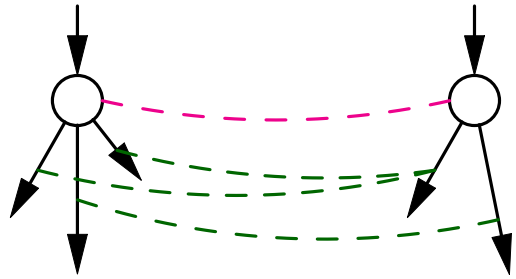
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

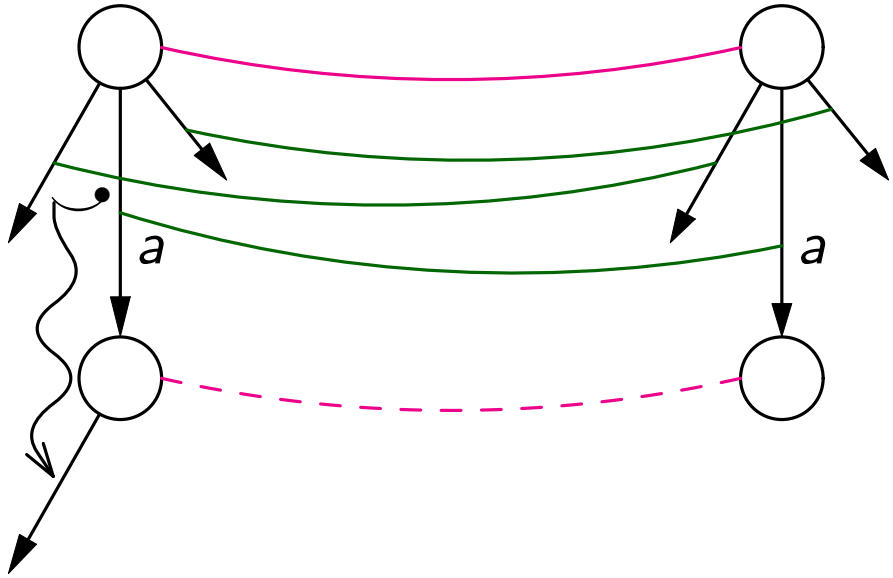
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

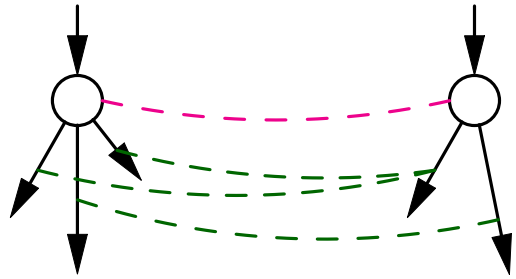
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

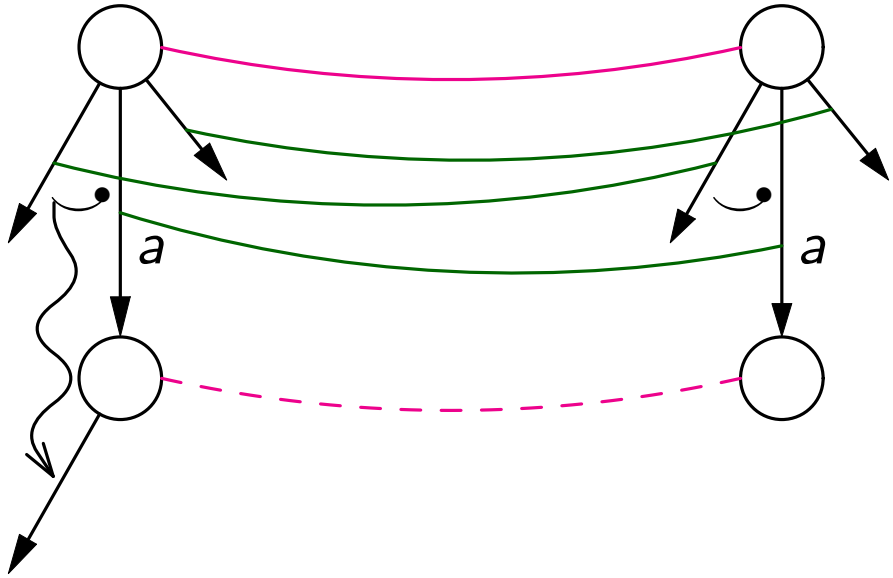
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

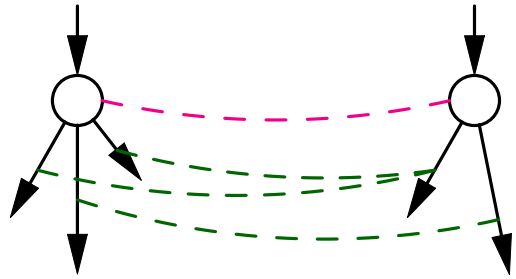
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

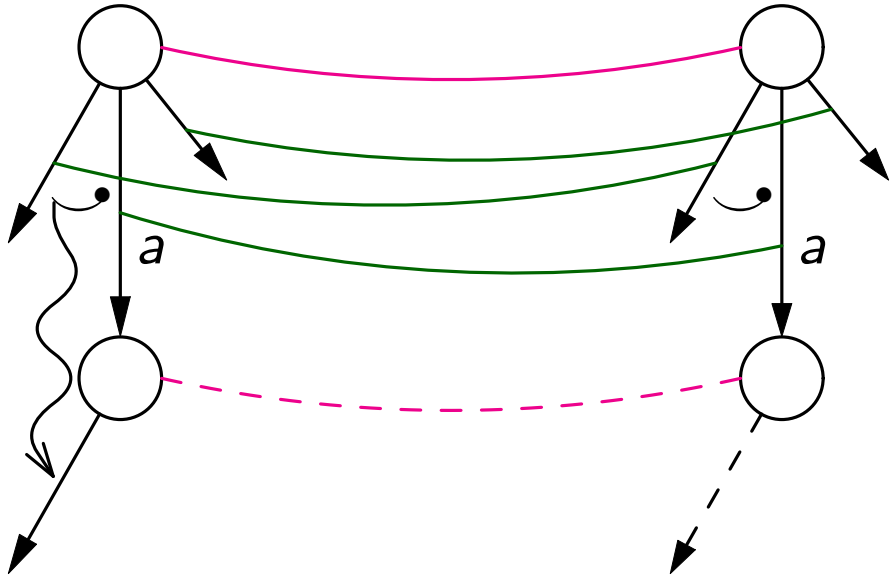
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

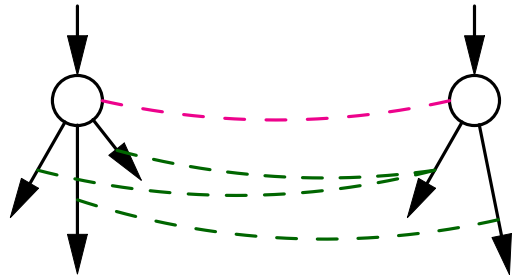
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

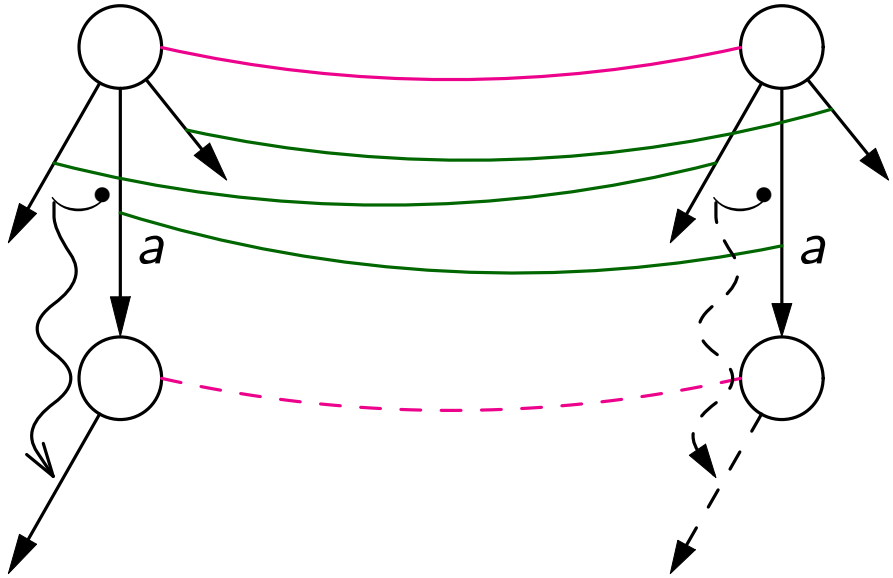
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

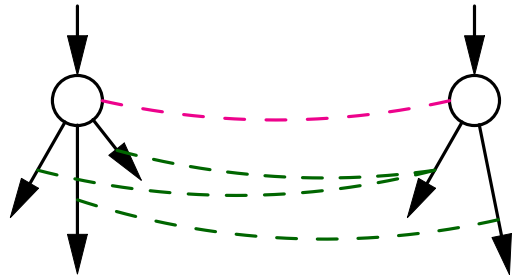
- The *transfer property* holds:



Enabling preserving bisimulation equivalence

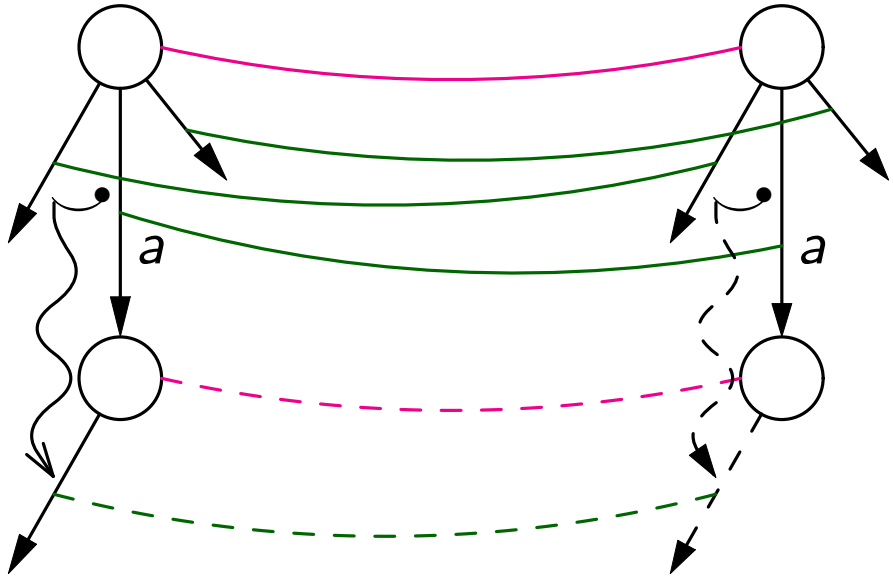
To show that two systems have the same liveness properties, one constructs an *enabling preserving bisimulation* between them.

This is a relation $\mathcal{R}$ between their states, where each pair of related states is equipped with a relation R between their enabled transitions, such that



- The initial states are related:

- The *transfer property* holds:



EP bisimulation is useful

This notion of bisimulation has the properties we want:

- it preserves liveness properties
  under the assumption of justness;
- it induces an equivalence relation,
- which is a congruence for parallel composition (and other operators), thus allowing compositional reasoning.

EP bisimulation is useful

This new bisimulation can be used
to prove implementations equivalent to specifications
in such a way that – under the assumption of justness –

all liveness properties of the specification
also hold for the implementation.

Applications: packet delivery

Once a framework for formalising liveness properties,
and right tools for proving them, have been established,
I aim to to apply them to realistic verifications of distributed systems.

Applications: packet delivery

Once a framework for formalising liveness properties,
and right tools for proving them, have been established,
I aim to to apply them to realistic verifications of distributed systems.

The *packet delivery* property for routing protocols in wireless networks.

"Under suitable conditions a data packet injected at a source node
will eventually be delivered at its destination node."

Applications: packet delivery

Once a framework for formalising liveness properties,
and right tools for proving them, have been established,
I aim to to apply them to realistic verifications of distributed systems.

The *packet delivery* property for routing protocols in wireless networks.

"Under suitable conditions a data packet injected at a source node
will eventually be delivered at its destination node."

Here fairness assumptions are not warranted:
they validate versions of packet delivery that do not hold.

Yet, without assuming justness no useful packet delivery property holds.

Applications: locks

Locks take the role of mutual exclusion protocols in efficient implementations of distributed systems.

ticket lock Mellor-Crummey Scott lock Craig Landin Hagersten lock

As for mutex protocols, the correctness properties of such locks require a justness assumption at the least, whereas fairness assumptions can be demonstrated to assume too much.

Moreover, the proposed language extensions come into play.

Applications: locks

Locks take the role of mutual exclusion protocols in efficient implementations of distributed systems.

ticket lock Mellor-Crummey Scott lock Craig Landin Hagersten lock

As for mutex protocols, the correctness properties of such locks require a justness assumption at the least, whereas fairness assumptions can be demonstrated to assume too much.

Moreover, the proposed language extensions come into play.

Many semi-formal verifications apply the justness assumption implicitly.

I here strive for better reusability through a higher degree for formalisation in which such an assumption becomes explicit.

Applications: garbage collection

In celebrated verifications of garbage collection in software (see, e.g., Gammie et al., PLDI 2015: 99-109), merely safety properties are verified, stating that only garbage will be removed.

Applications: garbage collection

In celebrated verifications of garbage collection in software (see, e.g., Gammie et al., PLDI 2015: 99-109), merely safety properties are verified, stating that only garbage will be removed.

The authors apologise for not showing liveness properties, stating that all garbage will be removed.

Applications: garbage collection

In celebrated verifications of garbage collection in software (see, e.g., Gammie et al., PLDI 2015: 99-109), merely safety properties are verified, stating that only garbage will be removed.

The authors apologise for not showing liveness properties, stating that all garbage will be removed.

"formal treatment of liveness is likely to be complex [...], given the need for dubious fairness hypotheses."

Applications: garbage collection

In celebrated verifications of garbage collection in software (see, e.g., Gammie et al., PLDI 2015: 99-109), merely safety properties are verified, stating that only garbage will be removed.

The authors apologise for not showing liveness properties, stating that all garbage will be removed.

"formal treatment of liveness is likely to be complex [...], given the need for dubious fairness hypotheses."

This is exactly the type of problem that will be solved by my proposed work.