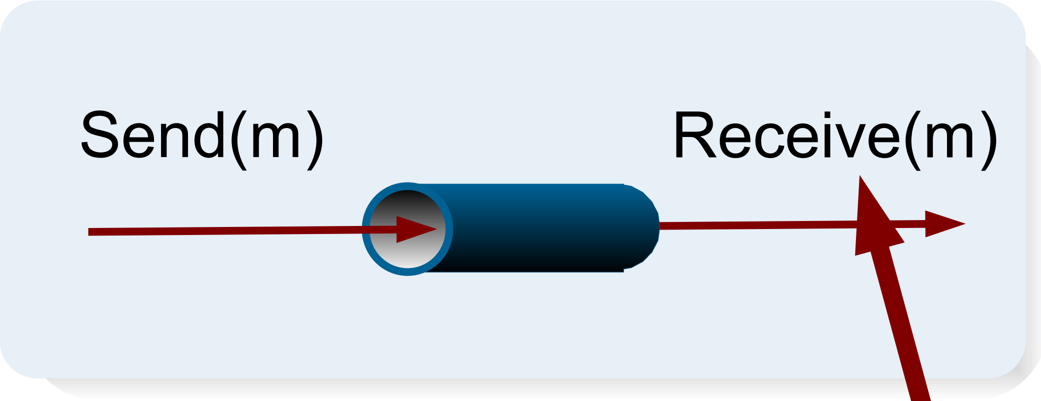
Trace properties

- A trace is the externally visible sequence of actions
- A trace property is a set of traces
- Proof strategy:
  - Add the trace as a variable to the state
  - Safety trace properties are then invariant assertions

Example: Reliable channel



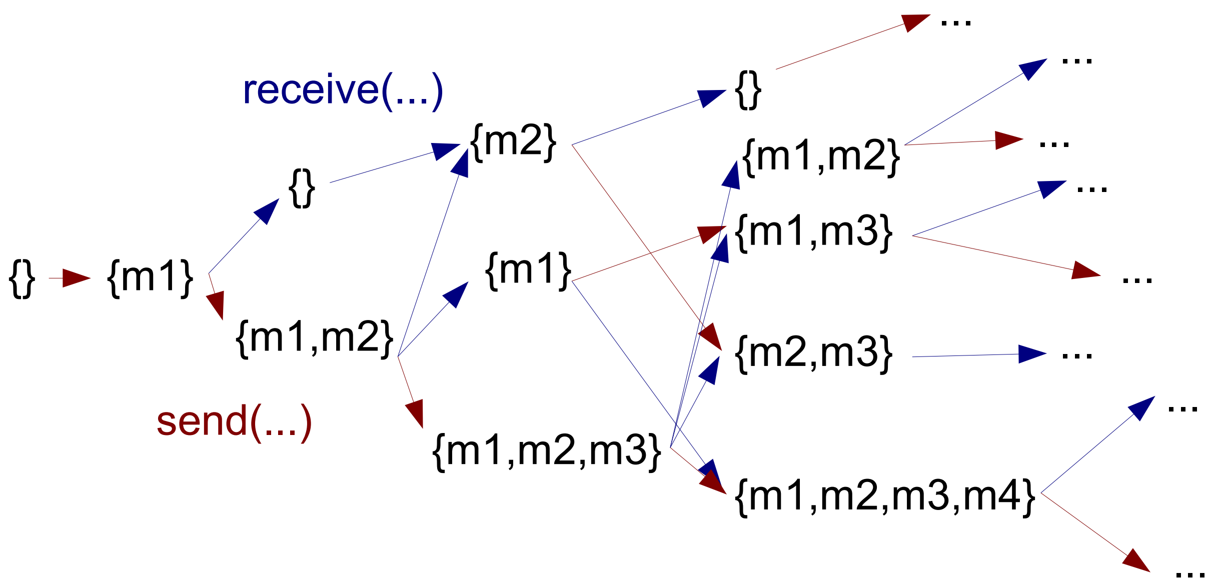
- Reliable channel:
  - Unordered
  - FIFO

Why *Receive(m)* and not *m := Receive()*?

Example: Reliable channel

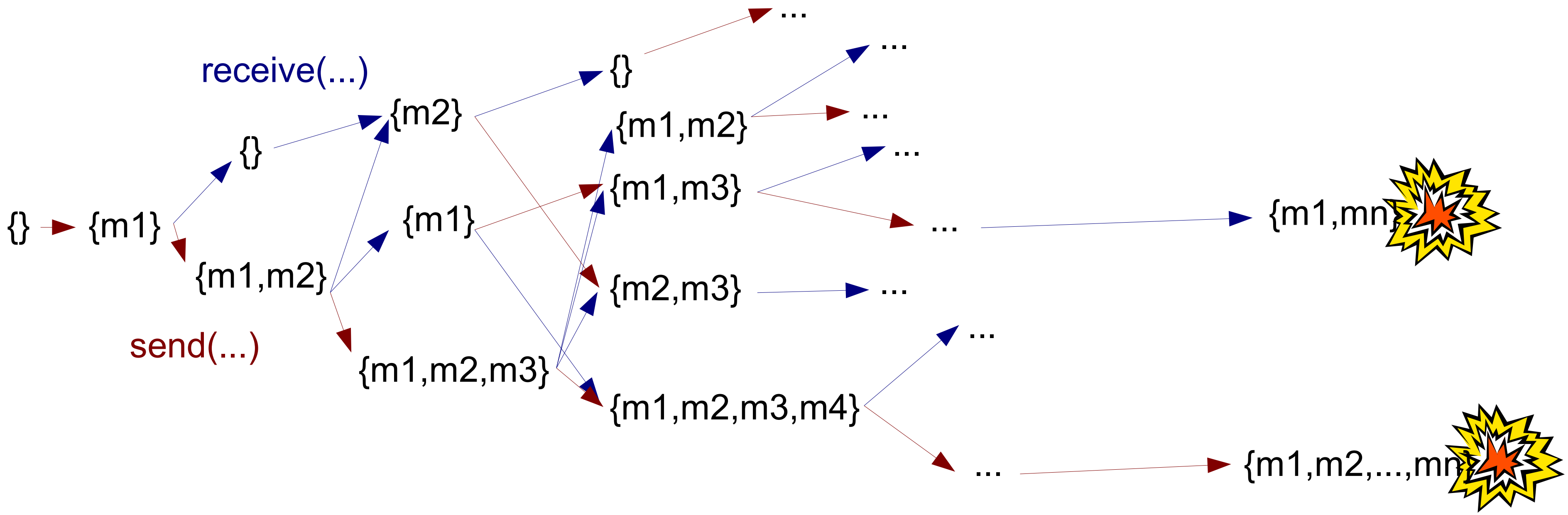
- State:
  - transit, bag of M, initially {}
- Send(m), $m \in M$:
  - Pre-condition:
    - True
  - Effect:
    - transit :=transit + {m}
- Receive(m), $m \in M$:
  - Pre-condition:
    - m in transit
  - Effect:
    - transit := transit - {m}

Behaviors of a channel



- Concurrency is modeled by alternative enabled transitions:
  - Sender and receiver
  - Within the channel (reordering)

Liveness and fairness

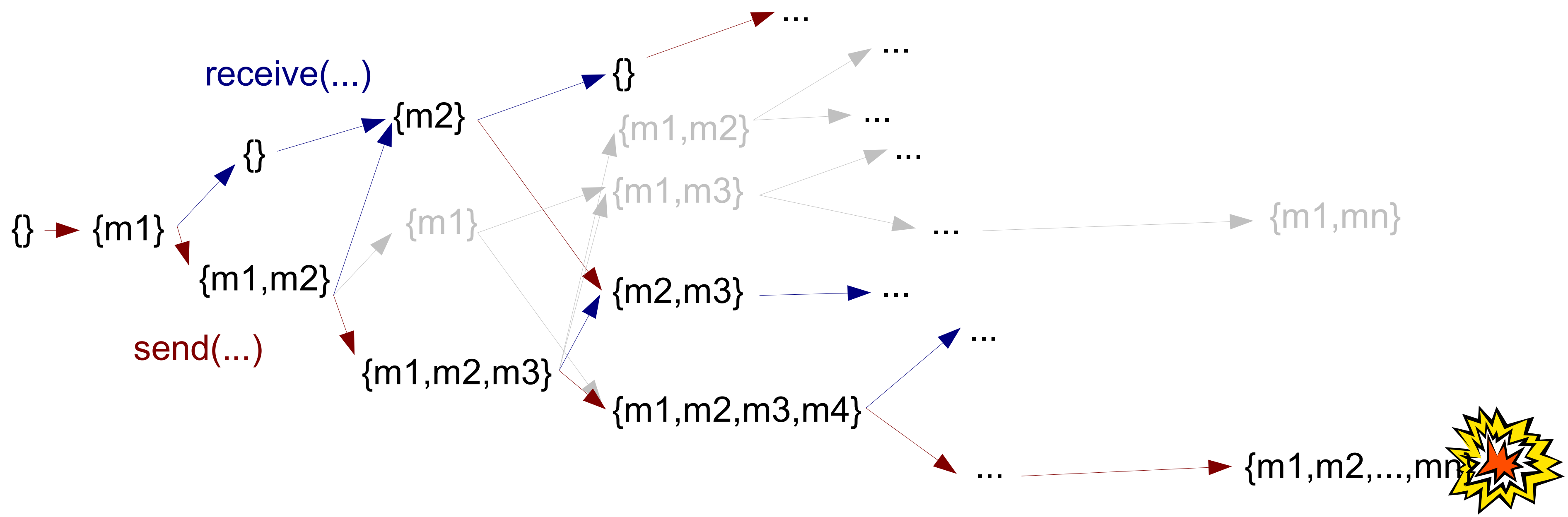


- Some behaviors do not satisfy liveness:
  - If m is sent, eventually m is received
- Some transitions don't get a fair chance to run:
  - receive(m1) and receive(m*)

Fairness

- Partition transitions in tasks:
  - Tasks:
    - For all m: {receive(m)}
- Assume that no task can be forever prevented to take a step
- What about a FIFO reliable channel?

Liveness and fairness



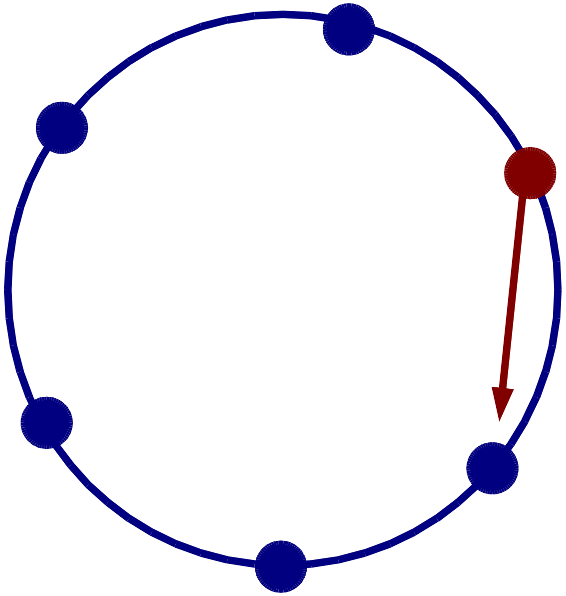
- FIFO order excludes a number of behaviors
  - Only executions with a finite number of receive(m) steps are unfair
- Fairness ensured by a single task:
  - {For all m: receive(m)}

Example: FIFO channel

- State:
  - transit, seq. of M, initially <>
- Send(m), $m \in M$:
  - Pre-condition:
    - True
  - Effect:
    - transit :=transit+<m>
- Receive(m), $m \in M$:
  - Pre-condition:
    - m=head(transit)
  - Effect:
    - transit := tail(transit)
- Tasks:
  - {For all m: receive(m)}

Example: Token ring

- Rotating token algorithm:



- Mutual exclusion?
- Deadlock freedom?

Example: Token ring

- State:
  - n is the number of nodes
  - token[0]=1
  - token[i]=0, for 0<i<n
- Move(i):
  - Pre-condition:
    - token[i]=1
  - Effect:
    - token[i]:=0
    - token[(i+1) mod n]:=1

Example: Token ring

- Mutual exclusion:
  - There is at most one token in the ring (i.e. sum of token[i]≤1)
- Proof by induction:
  - Base step:
    - $\sum$token[i]=1 trivially true
  - Induction step:
    - $\sum$token-before[i]≤1⇒$\sum$token-after[i]≤1

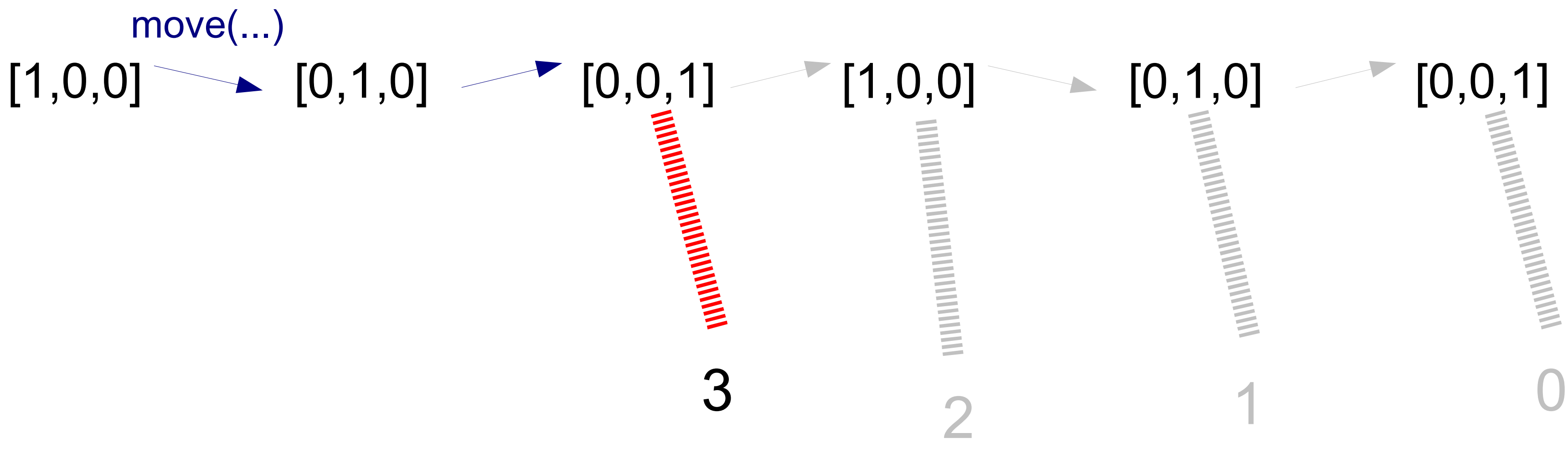
Example: Token ring

- No starvation:
  - Eventually i gets the token at least k times
- Proof with a progress function:
  - Function from state to a well-founded set
  - Helper actions decrease the value
  - Other actions do not increase the value
  - Helper actions are taken until goal is met (i.e. enabled and in separate tasks)

Invariant assertion

Progress function

- Define progress function f as:
  - Target is non-negative integers
  - Value is ((k-1) x n + i - 1) - length(trace)
- Example with n=3, k=2, and i=3:

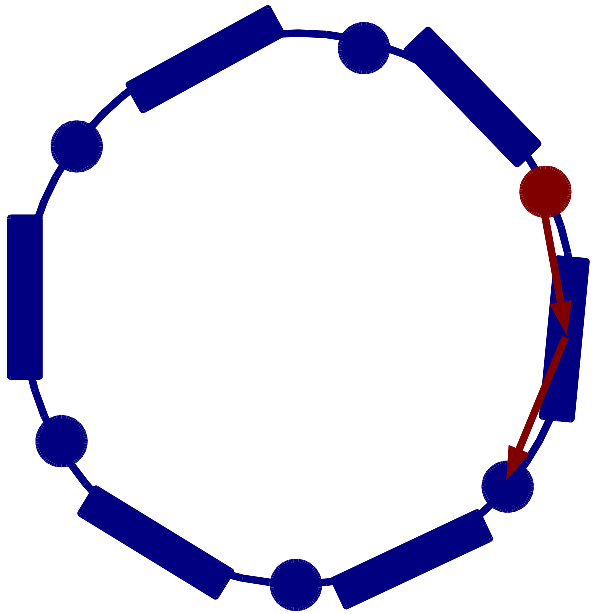


Summary

- I/O Automata definition
  - Safety specification
  - Fairness specification
- Proof strategies for:
  - Invariants
  - Trace properties
    - Safety
    - Liveness
- How to apply to large and complex specifications?

Example: Token ring with channels

- Refine the specification to include channels:



- Mutual exclusion?
- Deadlock freedom?

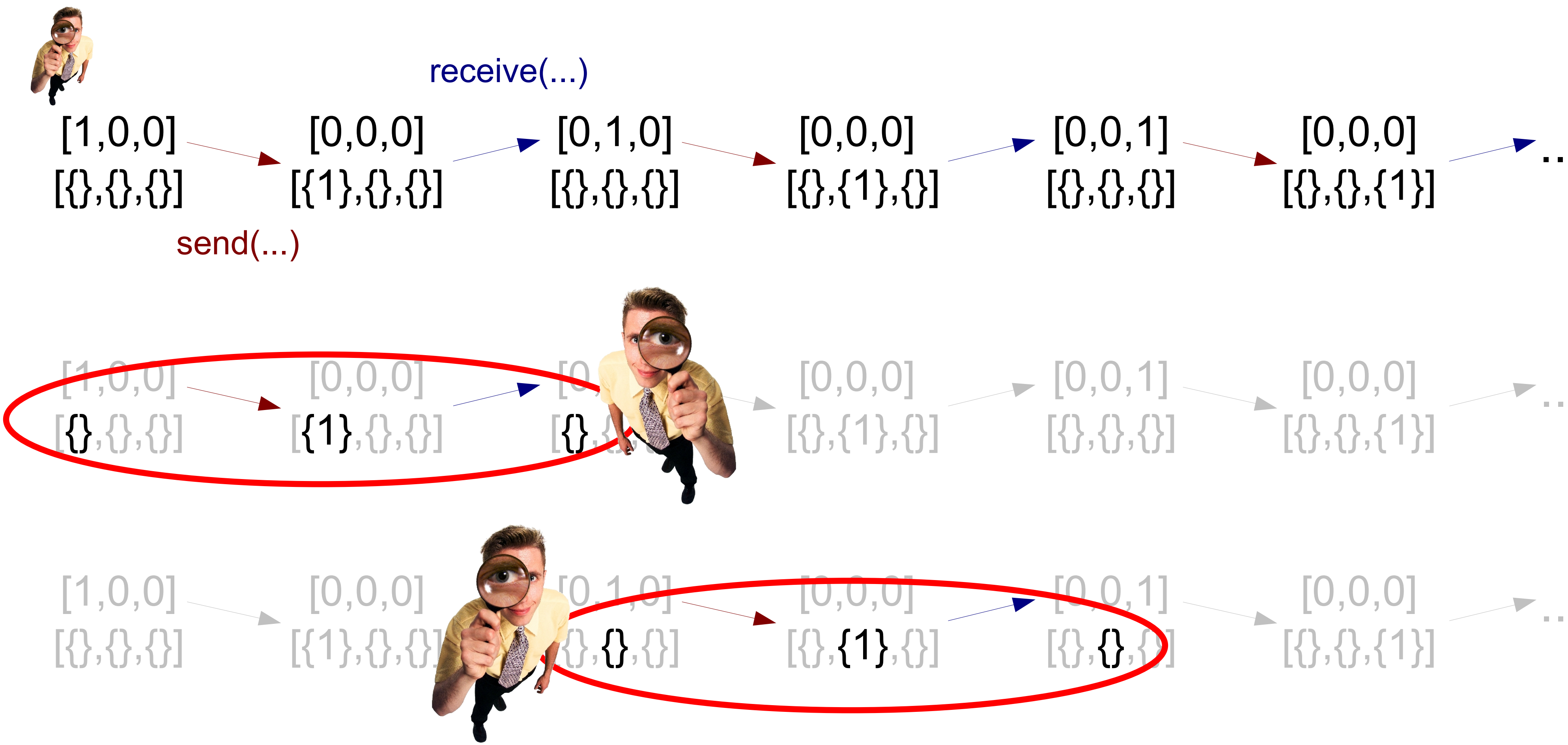
Example: Token ring with channels

- Initially:
  - n is the number of nodes
  - token[0]=1
  - token[i]=0, for 0<i<n
  - transit[i]={}, for all i
- Send:
  - Pre-condition:
    - token[i]=1
  - Effect:
    - token[i]:=0
    - transit[i]:={1}
- Receive:
  - Pre-condition:
    - 1 in transit[i]
  - Effect:
    - token[(i+1)mod n]:=1
    - transit[i]:={}

Example: Token ring with channels

- Proof of mutual exclusion?
- Seems to be true. But...
  - $\sum token[i] \leq 1$, with token=[1,0,0,...] and transit[0]={1}
  - after receive, $\sum token[i]=2$!
- Solution is to strengthen the invariant:
  - Prove by induction: $\sum token[i] + \sum elems(transit[i]) \leq 1$
  - Then conclude $\sum token[i] \leq 1$
    (assuming that transit[i] not negative, easy to prove)

Example: Token ring with channels



- One can observe valid executions of reliable channels embedded in the ring

Composition

- Compatible automata:
  - Internal actions do not overlap with any other actions
  - Output actions are disjoint
  - No action is contained in infinitely many automata
- This allows:
  - Several input actions to overlap
  - Input actions to overlap with a single output action

Composition

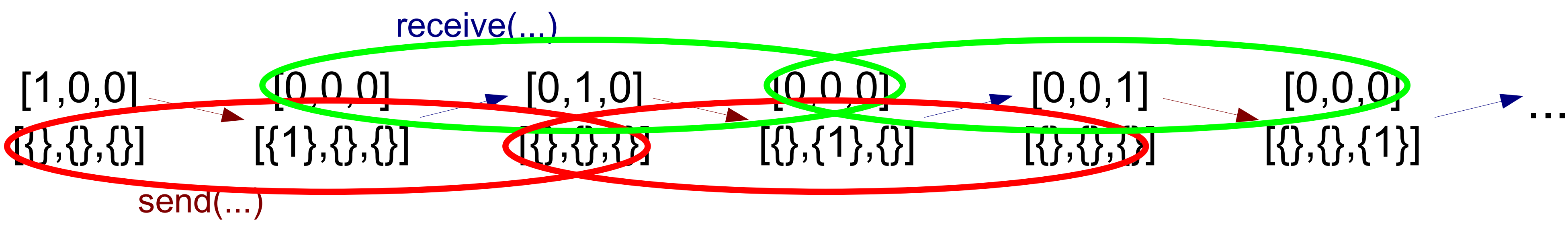
- A composition A with signature S from a set of Ai, with signature Si
- The state of the composed automaton A is:
  - state(A) = Π state(Ai)
  - start(A) = Π start(Ai)
- The signature of S is as follows:
  - out(S) = U out(Si)
  - int(S) = U int(Si)
  - in(S) = U in(Si) – out(S)
- Transitions and tasks likewise

Example: A process

- State:
  - token, integer, initially 0
- Send(m), $m \in M$:
  - Pre-condition:
    - token = 1
  - Effect:
    - token := 0
- Receive(m), $m \in M$:
  - Pre-condition:
    - true
  - Effect:
    - token := 1

Example: Composite token ring

- send(m) is an input to a channel
  - overlaps with receive(m) in a process
- receive(m) is an input to a process
  - overlaps with send(m) in a channel



Compositional reasoning

- A necessary condition for mutual exclusion in a ring is that the token is not duplicated while in transit
- Consider the following trace property:
  - For each receive(m) (i.e. lock), there is some corresponding send(m) (i.e. unlock)
- This property is true for each individual reliable channel
- Therefore it is true for the composed token ring