Verification of Computing Systems

Masami Hagiya
Cyrille Artho (RCIS, AIST)

Plan (Three Parts)

- Oct 24: introduction and LTL (Linear Time Temporal Logic)
- Oct 31: LTL (continued) and model checking
- Nov 7: abstract model checking and CEGAR
- Nov 14 21 28 Dec 5 12 19 (or 22): software model checking (with JFP, etc.) by Cyrille Artho
- Jan 16: verification of security protocols: symbolic analysis (model checking and strand space)
- Jan 23: computational analysis
- Jan 30: computational soundness (mapping lemma)

How to Mark

- Three reports
  - one for each of the three parts

References

- The first part
  - *Model Checking*, Clarke, Grumberg and Peled, The MIT Press.
  - A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs, *TACAS'99*, LNCS1579, 1999.
  - *Principles of the Spin Model Checker*, Ben-Ari, Springer.
  - *Principles of Model Checking*, Baier and Katoen, The MIT Press
  - http://research.microsoft.com/en-us/projects/slam/

Formal Verification

- For building more reliable computing systems
  - Hardware
  - Software
  - Protocols
  - Algorithms
- Dependability

  IT as social infrastructure like electricity and gas
- Methodologies/tools based on symbolic logic

  Maximum reliability but high human cost

  $\Rightarrow$ Mainly applied to safety critical systems

Software Failures in Japan

organization	year/month	cause
Mizuho Bank	2011/3	Batch processing capacity exceeded
JR East (shinkansen control system)	2011/1	Overflow in the number of data updates
Apple	2010/6	Too many reservation requests
ANA (reservation system)	2008/9	Authentication software license expired
Tokyo Stock Market	2008/3	Deadlock due to overconcentration of orders
Tokyo Stock Market	2005/11	Data transfer target unidentified
National Tax Agency (income tax document preparation system)	2004/4	Error in mutual exclusion

Methodologies/Tools

- Proof assistants
  - Higher-order logic/type theory + inductive def.
  - General purpose
  - Semi-automatic proof (tactic) + interactive proof
- State exploration/model checking
  - Temporal logic (discrete/continuous/probabilistic)
  - Targeted to state transition systems (such as distributed algorithms)
  - Fully automatic proof, subsuming static analysis (type analysis/inference, dataflow analysis, etc.)
- Merging the two in various approaches

Temporal Logic

- Logic that describes systems with respect to state transition and temporal evolution
- Linear-time temporal logic
  - LTL (Linear-time Temporal Logic)
- Branching-time temporal logic
  - CTL (Computation Tree Logic)
  - μ-calculus
- Model checking

  Method to verify that a target system satisfies a property expressed in temporal logic

Example: Peterson's Algorithm

```
me  = 0;
you = 1;
for (;;) {
    flags[me] = true;
    turn = you;
    while (flags[you] == true) {
        if (turn != you) break;
    }
    // the critical section
    flags[me] = false;
    // the idle part
}
```

Example: Peterson's Algorithm

```
me  = 1;
you = 0;
for (;;) {
    flags[me] = true;
    turn = you;
    while (flags[you] == true) {
        if (turn != you) break;
    }
    // the critical section
    flags[me] = false;
    // the idle part
}
```

Example: Peterson's Algorithm

0: flags[me] = true;

1: turn = you;

2: if (flags[you] != true) goto 4;

3: if (turn != you) goto 4; else goto 2;

4: critical section;

5: flags[me] = false;

6: either goto 6 or goto 0;

- state : (pc0, pc1, flags[0], flags[1], turn)
  - pc0, pc1: 0..6
  - flags[0], flags[1]: {true, false}
  - turn: {0, 1}

(pc0, pc1, flags[0], flags[1], turn) = (2,2,t,t,1) can make a transition to

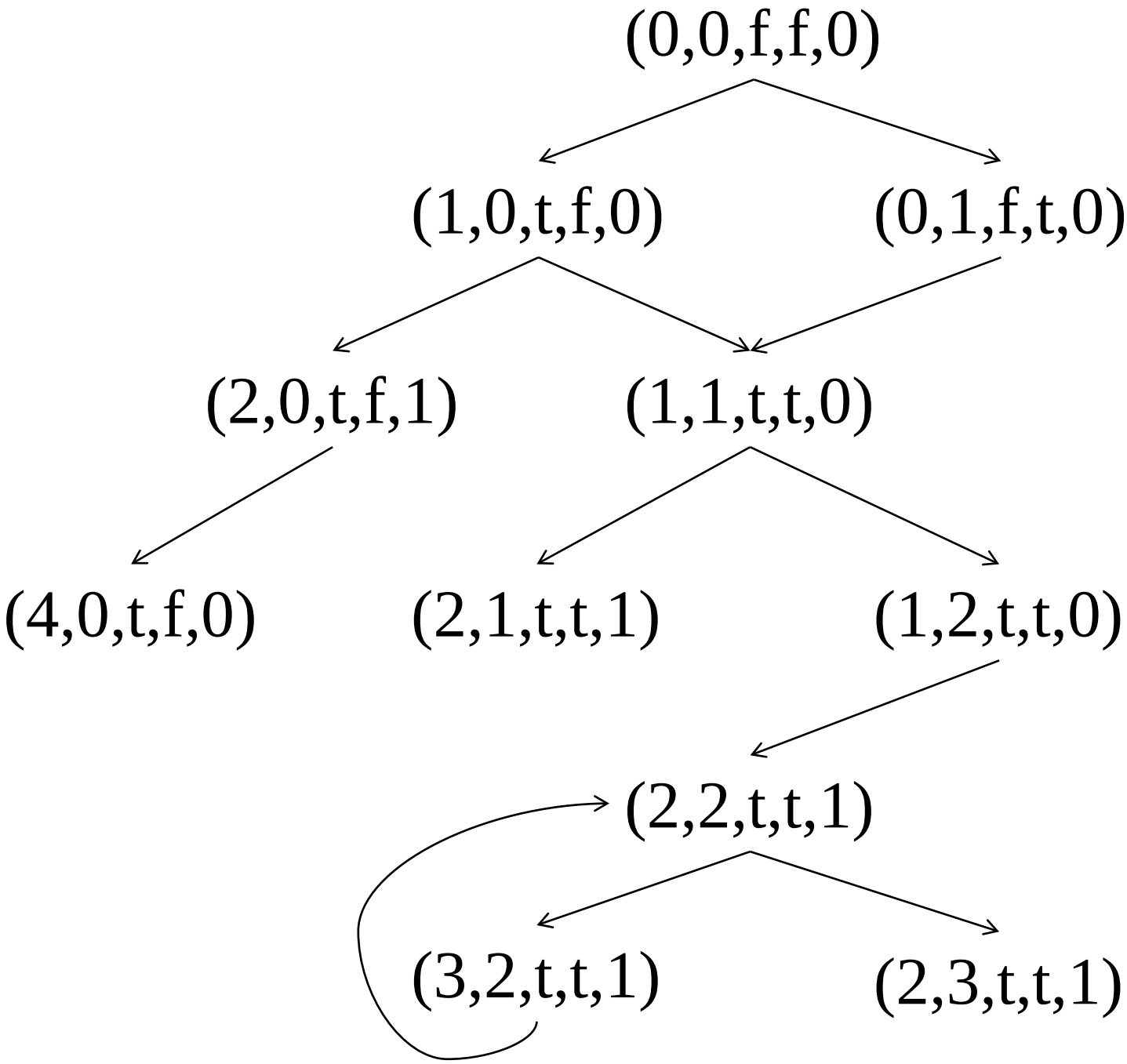
1. (2,4,t,t,1)
2. (3,2,t,t,1)
3. (4,2,t,t,1)

```
0: flags[me] = true;
1: turn = you;
2: if (flags[you] != true) goto 4;
3: if (turn != you) goto 4; else goto 2;
4: critical section;
5: flags[me] = false;
6: either goto 6 or goto 0;
```

(pc0, pc1, flags[0], flags[1], turn) = (3,2,t,t,1) can make a transition to

1. (2,2,t,t,1)
2. (3,4,t,t,1)
3. (4,2,t,t,1)

```
0: flags[me] = true;
1: turn = you;
2: if (flags[you] != true) goto 4;
3: if (turn != you) goto 4; else goto 2;
4: critical section;
5: flags[me] = false;
6: either goto 6 or goto 0;
```


(0,0,f,f,0)
(1,0,t,f,0)
(0,1,f,t,0)
(2,0,t,f,1)
(1,1,t,t,0)
(4,0,t,f,0)
(2,1,t,t,1)
(1,2,t,t,0)
(2,2,t,t,1)
(3,2,t,t,1)
(2,3,t,t,1)

Properties Verified on State Transition Systems

- Safety
  - Something wrong never occurs
- Liveness (no starvation)
  - Something good eventually occurs
- Liveness requires fairness
  - Without fairness, specific processes are only executed
- Fairness assumption
  - Unconditional fairness
  - Weak fairness
  - Strong fairness

Which is a liveness property?

1. No deadlock occurs
2. A file being opened always exists
3. An opened file is eventually closed
4. An index to an array is always within its bound

In Case of Peterson's Algorithm

- Safety
  - The two processes never enter their critical sections simultaneously
  - pc0=pc1=4 never holds
- Liveness (no starvation)
  - If a process has entered its header part, it eventually enters its critical section
- Liveness requires fairness
- Fairness assumption
  - Both processes are executed infinitely often (unconditional fairness)
  - Each process is assumed not to enter an infinite loop inside its critical section

Expressing Correctness in Temporal Logic

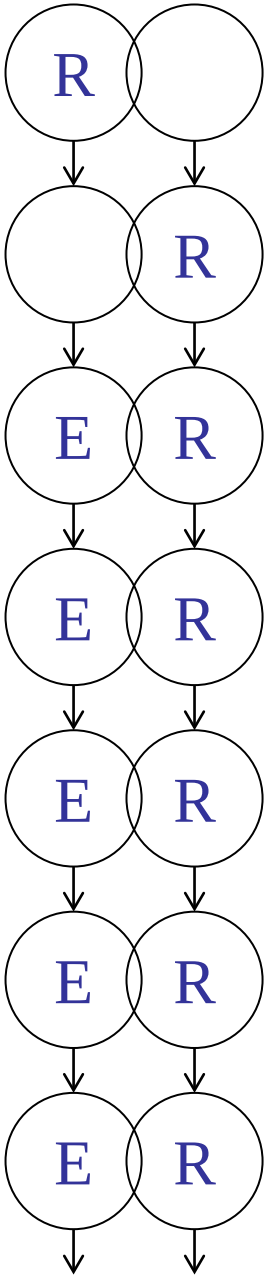
- Safety
  - LTL（Linear-time Temporal Logic)

    $\Box(\neg(pc0=4 \wedge pc1=4))$
  - CTL（Computation Tree Logic）

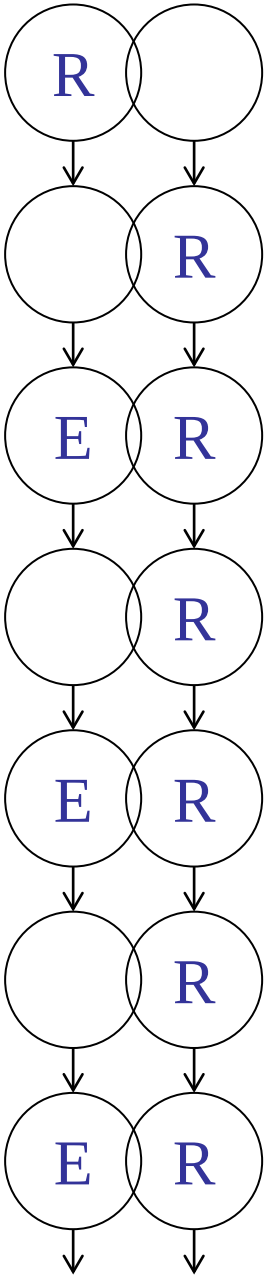
    $AG(\neg(pc0=4 \wedge pc1=4))$
- Liveness (no starvation)
  - LTL $\Box(pc0=0 \supset \Diamond(pc0=4))$
  - CTL $AG(pc0=0 \supset AF(pc0=4))$
- Fairness (assumption)
  - LTL $\Box(pc0=0 \supset \Diamond(pc0=1)) \wedge ...$
  - Fairness cannot be expressed in CTL

Fairness Assumption

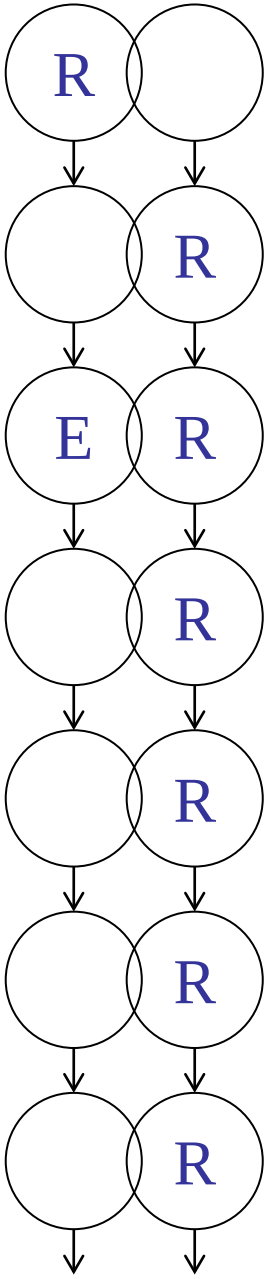
- Unconditional fairness
  - Each process is executed infinitely often
- Weak fairness
  - Any process that is continuously executable is eventually executed
  - No process is kept continuously executable for ever without being executed
- Strong fairness
  - No process becomes executable infinitely often without being executed



Violating
Weak fairness



Violating
Strong fairness



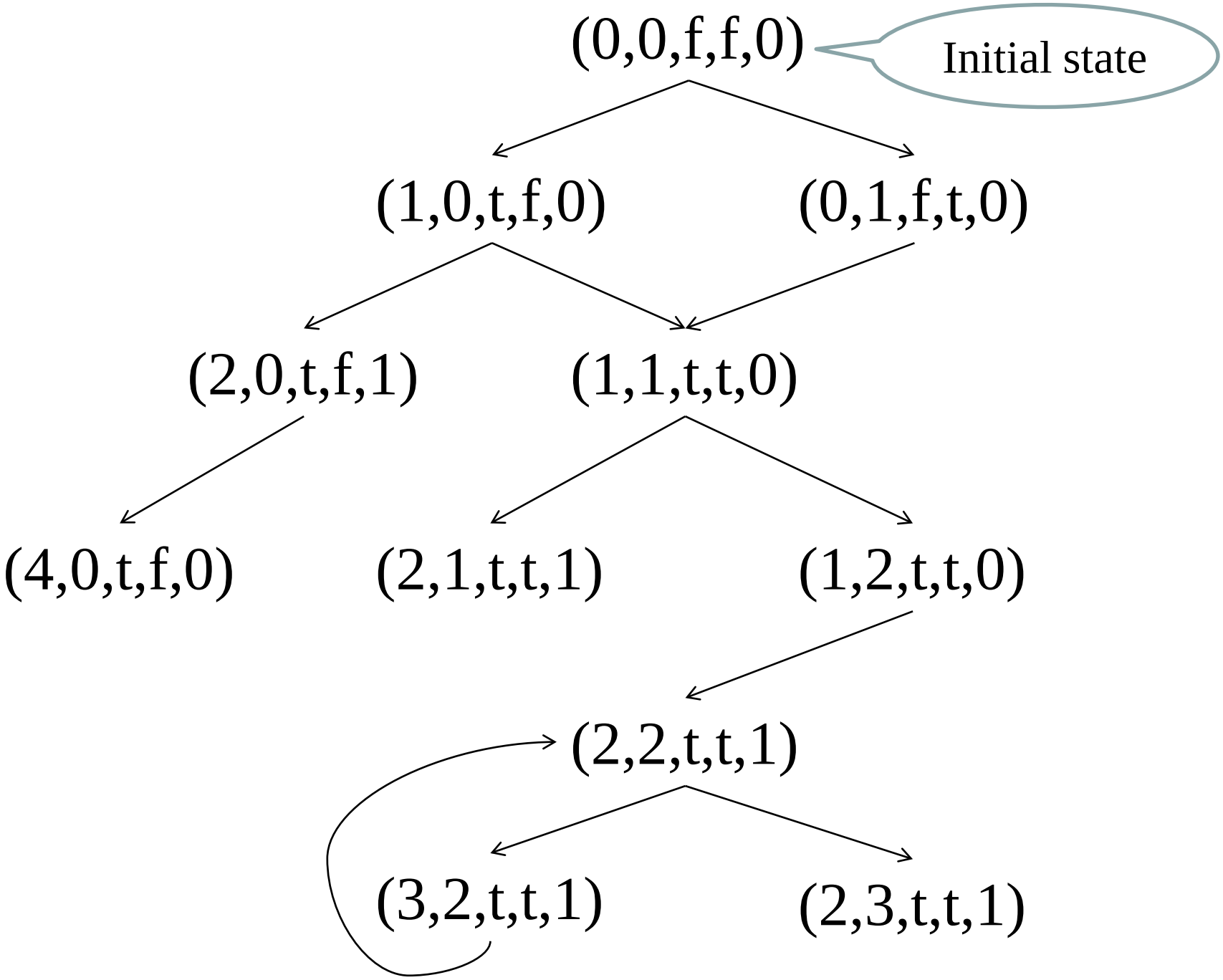
Violating
Unconditional fairness

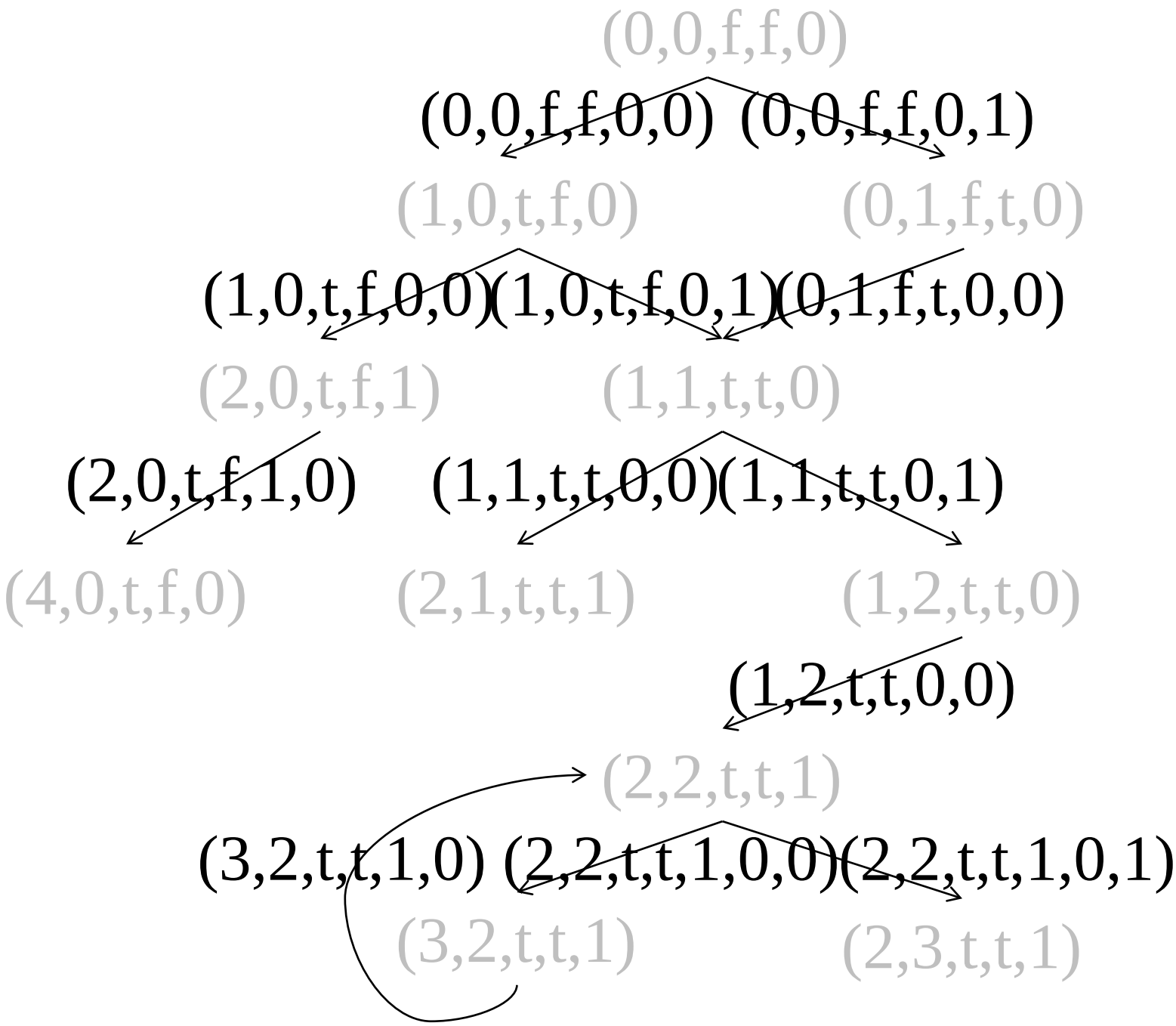
E: is executable
(enabled)
R: runs (is executed)
next

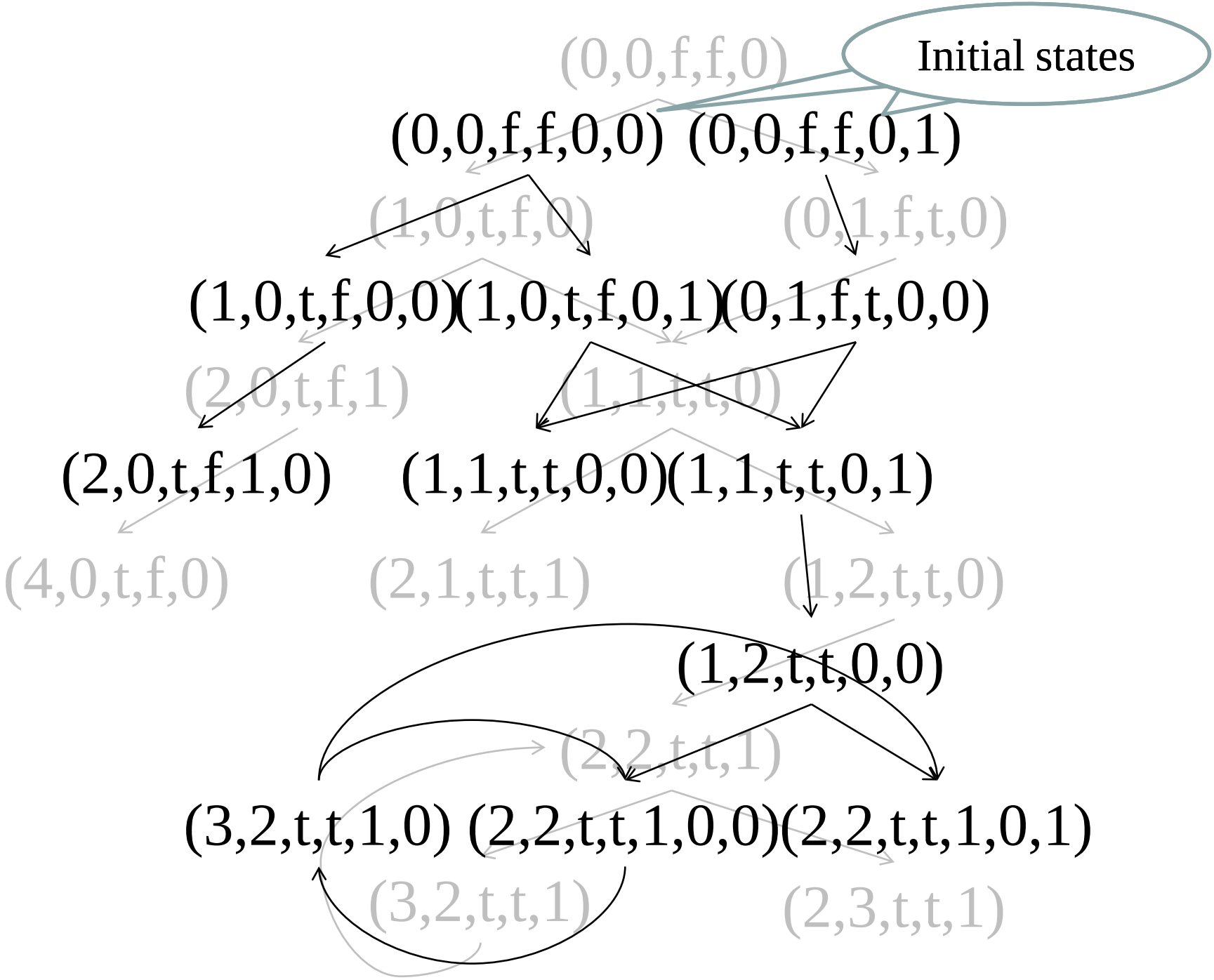
Peterson again

- One can augment a state with the information about which process is executed next

  (pc0, pc1, flags[0], flags[1], turn, next)
- If next=0, then process0 runs
- If next=1, then process1 runs
- Both processes are always enabled (unconditional fairness)


(0,0,f,f,0)
Initial state
(1,0,t,f,0)
(0,1,f,t,0)
(2,0,t,f,1)
(1,1,t,t,0)
(4,0,t,f,0)
(2,1,t,t,1)
(1,2,t,t,0)
(2,2,t,t,1)
(3,2,t,t,1)
(2,3,t,t,1)


(0,0,f,f,0)
(0,0,f,f,0,0)
(0,0,f,f,0,1)
(1,0,t,f,0)
(0,1,f,t,0)
(1,0,t,f,0,0)
(1,0,t,f,0,1)
(0,1,f,t,0,0)
(2,0,t,f,1)
(1,1,t,t,0)
(2,0,t,f,1,0)
(1,1,t,t,0,0)
(1,1,t,t,0,1)
(4,0,t,f,0)
(2,1,t,t,1)
(1,2,t,t,0)
(1,2,t,t,0,0)
(2,2,t,t,1)
(3,2,t,t,1,0)
(2,2,t,t,1,0,0)
(2,2,t,t,1,0,1)
(3,2,t,t,1)
(2,3,t,t,1)


Initial states
(0,0,f,f,0)
(0,0,f,f,0,0)
(0,0,f,f,0,1)
(1,0,t,f,0)
(0,1,f,t,0)
(1,0,t,f,0,0)
(1,0,t,f,0,1)
(0,1,f,t,0,0)
(2,0,t,f,1)
(1,1,t,t,0)
(2,0,t,f,1,0)
(1,1,t,t,0,0)
(1,1,t,t,0,1)
(4,0,t,f,0)
(2,1,t,t,1)
(1,2,t,t,0)
(1,2,t,t,0,0)
(2,2,t,t,1)
(3,2,t,t,1,0)
(2,2,t,t,1,0,0)
(2,2,t,t,1,0,1)
(3,2,t,t,1)
(2,3,t,t,1)

Report1

- Enumerate all processes that are reachable from the initial state(s) and check that no state with pc0=pc1=4 is reachable
- Check also that if pc0=2 is true, then pc0=4 eventually becomes true under unconditional fairness
- You should write a program!