

Safety & Liveness

Uwe R. Zimmer - The Australian National University



Safety & Liveness

Repetition

Correctness concepts in concurrent systems

Liveness properties:

 $(P(I) \land Processes(I,S)) \Rightarrow \Diamond Q(I,S)$ where $\Diamond Q$ means that Q does eventually hold (and will then stay true)

Examples:

- Requests need to complete eventually.
- The state of the system needs to be displayed eventually.
- No part of the system is to be delayed forever (fairness).
- a Interesting liveness properties can become very hard to proof



Safety & Liveness

Deadlocks

Most forms of synchronization may lead to

Reserving resources in reverse order

Towards synchronization

Safety & Liveness

var reserve_1, reserve_2 : semaphore := 1;

process P2; statement A;

process P1; statement X;

Deadlocks

(Avoidance / prevention of deadlocks is one central safety property)

- How to predict them?

□ ... or are there structurally dead-lock free forms of synchronization?

References for this chapter

Ben2006]

Ben-Ari, M Principles of Concurrent and Dis-tributed Programming second edition, Prentice-Hall 2006

[Silberschatz 2001] Silberschatz, Abraham, Gal-vin, Peter & Gagne, Greg Operating System Concepts John Wiley & Sons, Inc., 2001

 $(P(I) \land Processes(I,S)) \Rightarrow \Box Q(I,S)$ where $\Box Q$ means that Q does always hold

Correctness concepts in concurrent systems

Repetition

Safety & Liveness

Extended concepts of correctness in concurrent systems:

— Termination is often not intended or even considered a failure

 $(P(J) \land Processes (I,S)) \Rightarrow \diamondsuit Q (I,S)$ where $\diamondsuit Q$ means that Q does eventually hold land will then stay true) and S is the current state of the concurrent system



Safety & Liveness

Fairness Liveness

Liveness properties:

Revisiting

Correctness concepts in concurrent systems

- Specified responsiveness or free capabilities ra Real-time systems (typical in real-time / embedded systems or server applications)

• First-in, first-out: $\Diamond R \Rightarrow \Diamond G$... before any other process which applied for the same resource at a later point in time (common fairness in single-note systems).

 Strong fairness: □◇R ⇒ ◇G ... eventually, if a process requests infinitely often. • Linear waiting: $\mathcal{R}_R \Rightarrow \mathcal{O}_C$... before any other process had the same resource granted more than once (common fairness in distributed systems).

• Weak fairness: $\Diamond \Box R \Rightarrow \Diamond G \ldots$ eventually, if a process requests continually.

Fairness (as a means to avoid starvation): Resources will be granted ...

Safety & Liveness

Circular dependencies Towards synchronization

var reserve_1, reserve_2, reserve_3 : semaphore := 1;

process P3; statement K;

process P1; statement X;

statement B; -- employ all resources

tatement Y; -- employ all resources

statement Z;

statement M; statement C; Sequence of operations: $A \to B \to C$; $X \to Y \to Z$; $K \to L \to M$; $[X,Z] A,B,C \mid K,M] [A,C \mid X,Y,Z \mid K,M] [A,C \mid K,L,M \mid X,Z] \to [B \mid Y \mid L]$

or: $[A \mid X \mid K]$ followed by a deadlock situation.

Sequence of operations: $A \rightarrow B \rightarrow C$; $X \rightarrow Y \rightarrow Z$; $[XZ \mid A,B,C]$; $[A,C \mid X,Y,Z]$; $\rightarrow [B \mid Y]$ or: $[A \mid X]$ followed by a deadlock situation.

statement C; end P2;

statement Z; end P1;

Safety & Liveness

[Chandy1983]
Chandy, K. Misra, Jayadev & Haas, Laura
Distributed deadlock detection
Transactions on Computer Systems (TOCS) 1983 vol. 1 (2)



Safety & Liveness

Safety properties:

 $(P(I) \land Processes(I,S)) \Rightarrow \Diamond Q(I,S)$ where $\Diamond Q$ means that Q does eventually hold (and will then stay true)

 $(P(I) \land Processes(I,S)) \Rightarrow \Box Q(I,S)$ where $\Box Q$ means that Q does always hold

- Mutual exclusion (no resource collisions) is has been addressed
- (and other forms of 'silent death' and 'freeze' conditions) Absence of deadlocks set to be addressed now



Necessary deadlock conditions:

1. Mutual exclusion:

resources cannot be used simultaneously

472



Necessary deadlock conditions:

Deadlocks

1. Mutual exclusion:

resources cannot be used simultaneously.

2. Hold and wait:

a process applies for a resource, while it is holding another resource (sequential requests).

No pre-emption:
 resources cannot be pre-empted; only the process itself can release resources.

4. Circular wait: a ring list of processes exists,

where every process waits for release of a resource by the next one.

475



Deadlocks

Deadlock prevention

(Remove one of the four necessary deadlock conditions)

1. Break Mutual exclusion:

Hold and wait No pre-emption Circular wait

470 Safety & Liveness Deadlocks Necessary deadlock conditions: 1. Mutual exclusion: resources cannot be used simultaneously. 2. Hold and wait: a process applies for a resource, while it is holding another resource (sequential requests).

473



resources cannot be used simultaneously.

2. Hold and wait: a process applies for a resource, while it is holding another resource (sequential requests).

3. No pre-emption: resources cannot be pre-empted; only the process itself can release resources.

4. Circular wait: a ring list of processes exists,

where every process waits for release of a resource by the next one.

A system may become deadlocked, if all these conditions apply!

476



Deadlocks

Deadlock prevention

(Remove one of the four necessary deadlock conditions)

1. Break Mutual exclusion:

By replicating critical resources, mutual exclusion becomes unnecessary (only applicable in very specific cases).

2. Break Hold and wait:

Hold and wait No pre-emption Circular wait

Safety & Liveness

Deadlocks

Necessary deadlock conditions:

1. Mutual exclusion:

resources cannot be used simultaneously

2. Hold and wait:

a process applies for a resource, while it is holding another resource (sequential requests).

3. No pre-emption:

resources cannot be pre-empted; only the process itself can release resources.

474



Safety & Liveness

Deadlocks

Deadlock strategies:

• Ignorance & restart

Kill or restart unresponsive processes, power-cycle the computer, ...

· Deadlock detection & recovery

im find deadlocked processes and recover the system in a coordinated way

Deadlock avoidance

the resulting system state is checked before any resources are actually assigned

• Deadlock prevention

the system prevents deadlocks by its structure

page 474 of 758 (chapter 7: "Safety & Liveness" up to page 51

477



Safety & Liveness

Deadlocks

Deadlock prevention

(Remove one of the four necessary deadlock conditions)

1. Break Mutual exclusion:

By replicating critical resources, mutual exclusion becomes unnecessary (only applicable in very specific cases).

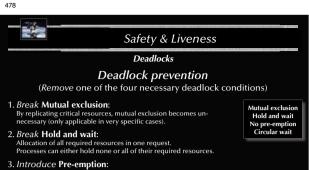
2. Break Hold and wait:

Allocation of all required resources in one request.

Processes can either hold *none* or *all* of their required resources.

3. Introduce Pre-emption::

Hold and wait No pre-emption Circular wait

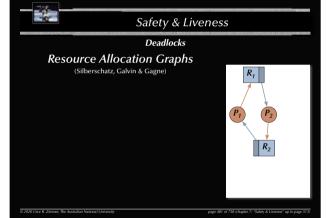


Provide the additional infrastructure to allow for pre-emption of resources. Mind that resources cannot be pre-empted, if their states cannot be fully stored and recovered.

4. Break Circular waits:

481

484



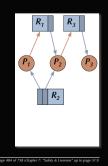
Safety & Liveness

Deadlocks

Resource Allocation Graphs

(Silberschatz, Galvin & Gagne)

™ No circular dependency ™ no deadlock:



Safety & Liveness

Deadlocks

Deadlock prevention

(Remove one of the four necessary deadlock conditions)

1. Break Mutual exclusion:

By replicating critical resources, mutual exclusion becomes un-necessary (only applicable in very specific cases).

2. Break Hold and wait:

479

Allocation of all required resources in one request.

Processes can either hold none or all of their required resources.

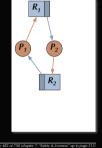
3. Introduce Pre-emption:
Provide the additional infrastructure to allow for pre-emption of resources. Mind that resources cannot be pre-empted, if their states cannot be fully stored and recovered.

4. Break Circular waits:

E.g. order all resources globally and restrict processes to request resources in that order only



Two process, reverse allocation deadlock:



Hold and wait

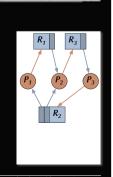
No pre-emption Circular wait

485

Safety & Liveness

Deadlocks

Resource Allocation Graphs (Silberschatz, Galvin & Gagne)



Derived rule:

are cycles in the resource allocation graph.

Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne) $RAG = \{V, E\}$; Resource allocation graphs consist of vertices V and edges E. $V = P \cup R$; Vertices V can be processes P or Resource types R. with processes $P = \{P_1, ..., P_n\}$ and resources types $R = \{R_1, ..., R_k\}$

with claims $E_c = \{P_i \rightarrow R_j, ...\}$ requests $E_r = \{P_i \rightarrow R_j, ...\}$ and assignments $E_a = \{R_j \rightarrow P_j, ...\}$ Note: any resource type R_i can have more than one instance of a resource.

 $E = E_c \cup E_r \cup E_a$; Edges E can be "claims" E_{cl} "requests" E, or "assignments".

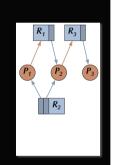
483

Safety & Liveness

Deadlocks

Resource Allocation Graphs

(Silberschatz, Galvin & Gagne)



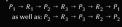
486



Deadlocks

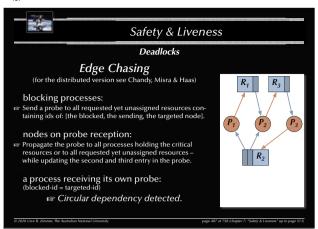
Resource Allocation Graphs (Silberschatz, Galvin & Gagne)

™ Two circular dependencies ™ deadlock:



If some processes are deadlocked then there

492



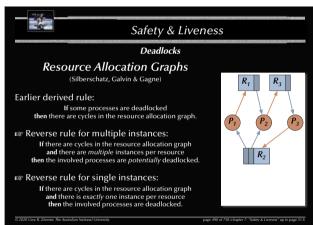
Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne) Knowledge of claims: Claims are potential future requests which have no blocking effect on the claiming process - while actual requests are blocking

488

491

Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne) Assignment of resources such that circular dependencies are avoided:

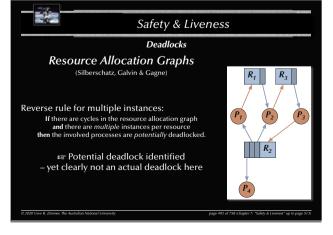
490

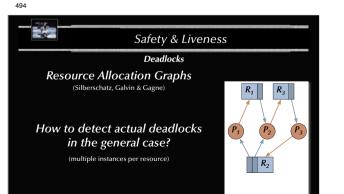


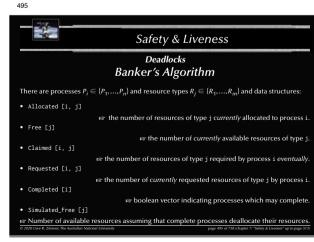
Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne) Reverse rule for single instances: If there are cycles in the resource allocation graph and there is exactly one instance per resource then the involved processes are deadlocked. Readlock identified

Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne) Reverse rule for multiple instances: If there are cycles in the resource allocation graph and there are multiple instances per resource then the involved processes are potentially deadlocked. Potential deadlock identified

493









Safety & Liveness

Deadlocks

Banker's Algorithm

1. Simulated_Free \Leftarrow Free; \forall i: Completed [i] \Leftarrow False;

2. While ∃i: ¬Completed [i]

and $\forall j$: Requested [i, j] < Simulated_Free [j] do:

 \forall j: Simulated_Free [j] \Leftarrow Simulated_Free [j] + Allocated [i, j]; Completed [i] ← True;

3. If ∀i: Completed [i] then the system is currently deadlock-free! else all processes i with —Completed [i] are involved in a deadlock!.

499



Safety & Liveness

Deadlocks

Distributed deadlock detection

Observation: Deadlock detection methods like Banker's Algorithm are too communication intensive to be commonly applied in full and at high frequency in a distributed system.

™ Therefore a distributed version needs to:

- FIF Split the system into nodes of reasonable locality (keeping most processes close to the resources they require).
- Organize the nodes in an adequate topology (e.g. a tree).
- r Check for deadlock inside nodes with blocked resource requests and detect/avoid local deadlock immediately.
- Exchange resource status information between nodes occasionally and detect global deadlocks eventually.

502



Safety & Liveness

Atomic & idempotent operations

Atomic operations

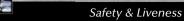
Definitions of atomicity:

An operation is atomic if the processes performing it ...

- . (by 'awareness') ... are not aware of the existence of any other active process, and no other active process is aware of the activity of the processes during the time the processes are performing the atomic operation.
- (by communication) ... do not communicate with other processes while the atomic operation is performed.
- (by means of states) ... cannot detect any outside state change and do not reveal their own state changes until the atomic operation is complete.

Short:

An atomic operation can be considered to be indivisible and instantaneous.



Deadlocks Banker's Algorithm

1. Simulated_Free \Leftarrow Free; \forall i: Completed [i] \Leftarrow False;

2.While $\exists i$: \neg Completed [i]

and $\forall j$: Claimed [i, j] < Simulated_Free [j] do:

 \forall j: Simulated_Free [j] \Leftarrow Simulated_Free [j] + Allocated [i, j]; Completed [i] ← True;

3. If $\forall i$: Completed [i] then the system is safe!

A safe system is a system in which future deadlocks can be avoided assuming the current set of available resources.

500

497



Safety & Liveness

Deadlocks

Deadlock recovery

A deadlock has been detected remains now what?

Breaking the circular dependencies can be done by:

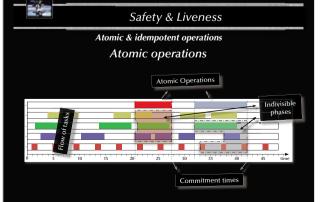
Fither pre-empt an assigned resource which is part of the deadlock. ☞ or stop a process which is part of the deadlock.

Usually neither choice can be implemented 'gracefully' and deals only with the symptoms.

Deadlock recovery does not address the reason for the problem! (i.e. the deadlock situation can re-occur again immediately)

page 500 of 758 (chapter 7: "Safety & Liveness" up to page 51.

503





Safety & Liveness

Deadlocks

Banker's Algorithm

Check potential future system safety by simulating a granted request: (Deadlock avoidance)

if (Request < Claimed) and (Request < Free) then

Free := Free - Request
Claimed := Claimed - Request

Allocated := Allocated + Request

if System_is_safe (checked by e.g. Banker's algorithm) then

Grant request

Restore former system state: (Free, Claimed, Allocated)

end if: end if;

501



Safety & Liveness

Deadlocks

Deadlock strategies:

• Deadlock prevention

System prevents deadlocks by its structure or by full verification

• Deadlock avoidance

System state is checked with every resource assignment.

Deadlock detection & recovery

Detect deadlocks and break them in a 'coordinated' way.

Ess computationally expensive (as lower frequent), yet usually 'messy'

• Ignorance & random kill

Kill or restart unresponsive processes, power-cycle the computer, .

504



Safety & Liveness

Atomic & idempotent operations Atomic operations

Important implications:

- 1. An atomic operation is either performed in full **or** not at all.
- 2. A failed atomic operation cannot have any impact on its surroundings (must keep or re-instantiate the full initial state).
- 3. If any part of an atomic operation fails, then the whole atomic operation is declared failed.
- 4. All parts of an atomic operations (including already completed parts) must be prepared to declare failure until the final global commitment.



Safety & Liveness

Atomic & idempotent operations Idempotent operations

Definition of idempotent operations:

An operation is idempotent if the observable effect of the operation are identical for the cases of executing the operation:

- · multiple times,
- infinitely often

Observations:

- · Idempotent operations are often atomic, but do not need to be.
- · Atomic operations do not need to be idempotent.
- · Idempotent operations can ease the requirements for synchronization.

508



Safety & Liveness

Reliability, failure & tolerance Faults in the logic domain

- Non-termination / -completion
 - Systems 'frozen' in a deadlock state, blocked for missing input, or in an infinite loop Watchdog timers required to handle the failure
- Range violations and other inconsistent states
 - Run-time environment level exception handling required to handle the failure
- Value violations and other wrong results
 - □ User-level exception handling required to handle the failure

511



Reliability, failure & tolerance

Fault prevention, avoidance, removal, ...

and / or

☞ Fault tolerance

Safety & Liveness Reliability, failure & tolerance

'Terminology of failure' or 'Failing terminology'?

Reliability ::= measure of success

with which a system conforms to its specification.

::= low failure rate.

Failure ::= a deviation of a system from its specification.

Error ::= the system state which leads to a failure.

Fault ::= the reason for an error.

509

506



Safety & Liveness

Reliability, failure & tolerance Faults in the time domain

- · Transient faults
- single 'glitches', interference, ... very hard to handle
- · Intermittent faults
- Faults of a certain regularity ... require careful analysis
- · Permanent faults

Faults which stay ... the easiest to find

512



Safety & Liveness

Reliability, failure & tolerance

Fault tolerance

- · Full fault tolerance
 - the system continues to operate in the presence of 'foreseeable' error conditions without any significant loss of functionality or performance even though this might reduce the achievable total operation time.
- Graceful degradation (fail soft)
 - the system continues to operate in the presence of 'foreseeable' error conditions, while accepting a partial loss of functionality or performance
- · Fail safe

the system halts and maintains its integrity

Full fault tolerance is not maintainable for an infinite operation time! FF Graceful degradation might have multiple levels of reduced functionality.



Safety & Liveness

Reliability, failure & tolerance Faults during different phases of design

• Inconsistent or inadequate specifications

Fir frequent source for disastrous faults

• Software design errors

Fire frequent source for disastrous faults

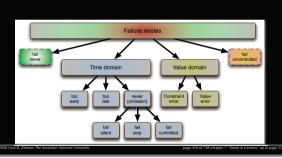
Component & communication system failures

rare and mostly predictable

510



Reliability, failure & tolerance Observable failure modes



513



Safety & Liveness

Summary

Safety & Liveness

- Liveness Fairness
- Safety
- Deadlock detection · Deadlock avoidance
- · Deadlock prevention
- Atomic & Idempotent operations
- · Definitions & implications
- · Failure modes
- · Definitions, fault sources and basic fault tolerance