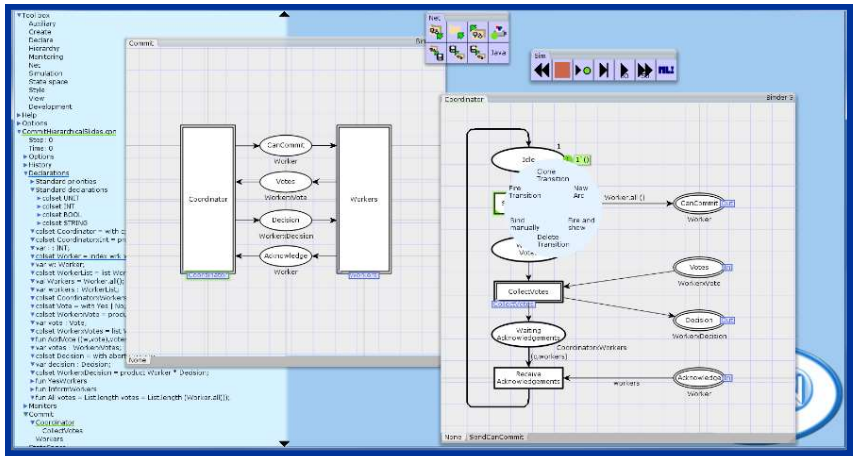
Lecture 6

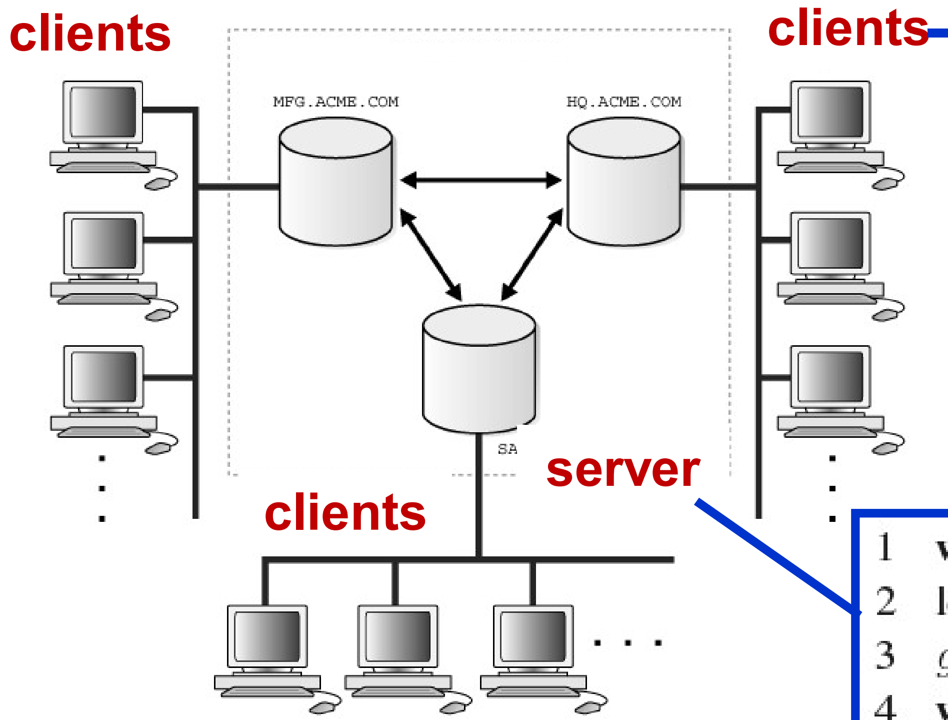
State Spaces Methods and Verification



Lars M. Kristensen
Department of Computing, Mathematics, and Physics
Western Norway University of Applied Sciences
Email: lmkr@hvl.no / WWW: home.hib.no/ansatte/lmkr

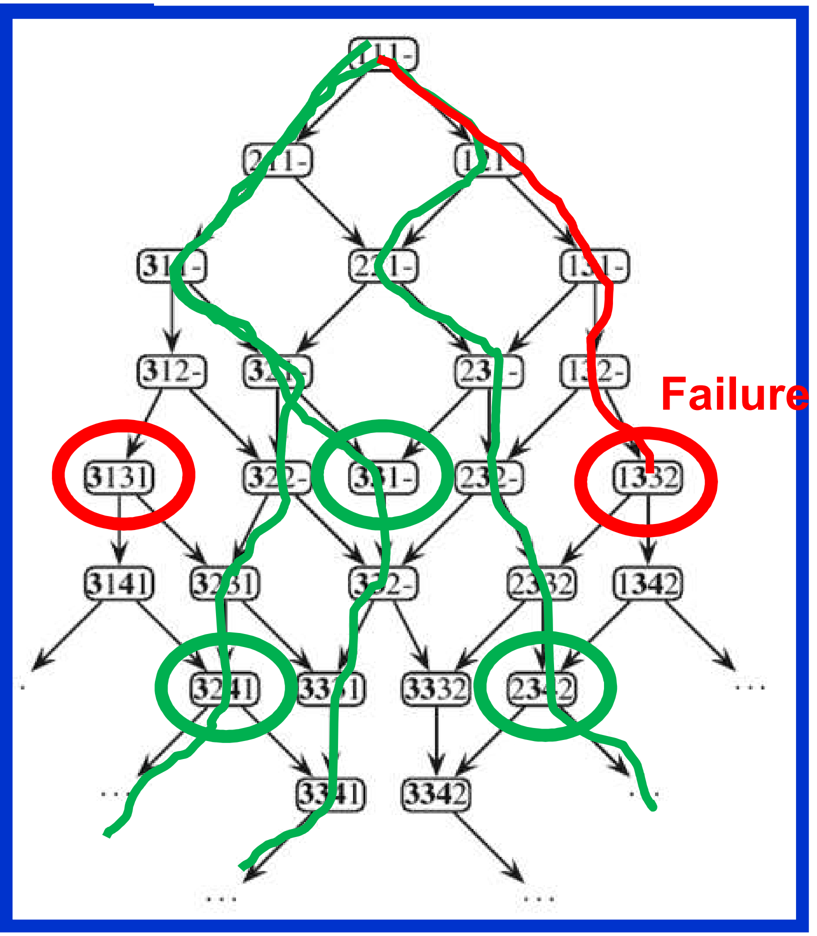
Complex Behaviour

- **A server controls access to databases which is to be used by at most one client at a time**



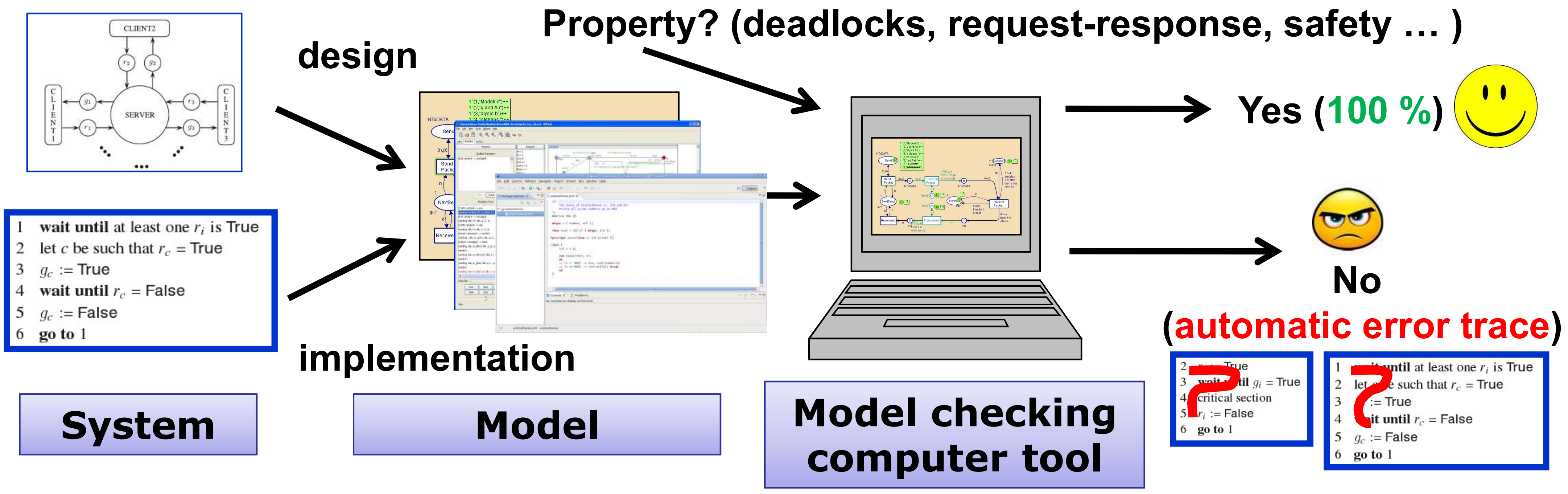
```
2  $r_i$ := True
3  wait until $g_i$
4  critical sectio
5  $r_i$ := False
6  go to 1
```

```
1  wait until at least one $r_i$
2  let $c$ be such that $r_c$ = Tr
3  $g_c$ := True
4  wait until $r_c$ = False
5  $g_c$ := False
6  go to 1
```



Verification

- **Executable models can be automatically analysed by computer tools**



```
wait until at least one $r_i$ is True
let $c$ be such that $r_c$ = True
$g_c$ := True
wait until $r_c$ = False
$g_c$ := False
go to 1
```

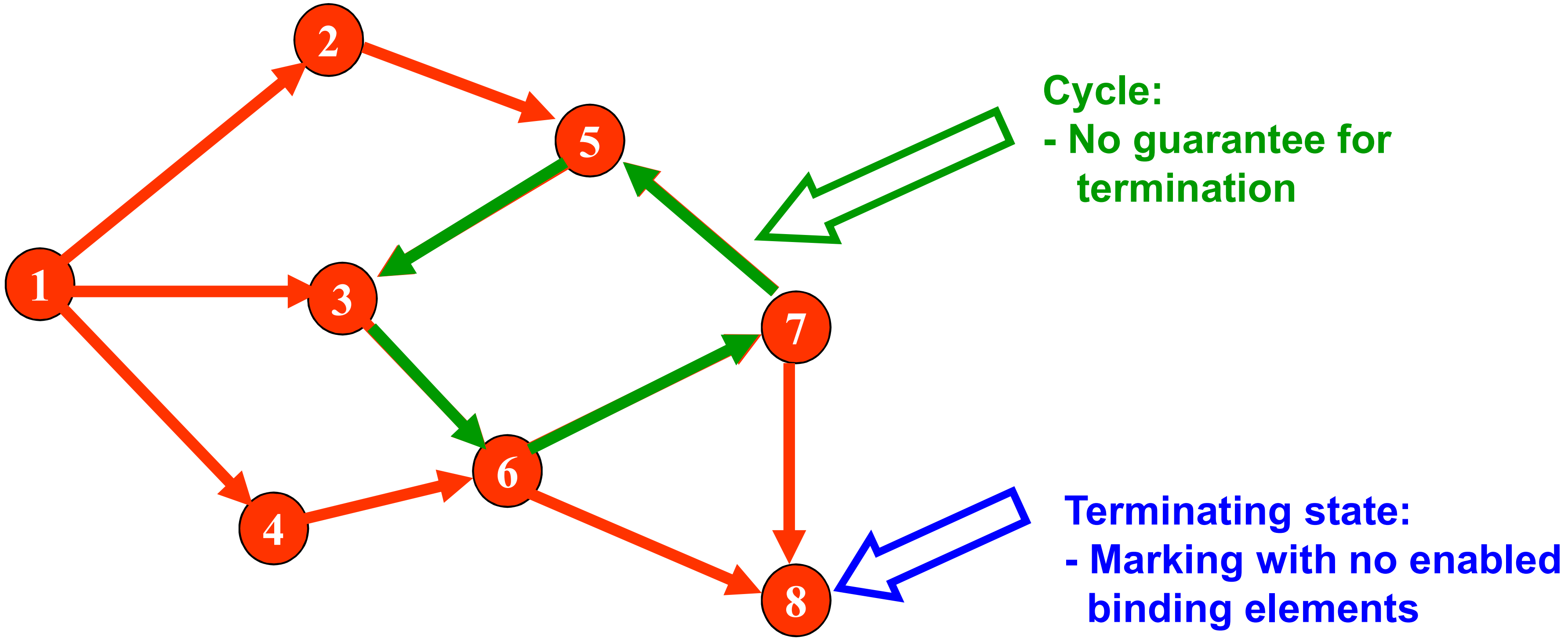
```
= True
til $g_i$ = True
critical section
$r_i$ := False
go to 1
```

```
until at least one $r_i$ is True
let e such that $r_c$ = True
:= True
it until $r_c$ = False
$g_c$ := False
go to 1
```

- **Facilitates early error-detection and verification of components and systems.**

State Spaces

- **The state space of a CPN is a directed graph with**
  - A node for each reachable marking (state).
  - An arc for each occurring binding element.
- **State spaces can be used to investigate the behavioural properties of the a model.**



State spaces

Definition 9.1. A **directed graph** with arc labels from a set L is a tuple $DG = (N, A)$, where:

1. N is a set of **nodes**.
2. $A \subseteq N \times L \times N$ is a set of **arcs**.

$\square$

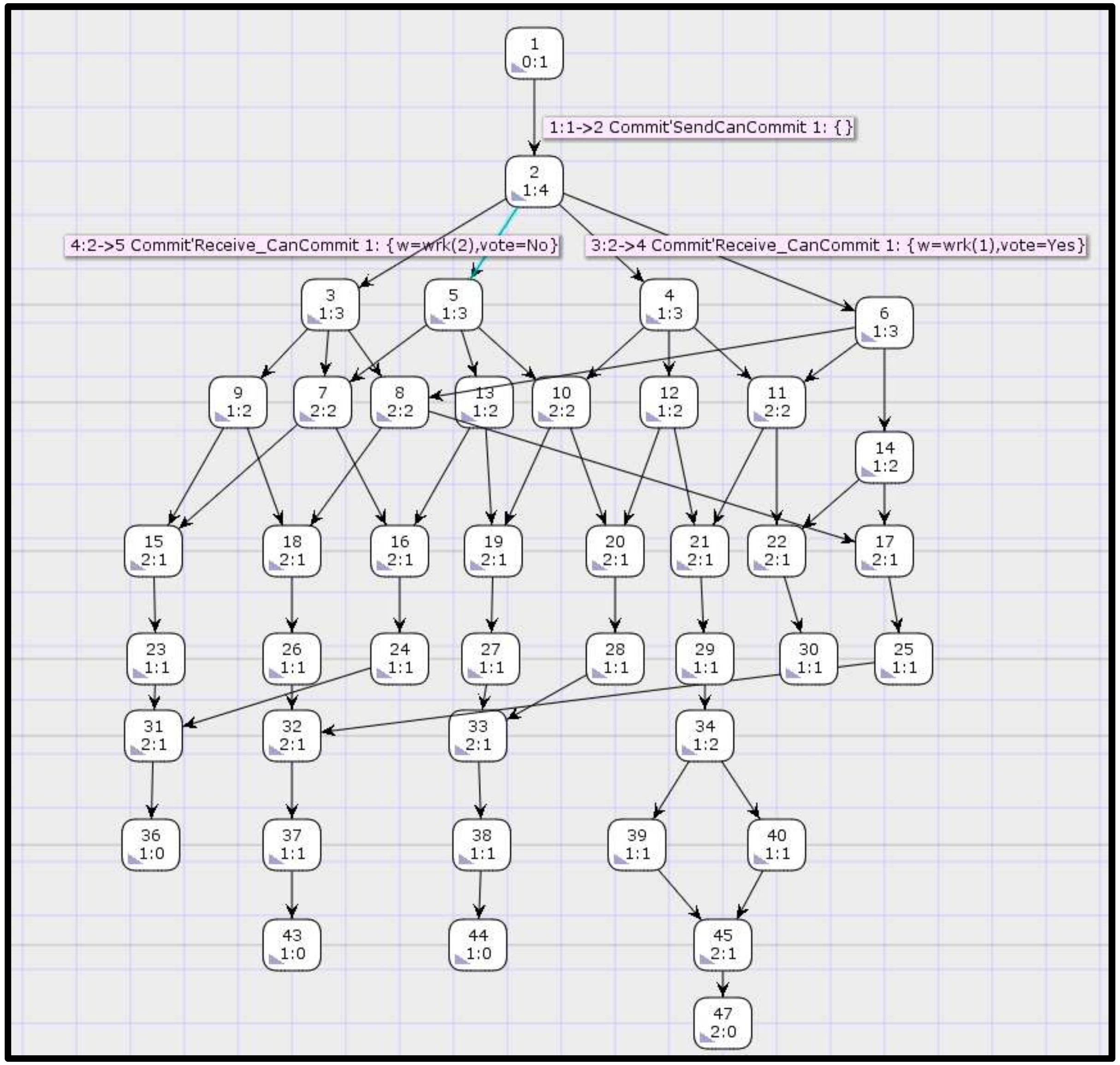
Definition 9.6. The **state space** of a Coloured Petri Net is a directed graph $SS = (N_{SS}, A_{SS})$ with arc labels from BE, where:

1. $N_{SS} = \mathscr{R}(M_0)$ is the set of **nodes**.
2. $A_{SS} = \{(M, (t,b), M') \in N_{SS} \times BE \times N_{SS} \mid M \xrightarrow{(t,b)} M'\}$ is the set of **arcs**.

SS is **finite** if and only if N_{SS} and A_{SS} are finite.

$\square$

State Space - Commit Protocol


1:1->2 Commit'SendCanCommit 1: {}
4:2->5 Commit'Receive_CanCommit 1: {w=wrk(2),vote=No}
3:2->4 Commit'Receive_CanCommit 1: {w=wrk(1),vote=Yes}

CPN Tools Demo

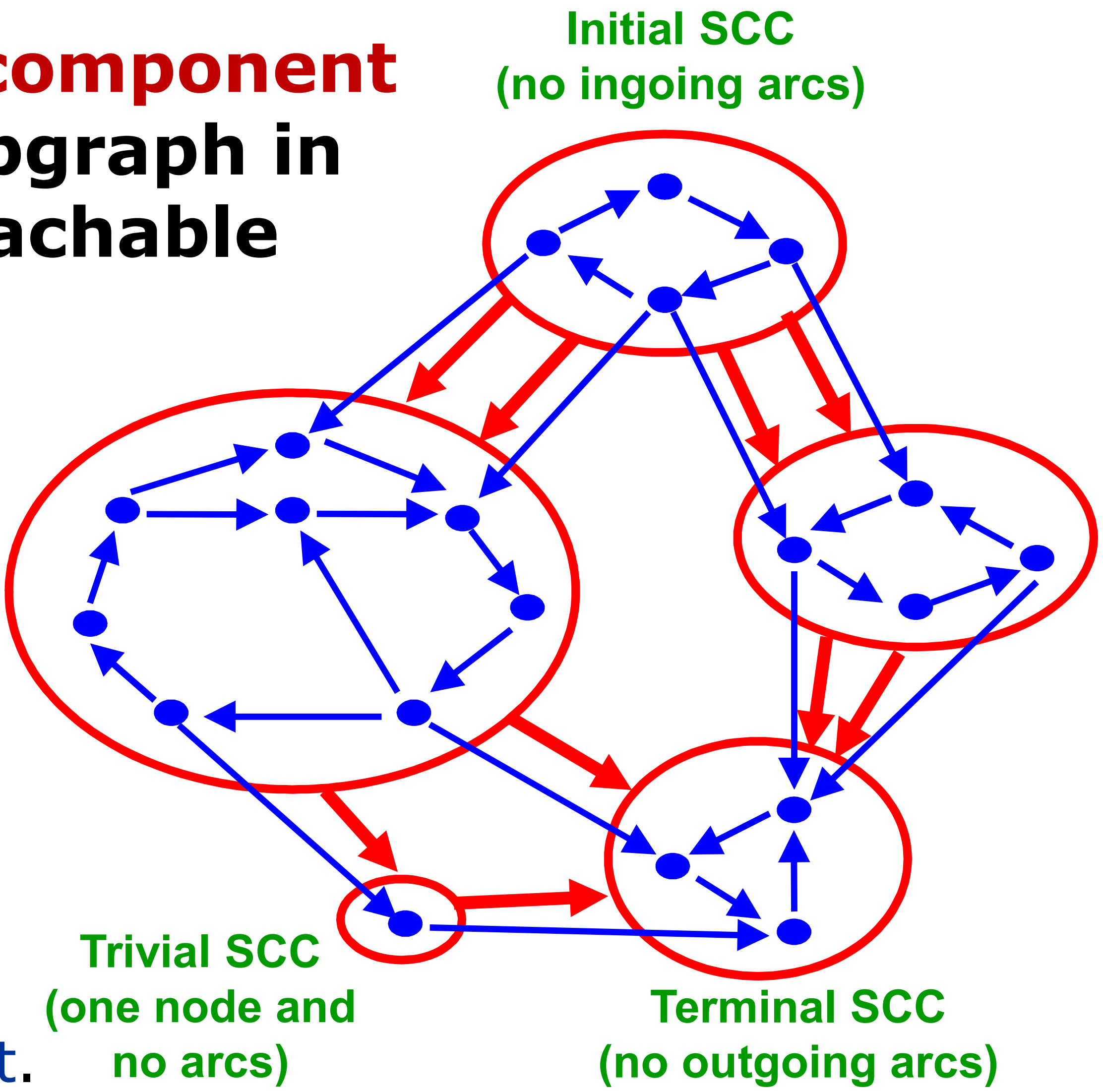
- **State space exploration of CPN models**
  - Variant of the two-phase commit protocol model
  - Calculation of state spaces
  - Stop options
  - Visualisation of fragments of state spaces

Construction Algorithm

```
$\text{NODES} \leftarrow \{M_0\}$
$\text{UNPROCESSED} \leftarrow \{M_0\}$
$\text{ARCS} \leftarrow \emptyset$
while $\text{UNPROCESSED} \neq \emptyset$ do
    Select a marking $M$ in $\text{UNPROCESSED}$
    $\text{UNPROCESSED} \leftarrow \text{UNPROCESSED} - \{M\}$
    for all binding elements $(t,b)$ such that $M \xrightarrow{(t,b)}$ do
        Calculate $M'$ such that $M \xrightarrow{(t,b)} M'$
        $\text{ARCS} \leftarrow \text{ARCS} \cup \{(M,(t,b),M')\}$
        if $M' \notin \text{NODES}$ then
            $\text{NODES} \leftarrow \text{NODES} \cup \{M'\}$
            $\text{UNPROCESSED} \leftarrow \text{UNPROCESSED} \cup \{M'\}$
        end if
    end for
end while
```

Strongly Connected Components

- **A strongly connected component (SCC) is a maximal subgraph in which all nodes are reachable from each other.**
- **SCC-graph contains**
  - A **node** for each SCC.
  - An **arc** from S_i to S_j for each state space arc from a node $n_i \in S_i$ to a node $n_j \in S_j$ $(i \neq j)$.
- The SCC-graph is acyclic.
- The SCCs are mutually disjoint.
- Each node is in exactly one SCC.



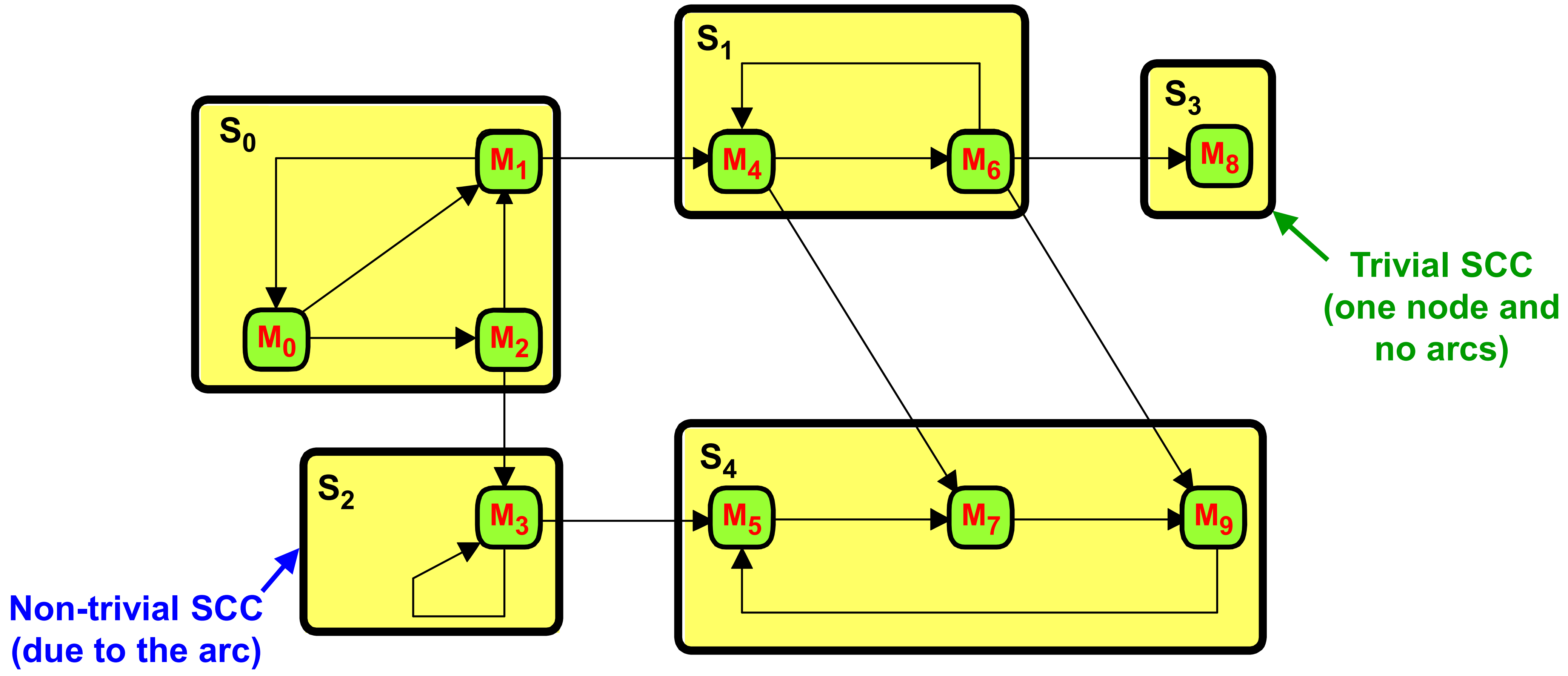
Example: State space

- **10 nodes and 16 arcs**

$$\begin{tikzcd}
& M_1 \arrow[dl, bend right] \arrow[r] & M_4 \arrow[r] \arrow[ddr] & M_6 \arrow[r] \arrow[ddr] \arrow[l, bend right] & M_8 \\
M_0 \arrow[ur] \arrow[r] & M_2 \arrow[u] \arrow[d] & & & \\
& M_3 \arrow[r] \arrow[loop below] & M_5 \arrow[r] & M_7 \arrow[r] & M_9 \arrow[ll, bend left]
\end{tikzcd}$$

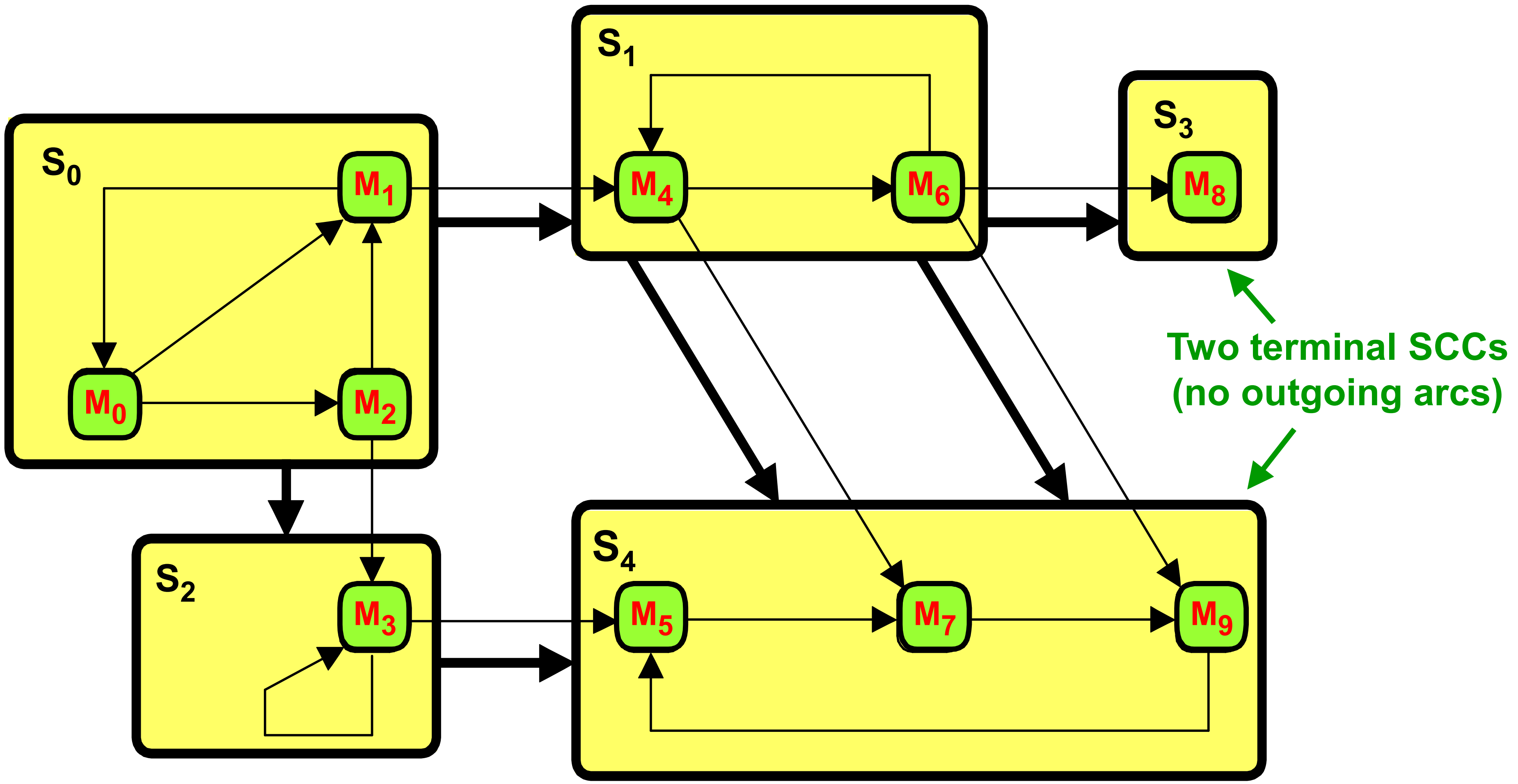
SCC Example

- **Five SCCs**



SCC graph

- 5 nodes (SCCs) and 6 arcs



State Space Analysis

- **State spaces may be very large and we need computer tools to construct and analyse them.**
- **Analysis of the state space typically starts with the generation of the state space report**
  - Textual report that can be generated fully automatically
  - Contains information about standard behavioural properties of the CPN model
  - Useful for locating errors or increase confidence in the correctness of the system

State Space Report - Content

- **The state space report contains information about standard behavioural properties**
  - Size of the state space and the time used to generate it.
  - Bounds for the number of tokens on each place and information about the possible token colours.
  - Home markings.
  - Dead markings.
  - Dead and live transitions.
  - Fairness properties for transitions.
- **The state space report can be generated for any CPN models with a finite state space.**

CPN Tools Demo

- **State space analysis of CPN models**
  - Generation of the strongly connected components
  - Generation and inspection of the state space report

Size and Exploration Time

State Space Statistics

State Space		Scc Graph	
Nodes:	23,497	Nodes:	23,497
Arcs:	52,192	Arcs:	52,192
Secs:	76	Secs:	1
Status:	Full		

- **The state space is Full: it contains all reachable markings.**
- **The SCC graph and the state space has the same number of nodes and arcs: no cycles.**

Integer Bounds

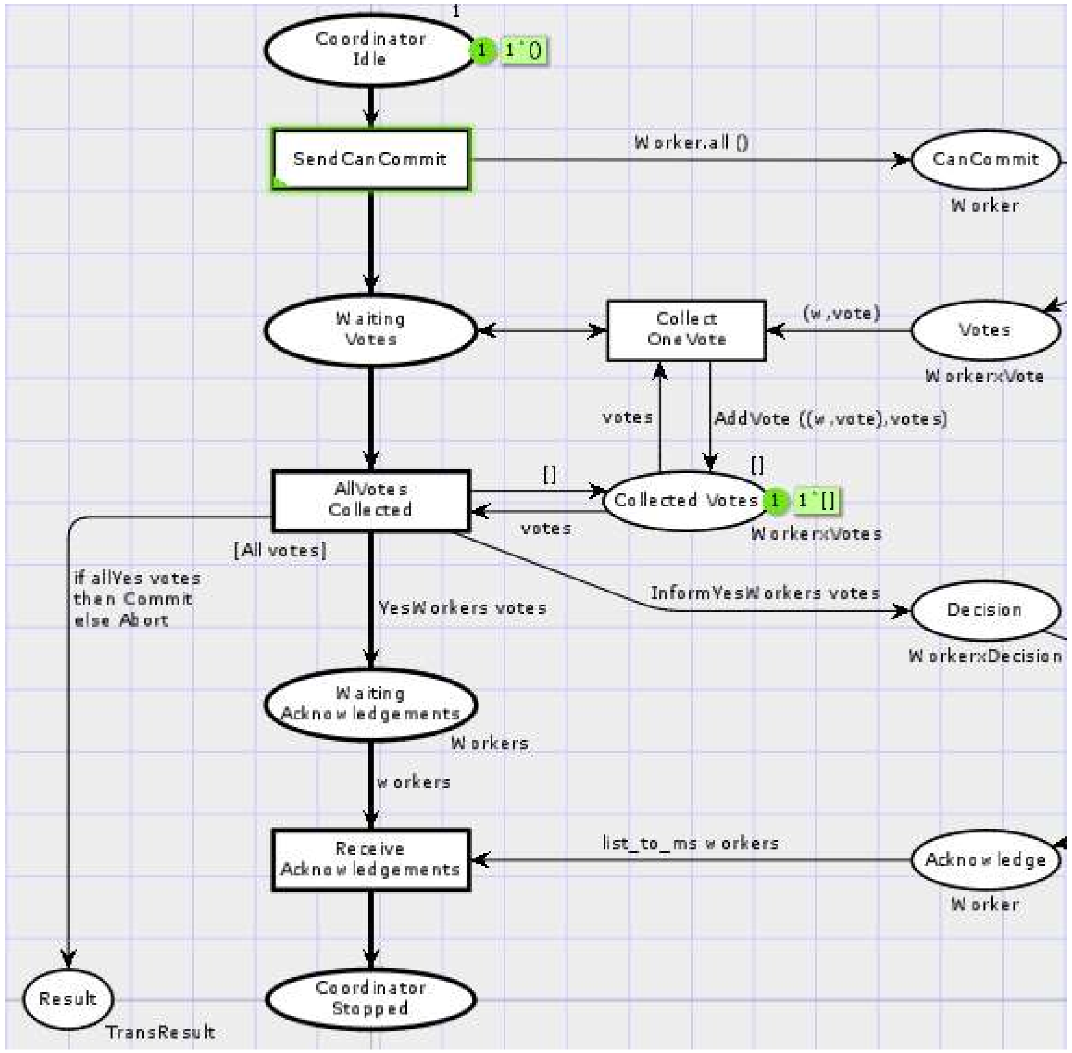
- **The integer bounds considers the number of tokens on a place**
  - The best upper integer bound for a place is the maximal number of tokens on the place in a reachable marking.
  - The best lower integer bound for a place is the minimal number of tokens on the place in a reachable marking.

- **Places with an upper integer bound are bounded.**
- **Places with no upper integer bound are unbounded.**

Integer Bounds - Coordinator

Best Integers Bounds

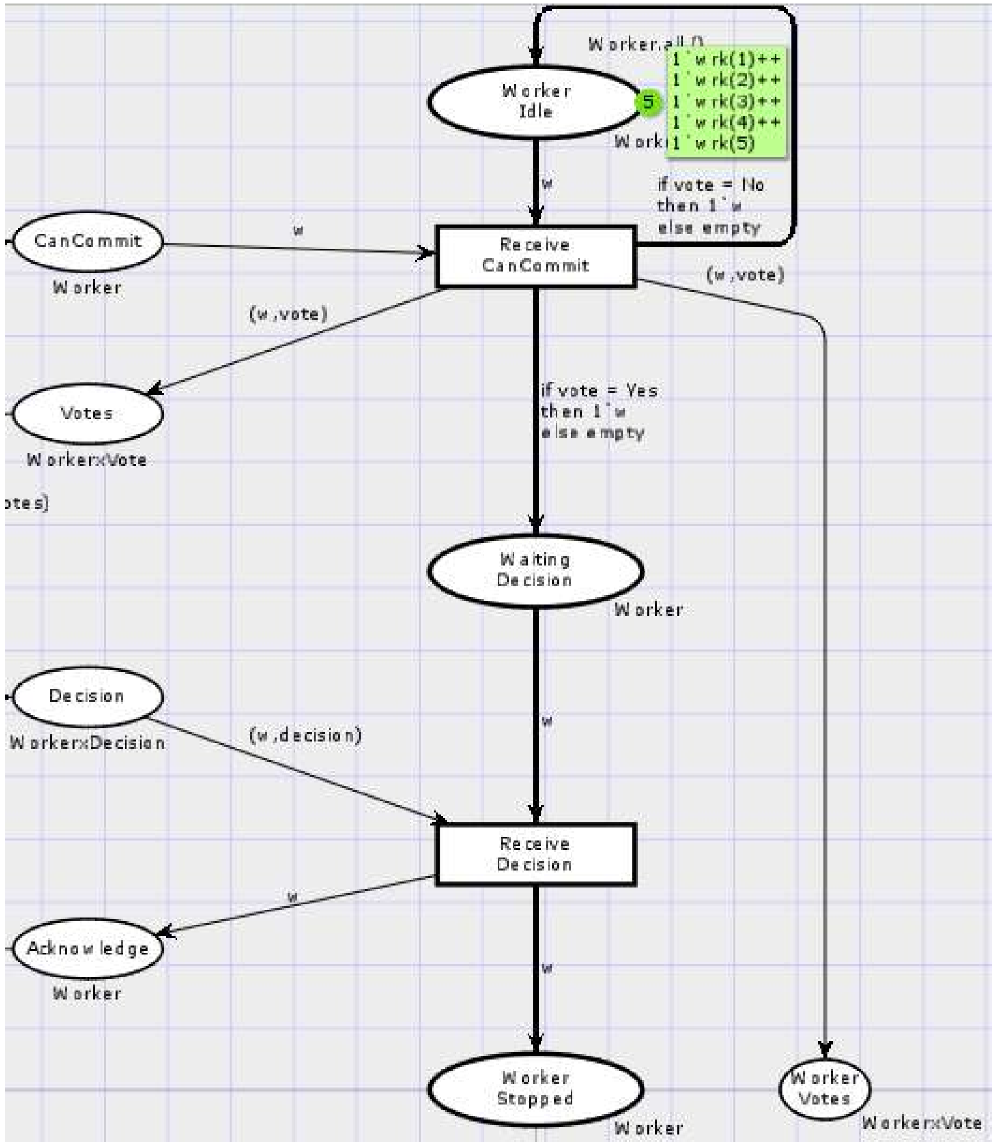
	Upper	Lower
CoordinatorIdle	1	0
CanCommit	5	0
WaitingVotes	1	0
CollectedVotes	1	1
Decision	5	0
Waiting Acknowledgements	1	0
CoordinatorStopped	1	0
Result	1	0



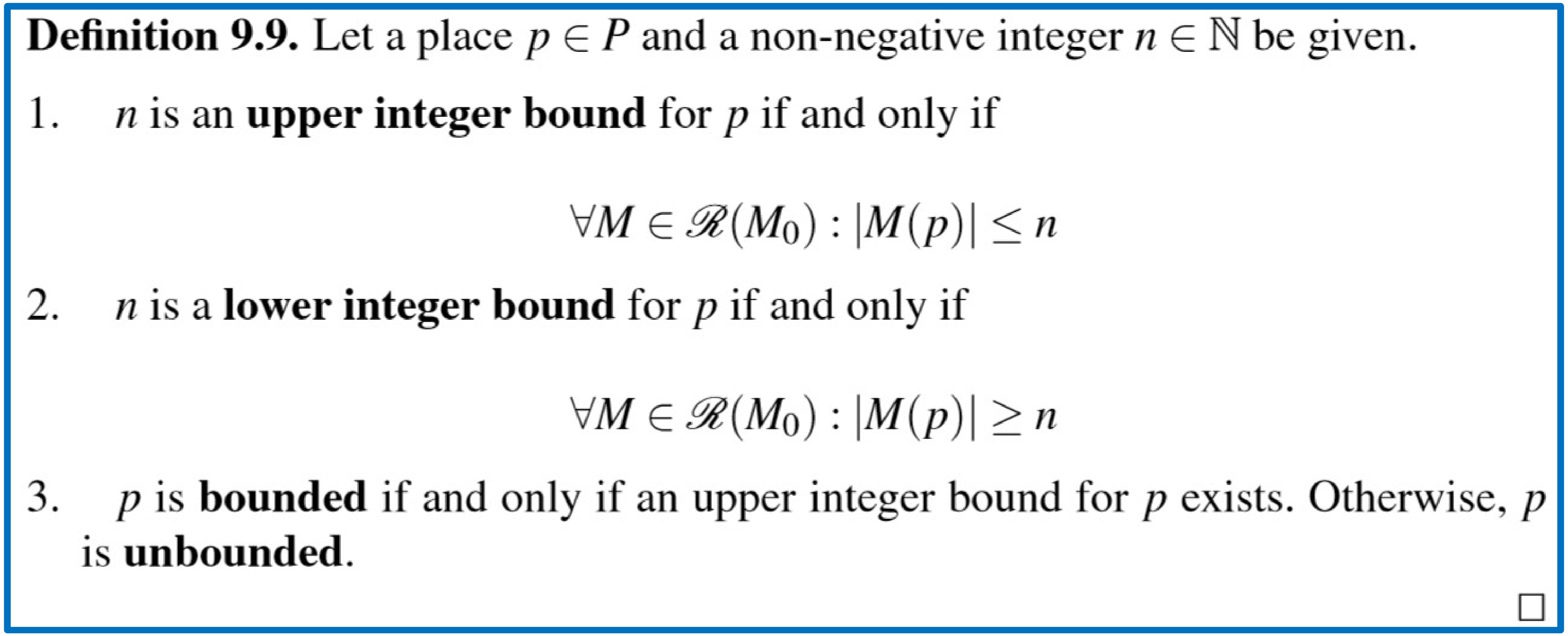
Integer Bounds - Workers

Best Integers Bounds

	Upper	Lower
WorkerIdle	5	0
Votes	5	0
WaitingDecision	5	0
Acknowledge	5	0
WorkerStopped	5	0
WaitingDecision	5	0



Definition - Integer bounds



Definition 9.9. Let a place $p \in P$ and a non-negative integer $n \in \mathbb{N}$ be given.

1. n is an **upper integer bound** for p if and only if

$$\forall M \in \mathscr{R}(M_0) : |M(p)| \leq n$$

2. n is a **lower integer bound** for p if and only if

$$\forall M \in \mathscr{R}(M_0) : |M(p)| \geq n$$

3. p is **bounded** if and only if an upper integer bound for p exists. Otherwise, p is **unbounded**.

$\square$

Multiset Bounds

- Integer bounds considers the number of tokens
  - ignores the token colours.
- Multiset bounds considers the possible token colours.

- The best upper multiset bound for a place is a multiset over the colour set of the place
  - the coefficient for a colour c is the maximal number of occurrences of tokens with colour c in a reachable marking

- The best lower multiset bound for a place is a multiset over the colour set of the place
  - the coefficient for a colour c is the minimal number of occurrences of tokens with colour c in a reachable marking

Upper Multiset - Coordinator

```
Best Upper Multiset Bounds

CoordinatorIdle     1`()

CanCommit           1`wrk(1) ++ 1`wrk(2) ++ 1`wrk(3) ++
                    1`wrk(4) ++ 1`wrk(5)

Decision            1`(wrk(1),abort) ++ 1`(wrk(1),commit) ++
                    1`(wrk(2),abort) ++ 1`(wrk(2),commit) ++
                    1`(wrk(3),abort) ++ 1`(wrk(3),commit) ++
                    1`(wrk(4),abort) ++ 1`(wrk(4),commit) ++
                    1`(wrk(5),abort) ++ 1`(wrk(5),commit)

CoordinatorStopped  1`()

Result              1`Abort++ 1`Commit
```

Upper Multiset - Workers

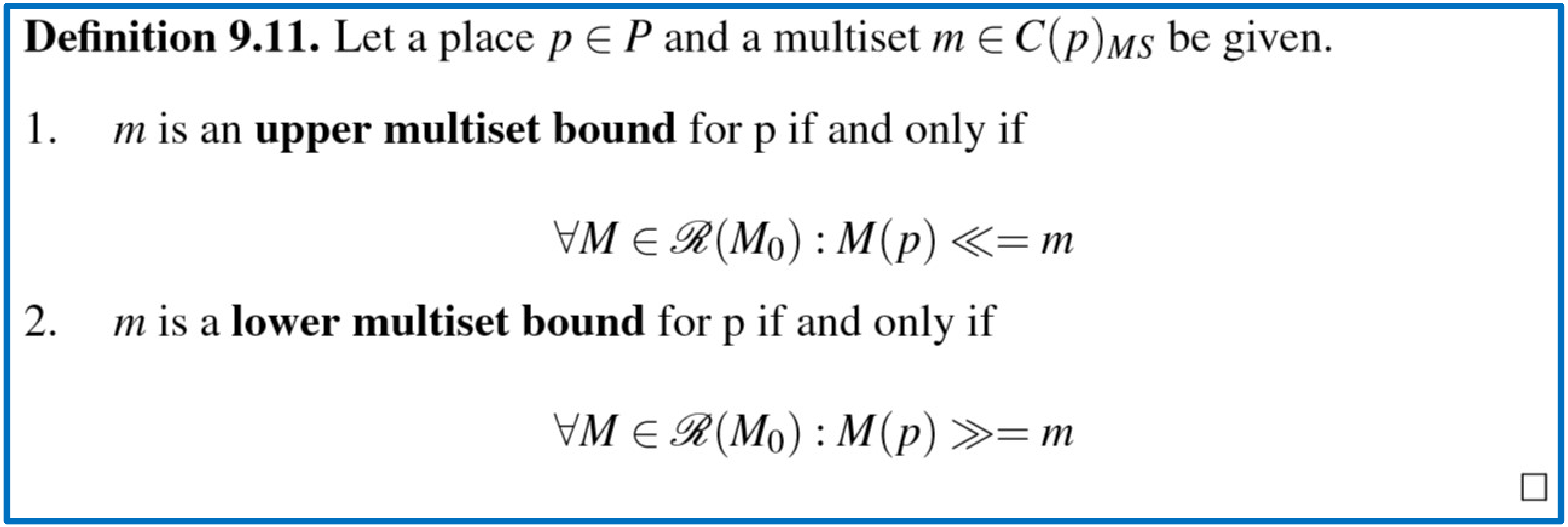
```
Best Upper Multiset Bounds

   WorkersIdle, WaitingDecision, WorkerStopped, Acknowledge

       1`wrk(1) ++ 1`wrk(2) ++ 1`wrk(3) ++ 1`wrk(4) ++ 1`wrk(5)

   Votes, WorkerVotes

       1`(wrk(1),Yes) ++ 1`(wrk(1),No) ++
       1`(wrk(2),Yes) ++ 1`(wrk(2),No) ++
       1`(wrk(3),Yes) ++ 1`(wrk(3),No) ++
       1`(wrk(4),Yes) ++ 1`(wrk(4),No) ++
       1`(wrk(5),Yes) ++ 1`(wrk(5),No)
```

Lower Multiset Bounds

```
Best Lower Multiset Bounds
CoordinatorIdle            empty          WorkerIdle        empty
CanCommit                  empty          Votes             empty
WaitingVotes               empty          WaitingDecision   empty
CollectedVotes             empty          Acknowledge       empty
Decision                   empty          WorkerVotes        empty
WaitingAcknowledgements    empty          WorkerStopped     empty
CoordinatorStopped         empty
Result                     empty
```

- **All are equal to the empty multi-set: all minimal coefficients are zero – no token is always present.**

Definition: multi-set bounds



Definition 9.11. Let a place $p \in P$ and a multiset $m \in C(p)_{MS}$ be given.

1. m is an **upper multiset bound** for p if and only if

$$\forall M \in \mathscr{R}(M_0) : M(p) \ll= m$$

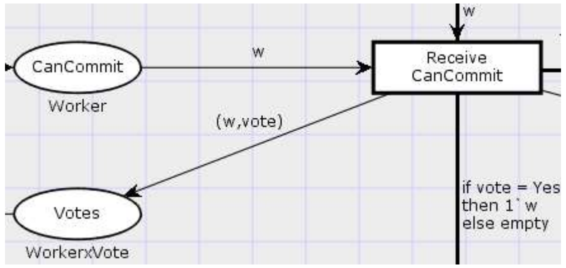
2. m is a **lower multiset bound** for p if and only if

$$\forall M \in \mathscr{R}(M_0) : M(p) \gg= m$$

□

Integer and Multiset bounds

- **The two kinds of bounds supplement each other and provides different kinds of information.**
- **Integer bounds**



```
Votes                    5        0
```

Tells us that Votes has at most five tokens and as little as zero tokens, but gives us no information about the token colours.

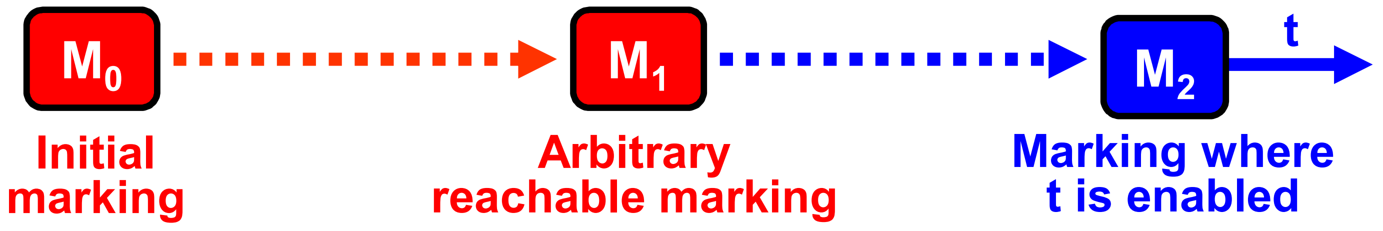
- **Multi-set bounds**

```
1`(wrk(1),Yes)++1`(wrk(1),No)++1`(wrk(2),Yes)++1`(wrk(2),No)++
1`(wrk(3),Yes)++1`(wrk(3),No)++1`(wrk(4),Yes)++1`(wrk(4),No)++
1`(wrk(5),Yes)++1`(wrk(5),No)
```

Tells us that Votes can have ten different tokens and as little as zero of each, but not how many can be present simultaneously.

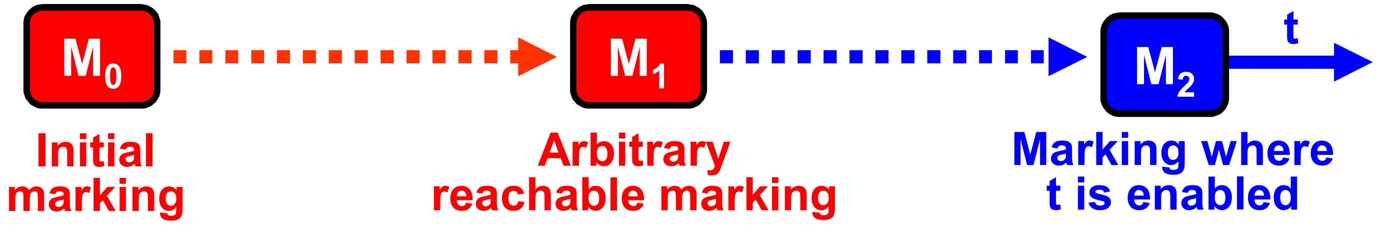
Liveness Properties

- A **marking M is dead** if M has no enabled transitions.
- A **transition t is dead** if t never can occur – i.e. is disabled in all reachable markings.
- A **transition t is live if** we from any reachable marking can reach a state where t is enabled



- Liveness tells that it is possible for t to occur.
- No guarantee that this will happen.

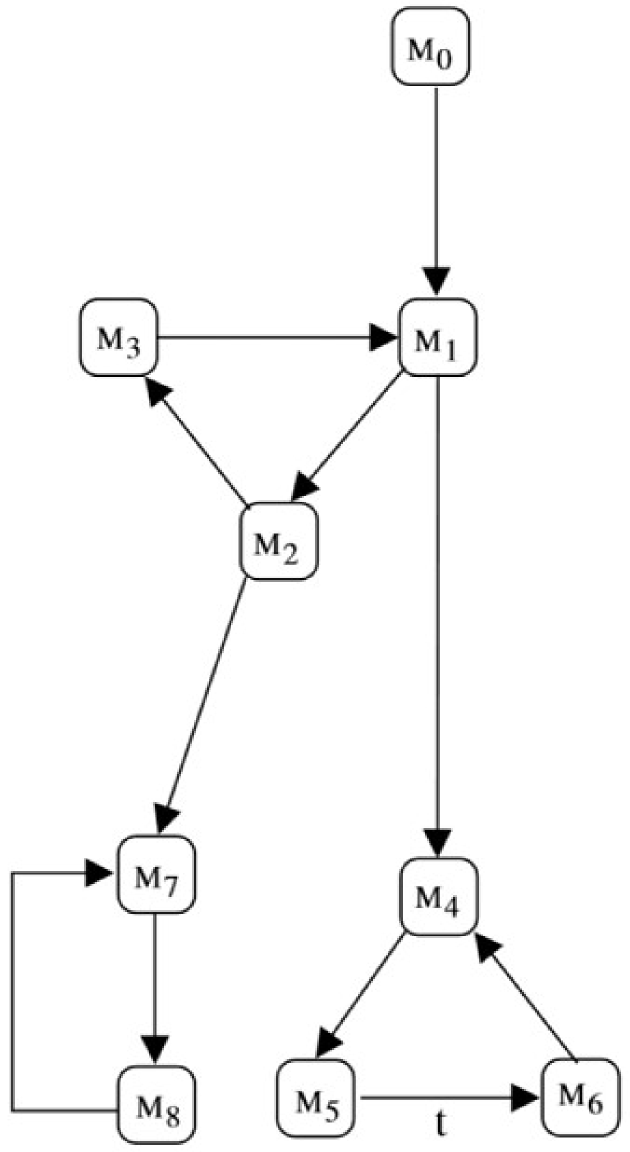
Liveness is a Strong Property

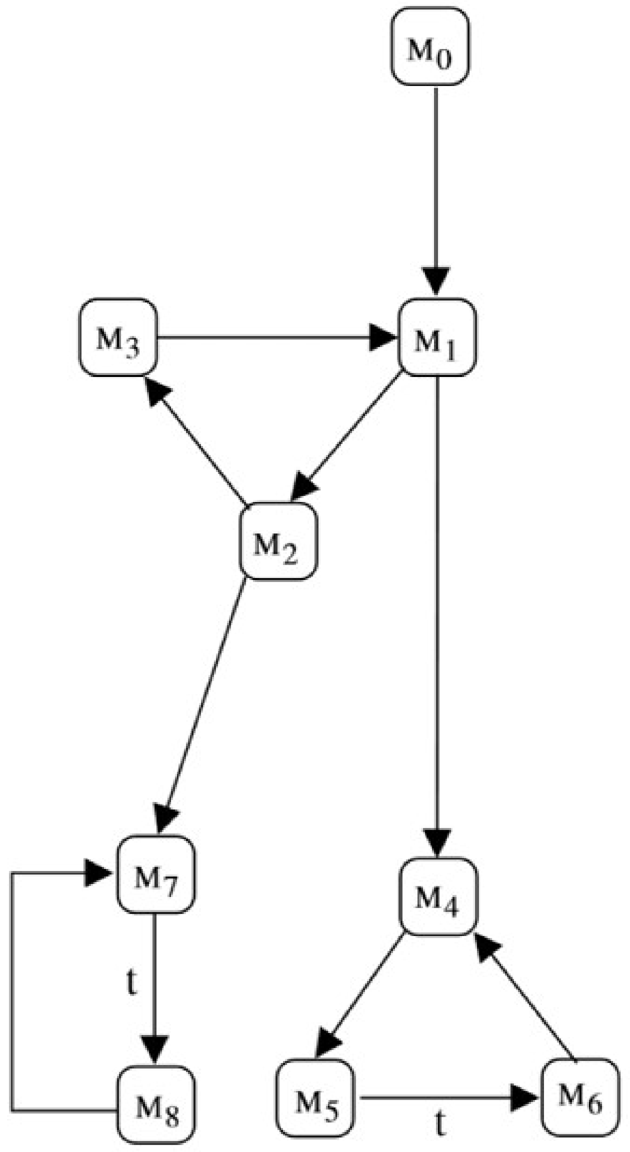


- **If the live transition t occurs in the marking M_2 we reach another reachable marking.**
- **We can use the new marking as M_1 and hence t is able to occur once more, and so on.**
- **There exists infinite occurrence sequences in which t occurs infinitely many times.**

- **It is possible to be non-dead without being live.**

Example: Liveness





Definition - Liveness Properties

Definition 9.19. Let a transition $t \in T$ and a marking M be given.

1. M is a **dead marking** if and only if

$$\forall t \in T : \neg\, (M \xrightarrow{t})$$

2. t is **dead in** M_0 if and only if

$$\forall M \in \mathscr{R}(M_0) : \neg\, (M \xrightarrow{t})$$

3. t is **live in** M_0 if and only if

$$\forall M \in \mathscr{R}(M_0)\ \exists M' \in \mathscr{R}(M) : M' \xrightarrow{t}$$

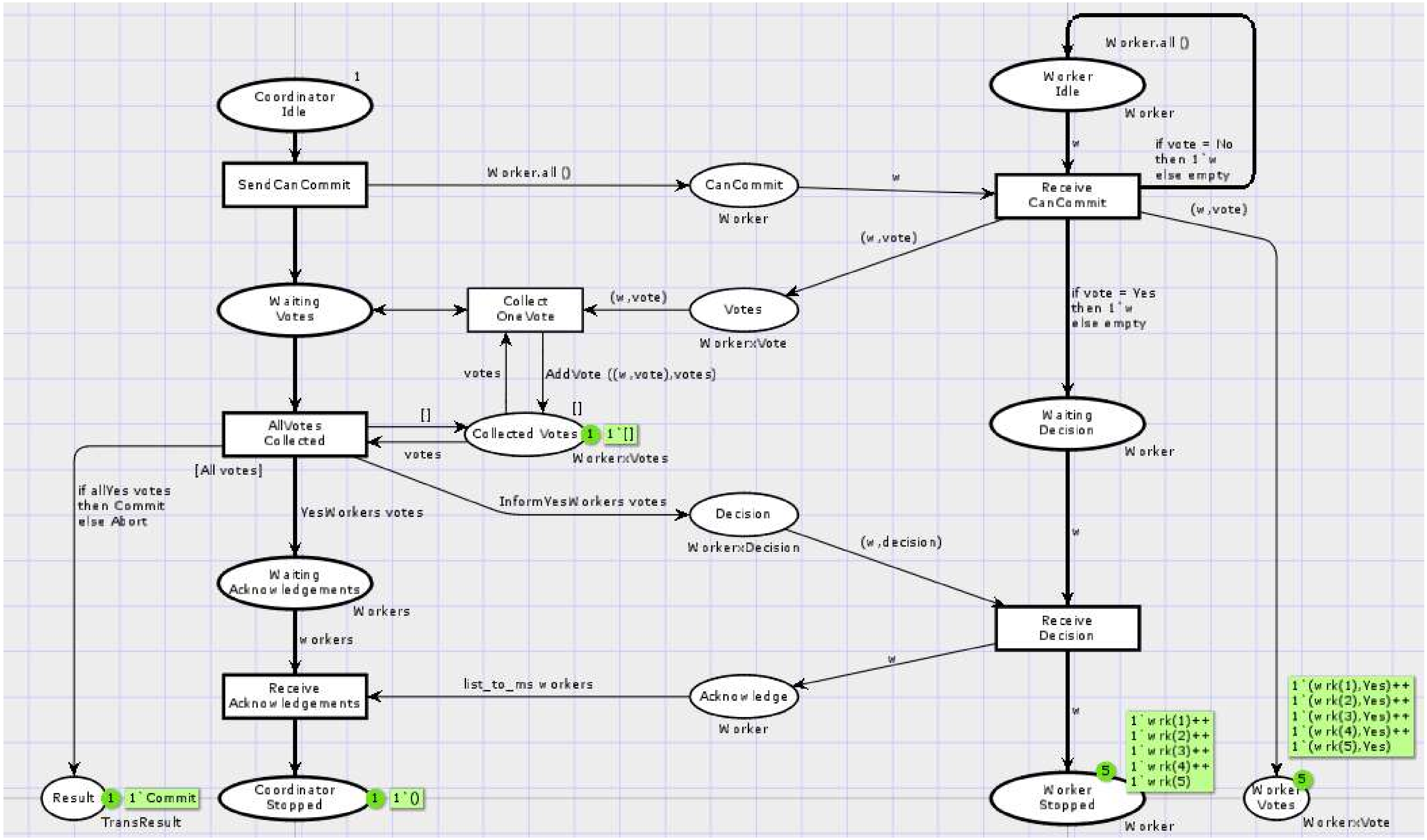
$\square$

Liveness Properties

```
Liveness Properties

   Dead Markings:  32 [23497,23376,23375,23374,23373,...]
   Dead Transitions: None
   Live Transitions: None
```

- **There are 32 dead markings represented by the nodes numbered: 23497, 23376,23375,…**
- **There are no dead transitions.**
- **There are no live transitions**
  - Consequence of the existence of a dead markings

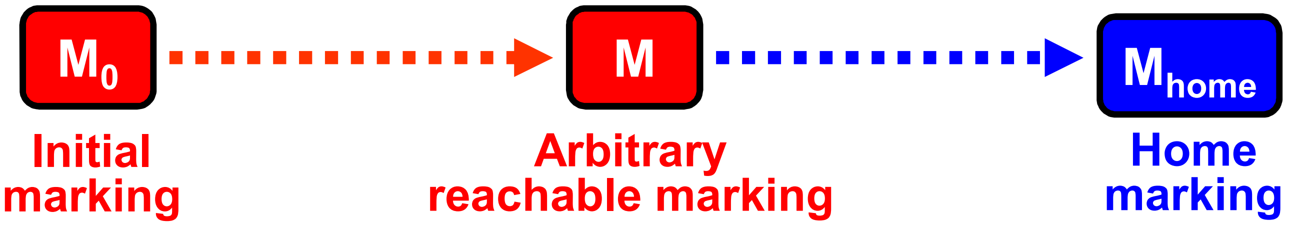
State 23497



Coordinator Idle
1
Send Can Commit
Worker.all ()
Can Commit
Worker
w
Worker.all ()
Worker Idle
Worker
w
if vote = No then 1`w else empty
Receive Can Commit
(w,vote)
(w,vote)
Waiting Votes
Collect One Vote
(w,vote)
Votes
WorkerxVote
if vote = Yes then 1`w else empty
votes
AddVote ((w,vote),votes)
[]
[]
All Votes Collected
Collected Votes
1
1`[]
WorkerxVotes
votes
Waiting Decision
Worker
[All votes]
if allYes votes then Commit else Abort
YesWorkers votes
InformYesWorkers votes
Decision
WorkerxDecision
(w,decision)
w
Waiting Acknowledgements
Workers
workers
Receive Decision
w
Receive Acknowledgements
list_to_ms workers
Acknowledge
Worker
w
1`wrk(1)++
1`wrk(2)++
1`wrk(3)++
1`wrk(4)++
1`wrk(5)
1`(wrk(1),Yes)++
1`(wrk(2),Yes)++
1`(wrk(3),Yes)++
1`(wrk(4),Yes)++
1`(wrk(5),Yes)
Result
1
1`Commit
TransResult
Coordinator Stopped
1
1`()
5
Worker Stopped
Worker
5
Worker Votes
WorkerxVote

Home Properties

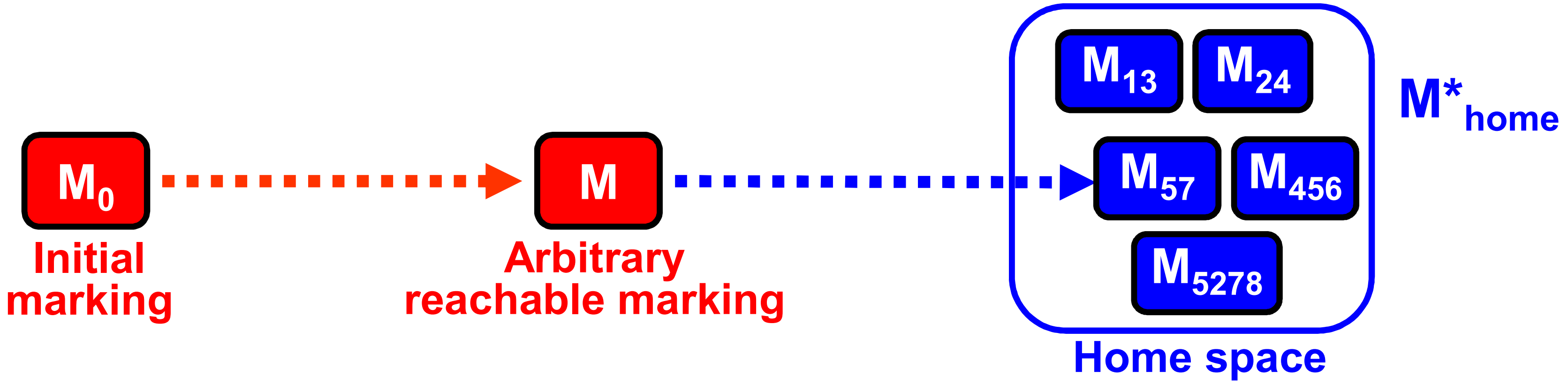
- A home marking is a marking M_{home} which can be reached from any reachable marking



- Impossible to have an occurrence sequence which cannot be extended to reach M_{home}.
  - The home property tells that it is possible to reach M_{home}.
  - No guarantee that this will happen.

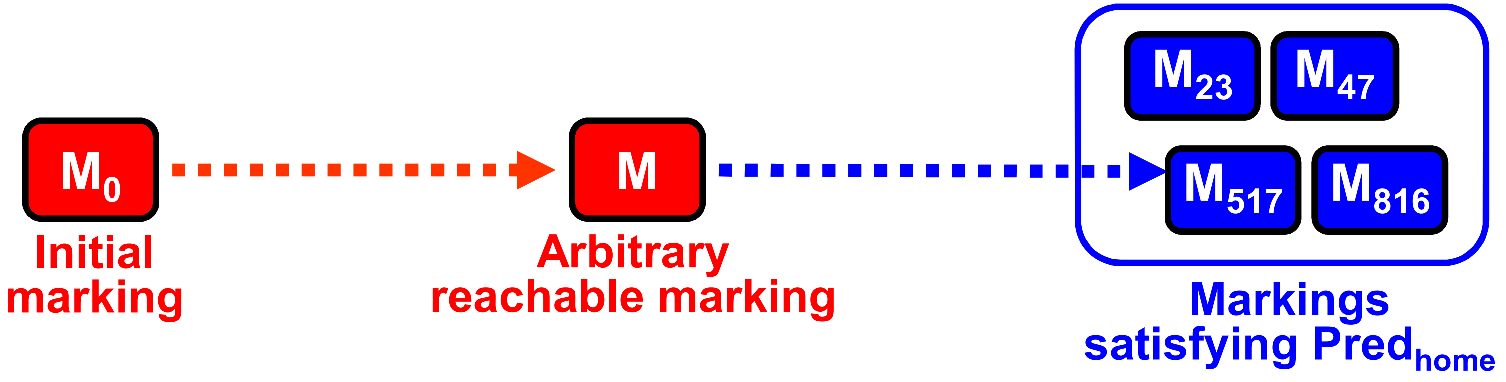
Home Space

- A **home space** is a set of markings M^*_{home} such that at least one marking in M^*_{home} can be reached from any reachable marking



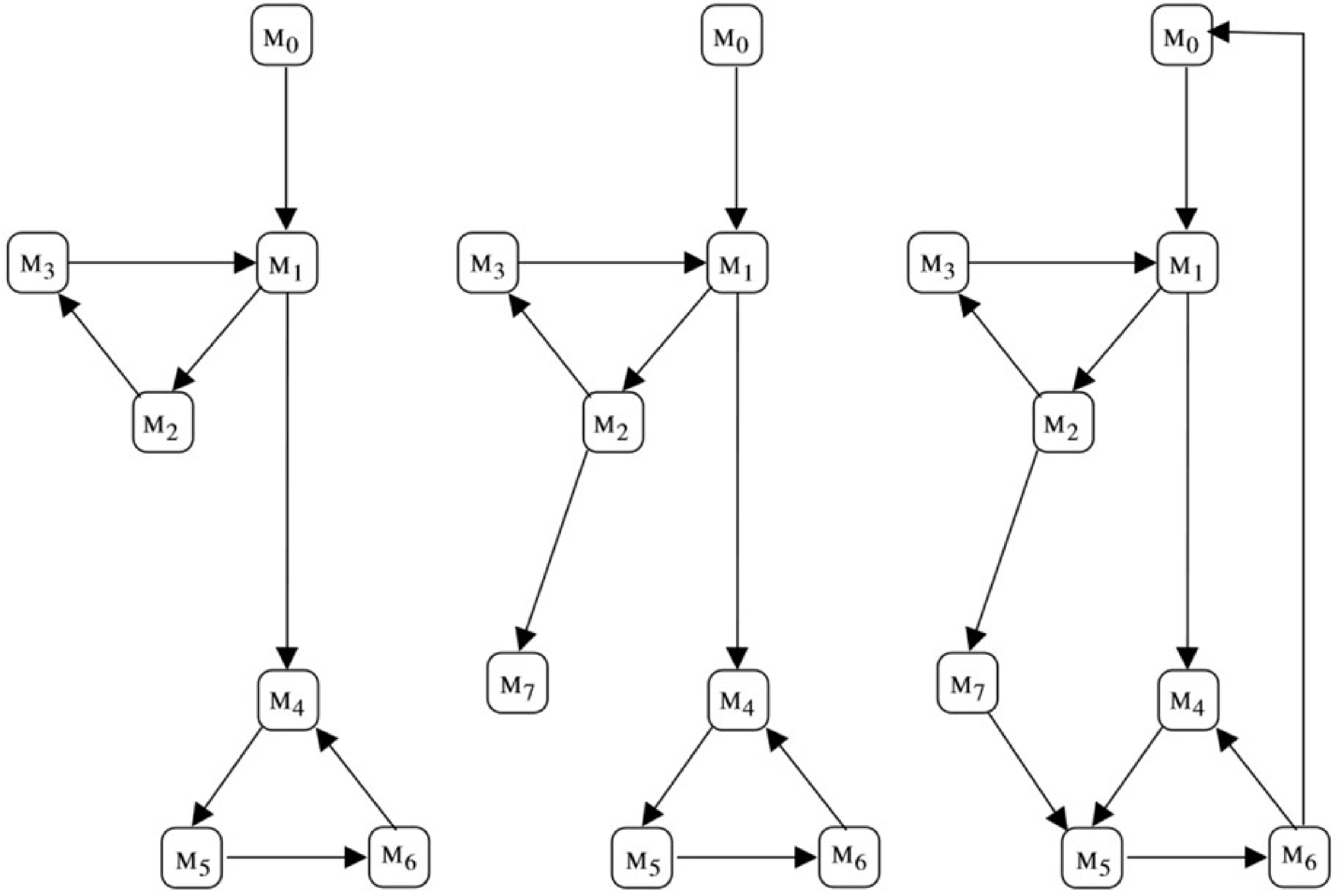
Home Predicate

- A **home predicate** is a predicate on markings $\text{Pred}_{\text{home}}$ such that at least one marking satisfying $\text{Pred}_{\text{home}}$ can be reached from any reachable marking

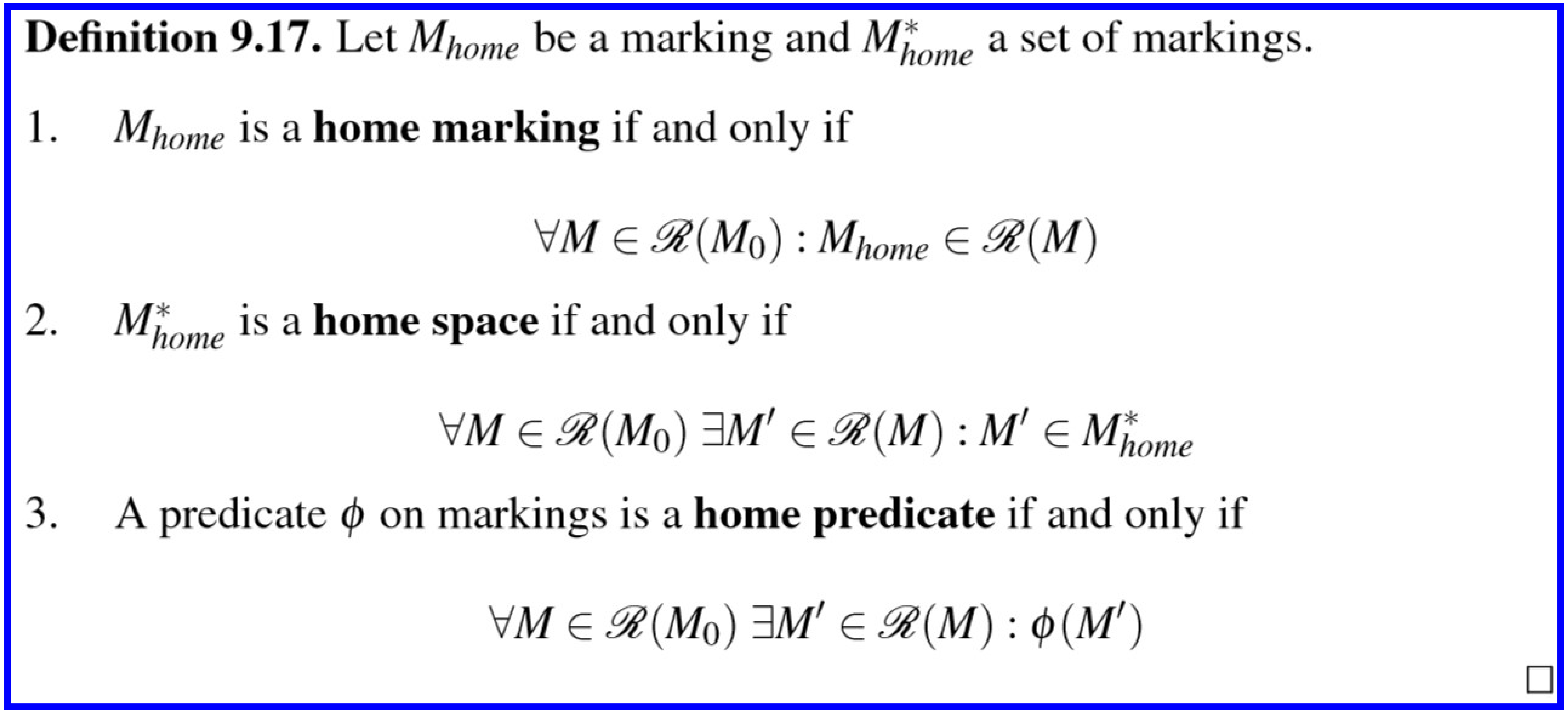


M_0
Initial marking
M
Arbitrary reachable marking
M_{23}
M_{47}
M_{517}
M_{816}
Markings satisfying $\text{Pred}_{\text{home}}$

Examples: home properties



Definition: home properties



Definition 9.17. Let M_{home} be a marking and M^*_{home} a set of markings.

1. M_{home} is a **home marking** if and only if

$$\forall M \in \mathscr{R}(M_0) : M_{home} \in \mathscr{R}(M)$$

2. M^*_{home} is a **home space** if and only if

$$\forall M \in \mathscr{R}(M_0)\ \exists M' \in \mathscr{R}(M) : M' \in M^*_{home}$$

3. A predicate ϕ on markings is a **home predicate** if and only if

$$\forall M \in \mathscr{R}(M_0)\ \exists M' \in \mathscr{R}(M) : \phi(M')$$

$\square$

Home Markings

- There is are no home markings in the two-phase commit protocol CPN model

Home Properties

    Home Markings: None

- A consequence of having more than one (32) dead markings.

Query Functions

- The state space report contains information about standard behavioural properties
- Non-standard behavioural properties can also be investigated by means of queries written in the Standard ML language:
  - provide arguments to a predefined query function – e.g. to check whether a set of markings constitute a home space.
  - write your own query functions using the Standard ML programming language.

Example – can we commit?

- **Check whether one of the dead markings corresponds to a commit state.**
- **Predicate checking the marking of the Result place in the coordinator**

```
fun hasCommit n = Mark.Commit'Result 1 n == 1`Commit
```

- **Check whether we have a commit in one of the dead markings**

```
List.exists hasCommit (ListDeadMarkings ());
```

Consistent Termination States

- The protocol commits iff all workers votes yes

```
fun correctTermination n =
  let
      val votes = Mark.Commit'Worker_Votes 1 n
      val votecount = size (Mark.Commit'Worker_Votes 1 n)
      val yescount = List.length (yesVotes votes)
      val result = ms_to_col (Mark.Commit'Result 1 n)
  in
      (votecount = W) andalso
      (
       ((result = Commit) andalso (yescount = W)) orelse
       ((result = Abort) andalso (yescount < W))
      )
  end
```

```
List.all correctTermination (ListDeadMarkings ())
```

Ability to Correctly Terminate?

- **Check if the terminating states of the protocol are consistent with the commit criteria.**
- **Check that the dead markings constitute a home space**

```
HomeSpace (ListDeadMarkings ())
```

- **The SCC-graph and the state space are equal for the two-phase commit protocol**
  - **One of the dead markings will always be reached**

- **Conclusion: the protocol eventually terminates and in a state consistent with the commit criteria.**

Property Violations

- **If a property is violated then state space methods can provide error-traces.**
- **Example: two-phase commit with wrong implementation of allYes predicate**

```
fun allYes votes = (List.length (yesVotes votes) = W-1)
```

- **Results of running the queries**

```
allCorrectTermination ()
```

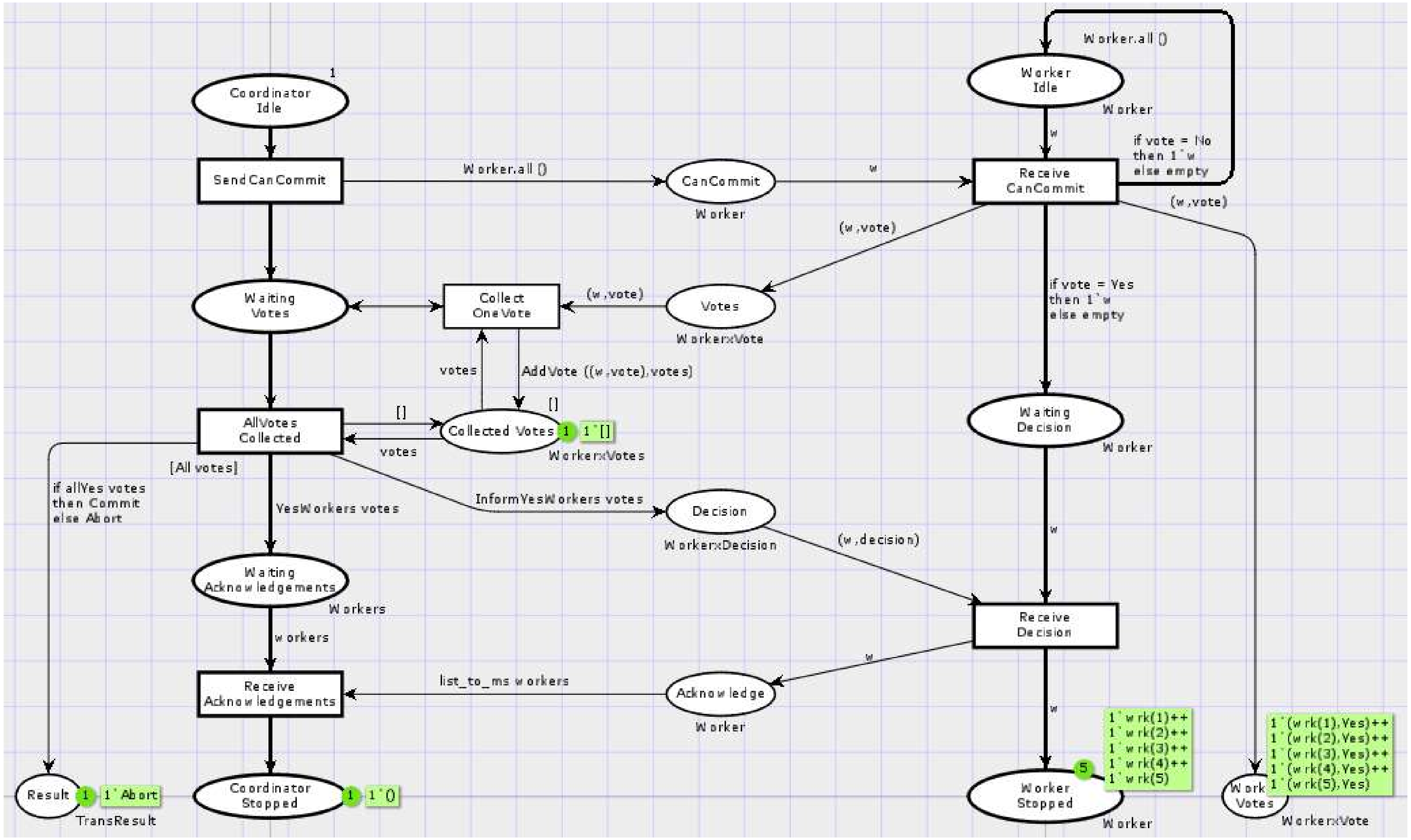
```
val it = false : bool
```

- **Find a dead marking with inconsistent termination**

```
List.find (not o correctTermination) (ListDeadMarkings ())
```

```
val it = SOME 23497 : Node option
```

State 23497



Worker.all ()
Worker Idle
Worker
1
Coordinator Idle
w
if vote = No then 1`w else empty
Send Can Commit
Worker.all ()
CanCommit
Worker
w
Receive CanCommit
(w,vote)
(w,vote)
Waiting Votes
Collect One Vote
(w,vote)
Votes
WorkerxVote
if vote = Yes then 1`w else empty
votes
AddVote ((w,vote),votes)
[]
[]
AllVotes Collected
Collected Votes
1
1`[]
WorkerxVotes
votes
Waiting Decision
Worker
[All votes]
if allYes votes then Commit else Abort
YesWorkers votes
InformYesWorkers votes
Decision
WorkerxDecision
(w,decision)
w
Waiting Acknowledgements
Workers
workers
Receive Decision
w
Receive Acknowledgements
list_to_ms workers
Acknowledge
Worker
w
1`wrk(1)++
1`wrk(2)++
1`wrk(3)++
1`wrk(4)++
1`wrk(5)
1`(wrk(1),Yes)++
1`(wrk(2),Yes)++
1`(wrk(3),Yes)++
1`(wrk(4),Yes)++
1`(wrk(5),Yes)
5
Result
1
1`Abort
TransResult
Coordinator Stopped
1
1`()
Worker Stopped
Worker
Work Votes
WorkerxVote

Counter Example Generation

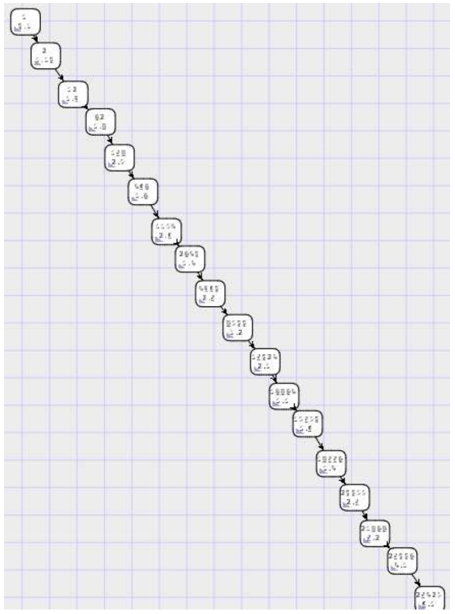
- **Find the arcs in a path form the initial state (node 1) to state 23497**

```
val path = ArcsInPath (1,23497)
```

```
val path = [1,11,93, 591, ... ] Arc list
```

- **Obtain sequence of binding elements corresponding to the arcs**

```
val errortrace = List.map ArcToBE path
```



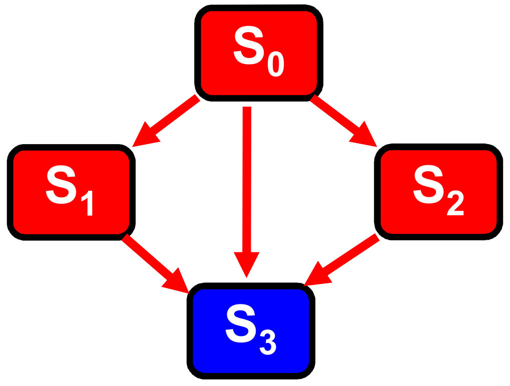
CPN Tools Demo

- **State space analysis of CPN models**
  - Variant where coordinator and workers returns to their idle state after having executed the protocol.

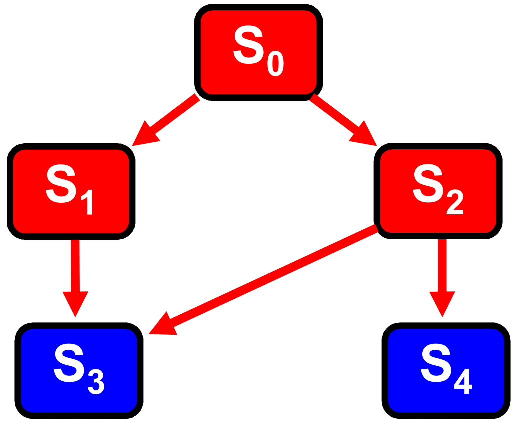
Home Markings and SCCs

- **The existence of home markings can be determined from the number of terminal SCCs.**

- **Only one terminal SCC**
  - All markings in the terminal SCC are home markings.
  - No other markings are home markings.

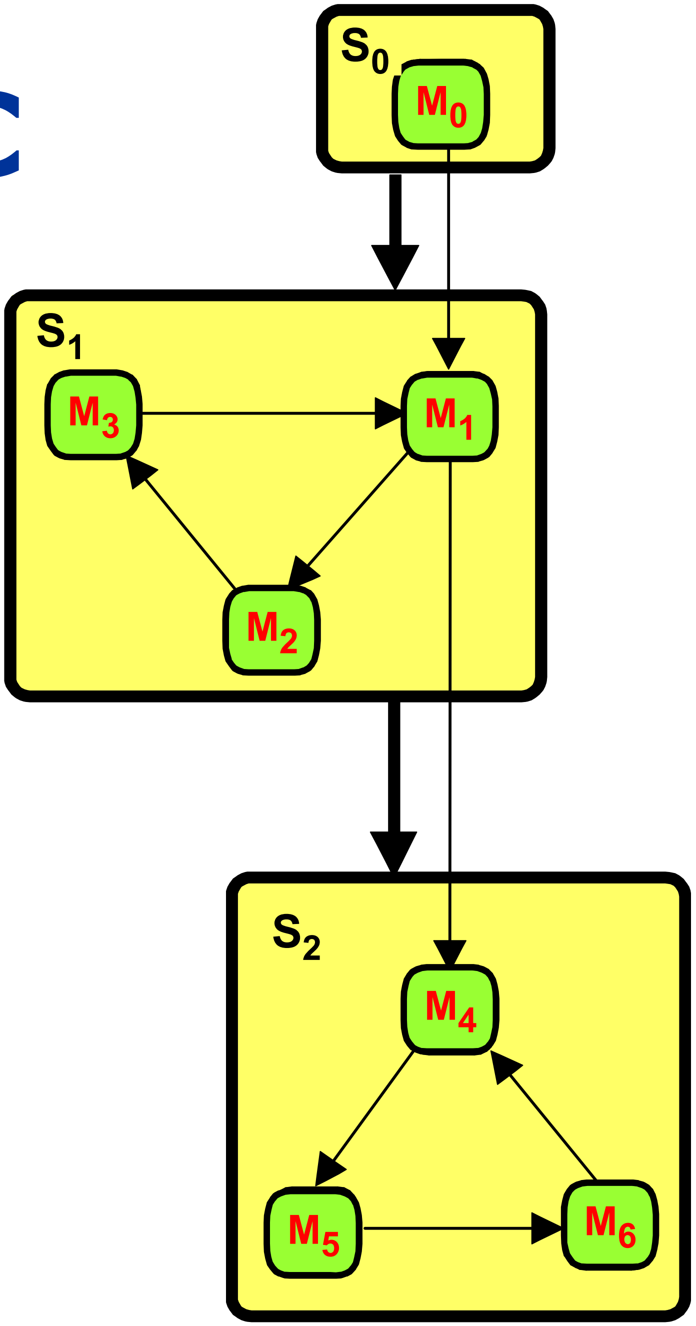


- **More than one terminal SCC**
  - No home markings.



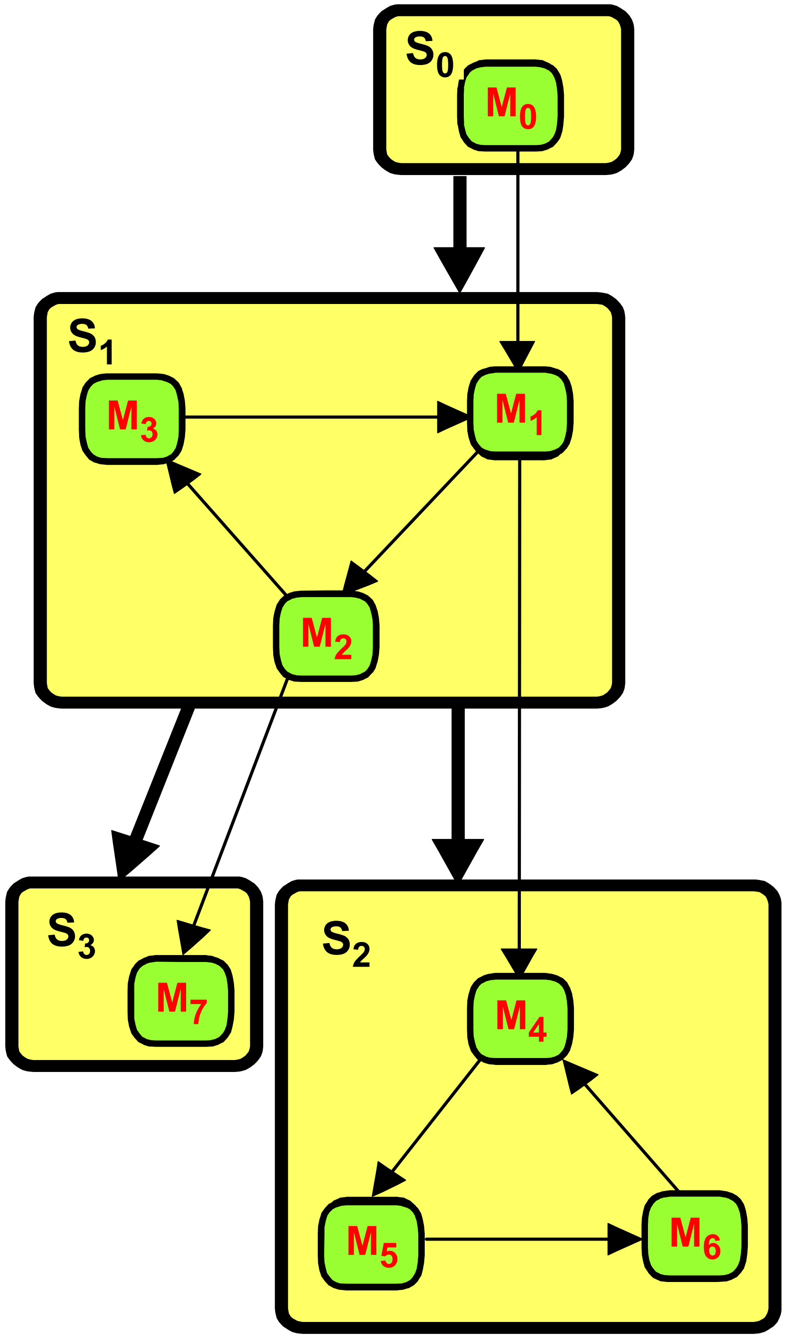
Single terminal SCC

- All markings in the terminal SCC S_2 are home markings.
- No other markings are home markings.



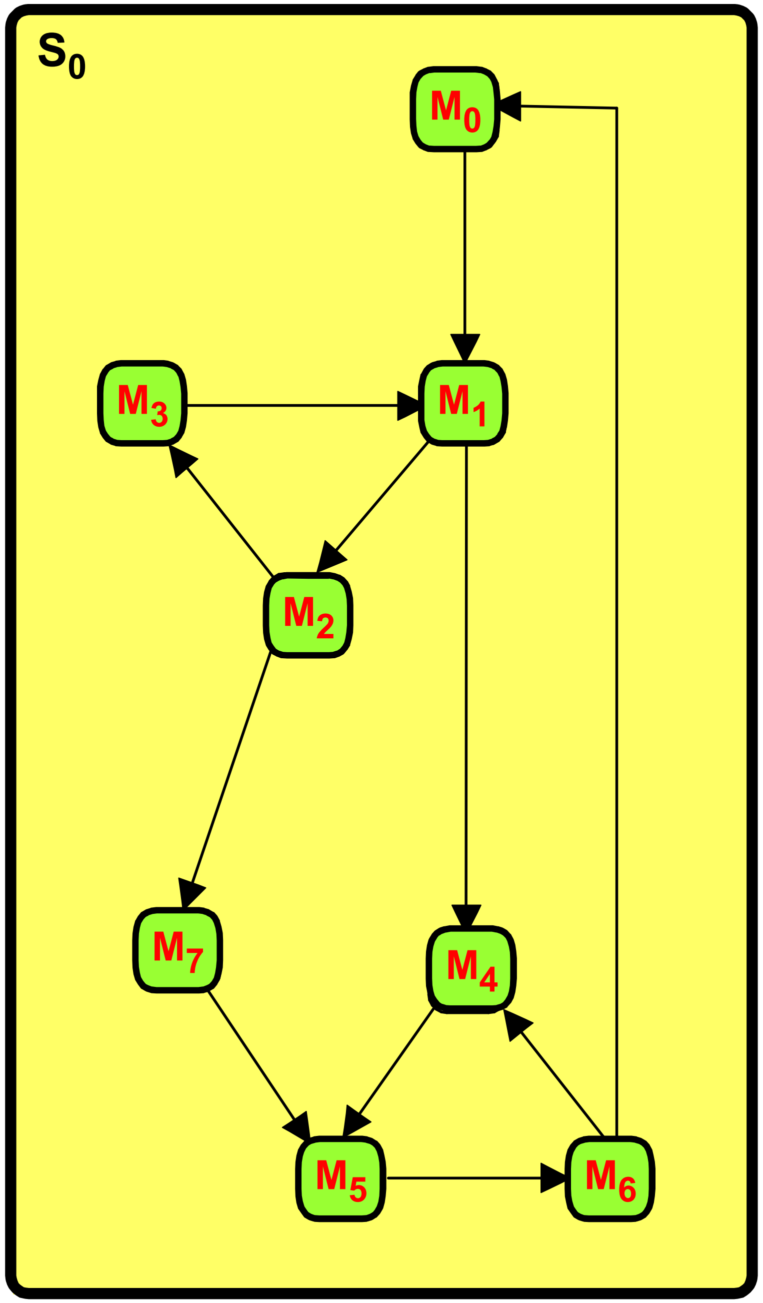
More than one terminal SCC

- No home markings.
- When one of the terminal SCCs S_2 and S_3 has been reached, it is impossible to leave it again.



Single SCC

- All reachable markings are home markings.
- They are mutually reachable from each other.

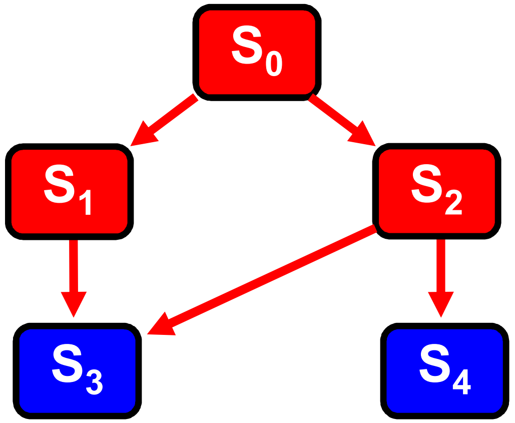


$$\begin{tikzcd}
M_3 \arrow[r] & M_1 \arrow[dl] \arrow[d] & M_0 \arrow[l] \\
M_2 \arrow[u] \arrow[d] & M_4 \arrow[d] & \\
M_7 \arrow[r] & M_5 \arrow[r] & M_6 \arrow[ul] \arrow[uu]
\end{tikzcd}$$

Home Spaces and SCCs

- **Home spaces can be determined from the terminal components in the SCC graph.**

- **A set of markings is a home space if and only if it contains a node from each terminal SCC.**
- **Home spaces must have at least as many elements as there are terminal SCCs.**



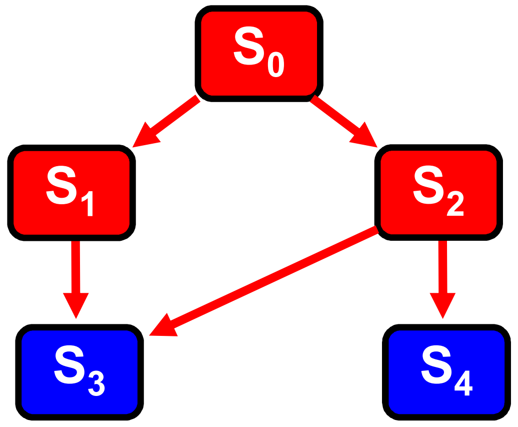
  - Each home marking is a home space with only one element.
  - A system may have home spaces without having home markings.

Liveness Properties and SCCs

- **Liveness properties of transitions can be determined from the SCC graph.**

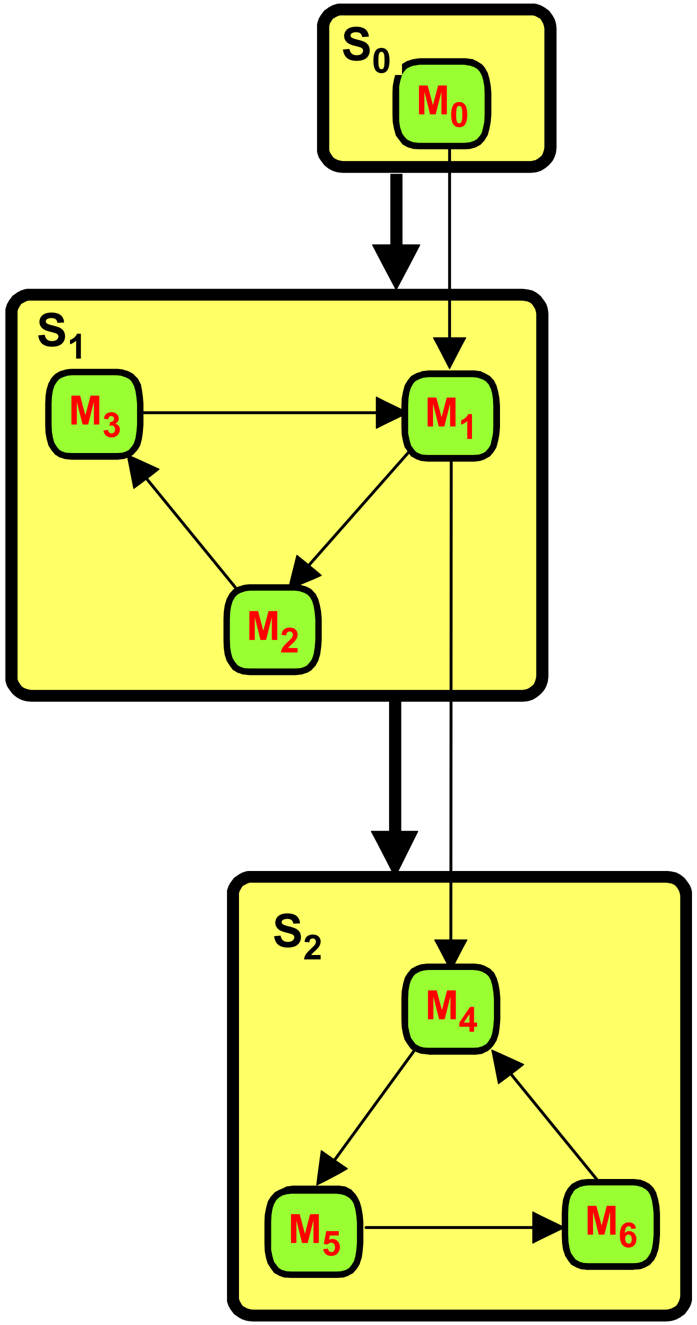
- A transition/binding element is live if and only if it appears on at least one arc in each terminal SCC.

- A set of transitions/binding elements is live if and only if each of the terminal SCCs contains at least one arc with a transition/binding element from the set.


$$\begin{tikzcd}
 & S_0 \arrow[dl] \arrow[dr] & \\
S_1 \arrow[d] & & S_2 \arrow[dll] \arrow[d] \\
S_3 & & S_4
\end{tikzcd}$$

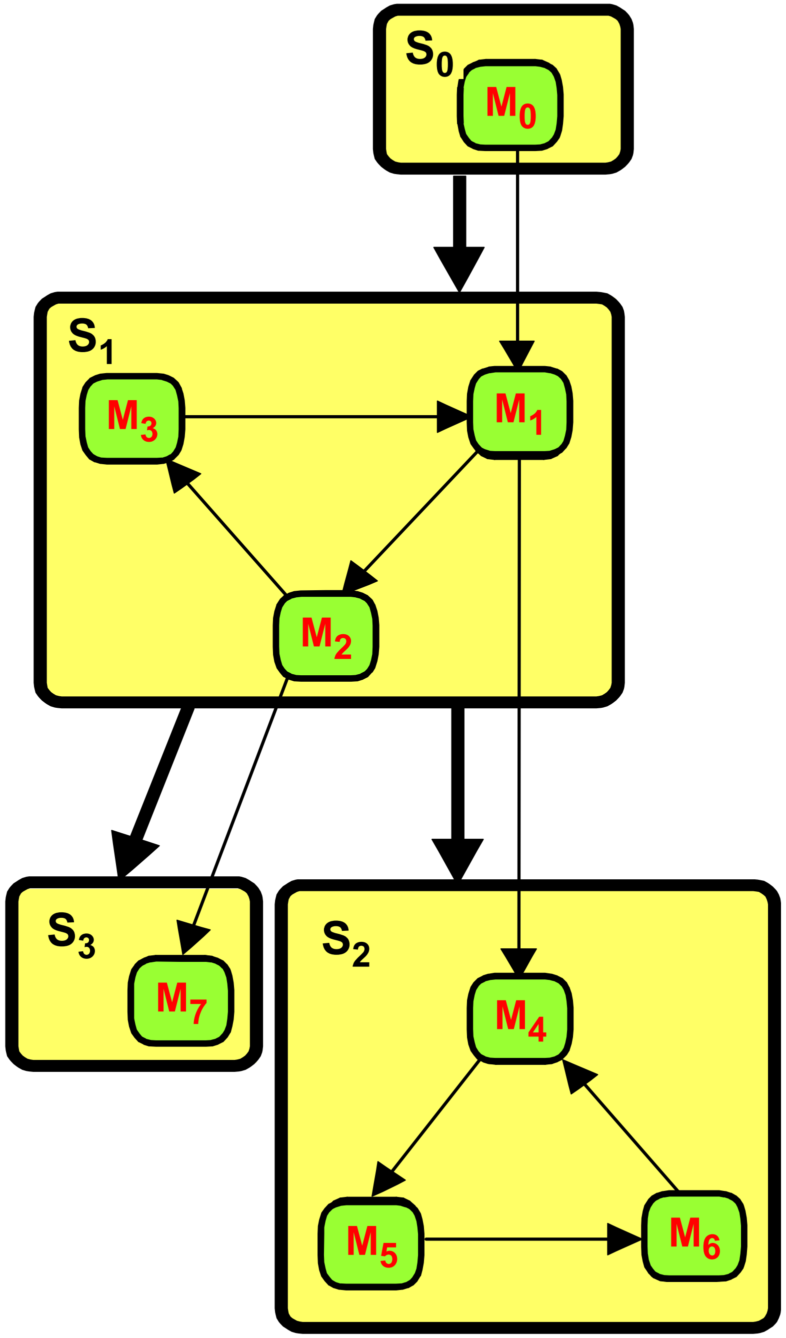
Single terminal SCC

- A transition is live if it appears on an arc in the terminal SCC S_2.



More than one terminal SCC

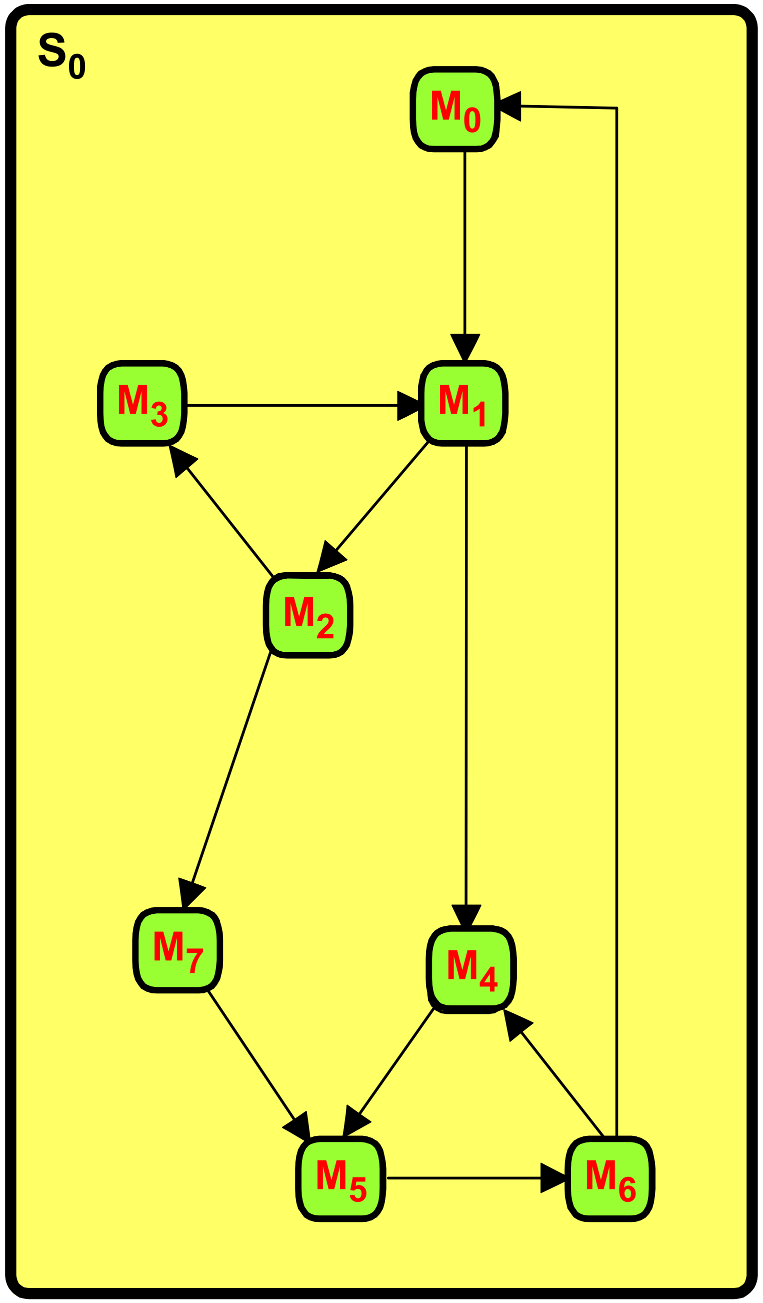
- A transition is live if it appears on an arc in each terminal SCC.
- No live transitions.
- S_3 is terminal and trivial.
- M_7 is a dead marking.



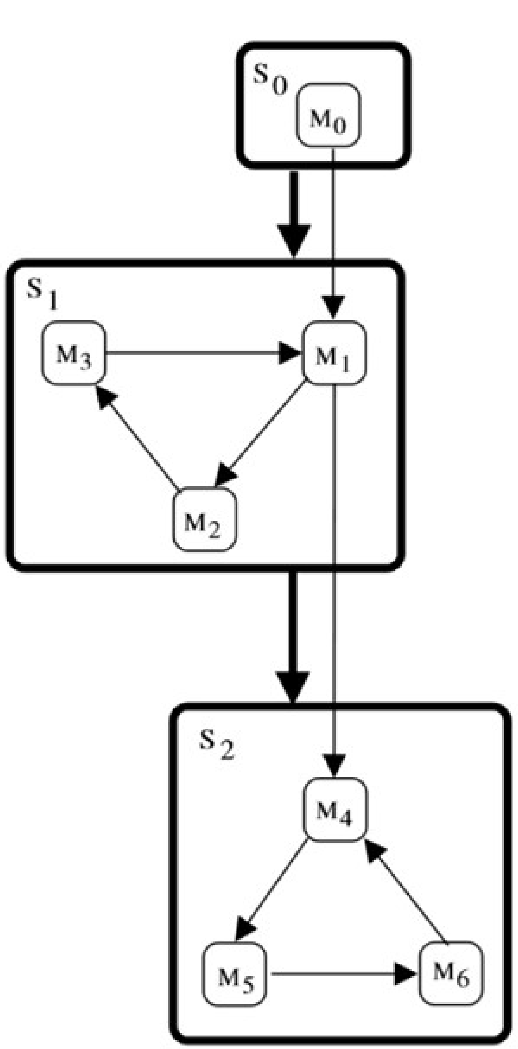
Single SCC

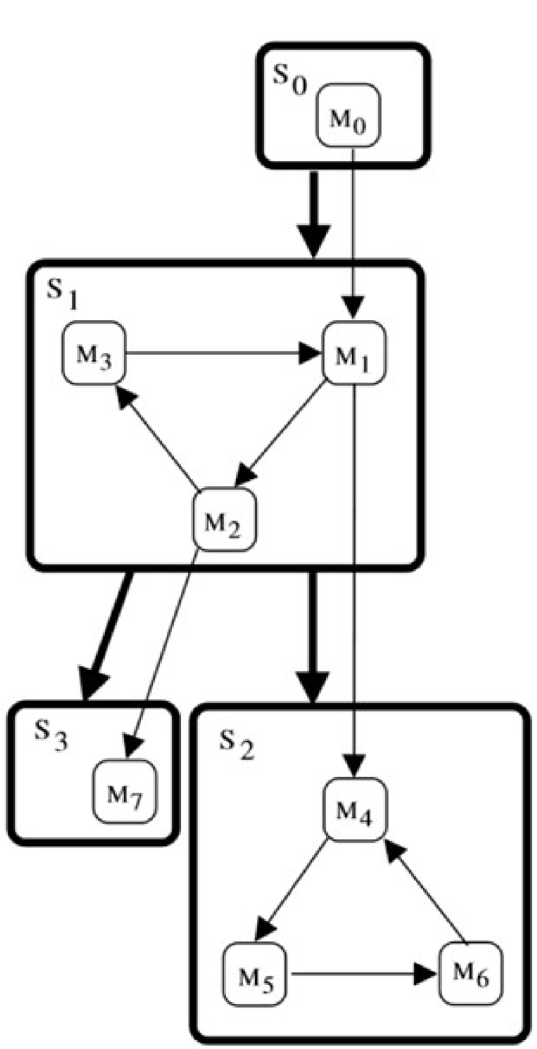
- A transition is live if it appears on an arc in the SCC.

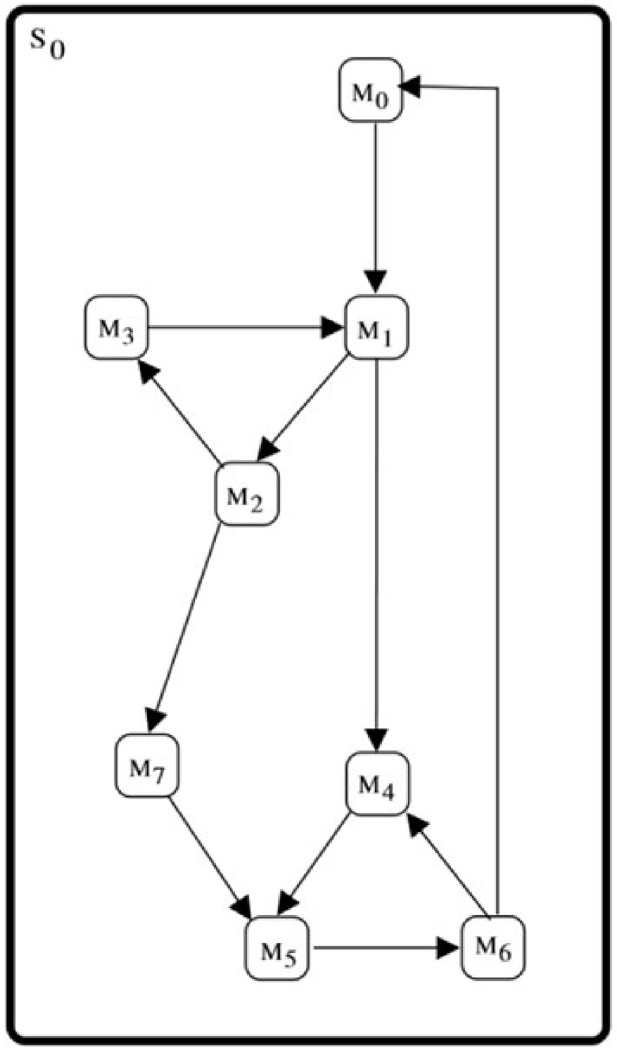
- In <u>this case</u> we have:
- A transition is live if and only is it is non-dead.



Home Properties and SCCs







System Configurations

- **Basic state spaces analyses a system for a particular configuration of its parameters**
  - in practice it is often sufficient to consider a few rather small configurations
  - we cannot be totally sure that larger configurations will have the same properties.
- **As system parameters increase the size of the state space often increases exponentially**
- **This phenomenon is also known as the space explosion problem**
  - one of the most severe limitations of state space methods.

Is it worthwhile?

- **It takes many hours (or days) to generate the state spaces and verify the desired properties**
  - it is fully automatic and hence requires little human work
- **A relatively small investment compared to**
  - the total number of resources used in a system development project.
  - cost of implementing, deploying and correcting a system with errors that could have been detected in the design phase.

State Spaces – Summery

- **State spaces are powerful and easy to use**
  - construction and analysis can be automated.
  - the user do not need to know the underlying mathematics.
- **The main drawback and limitation for practical use is the state explosion problem**
  - the present CPN state space tool can handle state spaces with up to several million states.
  - for many systems this is not sufficient.
  - efficient state space methods is an active area of research
  - an abundance of sophisticated techniques exists for alleviating the state explosion problem.