Looking good, behaving well

Michael Westergaard
Department of Computer Science
University of Aarhus
Example (1/4)

- 2 runners in a race, halfway through the race is a stand with water

- Either
  - run: a runner runs to the drink stand,
  - win: a runner wins the race, or
  - lose: a runner loses the race
Example (2/4)

- Only one runner can win the race
- In the beginning neither of the runners have finished any laps
- We can model this using Timed Automata or Coloured Petri nets
Example (3/4)
Example (3/4)
Example (3/4)
Example (3/4)
Example (4/4)
Motivation

We want the model to

- look good (even to people not familiar with the modeling formalism)
- behave well (e.g. ensure only one runner can win the race)
Outline

- The BRITNeY animation tool
- A state space tool
  - Memory-efficient state storage using the sweep-line method
  - Memory-efficient state storage using hash compaction and backtracking
Looking Good
The BRITNeY animation tool
Motivation
Motivation

Domain expert
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Motivation

Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.

Domain expert

FM expert

Formal model

Specification

FM expert

Domain expert
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Motivation

Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Methodology

Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Methodology

Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Figure 2 shows the approach taken to use CPN models to develop a prototype of the interoperability protocol. A CPN model (lower left of Fig. 2) has been developed by modelling the natural language protocol specification [22] (lower right) of the interoperability protocol. The modelling activity transforms the natural language specification into a formal executable specification represented by the CPN model. The CPN model captures the network architecture depicted in Fig. 1 and the protocol mechanisms of the interoperability protocol, e.g., the periodic transmission of advertisements, the dynamic updates of the DNS database, and traffic flows between hosts in the core network and nodes in the ad-hoc network.
Good-looking Runners
Executive Edition
Good-looking Runners
Engineer Edition
Good-looking Runners
Performance Analyst Edition
```
Step: 0
Time: 0

**Declarations**
- val laps
- fun no_laps
- fun model_time

**Color definitions**
- colset FLAG
- colset RUNNER = index r with 1..2 timed;
- colset LAP = int;
- colset RUNNER\times\text{LAP} = product \text{RUNNER} \times \text{LAP}

**Variable definitions**
- val d = discrete(30, 50);

New Page
```
BRITNeY animation (1/3)

Controller changes Model observes alerts View

Controller invokes Model

Model observes View
BRITNeY animation (2/3)

Controller + Model

invokes

View

observes

alerts

Formal executable model

BRITNeY animation
BRITNeY animation (3/3)

CPN Tools
editor

CPN simulator
Animation stubs
XML-RPC client

BRITNeY animation
XML-RPC server

Animation plugins
More Information about BRITNeY animation

Tool web-page: http://wiki.daimi.au.dk/tincpn

Screen-cast from CPN Workshop 2005 tutorial: http://www.daimi.au.dk/~mw/local/demo/BRITNeY_animation/

Behaving Well

A state space tool
State Space Analysis
State Space Analysis

Reachability: Does any possible state of the system satisfy a given property?
State Space Analysis

Reachability: Does any possible state of the system satisfy a given property?

E.g., can more than one runner win?
State Space Analysis

- Reachability: Does any possible state of the system satisfy a given property?
  - E.g., can more than one runner win?
- Analysis of CP-nets is impossible
State Space Analysis

- Reachability: Does any possible state of the system satisfy a given property?
  - E.g., can more than one runner win?
- Analysis of CP-nets is impossible
- How do we do it anyway? Try all possible states
State Space Analysis

Reachability: Does any possible state of the system satisfy a given property?

E.g., can more than one runner win?

Analysis of CP-nets is impossible

How do we do it anyway? Try all possible states

Loops? Build reachability graph
State Space for Runners

(r1, r2, flag)

r1, r2 ∈ {s, d, w, l}
flag ∈ {u, d}

((s, w, d), run 1) → (d, w, d), lose 1
((s, w, d), win 2) → (w, l, d), lose 2
((s, w, d), run 2) → (s, s, u)

((s, d, u), run 1) → (d, d, u), win 1
((s, d, u), run 2) → (w, d, d)

((d, s, u), run 1) → (w, s, d), win 1
((d, s, u), run 2) → (d, d, u)

((s, s, u), run 1) → (s, w, d)
((s, s, u), run 2) → (d, s, u)

((d, s, u), run 2) → (w, s, d)
Simple Algorithm for State Space Analysis

```java
Queue.add(Simulator.get_initial_state())

while !Queue.is_empty() do
  s := Queue.remove_first()
  Storage.add(s)
  process(s)
  forall s' in Simulator.get_next(s) do
    if !Storage.contains(s') then
      Queue.add(s')
    endif
  endfor
endwhile
```
Parametrizing the Algorithm

The algorithm relies on 3 data-structures:

- Simulator (get_initial_state, get_next)
- Queue (add, is_empty, remove_first)
- Storage (add, contains)

By providing different implementations, we can control which formalism to use (Simulator), how to traverse the state space (Queue - waiting/unprocessed), and how to store data efficiently (Storage - passed/processed)
Problems with State Space Analysis
Problems with State Space Analysis

Problem: The reachability graph is large, often even infinite
Problems with State Space Analysis

Problem: The reachability graph is large, often even infinite

Solution 1: Store only some of the graph
Problems with State Space Analysis

Problem: The reachability graph is large, often even infinite

Solution 1: Store only some of the graph

Solution 2: Store each node more efficiently
Problems with State Space Analysis

Problem: The reachability graph is large, often even infinite

Solution 1: Store only some of the graph

Solution 2: Store each node more efficiently

A lot of so-called reduction methods exist and new reduction methods are found out every day
We want to represent the entire state space.

A state of the system is $(r_1, r_2, \text{flag})$ with $r_1, r_2 \in \{s, d, w, l\}$ and $\text{flag} \in \{u, d\}$.

Only some (10) of the syntactically possible states ($4 \cdot 4 \cdot 2 = 32$) are reachable.

At least $\lceil \log(32) \rceil = 5$ bits are used to store each state, although $\lceil \log(10) \rceil = 4$ bits would suffice.
A Condensed Representation (2/2)

In realistic examples, the number of syntactically possible states is much larger than the number of reachable states, so distinguishing only between reachable states yields a good reduction.

Alas, we first know the number of reachable states, when we have constructed the reachability graph.
The Sweep-line Method
The Sweep-line Method

Already processed
The Sweep-line Method

Algorithms (s,w,d) (d,w,d) (w,l,d) (l,w,d)
(s,d,u) (d,d,u) (w,d,d) (w,s,d)
(s,s,u) (d,s,u) (w,l,d) (l,w,d)

- run 1: win 2, lose 1
- run 2: win 1, lose 2

Already processed: (s,w,d), (s,d,u), (s,s,u)
Discovered but not yet processed: (d,w,d), (d,d,u), (d,s,u), (w,d,d), (w,s,d), (w,l,d), (l,w,d)
The Sweep-line Method

- **Already processed**
- **Discovered but not yet processed**
- **Not yet discovered**
The Sweep-line Method

Already processed

Discovered but not yet processed

Not yet discovered

0 1 2 3
All states we will encounter during the rest of the exploration are located in front of the sweep-line.
The Sweep-line Method

Already processed

Discovered but not yet processed

Not yet discovered

0 1 2 3
The Sweep-line Method

Already processed

Discovered but not yet processed

Not yet discovered
The Sweep-line Method

Already processed

Discovered but not yet processed

Not yet discovered
A Neighbor List Representation

- (s,w,d) → run 1
- (d,w,d) → lose 1
- (s,d,u) → win 2
- (d,d,u) → win 2
- (w,d,d) → lose 2
- (s,s,u) → run 2
- (d,s,u) → run 2
- (w,s,d) → run 2

Diagram:

- (s,w,d) → run 1
- (d,w,d) → lose 1
- (s,d,u) → win 2
- (d,d,u) → win 2
- (w,d,d) → lose 2
- (s,s,u) → run 2
- (d,s,u) → run 2
- (w,s,d) → run 2
Assume we can enumerate all transitions:
0: run 1
1: run 2
2: win 1
3: win 2
4: lose 1
5: lose 2
Assume we can enumerate all transitions:
0: run 1
1: run 2
2: win 1
3: win 2
4: lose 1
5: lose 2
Assign a unique number, 0...9, to each state.
Assign a unique number, 0...9, to each state.

Neighbor List Representation

- (s,w,d) -> (d,w,d) with run 1, win 2, lose 1
- (s,d,u) -> (d,d,u) with run 1, win 1
- (s,s,u) -> (d,s,u) with run 1, win 1
- (w,l,d) with run 2, win 2, lose 2
- (w,d,d) with run 2, lose 2
- (l,w,d) with run 2, win 2, lose 2
A Neighbor List Representation

(s,w,d) -> (d,w,d) -> (w,l,d)
run 1

(s,d,u) -> (d,d,u) -> (l,w,d)
run 1

(s,s,u) -> (d,s,u) -> (w,s,d)
run 1

win 1

win 2

lose 1

lose 2

1 0
2 3
4 5
6 7
8 9

0 1 2 3 4 5 6 7 8 9
A Neighbor List Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>(0,1) (1,2)</td>
<td>2</td>
<td>(1,3) (2,6)</td>
<td>2</td>
<td>(0,3) (3,4)</td>
<td>2</td>
<td>(2,7) (3,5)</td>
<td>1</td>
<td>(0,5)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>(4,8)</td>
<td>1</td>
<td>(1,7)</td>
<td>1</td>
<td>(5,9)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The diagram represents a neighbor list with nodes connected by edges, indicating wins, runs, and losses.
A Neighbor List Representation

<table>
<thead>
<tr>
<th>State number</th>
<th>(0,1)</th>
<th>(1,2)</th>
<th>(1,3)</th>
<th>(2,6)</th>
<th>(0,3)</th>
<th>(3,4)</th>
<th>(2,7)</th>
<th>(3,5)</th>
<th>(0,5)</th>
<th>(4,8)</th>
<th>(1,7)</th>
<th>(5,9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

State transitions:
- (s,w,d) → (d,w,d) with run 1
- (d,μ,d) → (s,μ,d) with win 1
- (w,l,d) → (l,w,d) with lose 2

Diagram:
- States are represented as circles with edges indicating transitions.
- Colors indicate different states and transitions.
- Numbers (0, 1, 2, etc.) represent state numbers.
- Arrows show the direction of transitions.
A Neighbor List Representation

# successors

<table>
<thead>
<tr>
<th>x</th>
<th>Number of successors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>
A Neighbor List Representation

<table>
<thead>
<tr>
<th>Transition number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>(0,1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0,5)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(1,7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>(2,7)</td>
<td></td>
<td></td>
<td>(4,8)</td>
<td></td>
<td></td>
<td>(1,7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(5,9)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(4,8)</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# successors:

- 0: (0,1), (1,2), (0,5), (4,8)
- 1: (1,3), (2,6), (1,7), (5,9)
- 2: (2,7), (1,7), (1,7), (5,9)
- 3: (2,7), (3,4), (2,7), (3,5)
- 4: (0,5)
- 5: (4,8)
- 6: (1,7)
- 7: (5,9)
- 8: (0,5)
- 9: (4,8)
A Neighbor List Representation

Transition number

Successor state number

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>(0,1) (1,2)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>(2,7) (2,7)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>(0,5)</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>(4,8)</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>(1,7)</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>(5,9)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

# successors

Transition number

Successor state number

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>(0,1) (1,2)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>(2,7) (2,7)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>(0,5)</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>(4,8)</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>(1,7)</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>(5,9)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Building the Condensed Representation
Building the Condensed Representation
Building the Condensed Representation
Building the Condensed Representation

$(s, s, u)$
Building the Condensed Representation

(s,d,u)

run 1

(s,s,u)

run 2

(d,s,u)
Building the Condensed Representation

\[(d,s,u)\]

\[(s,d,u)\]

\[(s,s,u)\]

\[(d,s,u)\]

0

1

2

3

run 1

run 2
Building the Condensed Representation

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>2</th>
<th>(0,1)</th>
<th>(1,2)</th>
</tr>
</thead>
</table>

(s,d,u)

(s,s,u) → (d,s,u)

run 2

run 1
Building the Condensed Representation

New header:

# bits used to store each successor

0 2 2 (0,1) (1,2)

0 1 2

0 2 2 (0,1) (1,2)

0 1 2

(s,d,u)

2

run 2

1

run 2

(s,s,u)

run 1

0 0

(s,s,u)

run 1

(d,s,u)

1

0 1 2 3
Building the Condensed Representation

(0,1) (1,2)

(s,d,u) 

(0,1) (1,2)

(s,s,u) 

(d,s,u) 

run 1

run 2

run 1

run 2
Building the Condensed Representation

(s,w,d)  
 run 1  win 2

(s,d,u)  
 run 2  run 1

(d,u)  
 run 1

0 2 2 (0,1) (1,2)
Building the Condensed Representation

\[
\begin{pmatrix}
0 & 2 & (0,1) & (1,2) \\
1 & 1 & & \\
2 & 2 & &
\end{pmatrix}
\]

\[
\begin{pmatrix}
(s, w, d) \\
(s, d, u) \\
(s, s, u)
\end{pmatrix}
\]

\[
\begin{pmatrix}
(d, d, u) \\
(d, s, u)
\end{pmatrix}
\]

- Run 1
  - Win 2
  - Run 1
  - Run 1

- Run 2
  - Run 2

\[
\begin{pmatrix}
0 & 2 & 4 \\
3 & 0 & 0 \\
1 & 3 & 1
\end{pmatrix}
\]
Building the Condensed Representation

Run 1
(s,w,d)

Win 2
(s,w,d)

Run 1
(s,d,u)

Run 2
(s,s,u)

Run 1
(d,s,u)

Run 1
(d,d,u)
Building the Condensed Representation
Building the Condensed Representation

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>2</th>
<th>(0,1)</th>
<th>(1,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 2 2 (0,1) (1,2)

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

(s,w,d) run 1 (d,w,d)

(s,d,u) win 2 (d,d,u)

(s,s,u) run 2 (d,s,u)
Building the Condensed Representation

0 2 2 (0,1) (1,2)
1 2 3 (0,3) (3,4)
2 1 3 (0,5)
3
4
5

(s,w,d) run 1 (d,w,d)
(s,d,u) run 1 (d,d,u)
(s,s,u) run 1 (d,s,u)

run 1
run 2
run 1
run 2

0 1 2 3
Building the Condensed Representation

\( (d,s,u) \)

\( (d,d,u) \)

\( (s,w,d) \)

\( (s,d,u) \)

\( (s,s,u) \)

\( (d,w,d) \)

\( (d,d,u) \)

\( (s,w,d) \)

\( (s,d,u) \)

\( (s,s,u) \)
Building the Condensed Representation

- (d, w, d) at (5, 1)
- (d, d, u) at (3, 3)
- (d, s, u) at (1, 3)
Building the Condensed Representation

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>2</th>
<th>(0,1)</th>
<th>(1,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(d,w,d)  5
(d,d,u)  3
(d,s,u)  1
(w,s,d)  2

run 2
win 1
Building the Condensed Representation
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[(d,w,d)\] \[5\] \[(d,d,u)\] \[3\] \[(d,s,u)\] \[1\], \[2\] \[(w,s,d)\] \[6\]

run 2

win 1

0 1 2 3
Building the Condensed Representation

0 | 2 2 (0,1) (1,2)
1 | 2 3 (1,3) (2,6)
2 | 2 3 (0,3) (3,4)
3 | 1 3 (0,5)
4 | 5 |
5 |
6 |

0 2 2 (0,1) (1,2)
1 2 3 (1,3) (2,6)
2 2 3 (0,3) (3,4)
3 1 3 (0,5)
4 |
5 |
6 |

(d,w,d) 5
3 win 2
(d,d,u) 3 win 1
1 run 2
(d,s,u) 2 win 1
(w,d,d)

(w,s,d) 6
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(d,w,d) → 5

(d,d,u) → 3

win 2

(d,d,u) → 3

win 1

(d,s,u) → 1

run 2

(d,s,u) → 1

win 1

(w,d,d) → 7

(w,s,d) → 6

(w,d,d) → 2

(w,s,d) → 6
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[(d, w, d) \rightarrow (l, w, d)\]
\[(d, d, u) \rightarrow (w, d, d)\]
\[(d, s, u) \rightarrow (w, s, d)\]
\[(d, w, d) \rightarrow \text{win 2}\]
\[(d, d, u) \rightarrow \text{win 1}\]
\[(d, s, u) \rightarrow \text{win 1}\]
\[(d, w, d) \rightarrow \text{lose 1}\]
\[(d, d, u) \rightarrow \text{run 2}\]

\[(0,1), (1,2), (1,3), (2,6), (0,3), (3,4), (2,7), (3,5), (0,5)\]
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **(d,w,d)**: Lose 1
- **(d,d,u)**: Win 1
- **(d,s,u)**: Win 1
- **(w,d,d)**: Win 2
- **(w,s,d)**: Run 2
- **(l,w,d)**: Win 2

The image shows a condensed representation with transitions between different states, indicated by the arrows and colors. The states are labeled with numbers and connected by lines, each with a label indicating the outcome (win, lose, run).
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>2</th>
<th>(0,1)</th>
<th>(1,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Graph:
- Node 1: Run 2
- Node 2: Win 1
- Node 3: Win 2
- Node 4: Lose 1
- Node 5: Win 1
- Node 6: Win 1
- Node 7: Win 1
- Node 8: Win 1

Links:
- (d,w,d) to (l,w,d)
- (d,d,u) to (w,d,d)
- (d,s,u) to (w,s,d)
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building the Condensed Representation

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1) (1,2)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3) (2,6)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3) (3,4)</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7) (3,5)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>(1,7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- (w,d,d) at (2, 7)
- (w,s,d) at (2, 6)
- Run 2
Building the Condensed Representation
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 2</td>
<td>(0,1) (1,2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2 3</td>
<td>(1,3) (2,6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2 3</td>
<td>(0,3) (3,4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2 3</td>
<td>(2,7) (3,5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 3</td>
<td>(0,5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 4</td>
<td>(4,8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 3</td>
<td>(1,7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **(w,l,d)**
- **(w,d,d)**
- **(l,w,d)**
- **(w,s,d)**

- Lose 2
- Run 2
- Run 2
- Run 2
- Lose 2
Building the Condensed Representation

0 2 2 (0,1) (1,2)
1 2 3 (1,3) (2,6)
2 2 3 (0,3) (3,4)
3 2 3 (2,7) (3,5)
4 1 3 (0,5)
5
6
7
8
9

(w,s,d)
(w,l,d)
(l,w,d)
Building the Condensed Representation
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>(1,7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>(5,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Building the Condensed Representation

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1) (1,2)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3) (2,6)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3) (3,4)</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7) (3,5)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>(1,7)</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>(5,9)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building the Condensed Representation

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>2</th>
<th>(0,1)</th>
<th>(1,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>(1,7)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>(5,9)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Building the Condensed Representation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>(1,7)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>(5,9)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(l,w,d)
Building the Condensed Representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>(1,3)</td>
<td>(2,6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>(0,3)</td>
<td>(3,4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>(2,7)</td>
<td>(3,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>(0,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>(4,8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>(1,7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>(5,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation of the Algorithm

- We only store a few (6) actual states
- The condensed representation uses
  \[4 \cdot |R| \cdot w + |E| \cdot (\log|T| + \log|R|) + |F| \cdot \log|S|\text{ bits}\]
- \(R\): all reachable states
- \(w\): size of machine word
- \(E\): all reachable arcs
- \(T\): all transitions
- \(S\): all syntactically possible states
- \(F\): states on the front
- Efficient standard representation
  \[|R| \cdot (3 \cdot w + \log|S|) + |E| \cdot \log|S|\]
More Information about the Algorithm

T. Mailund, M. Westergaard: Obtaining Memory-Efficient Reachability Graph Representations Using the Sweep-Line Method, TACAS 2004
Backtrack the State Space
Backtrack the State Space

We need to support the operations add(s) and contains(s) but not any get operation.
Backtrack the State Space

- We need to support the operations add(s) and contains(s) but not any get operation

- Idea of hash compaction: Store hash value for each state only
Backtrack the State Space

- We need to support the operations `add(s)` and `contains(s)` but not any get operation.

- Idea of hash compaction: Store hash value for each state only.

- Hash collision?
Backtrack the State Space

- We need to support the operations add(s) and contains(s) but not any get operation.

- Idea of hash compaction: Store hash value for each state only.

- Hash collision?

- Hash compaction fails.
Backtrack the State Space

- We need to support the operations add(s) and contains(s) but not any get operation.

- Idea of hash compaction: Store hash value for each state only.

- Hash collision?

- Hash compaction fails.

- We will store for each state a predecessor and trace from the initial state.
Hash Compaction

Diagram showing the compaction process with different states and transitions:

- States: (s,w,d), (d,w,d), (l,w,d), (w,d,d), (s,d,u), (d,d,u), (w,s,d), (s,s,u)
- Transitions: run 1, run 2, win 1, win 2, lose 1, lose 2
Hash Compaction

We assume a hash function, assigning to each state a hash-value.
We assume a hash function, assigning to each state a hash-value.

Hash Compaction

(s,s,u) ▶ (w,s,d)]

run 1 ▶ lose 1 ▶ win 2 ▶ win 2 ▶ lose 2 ▶ win 2 ▶ lose 2

(s,d,u) ▶ (d,d,u) ▶ (w,d,d)
run 1 ▶ win 1 ▶ run 2 ▶ win 1 ▶ run 2

(s,w,d) ▶ (d,w,d) ▶ (l,w,d)
run 1 ▶ lose 1 ▶ win 2 ▶ win 2 ▶ lose 2
Hash Compaction
Hash Compaction

$(s, s, u)$

$(s, s, u)$
Hash Compaction

$h_0$
Hash Compaction

\[ h_0 \]

\((s, s, u)\)

run 1

run 2
Hash Compaction

\( h_0 \)
Hash Compaction

$h_0$  $h_1$

$s, s, u$  $(d, s, u)$

run 1  run 2
Hash Compaction

h0
h1
h2
h3
h4
Hash Compaction

h0
h1
h2
h3
h4

run 1
win 1

run 2
lose 2

(s,s,u) → (d,s,u) → (w,s,d) → (w,l,d) → (w,d,d)

h0
h1
h2
h3
h4
Hash Compaction

h0
h1
h2
h3
h4
h5

h0
h1
h2
h3
h4
h5
Hash Compaction

h0
h1
h2
h3
h4
h5

(s,s,u) → (d,s,u) → (w,s,d)
run 1
win 1

(w,l,d) → (w,d,d)
lose 2
run 2

(d,d,u) → (w,d,d)
win 2
run 2

((w,d,d) ← (w,l,d))
Hash Compaction

No need to re-explore this state
Hash Compaction

(s,s,u) → (w,s,d) → (w,l,d) → (w,s,d) → (d,d,u) → (d,d,u) → (s,s,u)

h0 → h1 → h2 → h3 → h4 → h5

run 1 → win 1 → run 2 → win 2 → lose 2
Hash Compaction

h0 → h1 → h2 → h3 → h4

h5 → h6 → h7

(s,s,u) → (d,s,u) → (w,s,d) → (w,l,d) → (l,w,d)

run 1 → win 1 → win 2 → lose 1 → lose 2

run 2 → win 1 → lose 2
Hash Compaction

- **h0**: (s,s,u) -> run 1
- **h1**: (d,s,u) -> win 1
- **h2**: (w,s,d) -> run 2
- **h3**: (l,w,d) -> lose 2
- **h4**: (w,l,d) -> lose 1
- **h5**: (d,w,d) -> win 1
- **h6**: (s,d,u) -> run 2
- **h7**: (w,l,d) -> lose 1

Numbers:
- h0
- h1
- h2
- h3
- h4
- h5
- h6
- h7
Hash Compaction

- h0
- h1
- h2
- h3
- h4
- h5
- h6
- h7
Hash Compaction

h3 has already been used

h0
h1
h2
h3
h4
h5
h6
h7
Hash Compaction

h0 h1 h2 h3 h4 h5 h6 h7

(s,s,u) (d,w,d) (w,l,d) (l,w,d)
(run 1 win 2 lose 1 lose 2)

(d,d,u) (w,d,d) (w,s,d)
(run 2 win 1 run 2 run 2)

(s,d,u) (d,s,u)
(run 1 run 2 win 1)

h0 h1 h2 h3 h4 h5 h6 h7
Hash Compaction

h0  h1  h2  h3  h4  h5  h6  h7

(s,w,d) (d,w,d) (w,l,d) (l,w,d)
(s,d,u) (d,d,u) (w,d,d) (w,s,d)
(s,s,u) (d,s,u)  win 1 win 2
run 1 run 2 lose 1 lose 2
win 1  win 2  lose 1  lose 2

h0  h1  h2  h3  h4  h5  h6  h7
Hash Compaction

Never discovered!

(s,w,d)
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm

(\(s, s, u\))

h0
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm

h0 0

0 (s,s,u) h0
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm

0

h0

h1

0

(s,s,u)

(d,s,u)

h0

h1

run 1

run 2
The ComBack Algorithm
The ComBack Algorithm

h0 0
h1 1

0

run 1

run 2

(s,s,u)

(d,s,u)
The ComBack Algorithm

The ComBack Algorithm is a machine learning technique that combines the strengths of different learning algorithms. It typically involves two runs: run 1 and run 2.

In run 1, the algorithm processes the data to learn the initial model parameters.

In run 2, the algorithm uses the information from run 1 to refine and improve the model parameters.

The diagram illustrates the process with nodes labeled with '0' and '1', and the transitions are indicated by arrows labeled 'run 1' and 'run 2'. The nodes labeled 'h0' and 'h1' represent the hidden states of the model.
The ComBack Algorithm
The ComBack Algorithm

<table>
<thead>
<tr>
<th>0</th>
<th>run 1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>win 1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>run 2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>lose 2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>run 2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>run 2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>h0</th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
<th>h5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?

<table>
<thead>
<tr>
<th></th>
<th>run 1</th>
<th>h0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>win 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>run 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>lose 2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>run 2</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)...

How do we know we have seen the state before (i.e. it is not a hash collision)?

We backtrack...
The ComBack Algorithm

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack...
The ComBack Algorithm

We rediscover the state (w,d,d) and compute the hash-value, h3...

How do we know we have seen the state before (i.e. it is not a hash-collision)?

We backtrack
We rediscover the state \((w,d,d)\) and compute the hash-value, \(h3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack
We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack.
The ComBack Algorithm

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack
The ComBack Algorithm

...and ComBack

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack...

...and ComBack
We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)... How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack... and ComBack
The ComBack Algorithm

...and ComBack

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack
The ComBack Algorithm...

...and ComBack

<table>
<thead>
<tr>
<th></th>
<th>run 1</th>
<th>0</th>
<th>h0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>run 1</td>
<td>0</td>
<td>h0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>win 1</td>
<td>1</td>
<td>h1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>run 2</td>
<td>2</td>
<td>h2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>lose 2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>run 2</td>
<td>2</td>
<td>h4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>run 2</td>
<td>1</td>
<td>h5</td>
<td>5</td>
</tr>
</tbody>
</table>

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)... How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack...

We backtracks...
The ComBack Algorithm

...and ComBack

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h3\)...

...how do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack...
The ComBack Algorithm

...and ComBack

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)... How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack...
The ComBack Algorithm

...and ComBack

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h_3\)...

How do we know, we have seen the state before (i.e. it is not a hash-collision)?
The ComBack Algorithm

...and ComBack

...and notice we have (re-)arrived at \((w,d,d)\). No need to proceed

We rediscover the state \((w,d,d)\) and compute the hash-value, \(h3\)... How do we know, we have seen the state before (i.e. it is not a hash-collision)?

We backtrack
The ComBack Algorithm
# The ComBack Algorithm

## Table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>run 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>win 1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>run 2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>lose 2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>run 2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>win 2</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>lose 1</td>
<td>6</td>
</tr>
</tbody>
</table>

## Diagram

```
0 ———> 1 ———> 2 ———> 3 ———> 4 ———> 5 ———> 6 ———> 7
```

```
run 1 0
win 1 1
run 2 2
lose 2 3
run 2 1
win 2 5
lose 1 6
```
By backtracking, we discover that the discovered state corresponding to h3 is (w,d,d), so (s,d,u) is new.
The ComBack Algorithm
The ComBack Algorithm

<table>
<thead>
<tr>
<th></th>
<th>run 1</th>
<th>win 1</th>
<th>run 2</th>
<th>lose 2</th>
<th>run 2</th>
<th>win 2</th>
<th>lose 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Diagram:
- ComBack Algorithm
- Moves through states
- Winning and losing conditions
- Transition arrows indicate state changes
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm
The ComBack Algorithm
We need to backtrack for state 3 and 8, and obtain \((w,d,d)\) and \((s,d,u)\), so \((s,w,d)\) is new.
The ComBack Algorithm

0
1 run 1 0 h0 0
2 win 1 1 h1 1
3 run 2 2 h2 2
4 lose 2 3 h3 3
5 run 2 1 h4 4
6 win 2 5 h5 5
7 lose 1 6 h6 6
8 run 2 0 h7 7

0 run 1 0 h0
1 win 1 1 h1
2 run 2 2 h2
3 lose 2 3 h3
4 run 2 1 h4
5 win 2 5 h5
6 lose 1 6 h6
7 run 2 0 h7

(w,l,d)

(w,s,d)

(d,s,u)

(s,d,u)

(w,d,d)

(d,d,u)

(s,w,d)

(l,w,d)
The ComBack Algorithm
The ComBack Algorithm

<table>
<thead>
<tr>
<th></th>
<th>run 1</th>
<th>0</th>
<th>h0</th>
<th>0</th>
<th>1</th>
<th>h1</th>
<th>1</th>
<th>2</th>
<th>h2</th>
<th>2</th>
<th>3</th>
<th>h3</th>
<th>3</th>
<th>8</th>
<th>h4</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>run 1</td>
<td>0</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>win 1</td>
<td>1</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>run 2</td>
<td>2</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>lose 2</td>
<td>3</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>run 2</td>
<td>1</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>win 2</td>
<td>5</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>lose 1</td>
<td>6</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>run 2</td>
<td>0</td>
<td>h0</td>
<td>0</td>
<td>1</td>
<td>h1</td>
<td>1</td>
<td>2</td>
<td>h2</td>
<td>2</td>
<td>3</td>
<td>h3</td>
<td>3</td>
<td>8</td>
<td>h4</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- **h0**: (s,s,u) -> run 1 -> 0
- **h1**: (d,s,u) -> win 1 -> 1
- **h2**: (w,s,d) -> run 2 -> 2
- **h3**: (s,w,d) -> run 1 -> 9
- **h4**: (d,w,d) -> lose 1 -> 4
- **h5**: (d,d,u) -> win 1 -> 5
- **h6**: (w,d,d) -> lose 2 -> 7
- **h7**: (l,w,d)
The ComBack Algorithm

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>run 1</td>
<td>0</td>
<td>h0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>run 1</td>
<td>0</td>
<td>h1</td>
<td>1</td>
<td>h2</td>
<td>2</td>
<td>h3</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>win 1</td>
<td>1</td>
<td>h4</td>
<td>4</td>
<td>h5</td>
<td>5</td>
<td>h6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>run 2</td>
<td>2</td>
<td>h7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>lose 2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>run 2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>win 2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lose 1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>run 2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>win 2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# The ComBack Algorithm

<table>
<thead>
<tr>
<th></th>
<th>run 1</th>
<th>run 2</th>
<th>win 1</th>
<th>win 2</th>
<th>lose 1</th>
<th>lose 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>run 1</td>
<td></td>
<td>win 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>win 1</td>
<td>run 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>run 2</td>
<td>lose 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>lose 2</td>
<td>run 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>run 2</td>
<td></td>
<td>win 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>win 2</td>
<td>lose 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lose 1</td>
<td>run 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>run 2</td>
<td></td>
<td>win 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>win 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Diagram

- **Nodes:**
  - **h0:** \((s,s,u)\)
  - **h1:** \((d,s,u)\)
  - **h2:** \((w,s,d)\)
  - **h3:** \((s,w,d)\)
  - **h4:** \((w,l,d)\)
  - **h5:** \((d,w,d)\)
  - **h6:** \((d,d,u)\)
  - **h7:** \((l,w,d)\)

- **Edges:**
  - **Run 1:** \(0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\)
  - **Win 1:** \(0 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9\)
  - **Lose 1:** \(0 \rightarrow 1 \rightarrow 2 \rightarrow 6 \rightarrow 8 \rightarrow 9\)
  - **Run 2:** \(0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow 9\)
  - **Win 2:** \(0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\)
  - **Lose 2:** \(0 \rightarrow 1 \rightarrow 2 \rightarrow 6 \rightarrow 8 \rightarrow 9\)

- **States:**
  - **h0:** \((s,s,u)\)
  - **h1:** \((d,s,u)\)
  - **h2:** \((w,s,d)\)
  - **h3:** \((s,w,d)\)
  - **h4:** \((w,l,d)\)
  - **h5:** \((d,w,d)\)
  - **h6:** \((d,d,u)\)
  - **h7:** \((l,w,d)\)
Evaluation of the Algorithm

The algorithm uses

$$|R| \cdot (w + (2 \cdot w + h)) + 2 \cdot |R| \cdot (h + \log|T|) =$$

$$|R| \cdot (3 \cdot (w + h) + \log|T|) \leq 7 \cdot w \cdot |R|$$

- $R$: reachable states
- $w$: size of machine word
- $h$: size of hash-value
- $T$: transitions

Experiments show that $16 \cdot w \cdot |R|$ is used

- $\log|T| = 3 \cdot w$

In SML-NJ everything is a reference
More Information about the Algorithm

L. Arge, G.S. Brodal, S. Christensen, L.M. Kristensen, and M. Westergaard: The ComBack State Space Exploration Method: Combining Hash Compaction with Backtracking, not yet submitted
Conclusion

- BRITNeY animation: formalism-independent tool for making formal models look good
- State space tool: formalism-independent platform for experimenting with multiple reduction methods of state spaces