University of Basel

Implementing and Evaluating Successor Generators in the Fast Downward Planning System

Bachelor Thesis

Yannick Zutter, 09.10.2020

Agenda.

Introduction – What is Planning?

Find sequence of operators to solve a given planning problem

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

https://www.jameswatkins.me/puzzles/2019/06/30/understanding-the-rubiks-cube.html (07.10.20)

Introduction – What is Planning?

Find sequence of operators to solve a given planning problem

Initial state

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

https://www.jameswatkins.me/puzzles/2019/06/30/understanding-the-rubiks-cube.html (07.10.20)

Introduction – What is Planning?

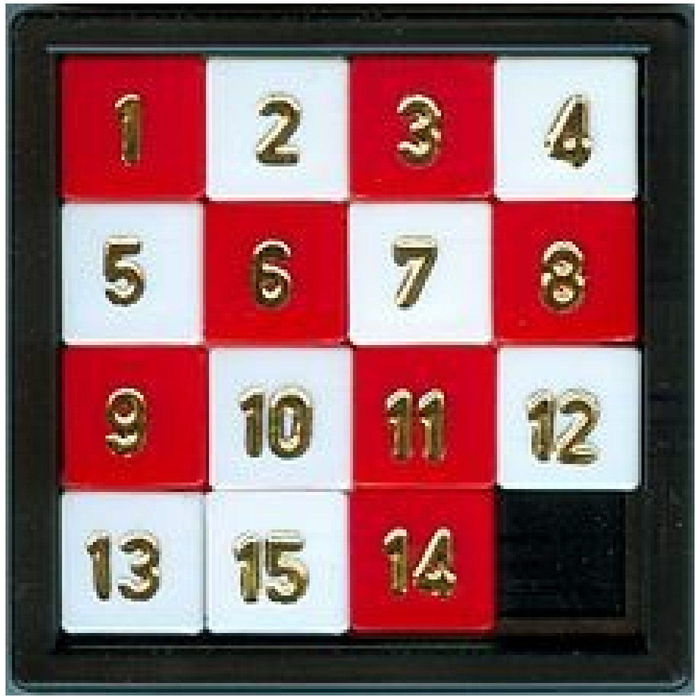
Find sequence of operators to solve a given planning problem

Initial state → operator

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

https://www.jameswatkins.me/puzzles/2019/06/30/understanding-the-rubiks-cube.html (07.10.20)

Introduction – What is Planning?



Find sequence of operators to solve a given planning problem

Initial state → operator → successor state → …

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

https://www.jameswatkins.me/puzzles/2019/06/30/understanding-the-rubiks-cube.html (07.10.20)

Introduction – What is Planning?

Find sequence of operators to solve a given planning problem

Initial state → operator → successor state → … → goal

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

https://www.jameswatkins.me/puzzles/2019/06/30/understanding-the-rubiks-cube.html (07.10.20)

Introduction – What is Planning?

FDR Planning Task

$\Pi = (V, s_0, s^*, O)$

- **V:** set of state variables with finite domain
- **s_0:** initial state as a set over V
- **s*:** set of goals as partial states
- **O:** set of operators with:
  - pre(o): preconditions as set a of facts
  - eff(o): effect of the operator
  - cost(o): cost

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

https://www.jameswatkins.me/puzzles/2019/06/30/understanding-the-rubiks-cube.html (07.10.20)

Introduction – What is Planning?

Operators:

- move_left
- move_right
- move_up
- move_down

move_left:

- Precondition: not in outer left column
- Effect: switch X with tile on left
- Cost: 1

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

https://www.jameswatkins.me/puzzles/2019/06/30/understanding-the-rubiks-cube.html (07.10.20)

Introduction – What is Planning?

Initial state

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix}$$

Introduction – What is Planning?

Initial state

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix} \xrightarrow{\text{move\_left}}$$

Introduction – What is Planning?

Initial state

successor state

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix} \xrightarrow{\text{move\_left}} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & X & 14 \end{pmatrix}$$

Introduction – What is Planning?

Initial state — successor state

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix} \xrightarrow{\text{move\_left}} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & X & 14 \end{pmatrix} \quad \Big\downarrow \; \ldots$$

Introduction – What is Planning?

Initial state

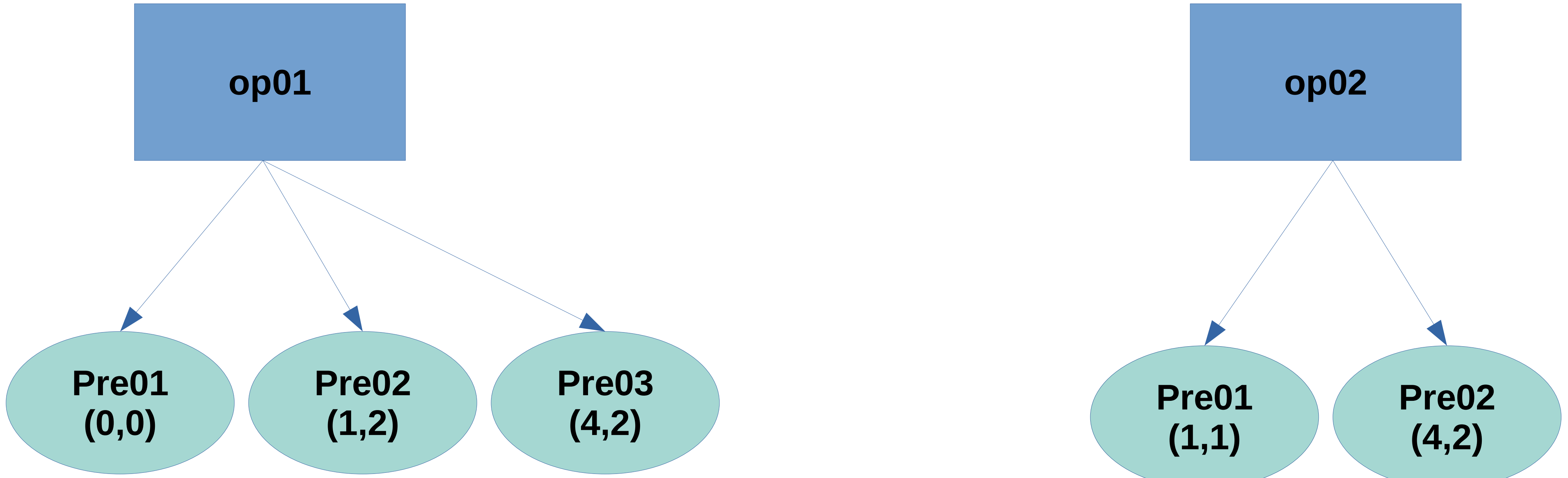
successor state

goal state

$$\begin{tikzcd}
\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & 14 & X \end{pmatrix} \arrow[r, "\text{move\_left}"] & \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 15 & X & 14 \end{pmatrix} \arrow[d, "\dots"] \\
& \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & X \end{pmatrix}
\end{tikzcd}$$

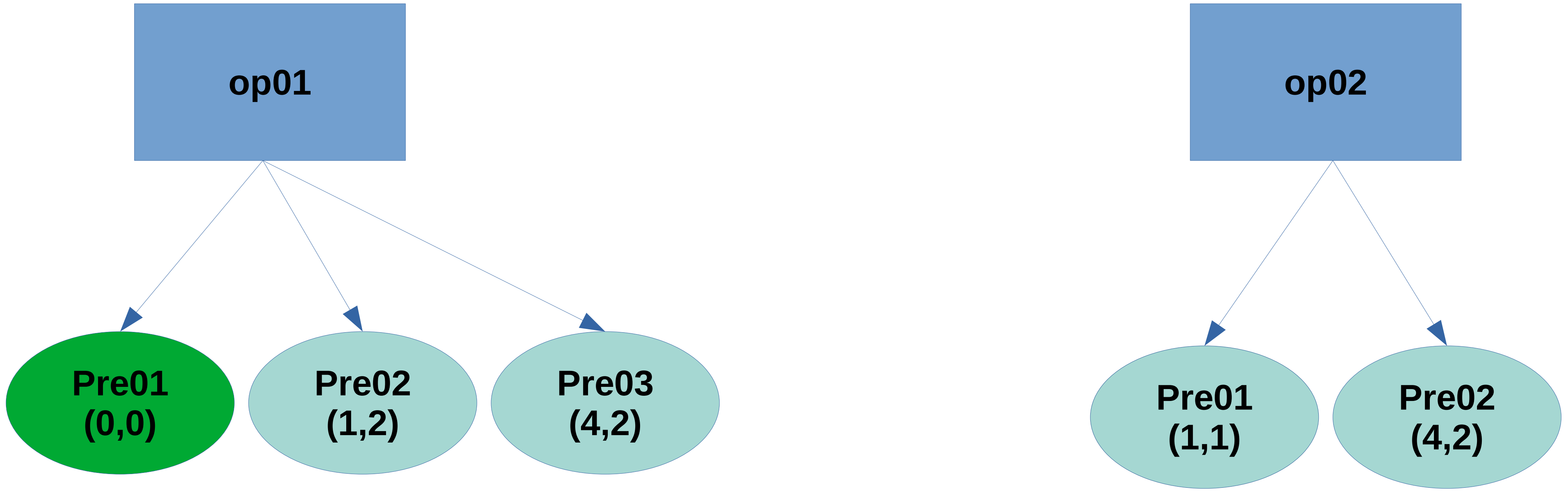
Agenda.

Naive Successor Generator



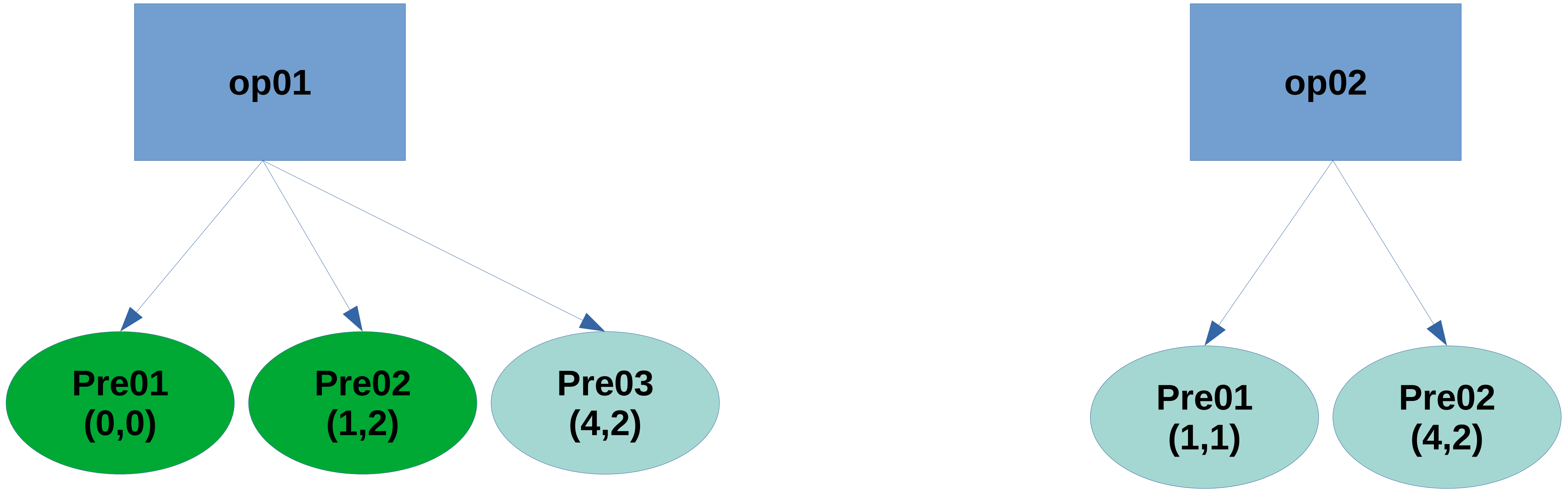
State:
(0, 2, 0, 2, 2)

Naive Successor Generator



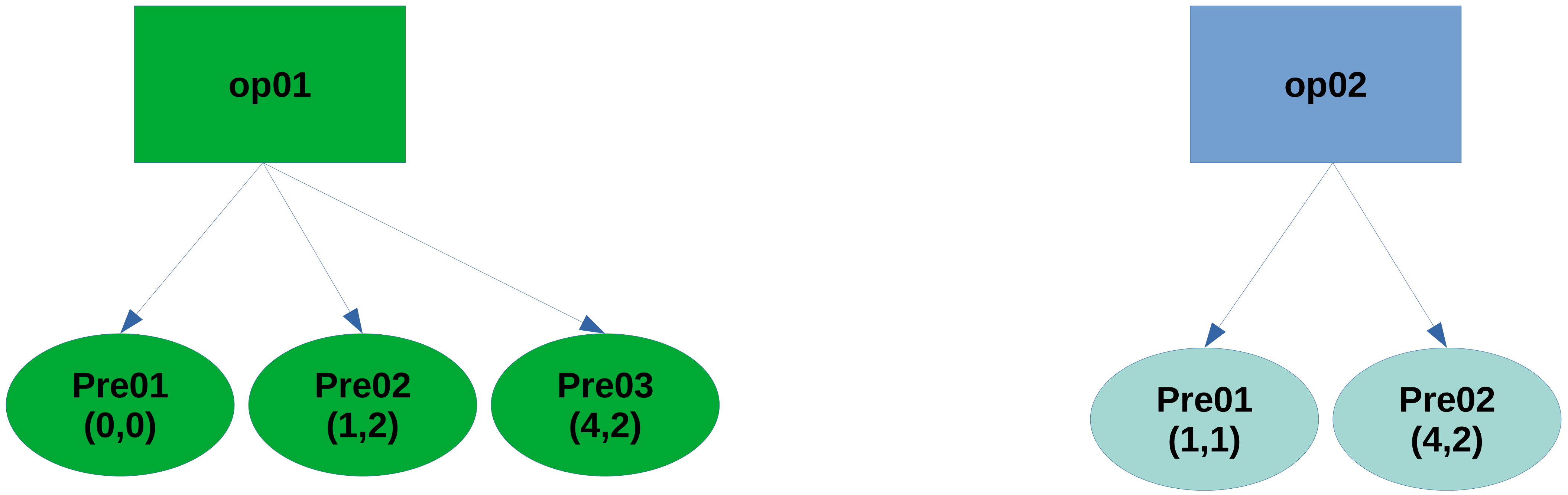
State:
(0, 2, 0, 2, 2)

Naive Successor Generator



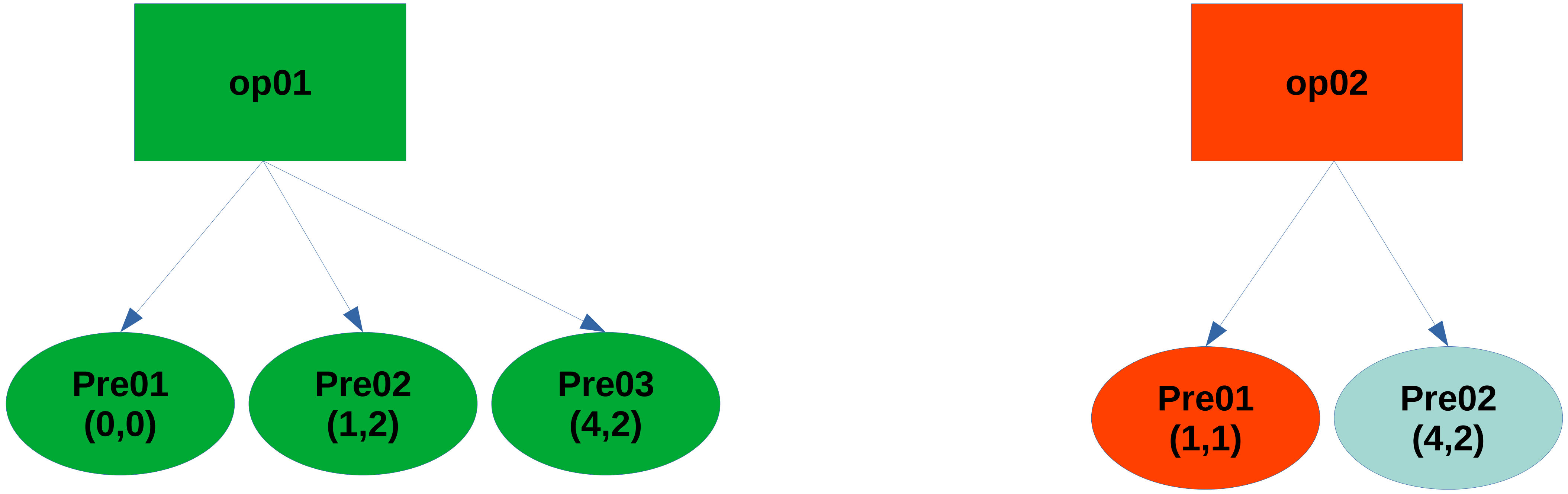
State:
(0, 2, 0, 2, 2)

Naive Successor Generator



State:
(0, 2, 0, 2, 2)

Naive Successor Generator



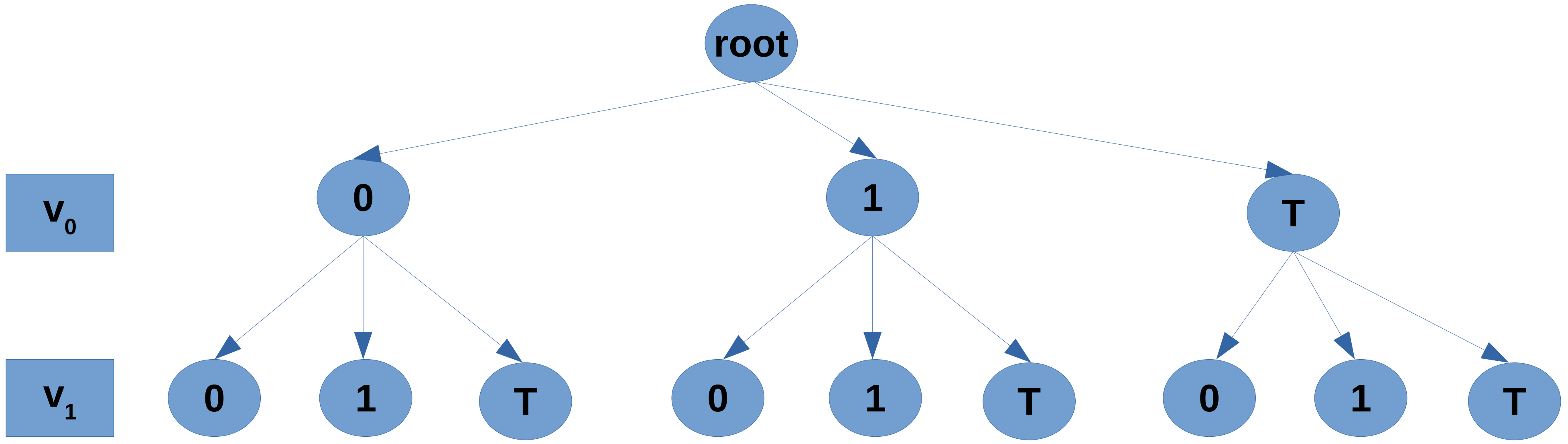
State:
(0, 2, 0, 2, 2)

Agenda.

Fast Downward Successor Generator

$V_0 = \{0, 1\}$

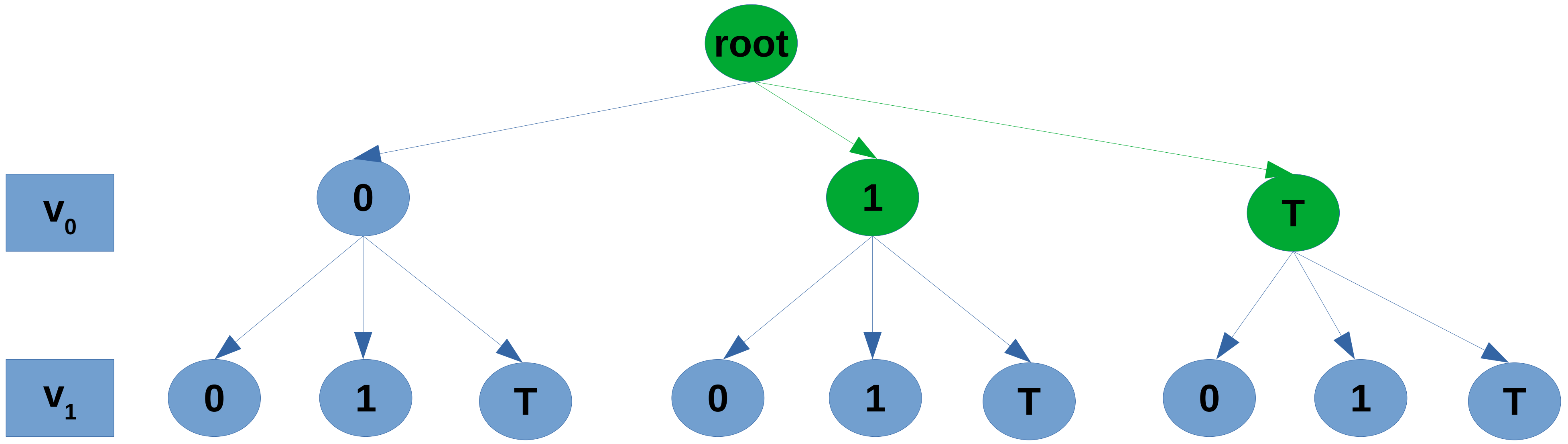
$v_1 = \{0, 1\}$



Fast Downward Successor Generator

$V_0 = \{0, 1\}$

$v_1 = \{0, 1\}$

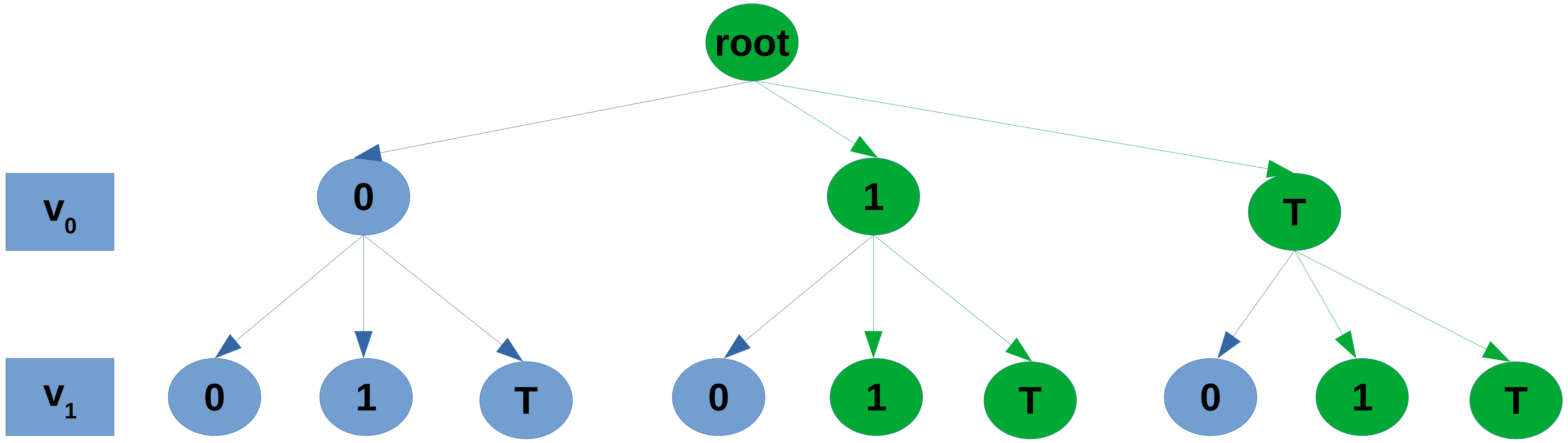


State:
(1,1)

Fast Downward Successor Generator

$V_0 = \{0, 1\}$

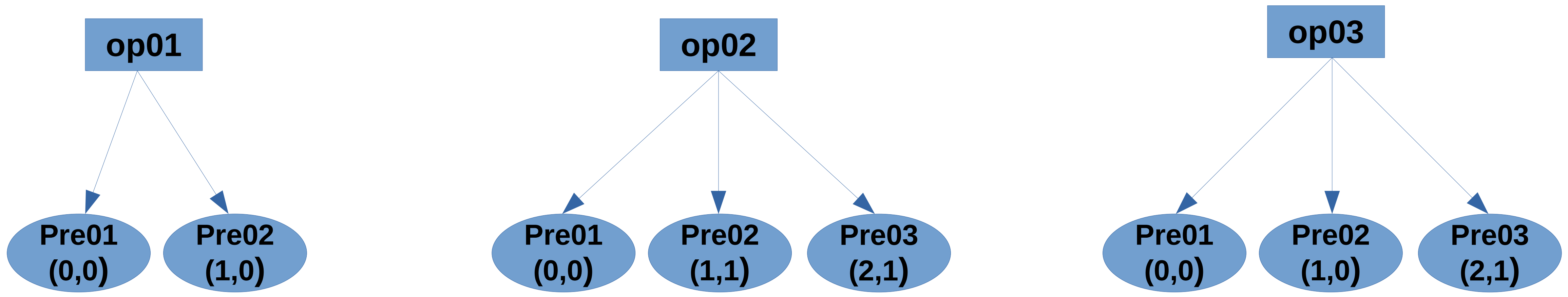
$v_1 = \{0, 1\}$



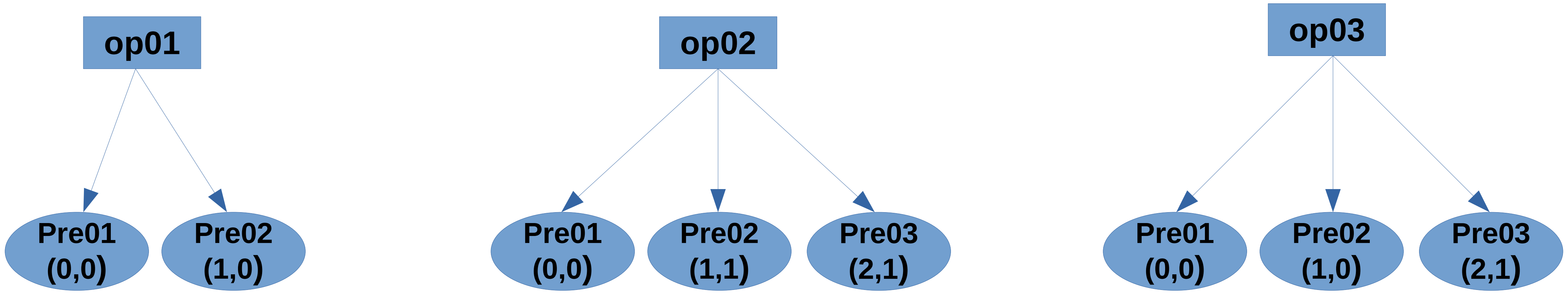
State:
(1,1)

Agenda.

Marking Successor Generator



Marking Successor Generator



State:
(0, 0, 1)

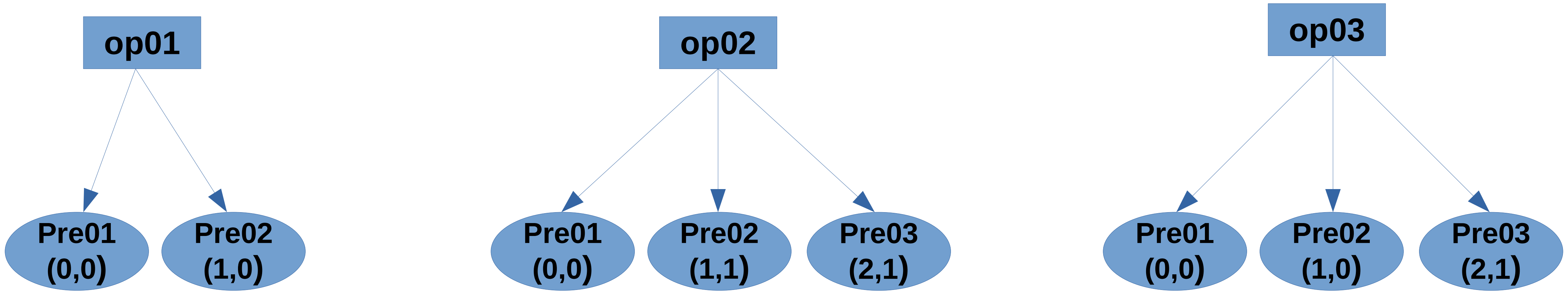
Counter: [2, 3, 3]

Precondition of:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01		op01	op02		op02
op02		op03			op03
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Marking Successor Generator



State:
(0, 0, 1)

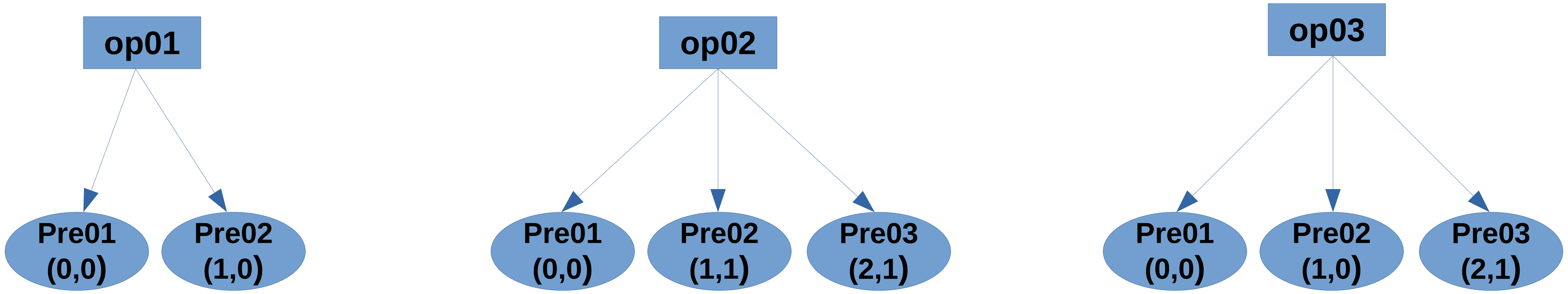
Counter: [2, 3, 3] → **Counter:** [1, 2, 2]

Precondition of:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01		op01	op02		op02
op02		op03			op03
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Marking Successor Generator



State:
(0, 0, 1)

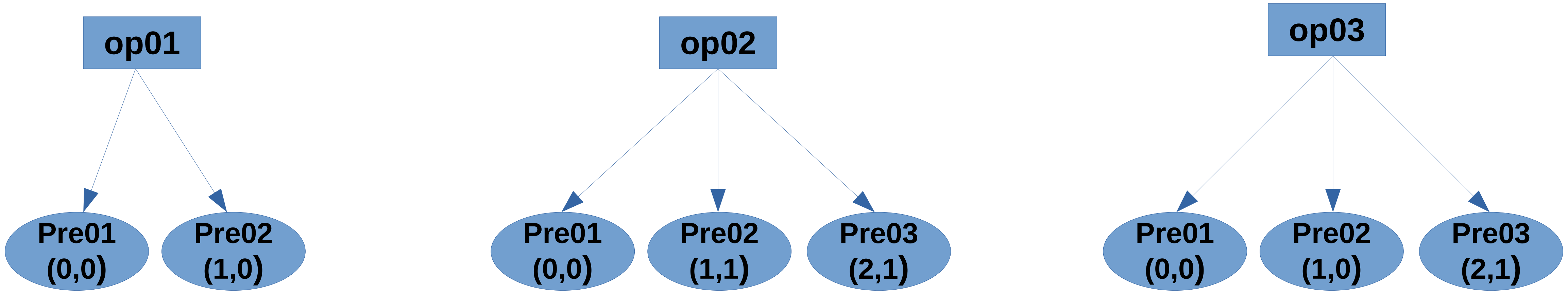
Counter: [1, 2, 2] → **Counter:** [0, 2, 1]

Precondition of:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01		op01	op02		op02
op02		op03			op03
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Marking Successor Generator



State:
(0, 0, <u>1</u>)

Counter: [0, 2, 1] → **Counter:** [0, 1, 0]

Precondition of:

(0,0)	**(0,1)**	**(1,0)**	**(1,1)**	**(2,0)**	**(2,1)**
op01		op01	op02		op02
op02		op02			op03
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Agenda.

PSVN Successor Generator

Vertex:

- Plausible operators
- Variable assignments
- Satisfied operators
- Children
- Choice
- Hash

PSVN Successor Generator

Vertex:

- Plausible operators
- Variable assignments
- Satisfied operators
- Children
- Choice
- Hash

Idea:

- Choose variable v, which has not been assigned
- For each value in D_v a outgoing edge
- For each outgoing edge, new vertex, with values from parent
- Apply value to plausible operators and split (sat/unsat/plaus) and remove satisfied precons
- Remove variable assignments which aren’t referenced anymore
- Check if vertex exists
  - If yes: edge goes to this one, stop recursion
  - If no: create new vertex and continue
- If DAG too big, restart and split operators in half

PSVN Successor Generator

Vertex:

- Plausible operators
- Variable assignments
- Satisfied operators
- Children
- Choice
- Hash

Op01: {(0,0), (1,1)}

Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator

Plaus: [1,2,3,4]
Vars: [-1, -1]
Sat: []
Choice: -1
Hash: ####

Op01: {(0,0), (1,1)}

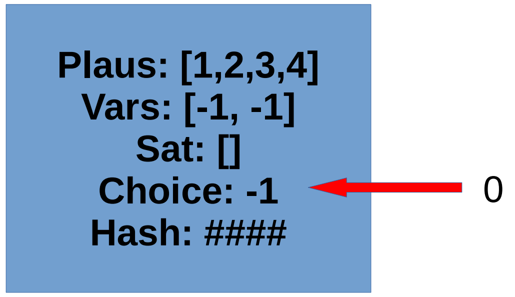
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variables Domains:
({0,1}, {0,1})

PSVN Successor Generator



Op01: {(0,0), (1,1)}

Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator

Plaus: [1,2,3,4]
Vars: [-1, -1]
Sat: []
Choice: 0
Hash: ####

Op01: {(0,0), (1,1)}

Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator

Plaus: [1,2,3,4]
Vars: [-1, -1]
Sat: []
Choice: 0
Hash: #####

Create Children

Op01: {(0,0), (1,1)}

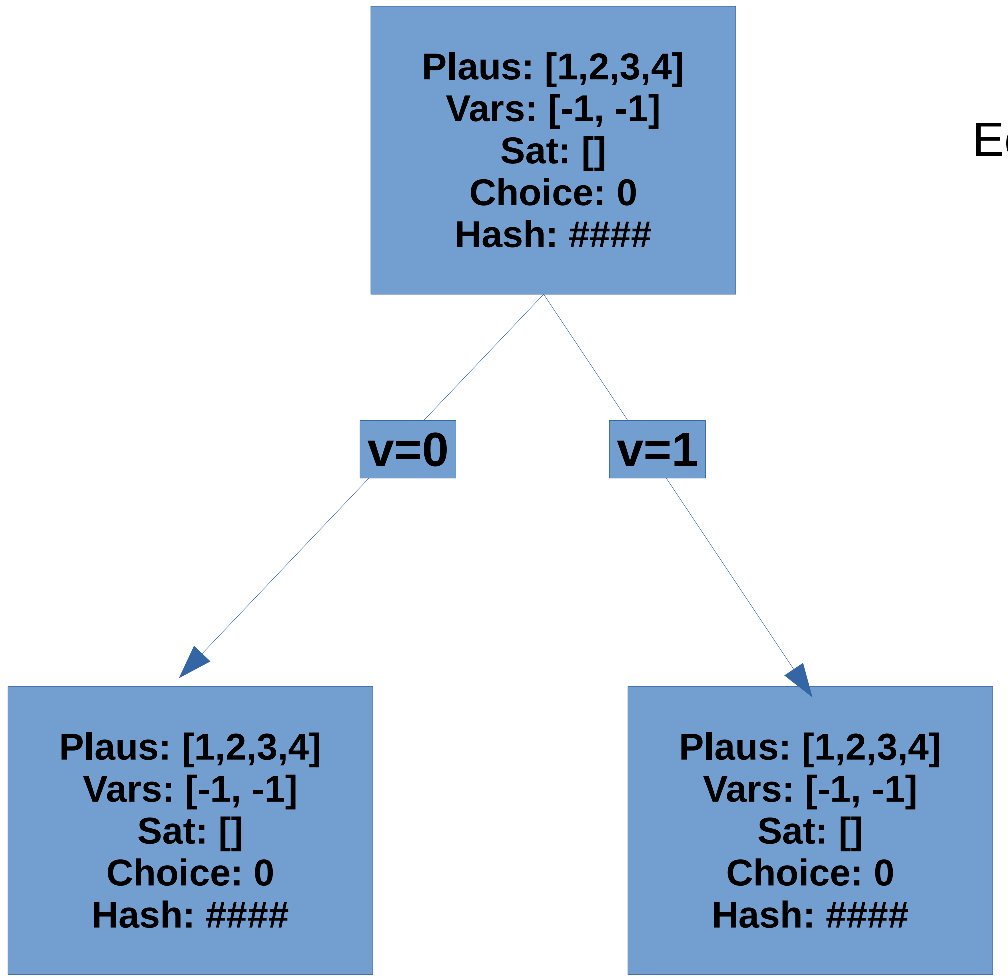
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Edit Children

Op01: {(0,0), (1,1)}

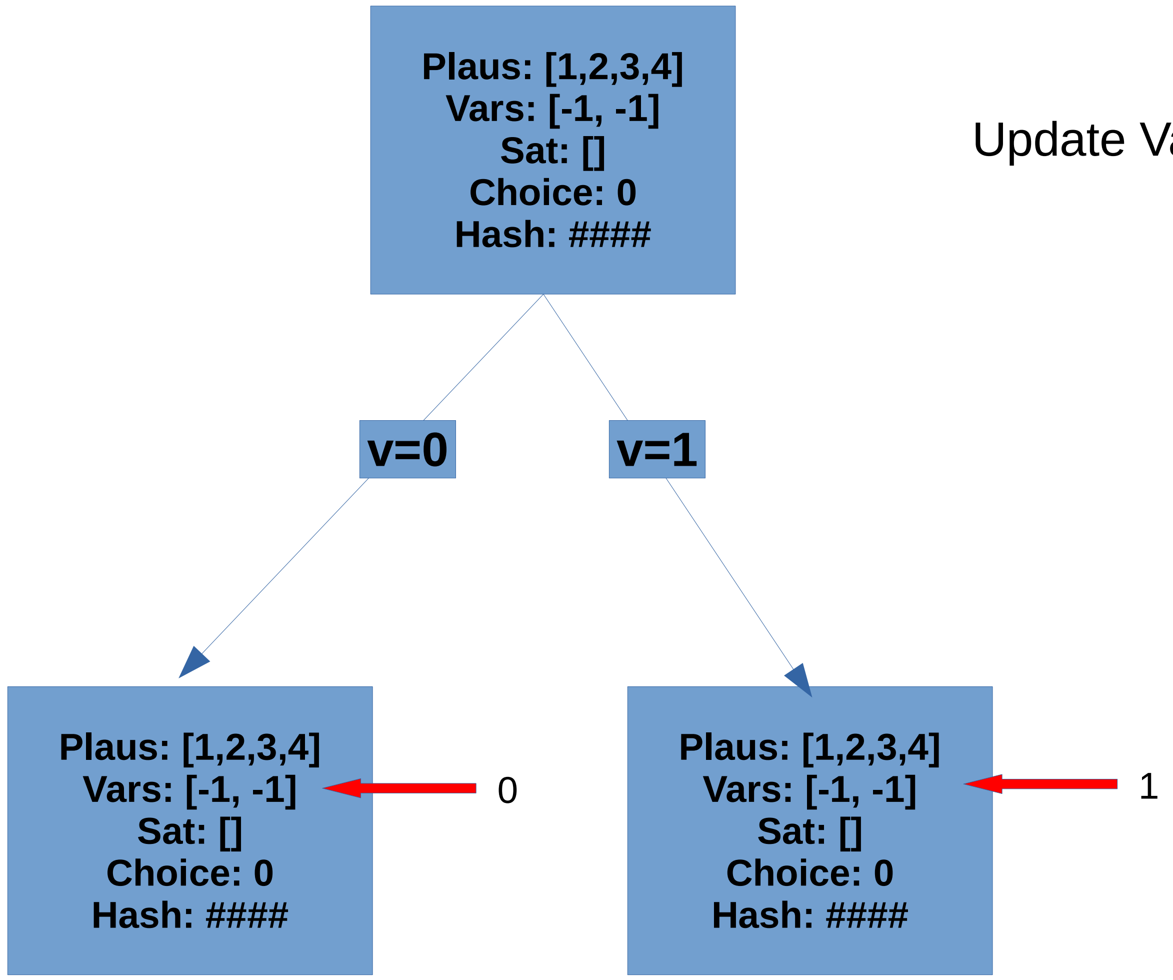
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Update Vars

Op01: {(0,0), (1,1)}

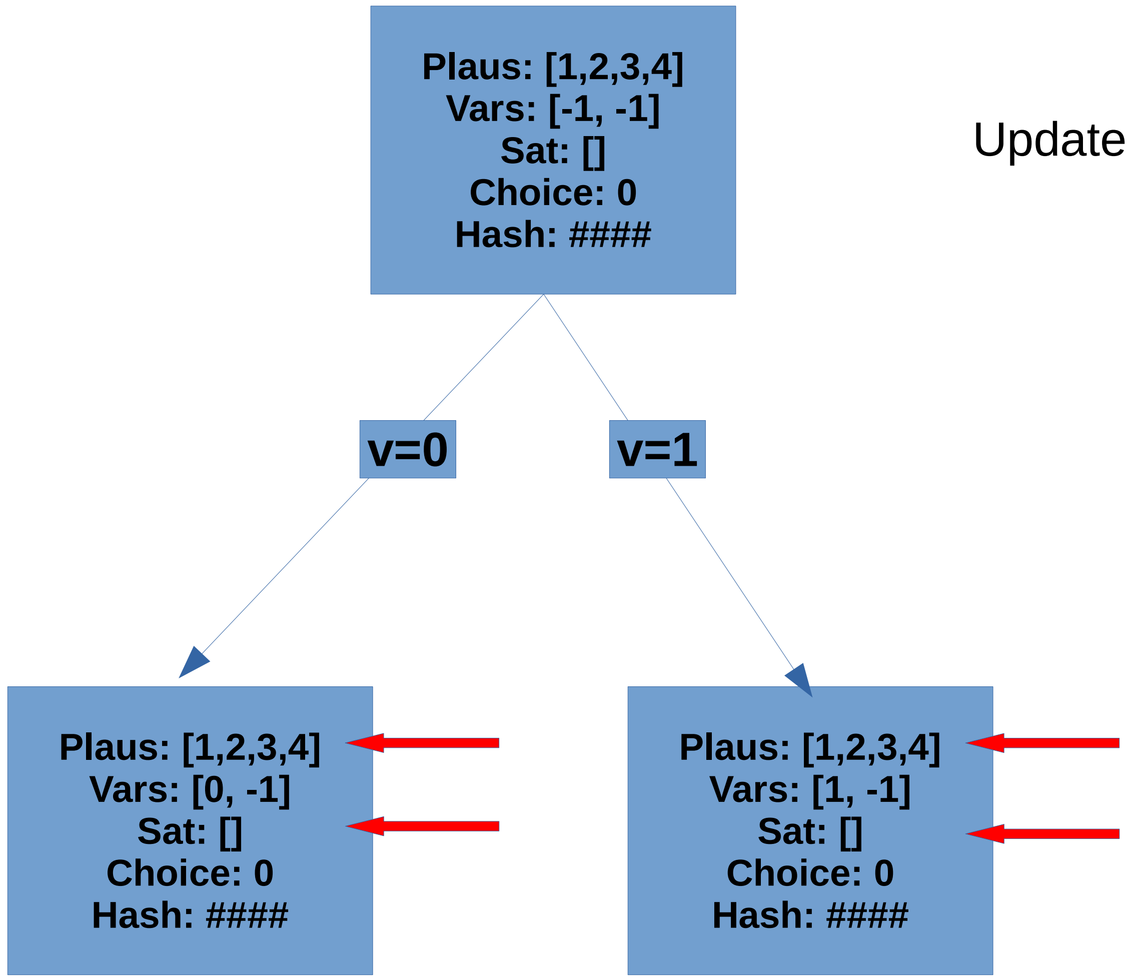
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Update Ops

Op01: {(0,0), (1,1)}

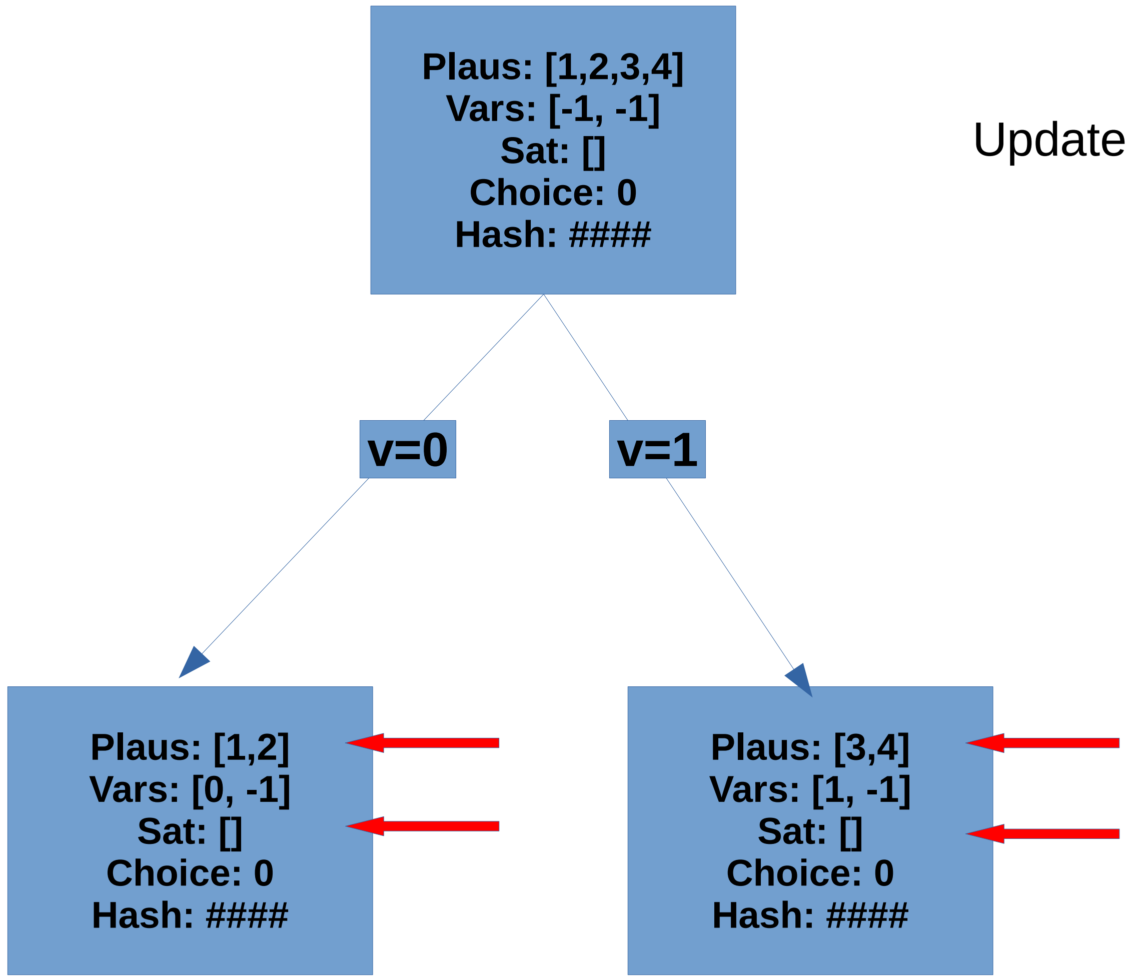
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Update Ops

Op01: {(0,0), (1,1)}

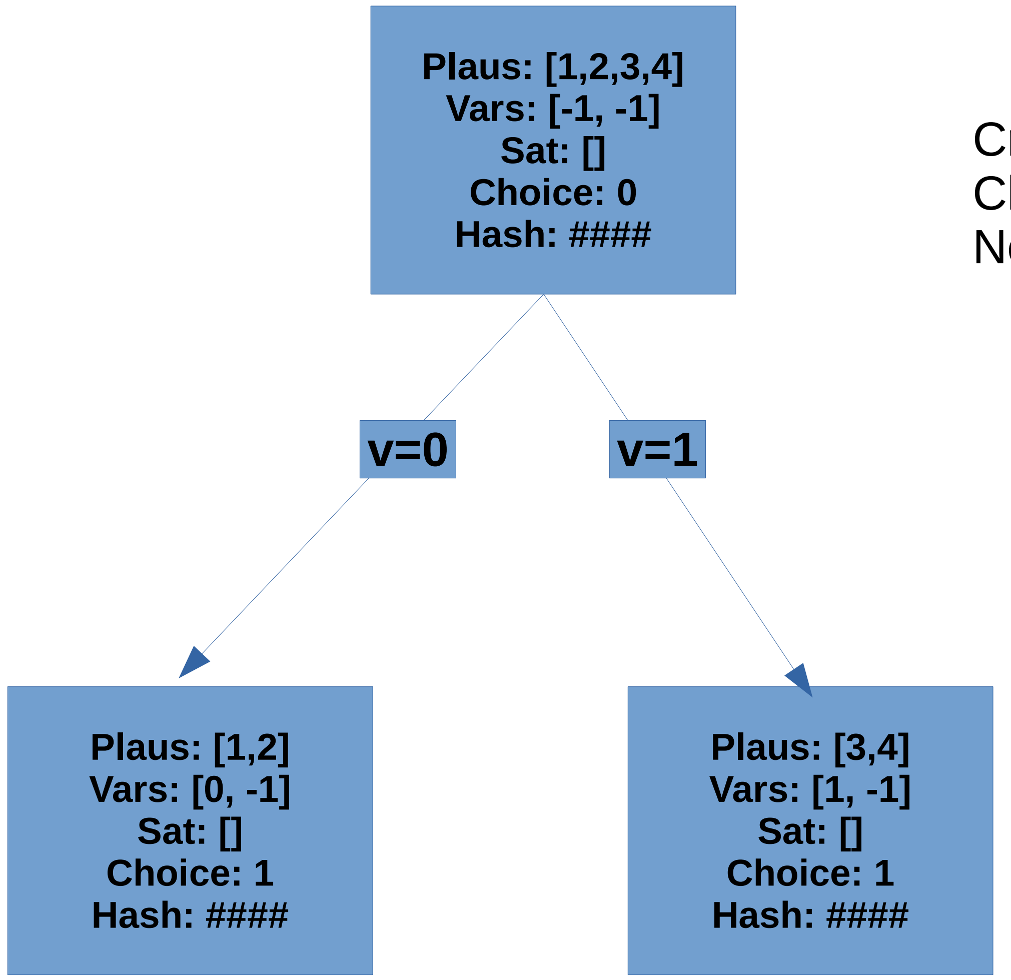
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Create Hash &
Check for Existence &
New Choice

Op01: {(0,0), (1,1)}

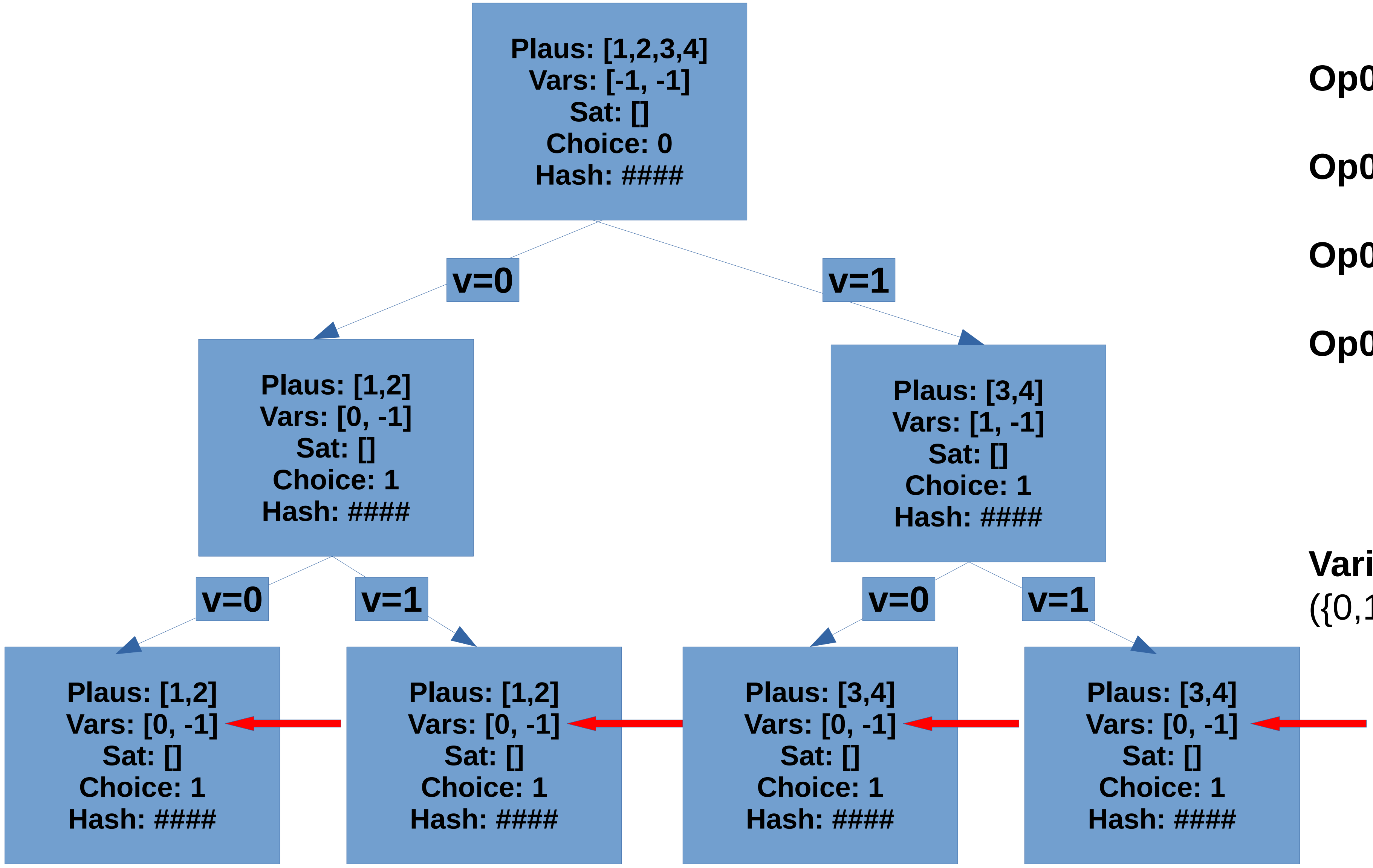
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Op01: {(0,0), (1,1)}

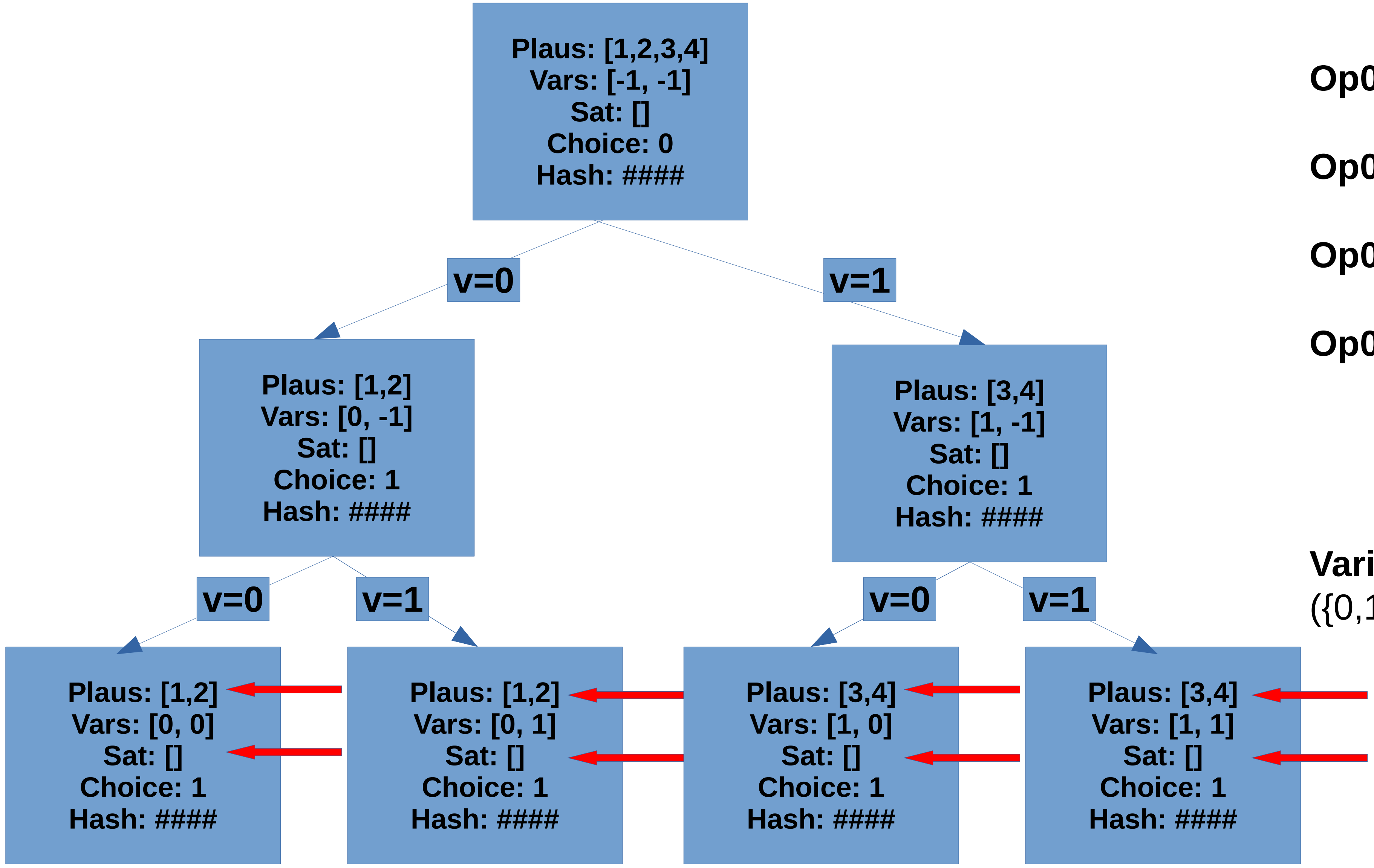
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Op01: {(0,0), (1,1)}

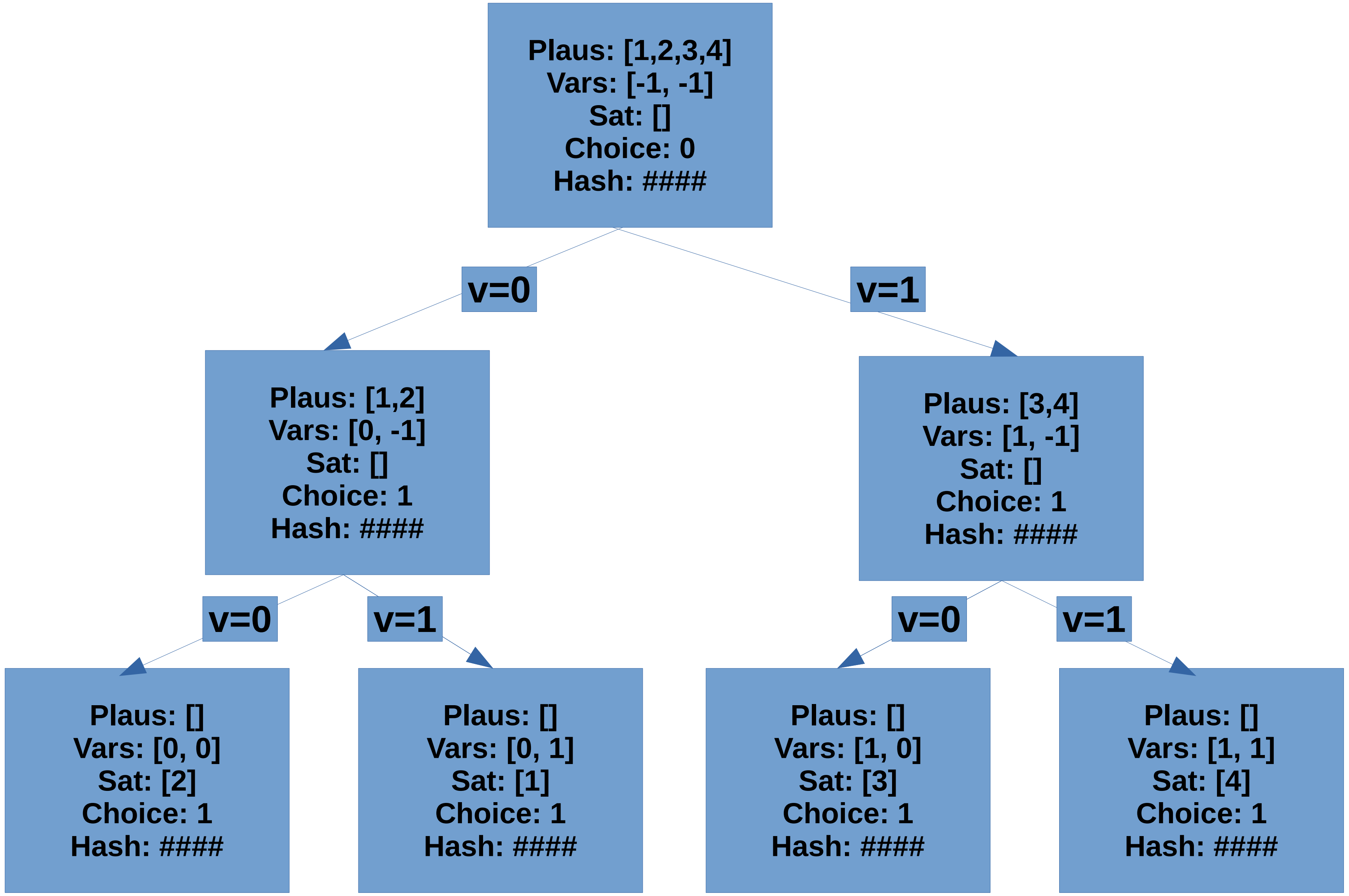
Op02: {(0,0), (1,0)}

Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

PSVN Successor Generator



Op01: {(0,0), (1,1)}

Op02: {(0,0), (1,0)}

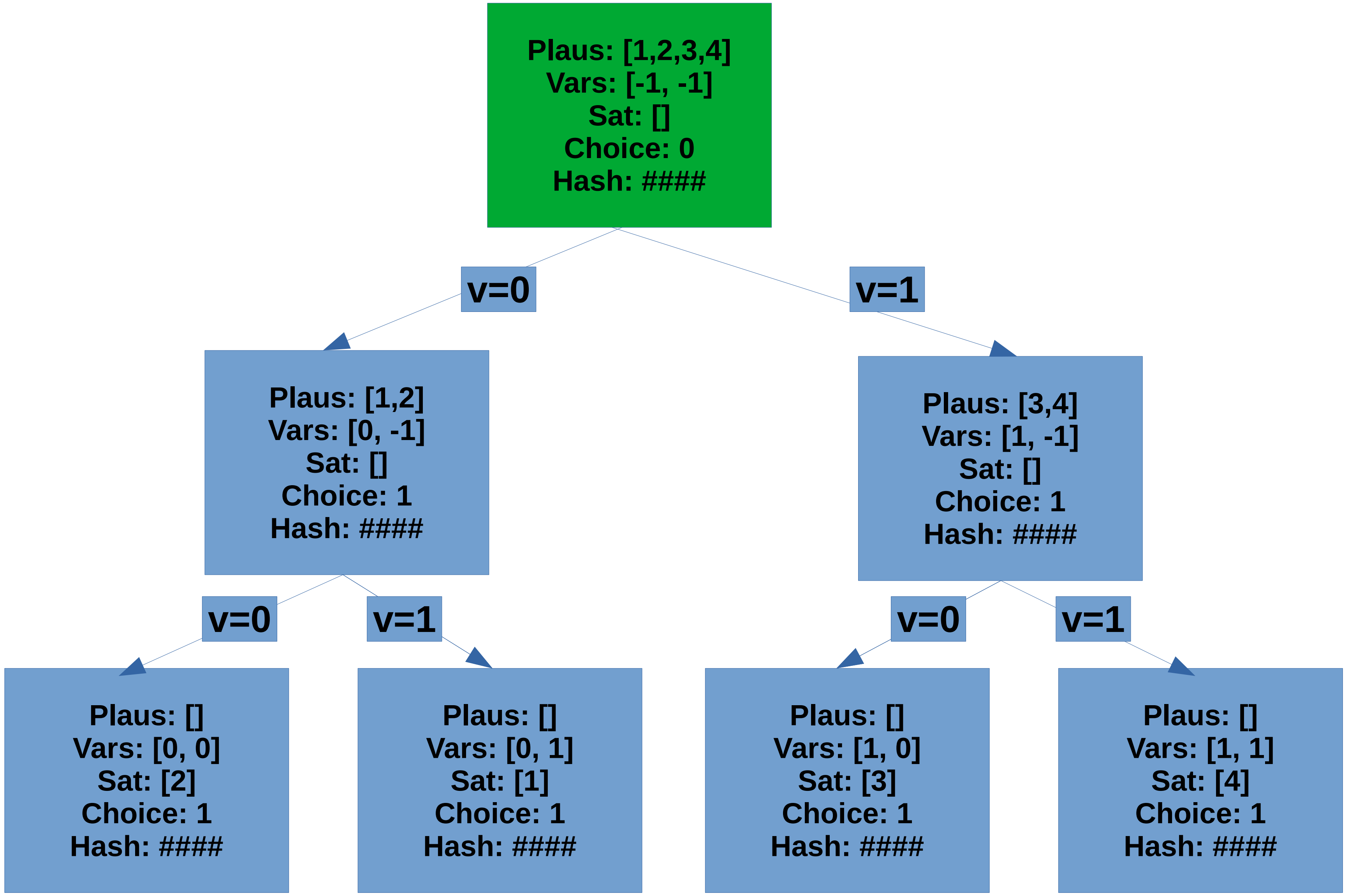
Op03: {(0,1), (1,0)}

Op04: {(0,1), (1,1)}

Variable Domains:
({0,1}, {0,1})

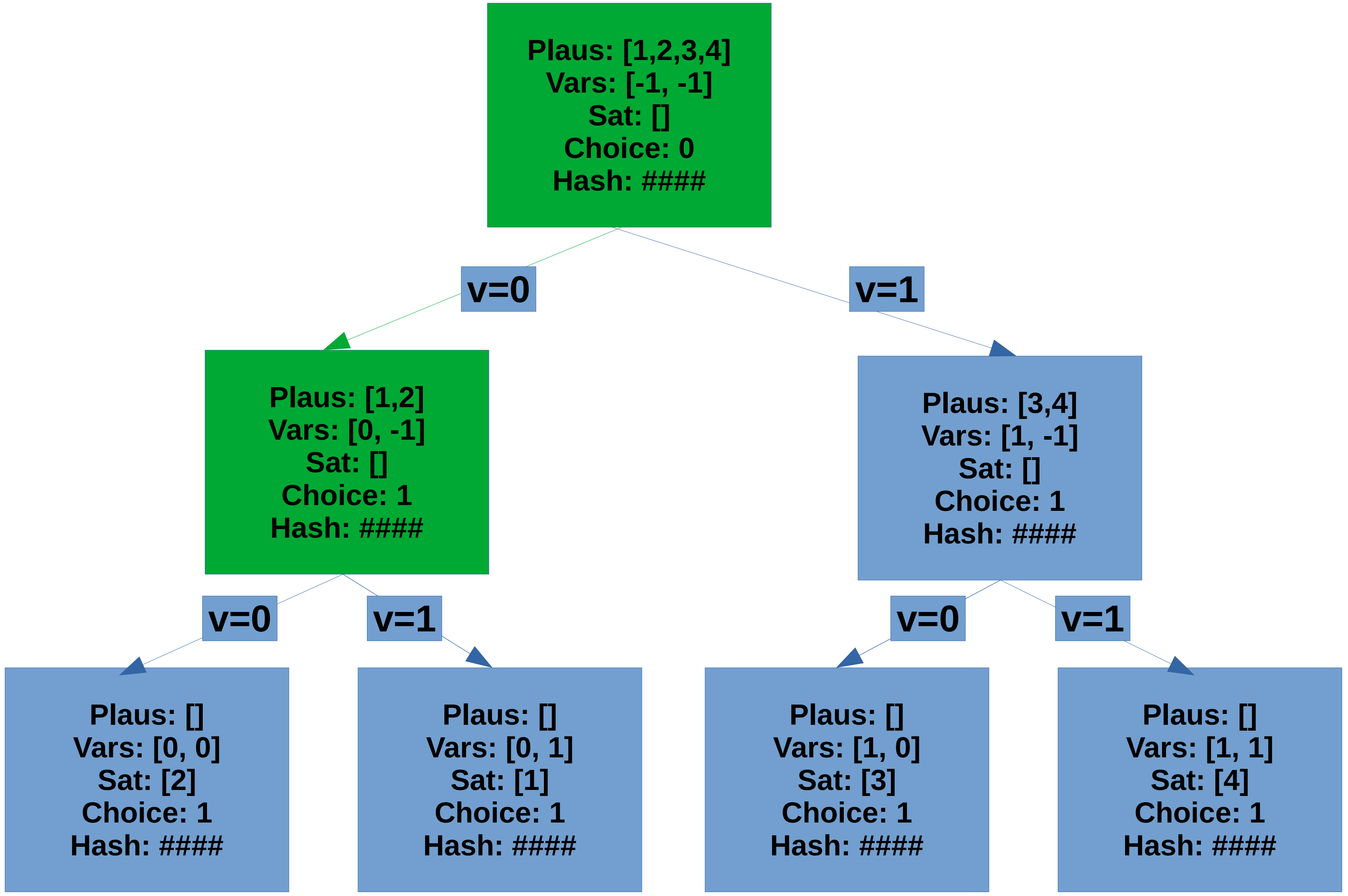
PSVN Successor Generator

State:
(0, 0)



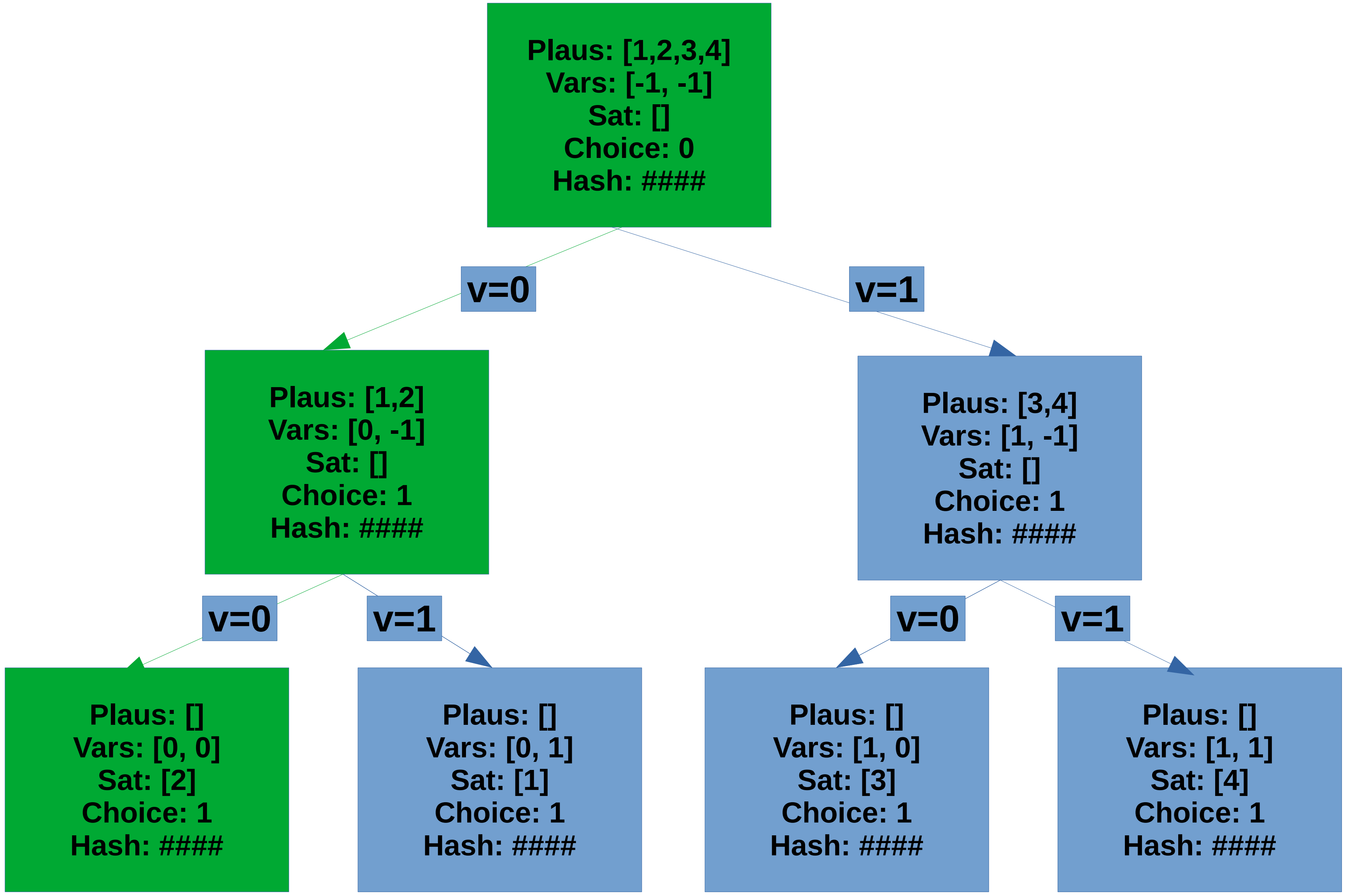
PSVN Successor Generator

State:
(0, 0)



PSVN Successor Generator

State:
(0, 0)



Agenda.

Watched Literals Successor Generator

SAT Solving:

- $(a \lor c) \land (\neg b \lor a) \land (\neg a \lor c \lor d)$

Watched Literals Successor Generator

SAT Solving:
- $(a \lor c) \land (\neg b \lor a) \land (\neg a \lor c \lor d)$

Backtracking algorithm:
- Assign unassigned literal
- Check if satisfied
- If not, assign other value and check again
  → creates two sub problems

Watched Literals Successor Generator

SAT Solving:

- (a ∨ c) ∧ (¬b ∨ a) ∧ (¬a ∨ c ∨ d)

Backtracking algorithm:

- Assign unassigned literal
- Check if satisfied
- If not, assign other value and check again
  → creates two sub problems

Update to DPLL:

- If only one literal left, we know how to assign that variable (unit propagation)
- If clause is satisfied, it stays that way

Watched Literals Successor Generator

SAT Solving:
- $(a \lor c) \land (\neg b \lor a) \land (\neg a \lor c \lor d)$

Backtracking algorithm:
- Assign unassigned literal
- Check if satisfied
- If not, assign other value and check again
  → creates two sub problems

Update to DPLL:
- If only one literal left, we know how to assign that variable (unit propagation)
- If clause is satisfied, it stays that way

Improving DPLL → 2 Watched Literals:
- Only want to know if one literal left for unit propagation
- Watch two literals:
  - If one satisfied, then clause is satisfied
  - If one unsatisfied, choose new unassigned to watch
  - If not possible → unit propagation

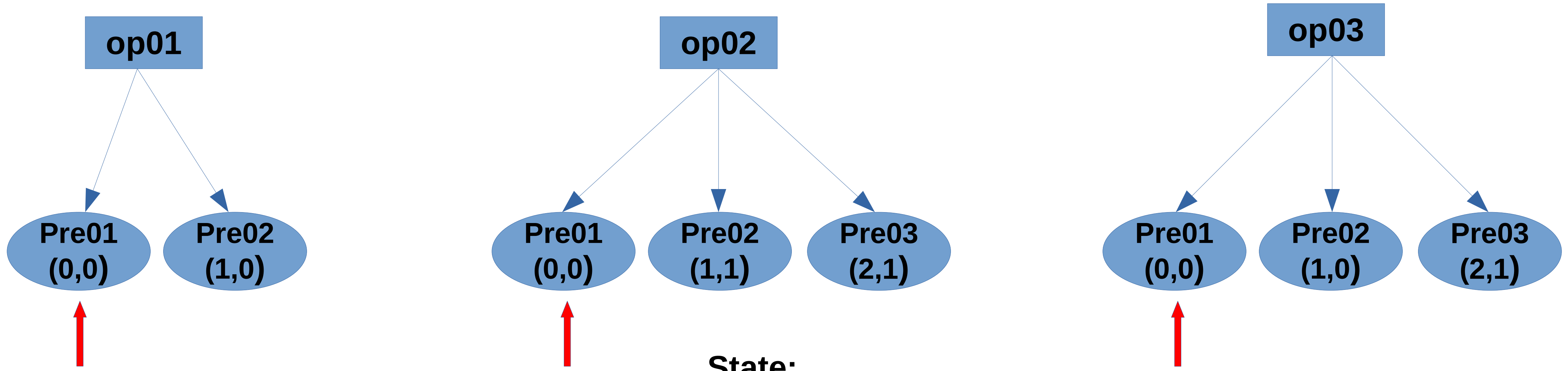
Watched Literals Successor Generator

Adaption:

- All preconditions must be satisfied

- When checking state:
  - For each variable assignment in the state:
    - Check each operator watching that variable assignment
      - If any precondition unsatisfied, watch unsatisfied precondition
      - If all preconditions satisfied, operator is applicable

Watched Literals Successor Generator



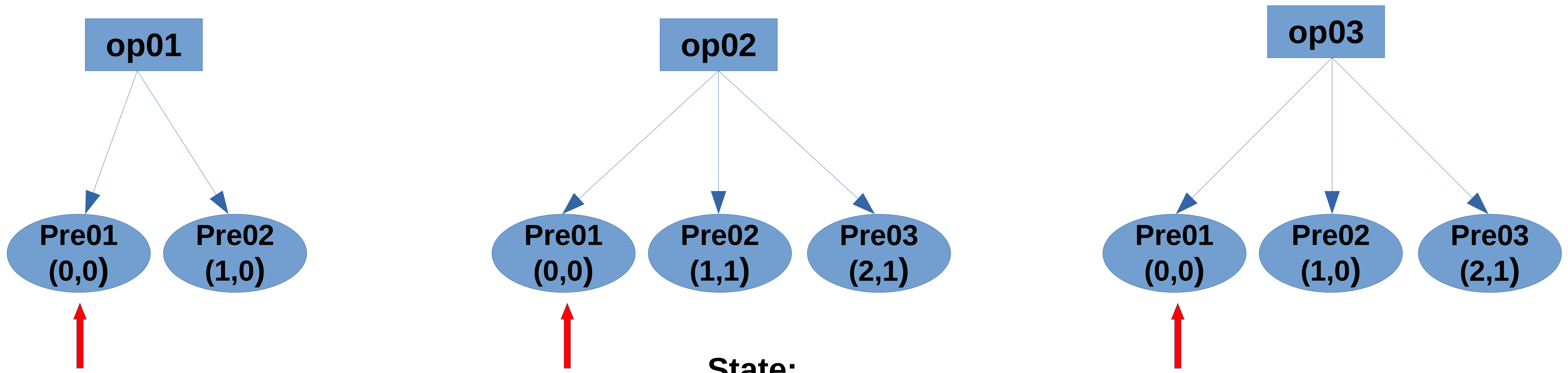
State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01					
op02					
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Watched Literals Successor Generator



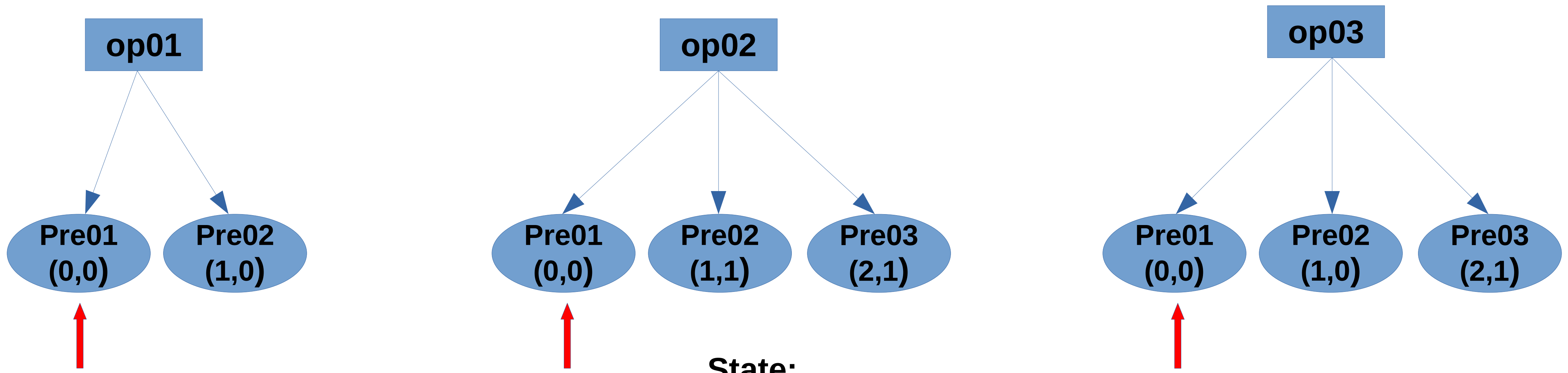
State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01 ←					
op02					
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Watched Literals Successor Generator



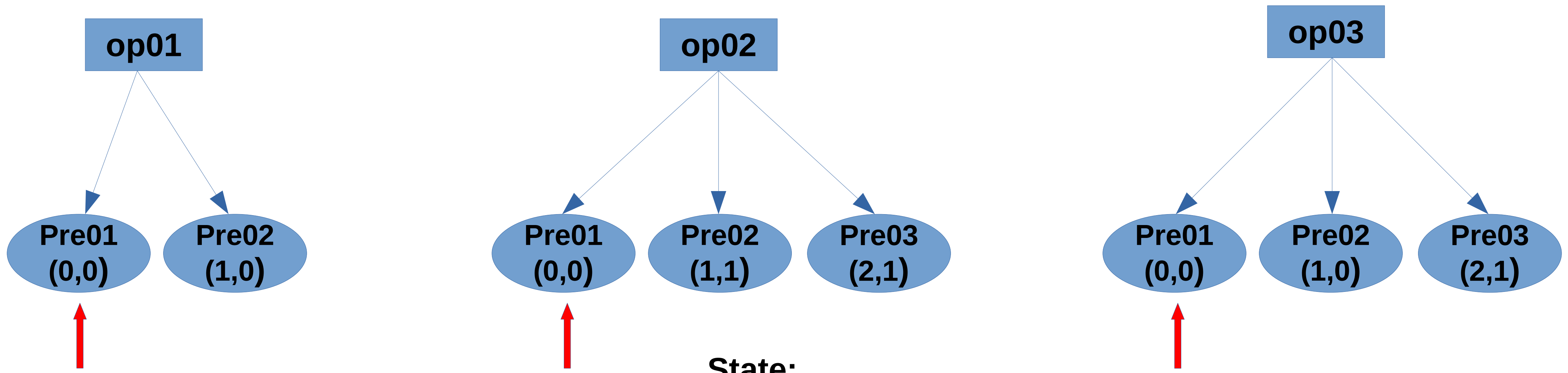
State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01	←	applicable			
op02					
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Watched Literals Successor Generator



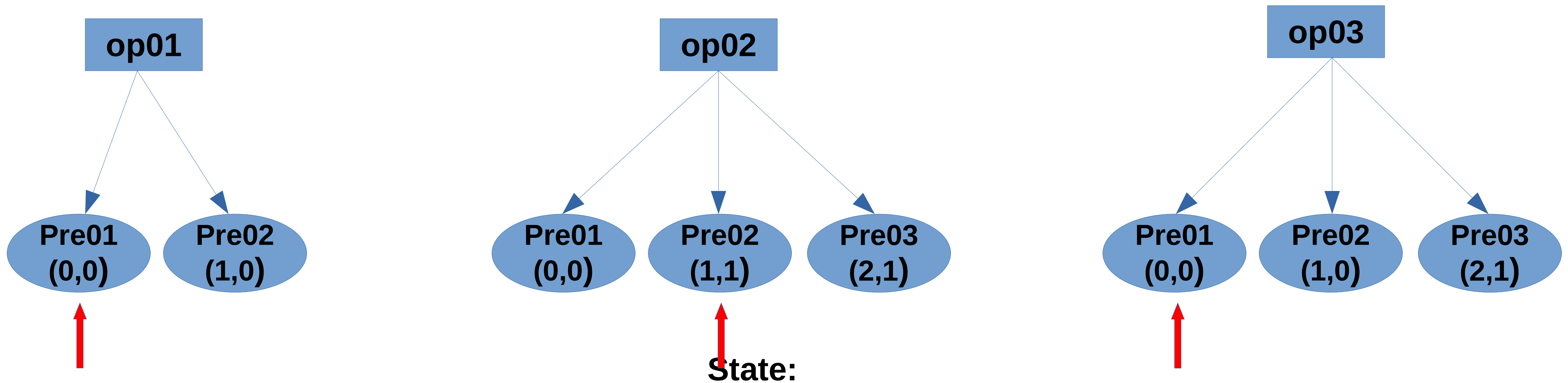
State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01					
op02 ←	Pre02 !!				
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Watched Literals Successor Generator



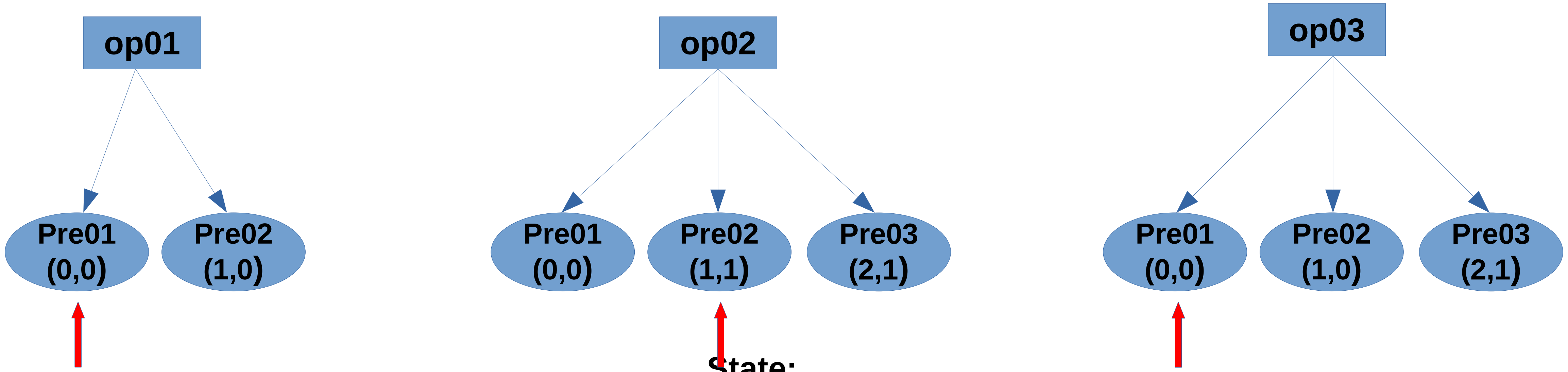
State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01			op02		
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Watched Literals Successor Generator



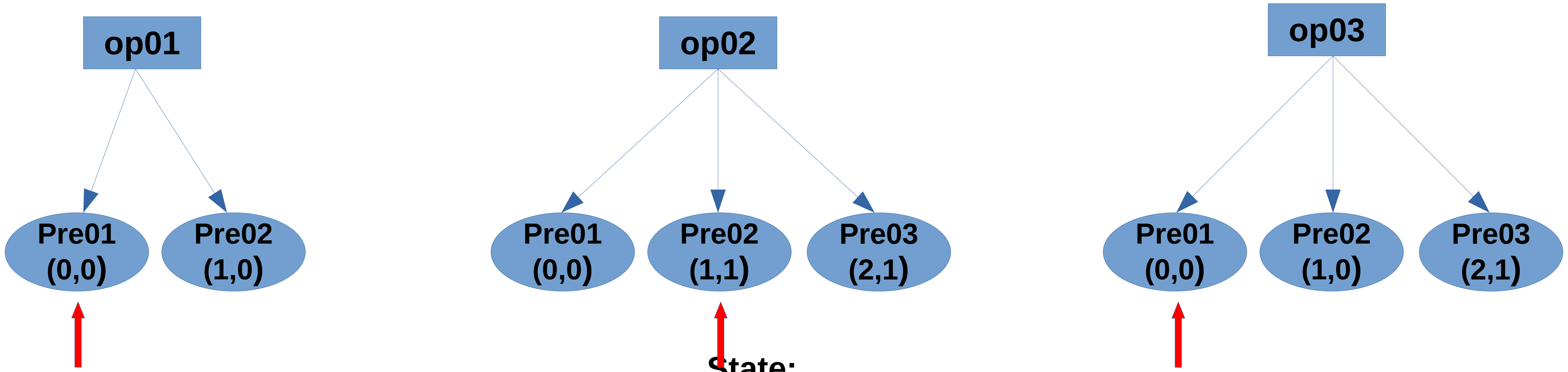
State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01			op02		
op03 ← applicable					

Variable Domains:
({0,1}, {0,1}, {0,1})

Watched Literals Successor Generator



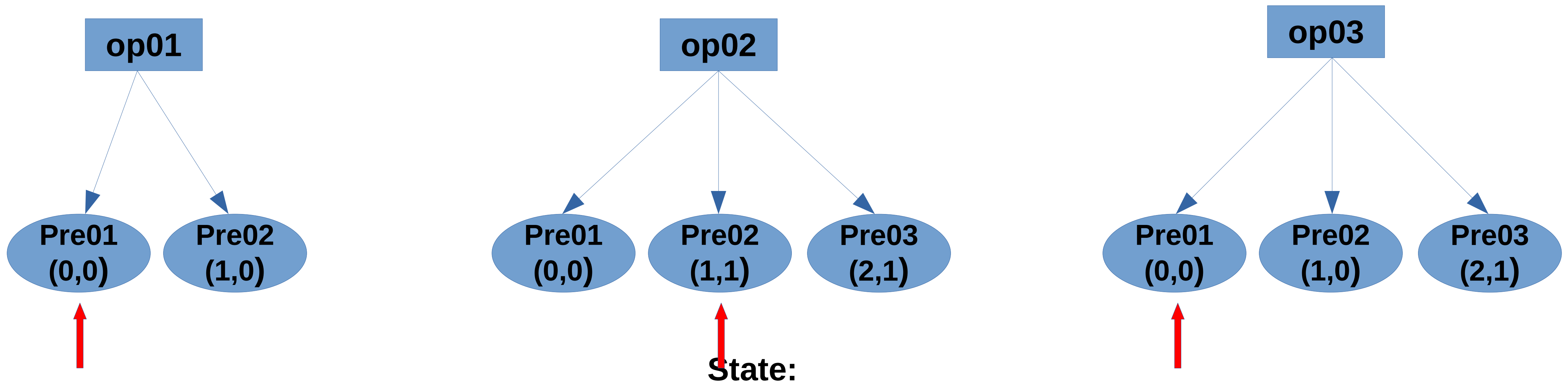
State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01			op02		
op03					

Variable Domains:
({0,1}, {0,1}, {0,1})

Watched Literals Successor Generator



State:
(0, 0, 1)

Watcher:

(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)
op01			op02		🟨
op03					🟨
					🟨

Variable Domains:
({0,1}, {0,1}, {0,1})

Agenda.

Evaluation – How was tested

- A* with blind search
- 1827 different planning tasks
  from 65 different domains

Evaluation - Results

Summary Unbound	Fast Downward	PSVN	Marking	Watched Literals	Naive
Coverage	712	253	680	658	689
Out Of Memory	1’098	1’522	1’006	866	773
Out Of Time	0	0	94	256	348
SG Init Time	0.08	335.54	0.59	0.02	0.01
GAO Time	841.53	873.75	1’592.82	3’079.83	3’735.03
GAO Mean	0.0014	0.0014	0.0026	0.0050	0.0061
Total Time - Mean	0.09	0.51	0.10	0.11	0.12

Evaluation – Conclusion

- No precomputation → faster init time, less out of memory
- A lot precomputation → faster GAO time, less out of time
- Trade off between faster initialization and faster GAO
- Choose correct successor generator for planning task!

University
of Basel

Thank you
for your attention.

Questions?