fc
Forward Checking

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

A solution is 3--3--2--1

Remember how bt thrashed?

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2
V3
V4
V5
V6
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3
V4
V5
V6
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4
V5
V6
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5
V6
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 1
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 1
V7 = 1
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 1
V7 = 2
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 1
V7 = 3
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 2
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 2
V7 = 1
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 2
V7 = 2
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 2
V7 = 3
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 3
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 3
V7 = 1
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 3
V7 = 2
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 1
V6 = 3
V7 = 3
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 1
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 1
V7 = 1
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 1
V7 = 2
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 1
V7 = 3
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 2
V7
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 2
V7 = 1
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 2
V7 = 2
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 2
V7 = 3
V8
V9
V10

Consider the following problem (csp5)

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

V1 = 1
V2 = 1
V3 = 1
V4 = 1
V5 = 2
V6 = 3
V7
V8
V9
V10

Thrashing:

Slavishly repeating the same set of actions with the same set of outcomes.

Can we minimise thrashing?

Forward Checking

Rather than checking backwards (from current to past)
check forwards (from current to future)

- When we instantiate v[i] with a value x
    - remove from the d[j] values inconsistent with v[i] = x
    - where j is in the future

Consequently, when we instantiate v[i] we know it is compatible with the past

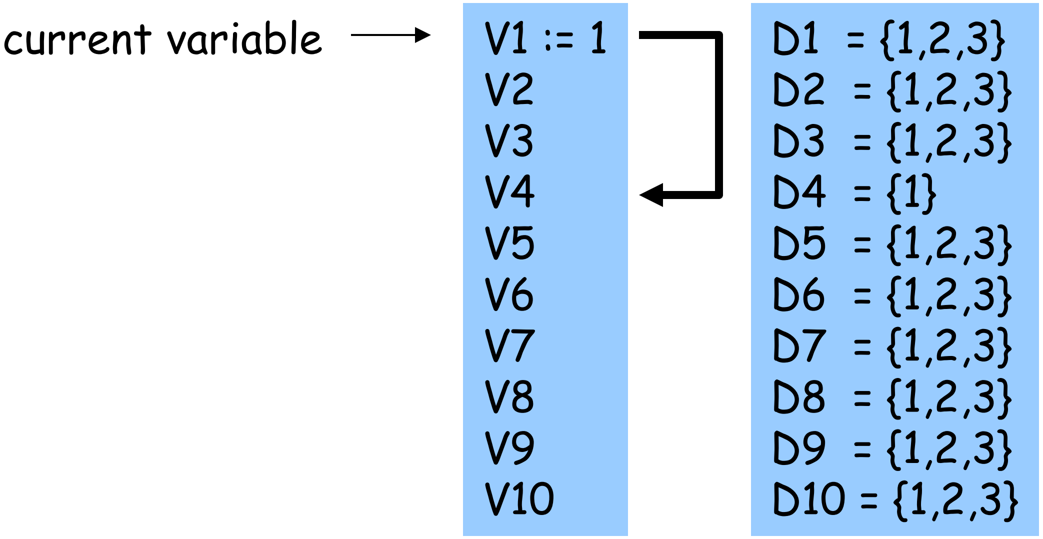
Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

V1	D1 = {1,2,3}
V2	D2 = {1,2,3}
V3	D3 = {1,2,3}
V4	D4 = {1,2,3}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {1,2,3}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable →

V1 := 1
V2
V3
V4
V5
V6
V7
V8
V9
V10

D1 = {1,2,3}
D2 = {1,2,3}
D3 = {1,2,3}
D4 = {1}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- instantiate V1 with value 1

- remove from D4 incompatible values {2,3}

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

current variable →

V1 := 1	D1 = {1,2,3}
V2 := 1	D2 = {1,2,3}
V3	D3 = {1,2,3}
V4	D4 = {1}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {1,2,3}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

- instantiate V2 with value 1

- no forward checking to perform

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

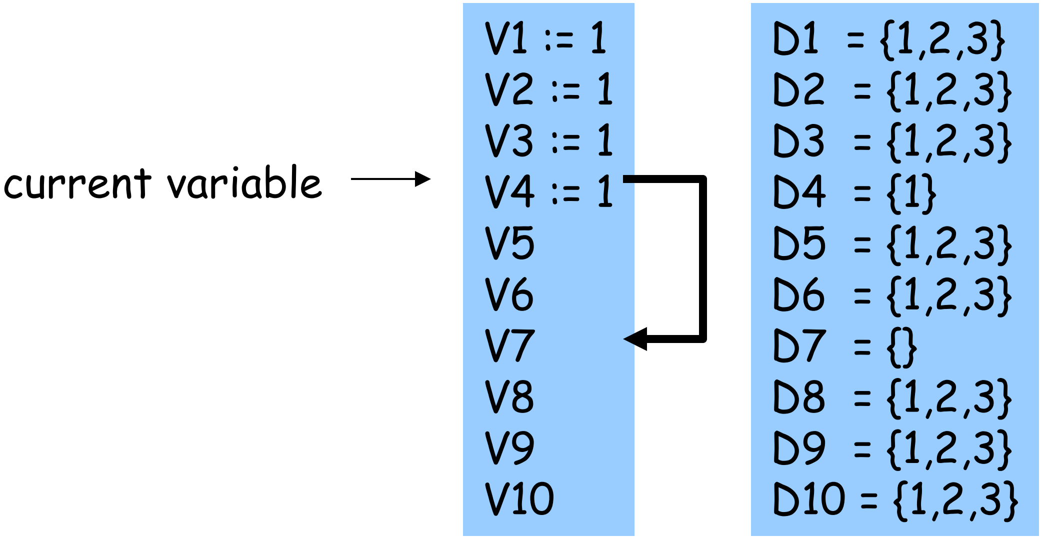
	V1 := 1	D1 = {1,2,3}
	V2 := 1	D2 = {1,2,3}
current variable →	V3 := 1	D3 = {1,2,3}
	V4	D4 = {1}
	V5	D5 = {1,2,3}
	V6	D6 = {1,2,3}
	V7	D7 = {1,2,3}
	V8	D8 = {1,2,3}
	V9	D9 = {1,2,3}
	V10	D10 = {1,2,3}

- instantiate V3 with value 1

- no forward checking to perform

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

V1 := 1	D1 = {1,2,3}
V2 := 1	D2 = {1,2,3}
V3 := 1	D3 = {1,2,3}
V4 := 1	D4 = {1}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

- instantiate V4 with value 1 (no choice!)
- remove from D7 incompatible values {1,2,3}

Dead end!

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

	V1 := 1	D1 = {1,2,3}
	V2 := 1	D2 = {1,2,3}
current variable →	V3 := 2	D3 = {2,3}
	V4	D4 = {1}
	V5	D5 = {1,2,3}
	V6	D6 = {1,2,3}
	V7	D7 = {1,2,3}
	V8	D8 = {1,2,3}
	V9	D9 = {1,2,3}
	V10	D10 = {1,2,3}

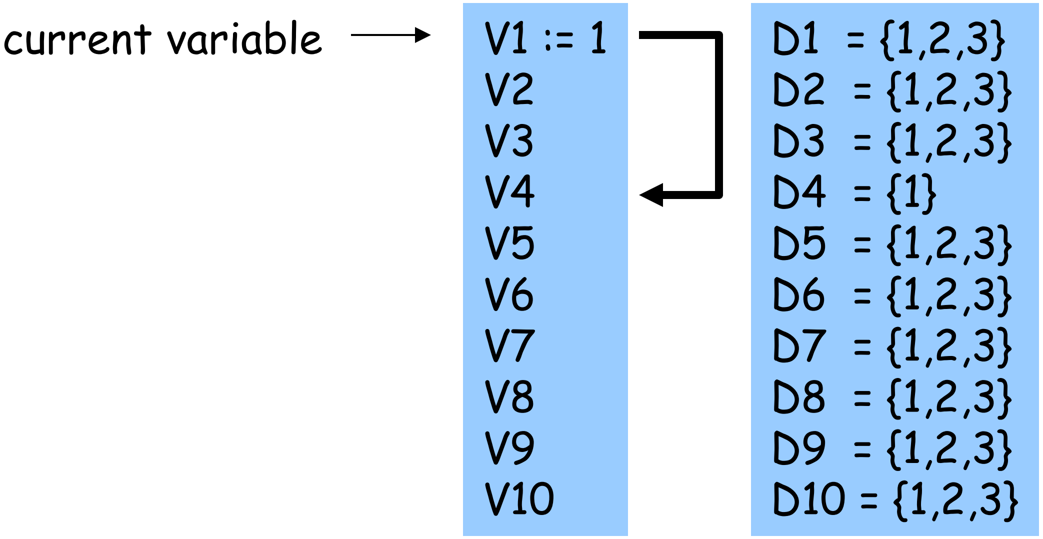
- backtrack to v3

We are still going to thrash!

But, could you see how we could “heuristically” exploit the FC information?

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

V1 := 1
V2
V3
V4
V5
V6
V7
V8
V9
V10

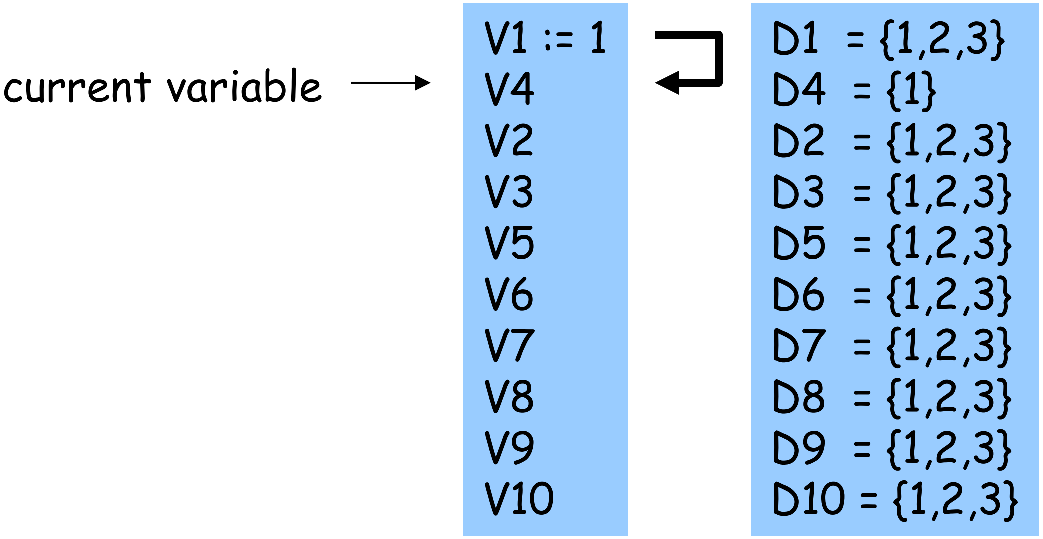
D1 = {1,2,3}
D2 = {1,2,3}
D3 = {1,2,3}
D4 = {1}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- instantiate V1 with value 1
- remove from D4 incompatible values {2,3}

Why not select V4 immediately after V1?

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

V1 := 1
V4
V2
V3
V5
V6
V7
V8
V9
V10

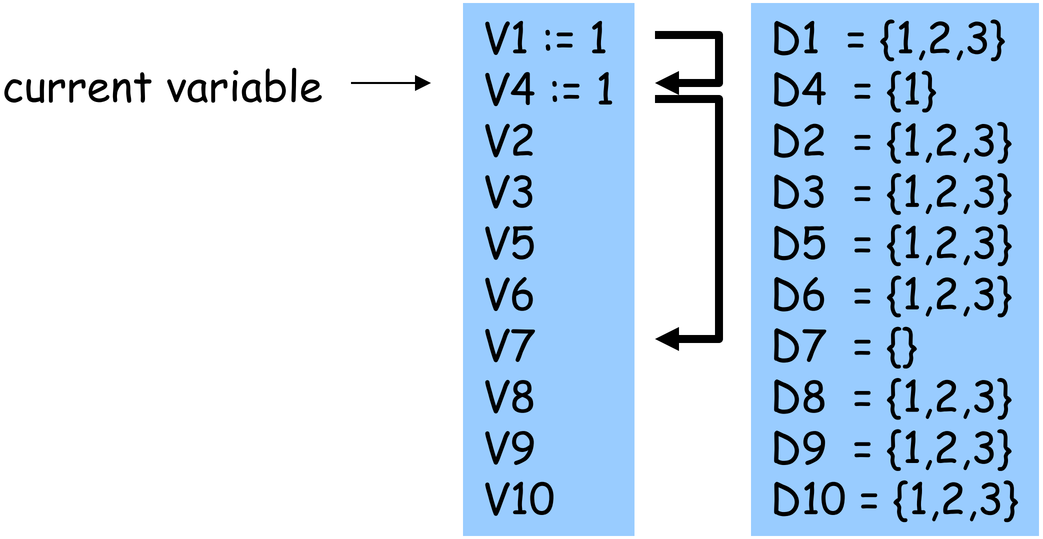
D1 = {1,2,3}
D4 = {1}
D2 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- select as current variable the variable with smallest domain

- instantiate V4

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

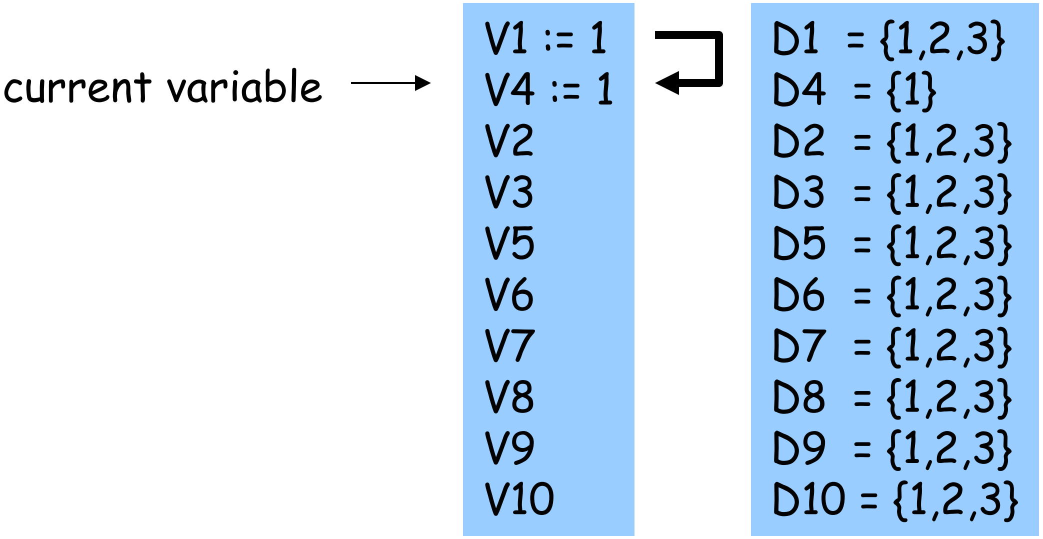
V1 := 1	D1 = {1,2,3}
V4 := 1	D4 = {1}
V2	D2 = {1,2,3}
V3	D3 = {1,2,3}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

- select as current variable the variable with smallest domain

- instantiate V4
- check forwards, against V7
- domain wipe out! Backtrack!

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1



current variable

V1 := 1
V4 := 1
V2
V3
V5
V6
V7
V8
V9
V10

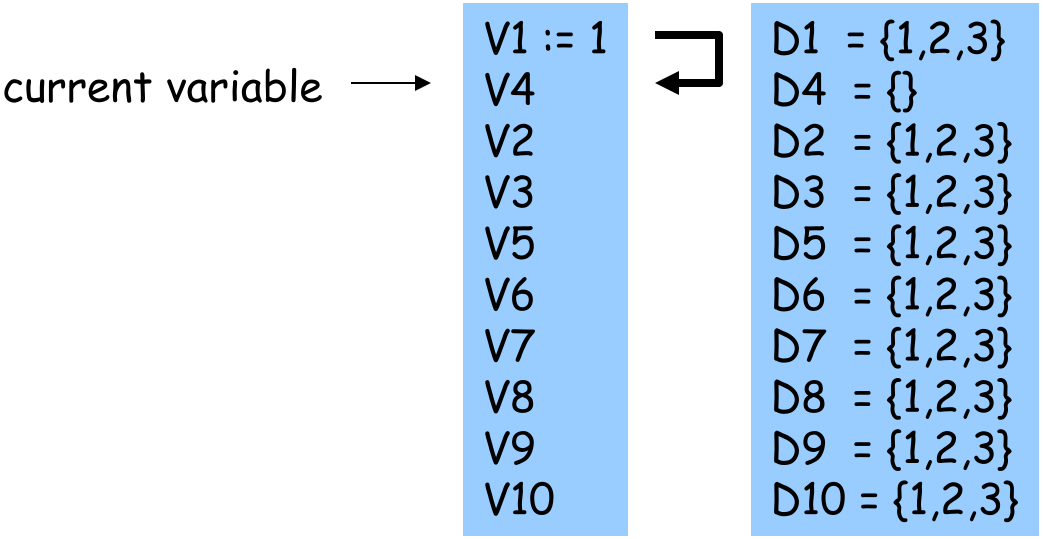
D1 = {1,2,3}
D4 = {1}
D2 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- backtrack

- return values to V7 removed by V4

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable →

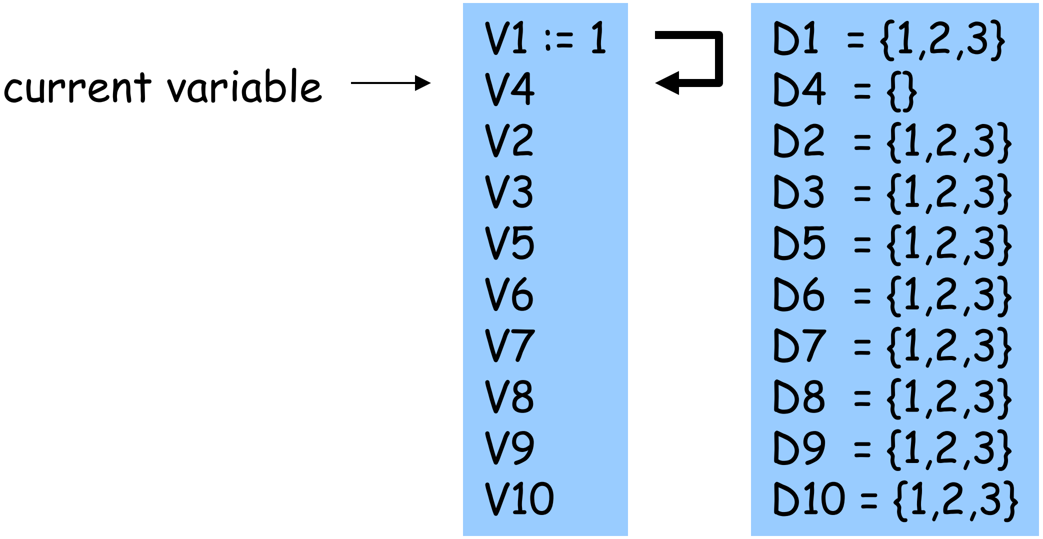
V1 := 1	D1 = {1,2,3}
V4	D4 = {}
V2	D2 = {1,2,3}
V3	D3 = {1,2,3}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {1,2,3}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

- backtrack

- remove from V4 the value it currently has

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

V1 := 1
V4
V2
V3
V5
V6
V7
V8
V9
V10

D1 = {1,2,3}
D4 = {}
D2 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- backtrack!

- V4 has a domain wipe out!

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

current variable →

V1 := 1	D1 = {1,2,3}
V4	D4 = {1,2,3}
V2	D2 = {1,2,3}
V3	D3 = {1,2,3}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {1,2,3}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

- backtrack!

- return to V4 values removed by V1

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

current variable →

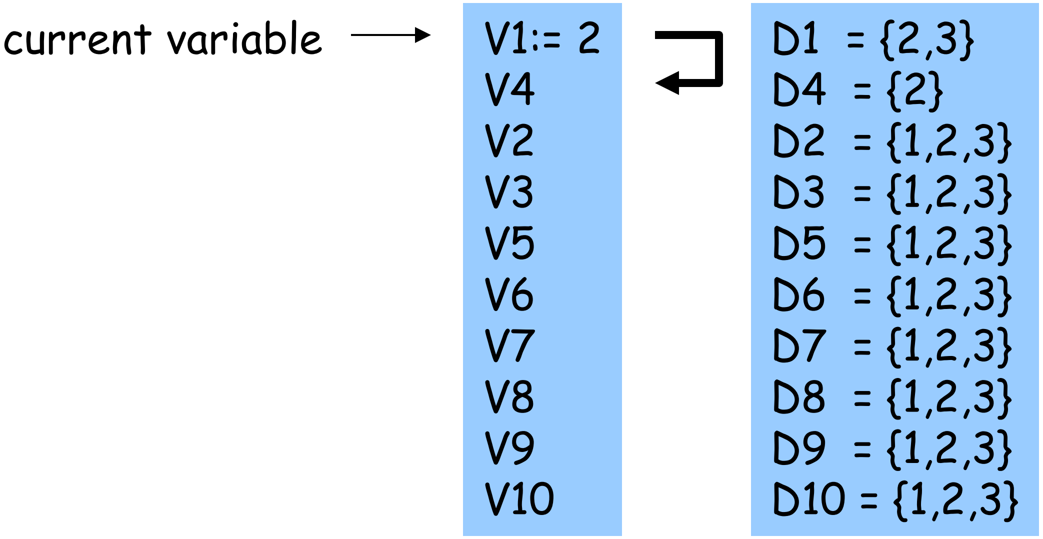
V1	D1 = {2,3}
V4	D4 = {1,2,3}
V2	D2 = {1,2,3}
V3	D3 = {1,2,3}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {1,2,3}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

- backtrack!

- remove the value V1 currently has from its domain

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



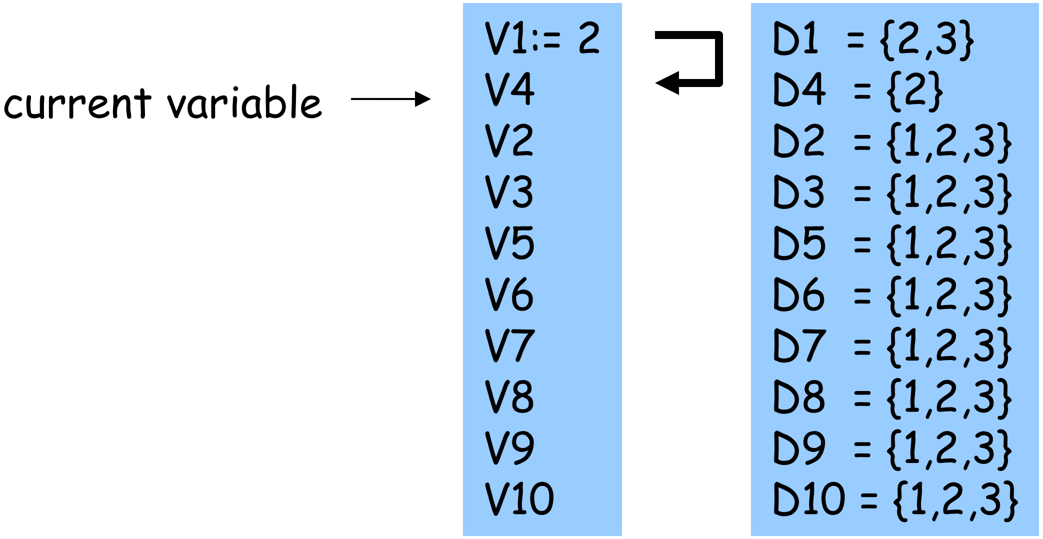
current variable →

V1:= 2	D1 = {2,3}
V4	D4 = {2}
V2	D2 = {1,2,3}
V3	D3 = {1,2,3}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V7	D7 = {1,2,3}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {1,2,3}

- instantiate V1 with next value
- check forwards to V4

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

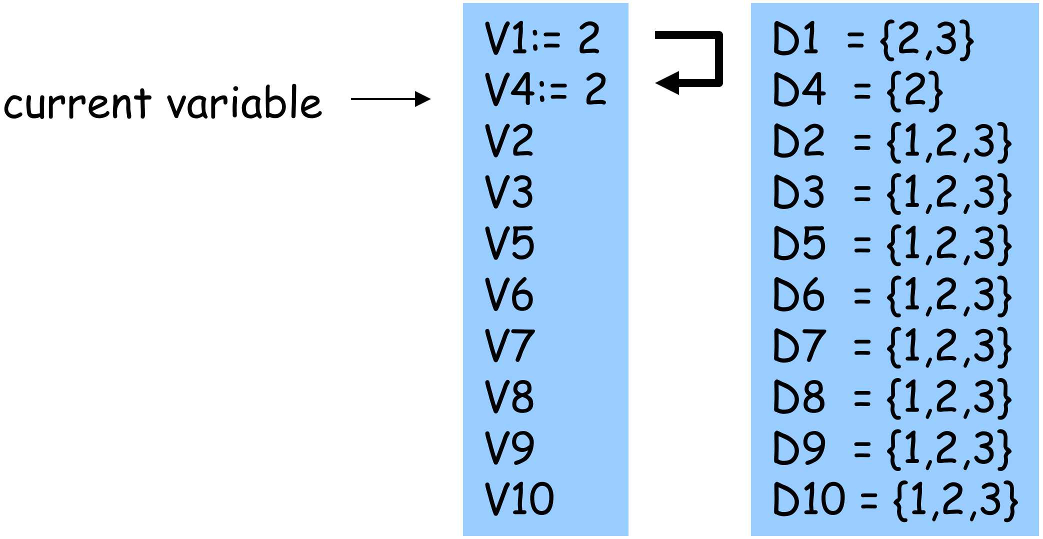
V1:= 2
V4
V2
V3
V5
V6
V7
V8
V9
V10

D1 = {2,3}
D4 = {2}
D2 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- *select variable with smallest domain*

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

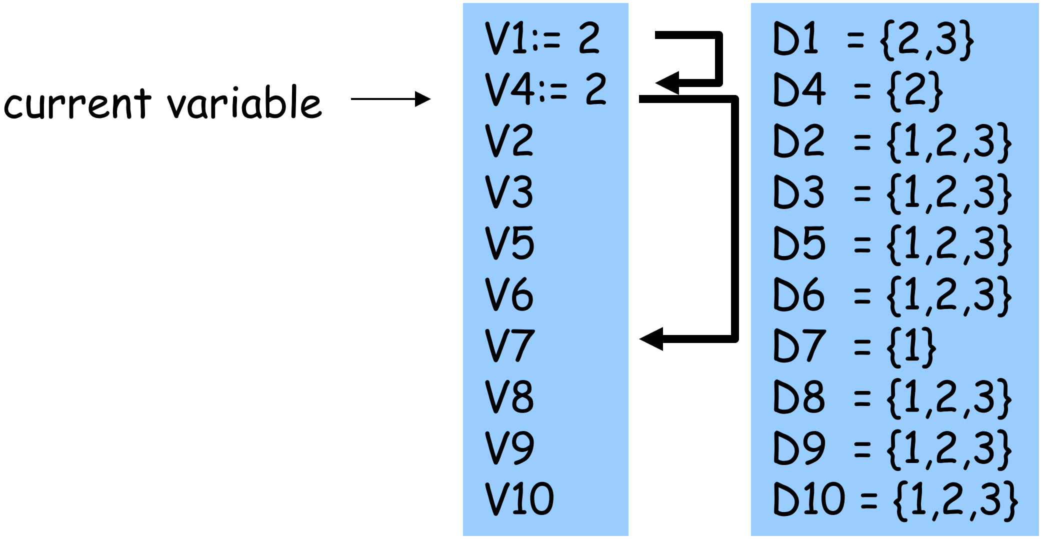
V1:= 2
V4:= 2
V2
V3
V5
V6
V7
V8
V9
V10

D1 = {2,3}
D4 = {2}
D2 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- instantiate V4

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

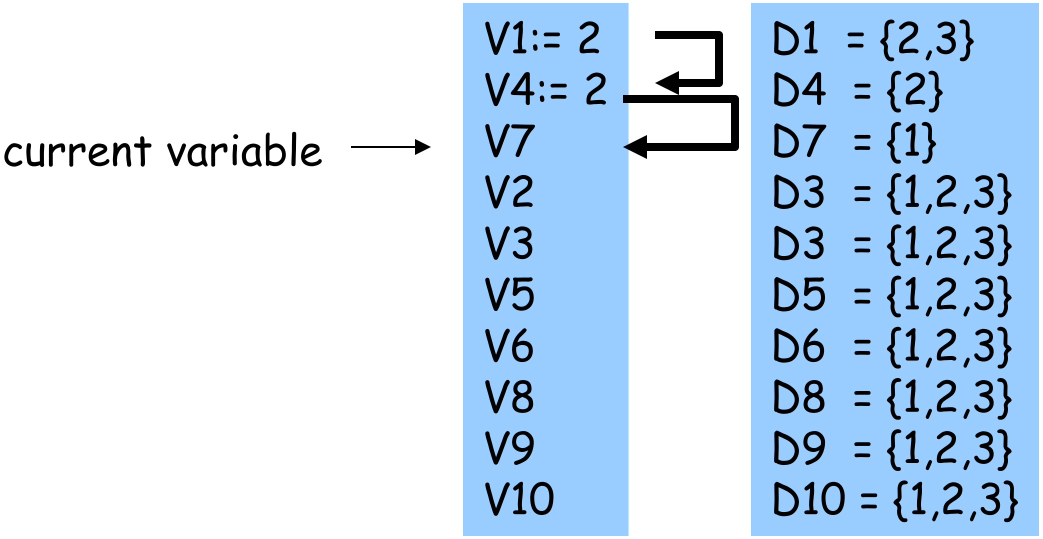
V1:= 2
V4:= 2
V2
V3
V5
V6
V7
V8
V9
V10

D1 = {2,3}
D4 = {2}
D2 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D7 = {1}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- check forwards to V7

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

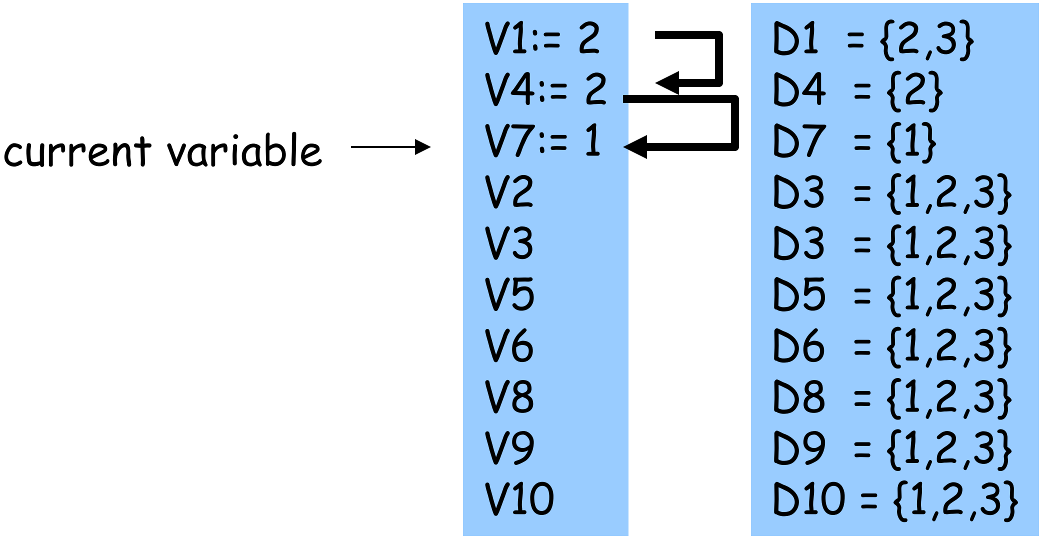
V1:= 2
V4:= 2
V7
V2
V3
V5
V6
V8
V9
V10

D1 = {2,3}
D4 = {2}
D7 = {1}
D3 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- select variable with smallest domain

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

V1:= 2
V4:= 2
V7:= 1
V2
V3
V5
V6
V8
V9
V10

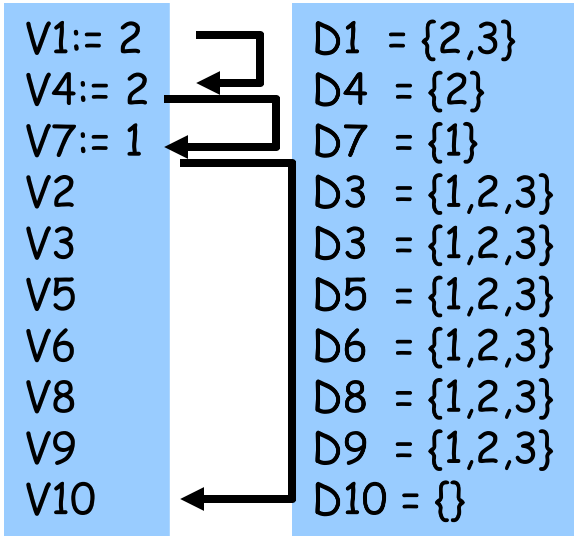
D1 = {2,3}
D4 = {2}
D7 = {1}
D3 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- instantiate current variable

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1

current variable →

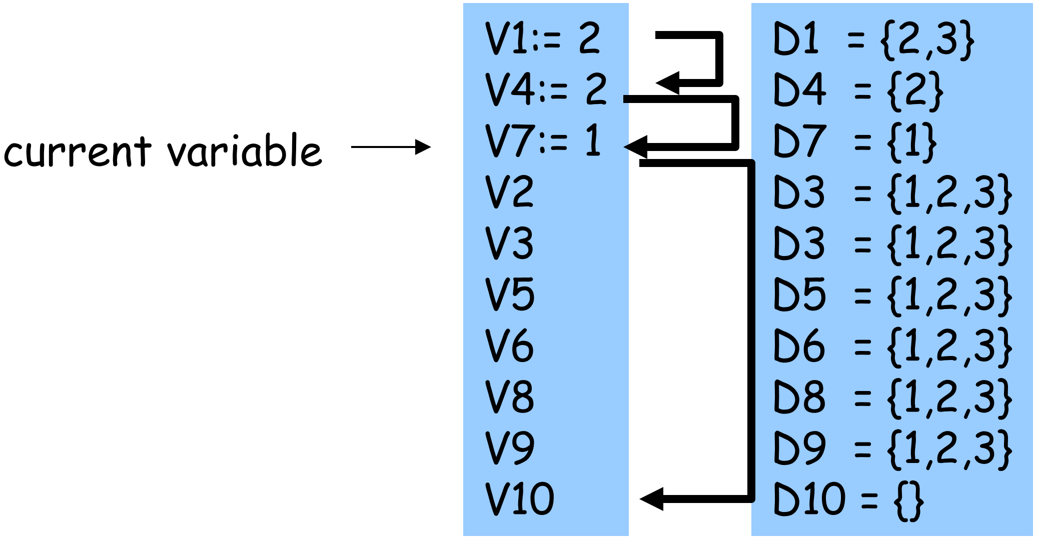


V1:= 2	D1 = {2,3}
V4:= 2	D4 = {2}
V7:= 1	D7 = {1}
V2	D3 = {1,2,3}
V3	D3 = {1,2,3}
V5	D5 = {1,2,3}
V6	D6 = {1,2,3}
V8	D8 = {1,2,3}
V9	D9 = {1,2,3}
V10	D10 = {}

- check forwards

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
  - V[1] = V[4]
  - V[4] > V[7]
  - V[7] = V[10] + 1



current variable

V1:= 2
V4:= 2
V7:= 1
V2
V3
V5
V6
V8
V9
V10

D1 = {2,3}
D4 = {2}
D7 = {1}
D3 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {}

- domain wipeout! Backtrack!

Forward checking

- variables V[1] to V[10]
- uniform domains D[1] to D[10] = {1,2,3}
- constraints
    - V[1] = V[4]
    - V[4] > V[7]
    - V[7] = V[10] + 1

current variable →

V1
V4
V7
V2
V3
V5
V6
V8
V9
V10

D1 = {3}
D4 = {1,2,3}
D7 = {1,2,3}
D3 = {1,2,3}
D3 = {1,2,3}
D5 = {1,2,3}
D6 = {1,2,3}
D8 = {1,2,3}
D9 = {1,2,3}
D10 = {1,2,3}

- backtrack to V1

dvo

That was a dvo, a dynamic variable ordering heuristic

select next the variable with smallest current domain
aka sdf, mrv, ff

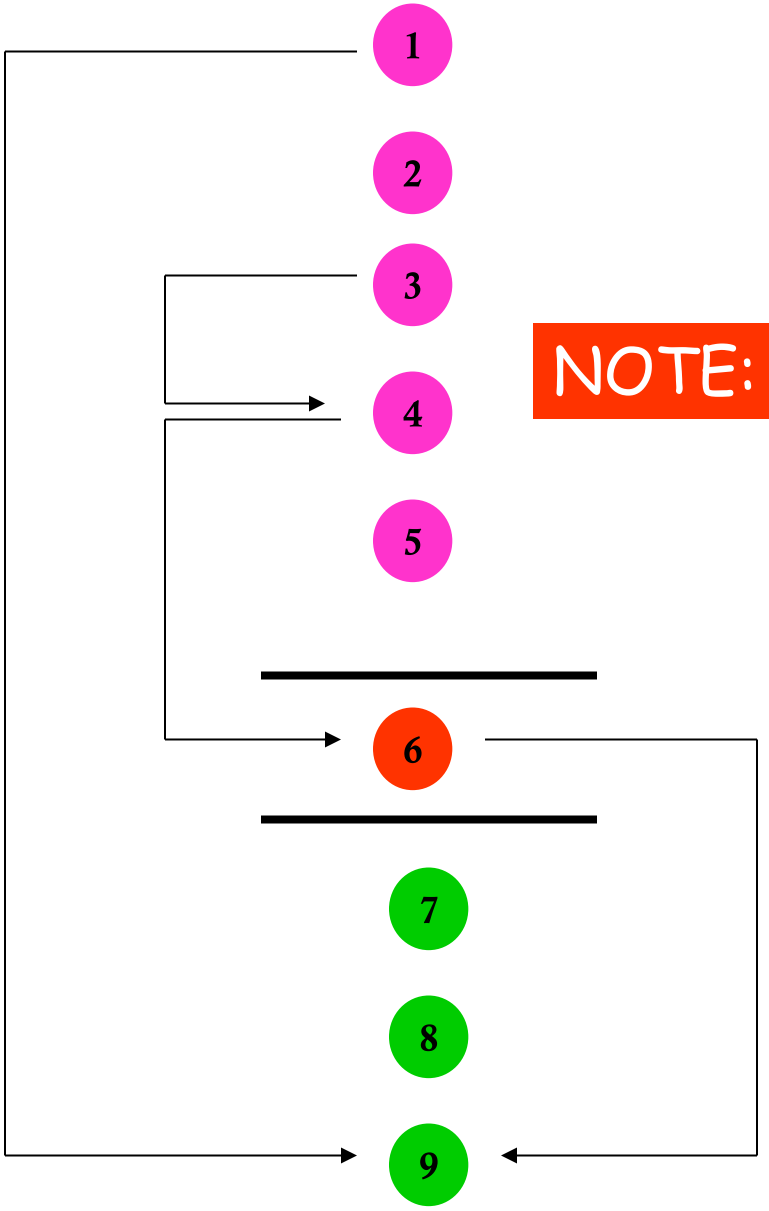
Was it a good heuristic?
If so, why was it a good heuristic?
Will it always be a good heuristic?
Can you think of a situation where it will be a bad heuristic?
Could you use this heuristic with bt or with cbj?

Is forward checking always better than backward checking?
That is, is fc better than bt (cbj) always

Does fc entirely eliminate thrashing?

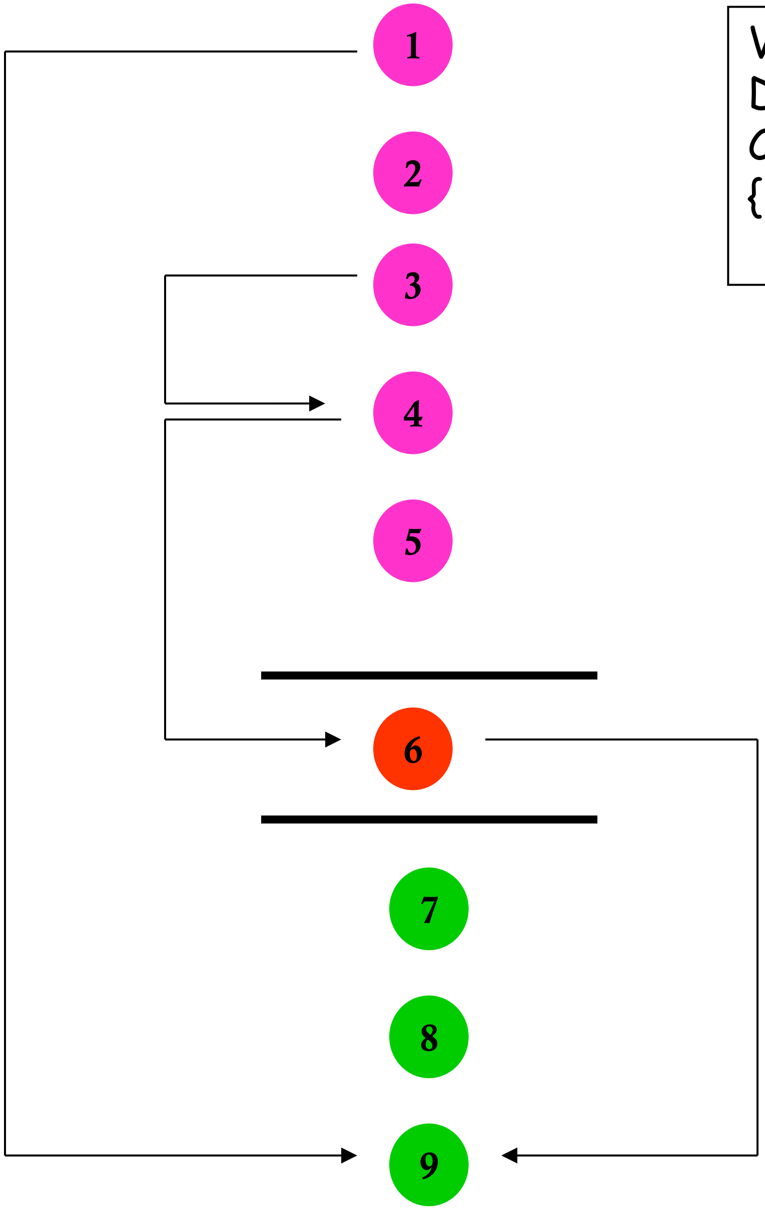
Could we incorporate cbj into fc?
That is check forwards and jump back?

Forward Checking



NOTE: arrows go forward!

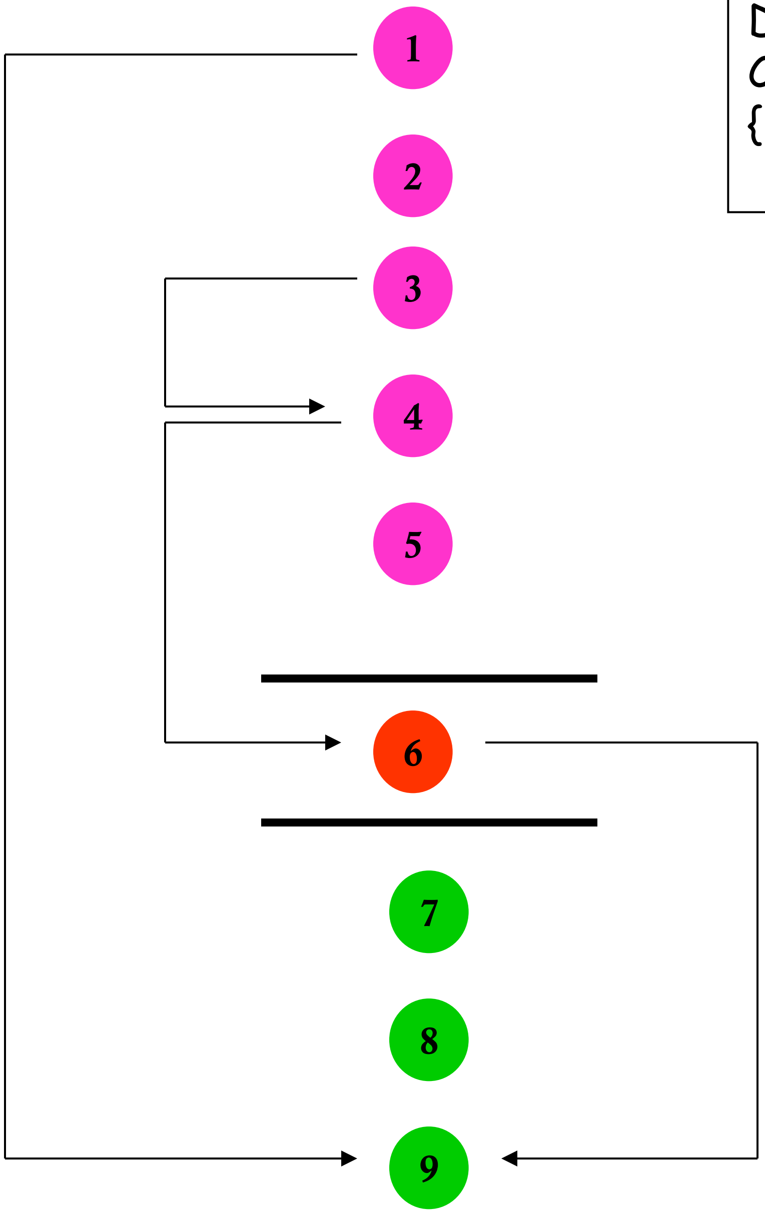
Forward Checking



Variables 1 to 9
Domains {A,B}
Constraints/nogoods:
{(1/A,9/A),(3A/,4/A), (4/B,6/A),
(6/B,9/B)}

State is:
1/A
2/A
3/A
4/B
5/A
6/B
7/{A,B}
8/{A,B}
9/{}

Forward Checking



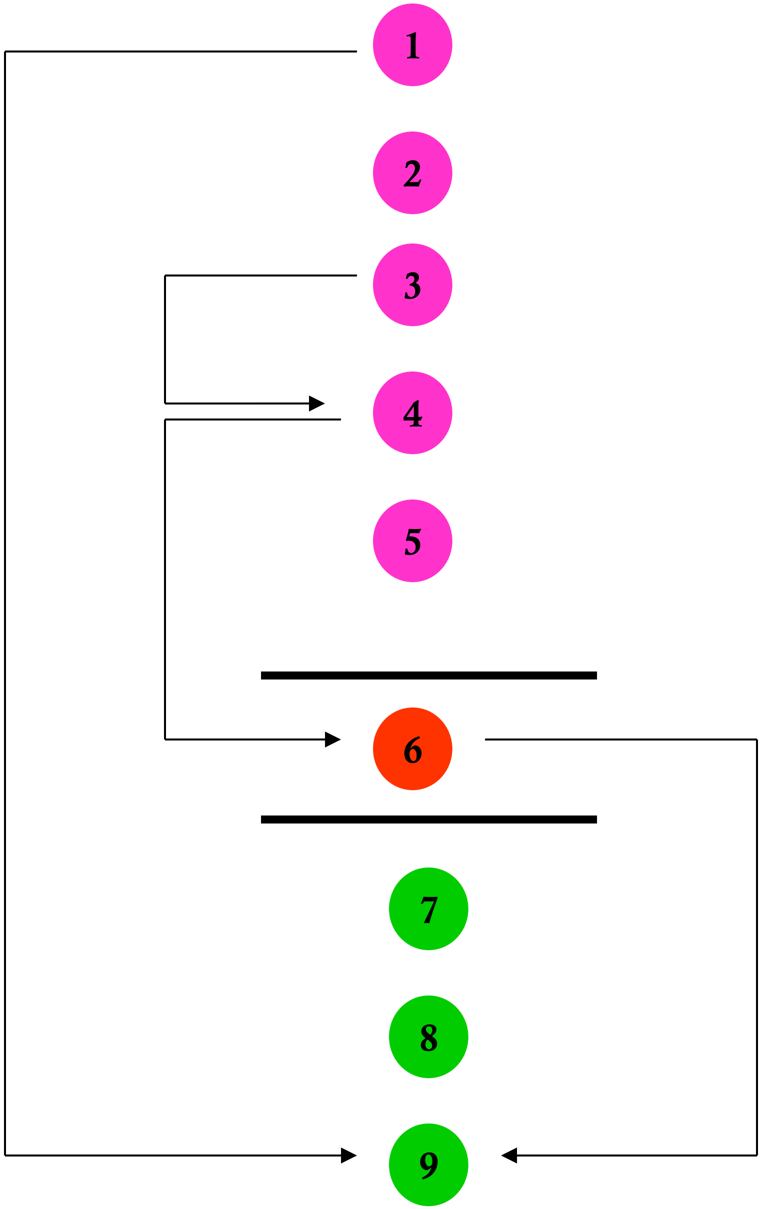
Variables 1 to 9
Domains {A,B}
Constraints/nogoods:
{(1/A,9/A),(3A/,4/A), (4/B,6/A),
(6/B,9/B)}

State is:
1/A
2/A
3/A
4/B
5/A
6/B
7/{A,B}
8/{A,B)
9/{}

Possible action:
Backjump to V4 so that V6 can then take value A

Backjump (step) to V3 so that V4 can take value A and Then V6 can take A also

Forward Checking Can Thrash



Without backjumping
FC can thrash!

Replace 5 with a collection
of variables with large domains
to convince yourself

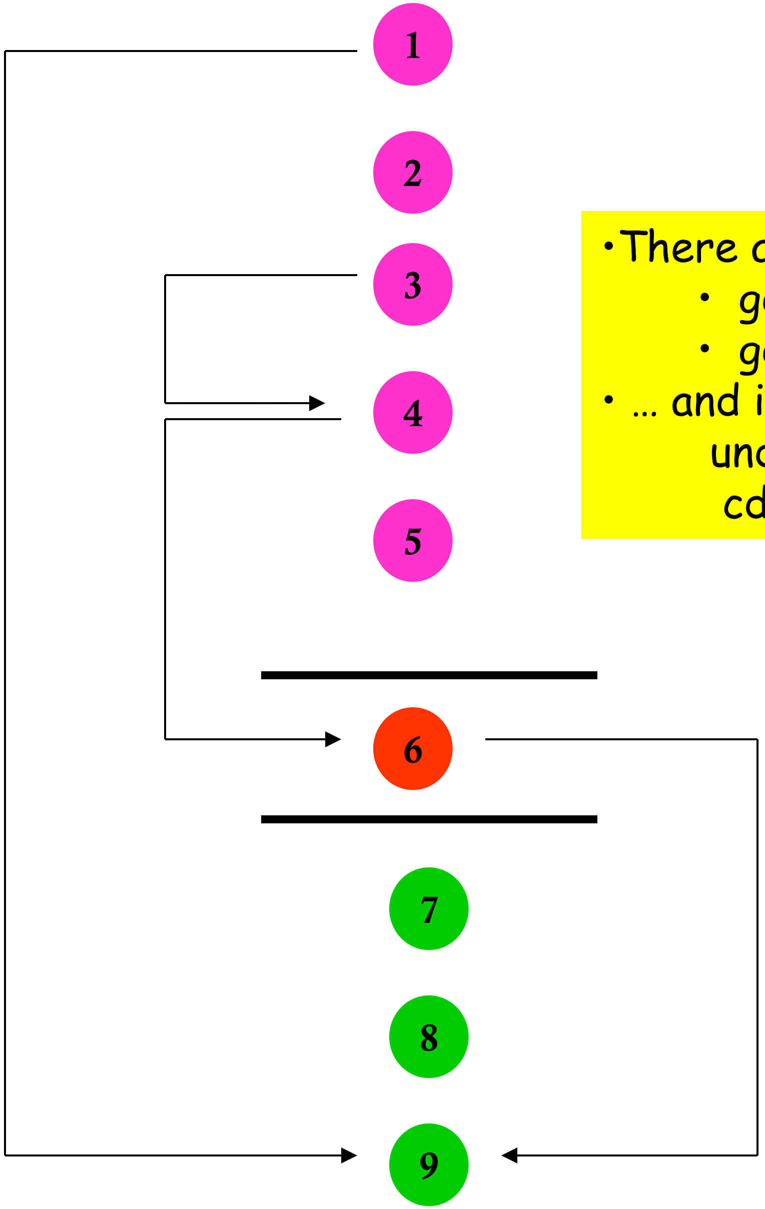
The mechanics of FC

- assume we have n variables each with domain of m values
- two dimensional array fcRed[1..n][1..n]
  - if v[i] checks forwards against v[j] and removes values
  - then fcRed[i][j] := true
- two dimensional array disallowed[1..n][1..m]
  - if v[i] removes value x from domain of v[j]
  - then disallowed[j][x] := i
    - i.e. the "culprit" for x removed from domain[j] is v[i]

When we uninstantiate v[i] we must undo the effects of forward checking

```
for j in (i+1 .. n)
  if fcRed[i][j]                                  // variable v[i] disallows values in v[j]
  then for x in (1 .. m)
          if disallowed[j][x] = i                 // this is a value disallowed by v[i]
          then domain[j] := domain[j]  U  {x}  // return a disallowed value
```

Check Forwards, Jump Back!



- There are no values in cd[6] compatible with v[9]
  - get more values into cd[9] (undo v[1]?) OR
  - get more values into cd[6] (undo v[4])
- … and if that doesn't work?

  undo v[3] so cd[4] gets value compatible with cd[6] that is then compatible with cd[9]

- why FC?
  - Fail 1st?
- Does FC suggest heuristics?
  - Static?
  - Dynamic
- But FC still thrashes!
  - We saw that
- FC could be worse than BT!
  - An example
- Could we combine FC with CBJ?

THE
END