INF623

2024/1

UFV

Inteligência Artificial

A10: Problemas de satisfação de restrição II

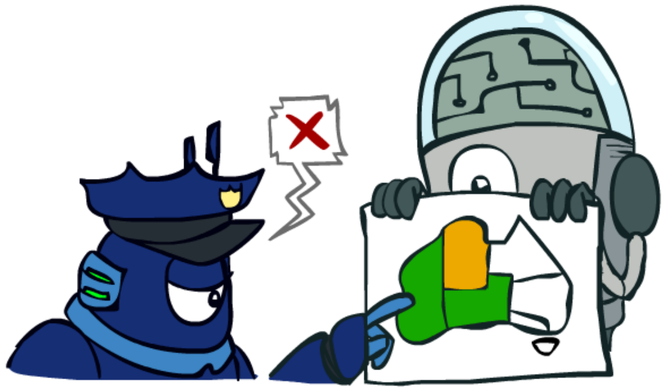
Plano de aula

- Ordenação de variáveis
  - Valores restantes mínimos
  - Heurística de grau
- Ordenação de valores
  - Valores menos restritivos
- Inferência em PSRs
  - Consistência de nó e arco
  - Algoritmo AC3 para consistência de arco
  - Verificação à frente

Busca com retrocesso para PSRs

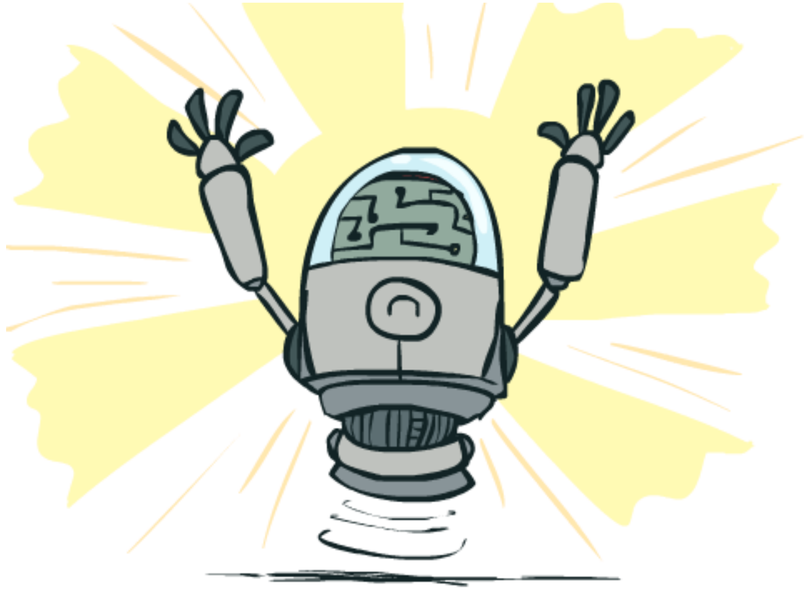
```
def busca-retrocesso(X, D, C):
1.      return retrocesso-recursivo({}, X, D, C)

def retrocesso-recursivo(assign, X, D, C):
1.      if is-complete(assign):
2.          return assign
3.      var = select-unsassigned-variable(X, assign) # Ideia 1: ordenar variáveis
4.      for value in sort(var, assign, X, D):
5.          if is-consistent(assign, value, C): # Ideia 2: verificar restrições
6.              assign[var] = value
7.              result = retrocesso-recursivo(assign, X, D, C)
8.              if result != None:
9.                  return result
10.             del assign[var]
11.     return None
```



Melhorias para a busca com retrocesso

- Novas ideias para guiar a busca em retrocesso
- **Ordenação de variáveis e valores:**
  - Quais variáveis devemos olhar primeiro?
  - Em que ordem os valores devem ser avaliados?
- **Inferência em PSRs:** podemos identificar falhas antes?
- **Estrutura do problema:** podemos explorar a estrutura dos problemas?



Ordenação de variáveis

```
def busca-retrocesso(X, D, C):
1.      return retrocesso-recursivo({}, X, D, C)

def retrocesso-recursivo(assign, X, D, C):
1.      if is-complete(assign):
2.          return assign
3.      var = select-unsassigned-variable(X, assign)
4.      for value in sort(var, assign, X, D):
5.          if is-consistent(assign, value, C):
6.              assign[var] = value
7.              result = retrocesso-recursivo(assign, X, D, C)
8.              if result != None:
9.                  return result
10.             del assign[var]
11.     return None
```

A ordem com a qual escolhemos uma variável pode tornar a busca mais eficiente. Existem 2 estratégias mais comuns:

- **Valores restantes mínimos**
- **Heurística de grau**

Ordenação de variáveis

```
def busca-retrocesso(X, D, C):
1.      return retrocesso-recursivo({}, X, D, C)

def retrocesso-recursivo(assign, X, D, C):
1.      if is-complete(assign):
2.          return assign
3.      var = select-unsassigned-variable(X, assign)
4.      for value in sort(var, assign, X, D):
5.          if is-consistent(assign, value, C):
6.              assign[var] = value
7.              result = retrocesso-recursivo(assign, X, D, C)
8.              if result != None:
9.                  return result
10.             del assign[var]
11.     return None
```

Valores Restantes Mínimos (VRM)

Escolher a variável com o menor número de valores válidos.

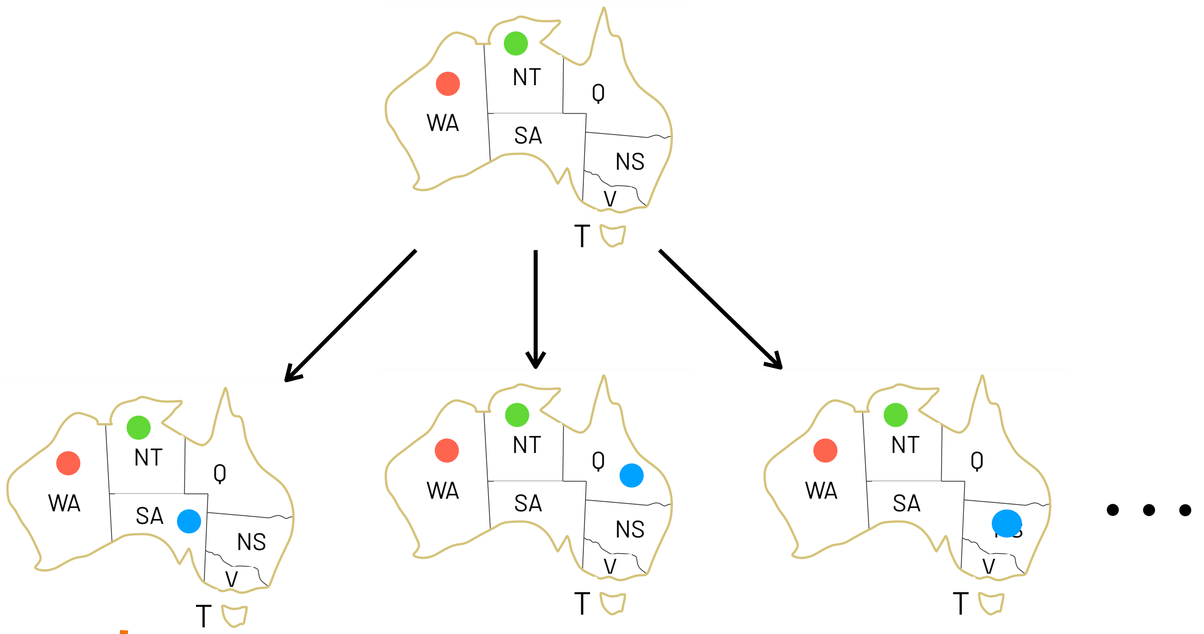
- Intuição: causar falhas o mais cedo possível
- Se uma variável X não tem mais valores válidos restantes, X será selecionado e a falha será detectada de imediato — evitando buscas inúteis por outras variáveis.

Ordenação de variáveis

Valores Restantes Mínimos (VRM)

Escolher a variável com o menor número de valores válidos.

- Intuição: causar falhas o mais cedo possível
- Se uma variável X não tem mais valores válidos restantes, X será selecionado e a falha será detectada de imediato — evitando buscas inúteis por outras variáveis.



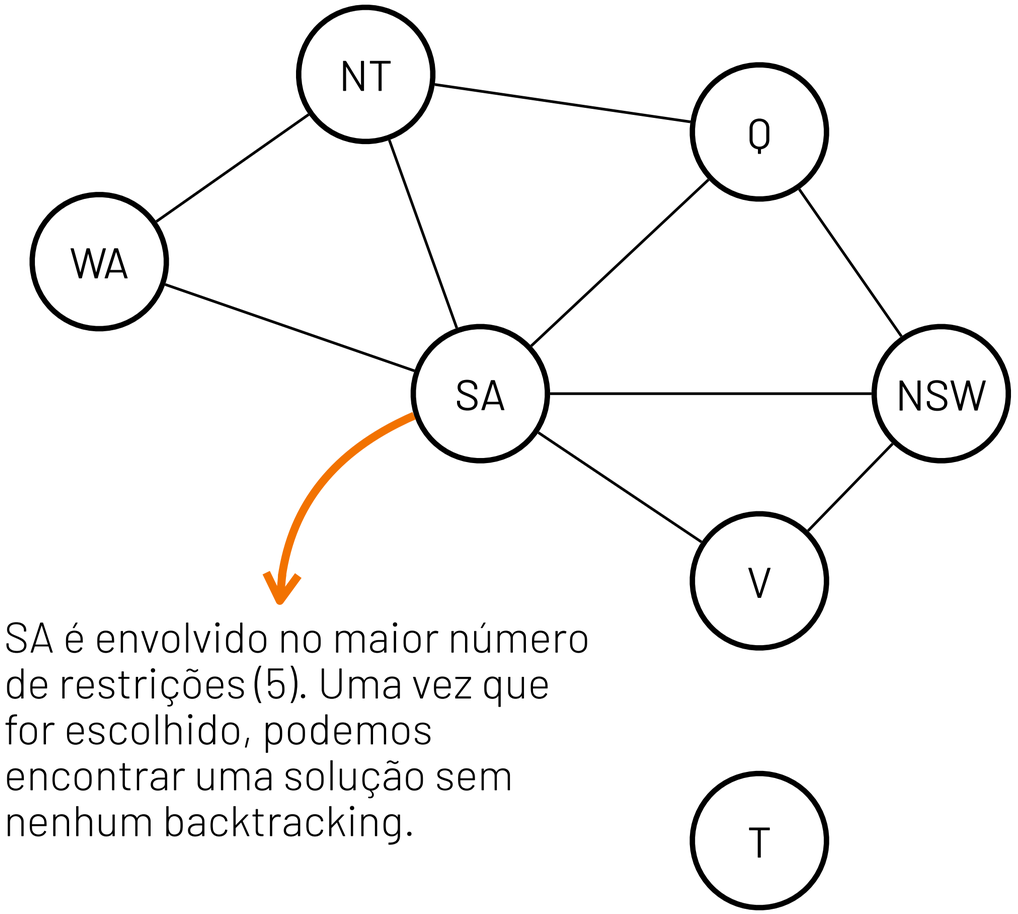
Depois de escolher **azul** para SA, as escolhas para Q, NS e V são todas forçadas!

Ordenação de variáveis

Heurística de grau

Escolher a variável envolvida com o maior número de restrições sobre outras variáveis não atribuídas.

- Intuição: reduzir o fator de ramificação em escolhas futuras
- No começo da busca no mapa da Austrália, todas as variáveis tem o mesmo o número de valores válidos, então a heurística VRM não nos ajuda.



SA é envolvido no maior número de restrições (5). Uma vez que for escolhido, podemos encontrar uma solução sem nenhum backtracking.

Ordenação de valores

```
def busca-retrocesso(X, D, C):
1.      return retrocesso-recursivo({}, X, D, C)

def retrocesso-recursivo(assign, X, D, C):
1.      if is-complete(assign):
2.          return assign
3.      var = select-unsassigned-variable(X, assign)
4.      for value in sort(var, assign, X, D):
5.          if is-consistent(assign, value, C):
6.              assign[var] = value
7.              result = retrocesso-recursivo(assign, X, D, C)
8.              if result != None:
9.                  return result
10.             del assign[var]
11.     return None
```

Uma vez que a variável foi selecionada, o algoritmo deve decidir pela ordem em que seus valores serão examinados.

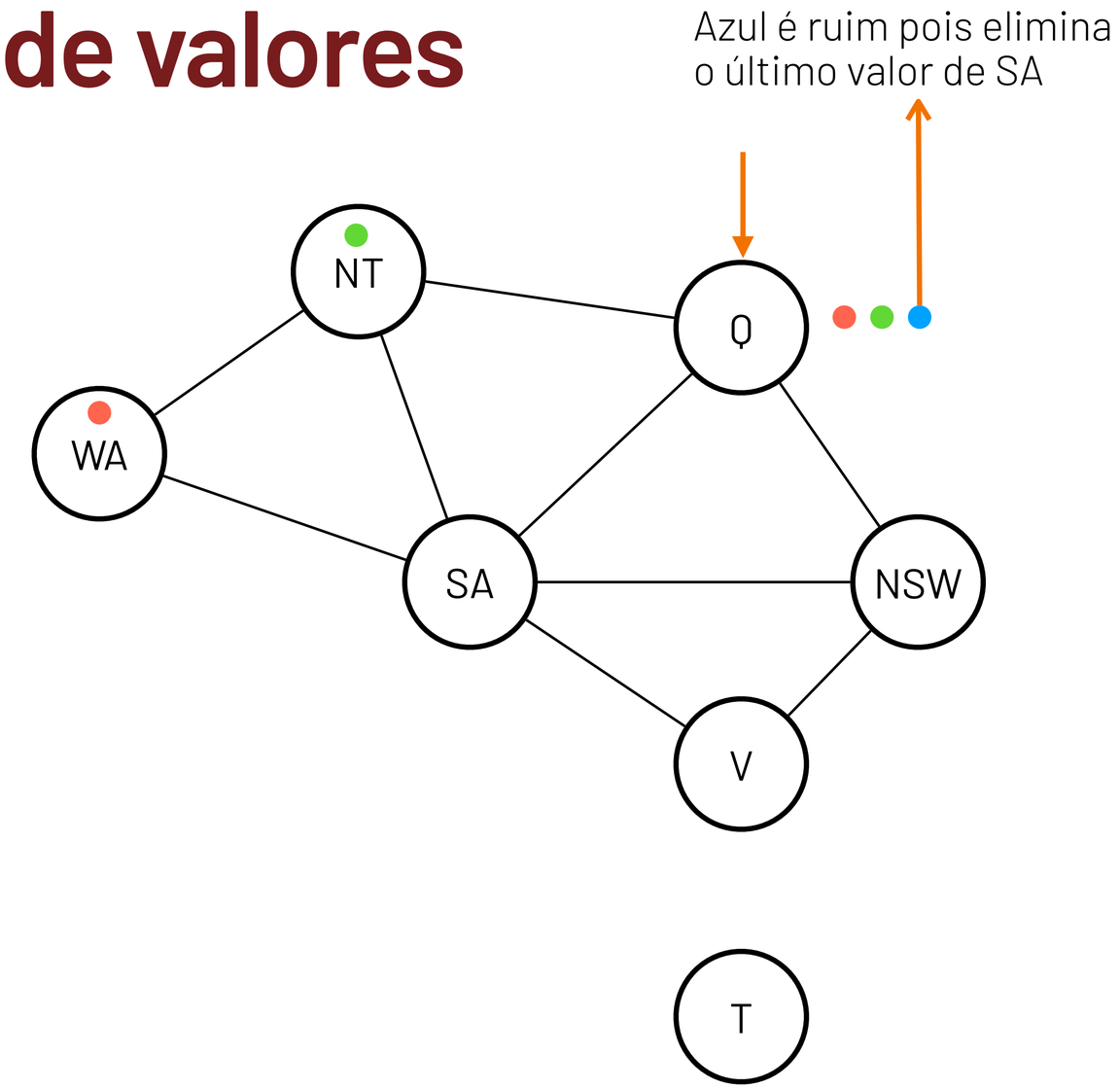
- **Valor menos restritivo**

Ordenação de valores

- **Valor menos restritivo**

Escolher o valor que elimina o menor número possível de escolhas para as variáveis vizinhas no grafo de restrições.

- Intuição: deixar a máxima flexibilidade para atribuições de variáveis subsequentes



Inferência em PSRs

```
def busca-retrocesso(X, D, C):
1.      return retrocesso-recursivo({}, X, D, C)

def retrocesso-recursivo(assign, X, D, C):
1.      if is-complete(assign):
2.          return assign
3.      var = select-unsassigned-variable(X, assign)
4.      for value in sort(var, assign, X, D):
5.          if is-consistent(assign, value, C):
6.              assign[var] = value
7.              infer ← inference(X, D, C,assign,var)
8.              if infer != None:
9.                  result = retrocesso-recursivo(assign, X, D, C)
10.                 if result != None:
11.                     return result
12.                 else:
13.                     del assign[var]
14.     return None
```

Utilizar restrições para inferir reduções no domínio de variáveis ao longo da busca:

- **Consistência de nó**
- **Consistência de arco**
- **Consistência de caminho**

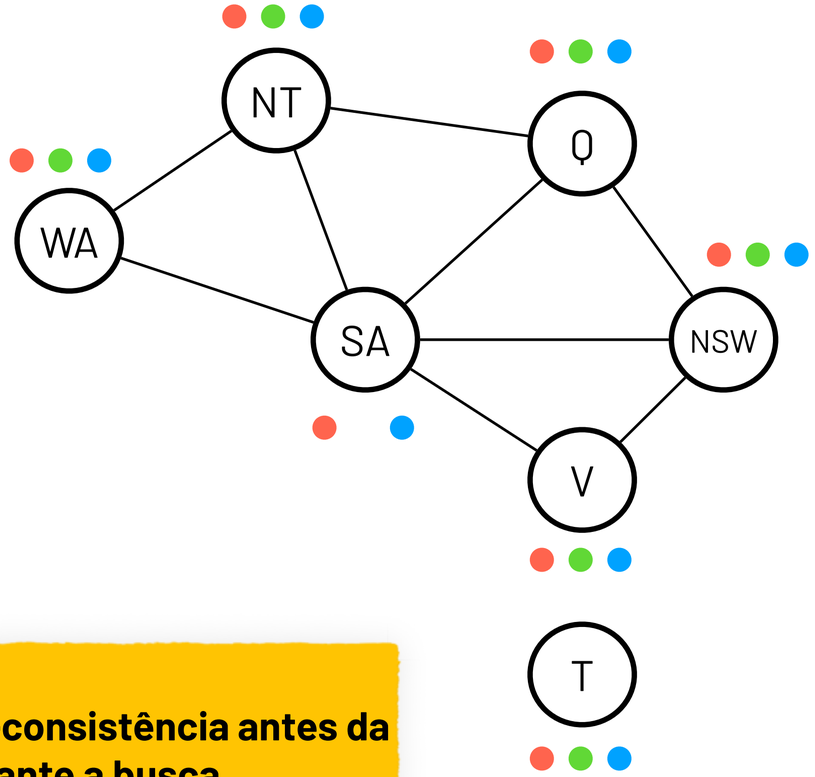
UFV

Consistência de nó

Uma variável X é **nó-consistente** se satisfizer suas restrições unárias

Considere uma variante do problema de coloração da Austrália onde os australianos do sul (moradores de SA) não gostam de verde:

- A variável SA começa com domínio $\{R, G, B\}$
- Podemos torna-lá nó-consistente eliminando G, deixando SA com domínio $\{R, B\}$

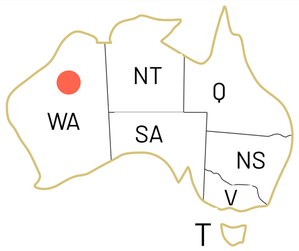


É sempre possível eliminar todas as restrições unárias em um PSR executando nó-consistência antes da busca, portanto não precisamos nos preocupar com esse tipo de consistência durante a busca.

Consistência de arco

As restrições binárias são as mais comuns em PSRs, pois é possível transformar todas as restrições n-árias em binárias.

Uma variável X é **arco-consistente** se satisfizer suas restrições binárias



- WA-NT: Deletar R do domínio de NT torna o arco WA-NT consistente
- WA-SA: Deletar R do domínio de SA torna o arco WA-SA consistente

As duas reduções de domínio acima tornam a variável WA arco-consistente!

O algoritmo AC3 para consistência de arco

Verificação à frente é o algoritmo mais simples de inferência para PSRs:

```
def ac3(X, D, C):
1.    Q = gerar-todos-os-arcos(X, C)
2.    while len(Q) > 0:
3.     (Xi, Xj) = Q.pop(0) # remover primeiro elemento
4.     if revisar((Xi, Xj), D, C):
5.       if len(Di) == 0:
6.         return false
7.       for Xk in vizinhos(Xi) - {Xj}:
8.         a.append((Xk, Xi))
9.    return True

def revisar((Xi, Xj), D, C):
1.      revisado = False
2.      for d in Di:
3.          if não existe valor v em Dj que satisfaça consistência de (Xi, Xj):
4.              remova d de Xi
5.              revisado = True
6.      return revisado
```

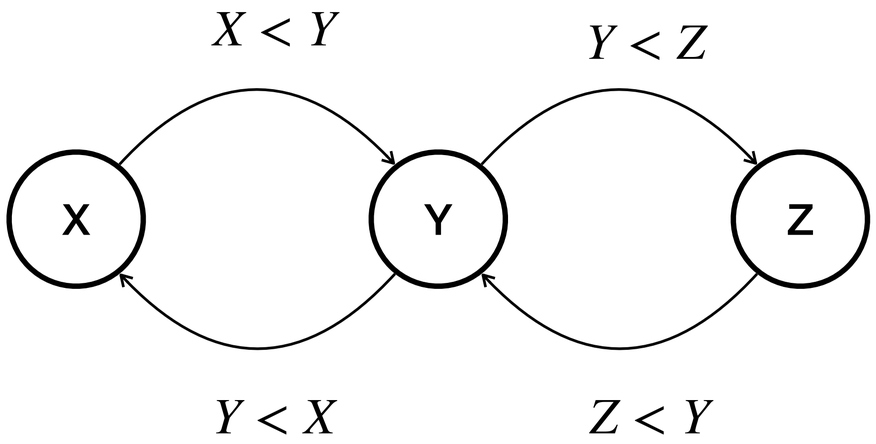
3 saídas possíveis:

- Uma variável possui domínio vazio: PSR não possui solução
- Todas variáveis possuem apenas um valor no domínio: solução encontrada
- Variáveis possuem um ou mais valores no domínio: solução ainda não foi encontrada.

Exemplo de execução do AC3

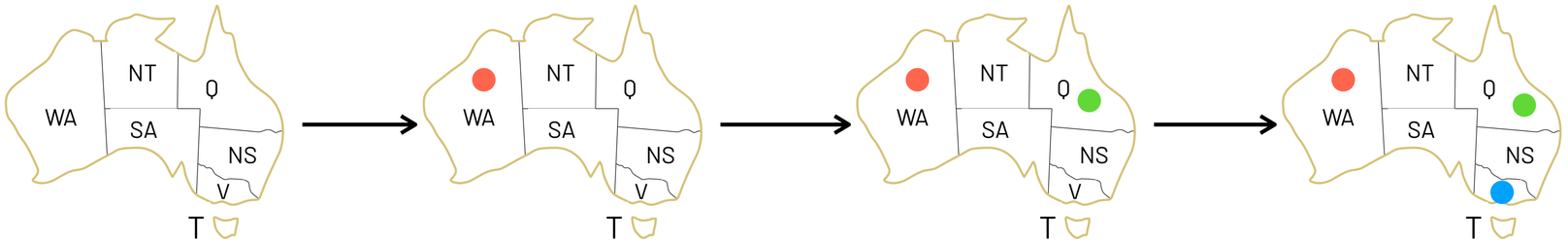
Fila	Ação
(X,Y), (Y,X), (Y,Z), (Z,Y)	Remover 3 de X
(Y,X), (Y,Z), (Z,Y)	Remover 0 de Y
(X,Y), (Y,Z), (Z,Y)	-
(Y,Z), (Z,Y)	Remover 3 de Y
(X,Y), (Z,Y)	Remover 2 de X
(Y,X), (Z,Y)	-
(Z,Y)	Remover 0 e 1 de Z
(Y,Z)	-

- **Variáveis:** X, Y, Z
- **Domínios:** $D = \{0,1,2,3\}$
- **Restrições:** $X < Y < Z$
- **Conjunto de restrições:** $\{(X, Y), (Y, X), (Y, Z), (Z, Y)\}$



Verificação à frente

Verificação à frente é o algoritmo mais simples de inferência para PSRs:



WA	NT	Q	NSW	V	SA
					None

Esse algoritmo força a consistência de arcos que apontam para cada nova variável atribuída!

Esse exemplo mostra como a inferência verificação à frente consegue identificar falhas antes delas acontecerem!

Próxima aula

Prova P1!

Busca no espaço de estados, busca local, busca competitiva e problemas de satisfação de restrição.