

Изкуствен интелект - зимен семестър, 2010/2011 учебна година

Лекция 2:
Търсене в пространството на
състоянията

Пространство на състоянията – основни понятия и задачи

Обща постановка. Решаването на много задачи, традиционно смятани за интелектуални, може да бъде сведено до последователно преминаване от едно описание (*формулировка*) на задачата към друго, *еквивалентно* на първото или *по-просто* от него, докато се стигне до това, което се смята за решение на задачата.

Примери: задачи от областта на т. нар. интелектуални игри, аналитични преобразования на алгебрични и тригонометрични изрази, решаване на уравнения и т.н.

Основни дефиниции

Състояние: едно описание (формулировка) на задачата в процеса на нейното решаване.

Видове състояния: начално, междинни, крайни (целеви).

Оператор: начин (правило, алгоритъм), по който от дадено състояние се получава друго.

Пространство на състоянията (ПС): съвкупността от всички възможни състояния, които могат да се получат от дадено начално състояние.

Представяне на ПС: чрез ориентиран граф (*граф на състоянията, ГС*) с възли – състоянията и дъги – операторите. Когато ПС може да се представи във вид на дърво, се говори за т. нар. *дърво на състоянията (ДС)*.

Действия, свързани с ПС

- **Генериране на състояния**

- генериране на следващ наследник
- генериране на всички наследници

- **Оценяване на състояние**

- двоични оценки (true/false)
- числови оценки в определен интервал (оценката на целта съвпада с единия край на интервала)

Основни типове задачи, свързани с ПС

- генериране на ПС
- търсене върху генерирано ПС
- комбинирано генериране и търсене в ПС

Стратегиите за решаване на тези задачи са сходни и затова обикновено се говори само за търсене (а се подразбира и/или генериране).

Основни типове задачи за търсене в ПС

1. **Търсене на път до (определена) цел** – търси се път от дадено начално състояние до определено целево състояние (целевото състояние е описано явно или може да бъде разпознато). Пътят може да се търси под формата на списък от възли или списък от дъги в ГС.

Варианти: търсене на минимален (най-къс или най-икономичен) път до цел и др.

Примери: търсене на път върху географска карта, задача за търговския пътник и др.

2. **Търсене на пчеливша стратегия** (при игри за двама играчи).

Примери: шахмат, шашки, “кръстчета и нулички” и др.

3. **Търсене на цел при спазване на ограничителни условия** (задачи за удовлетворяване на ограничения).

Примери: задача за осемте царици, решаване на криптограми, съставяне на разписания и др.

Задачи за удовлетворяване на ограничения (Constraint Satisfaction Problems, CSP)

Формулировка

Дадени са:

- множество от **променливи** $v_1, v_2, \dots, v_n$ (със съответни области на допустимите стойности D_{v_i} – дискретни (крайни или изброими безкрайни) или непрекъснати)
- множество от **ограничения** (допустими/недопустими комбинации от стойности на променливите)

Целево състояние (състояния): множество от свързвания със стойности на променливите $\{v_1=c_1, v_2=c_2, \dots, v_n=c_n\}$, които удовлетворяват всички ограничения.

Примерни задачи от разглеждания тип

- задача за осемте (n-те) царици, криптоаритметика, оцветяване на географска карта, пъзели, кръстословици, sudoku и др.
- планиране, проектиране, съставяне на разписания и др.

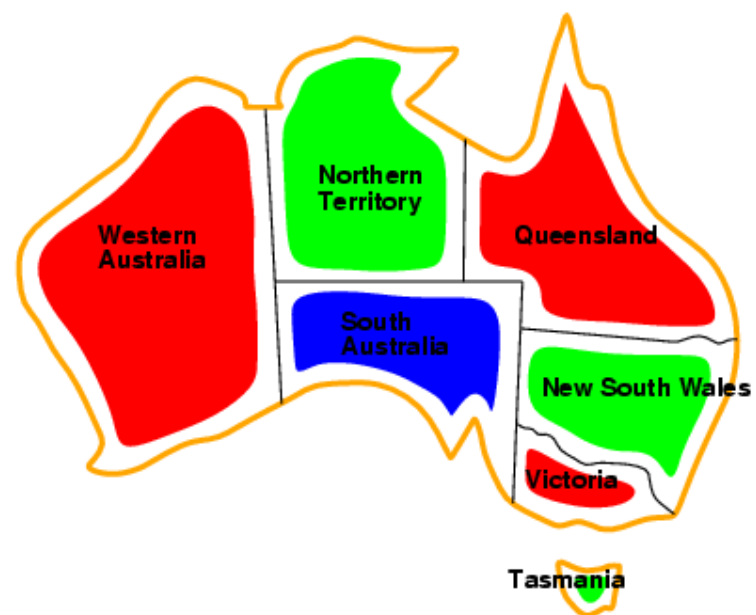
Пример: оцветяване на географска карта. Целта е всяка област (държава) върху картата да бъде оцветена с подходящ цвят по такъв начин, че да няма две съседни (граничещи) области, оцветени с един и същ цвят.



- Променливи: WA, NT, Q, NSW, V, SA, T
- Области на допустимите стойности: $D_i = \{\text{red}, \text{green}, \text{blue}\}$
- Ограничения: съседните области трябва да бъдат оцветени с различни цветове, например $WA \neq NT$ или $(WA, NT) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), (\text{green}, \text{blue}), (\text{blue}, \text{red}), (\text{blue}, \text{green})\}$

Забележка. Според теоремата за четирите цвята за решаването на задачата в общия случай са достатъчни 4 цвята. В нашия случай картата на Австралия може да бъде оцветена с 3 цвята.

Примерно решение:



Типове ограничения

- Унарни ограничения – включват по една променлива
Например: $SA \neq \text{green}$
- Бинарни ограничения – включват двойки променливи
Например: $SA \neq WA$
- Ограничения от по-висок ред – включват по 3 или повече променливи
Например: аритметичните ограничения в криптограмите
- Предпочитания (слаби ограничения) – определят критерии за избор между няколко решения (дефинират цената на някои свързвания на променливи)

Представяне на ограниченията

Бинарни ограничения: за представянето им се използва граф

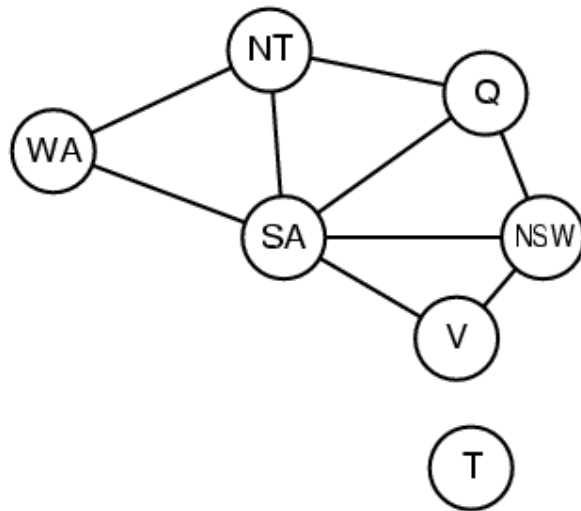
- възли – променливи ($X, Y, \dots$)

- дъги – ограничения

На всяко ограничение $p(X, Y)$ съответстват по две насочени дъги в графа: $\langle X, Y \rangle$ и $\langle Y, X \rangle$

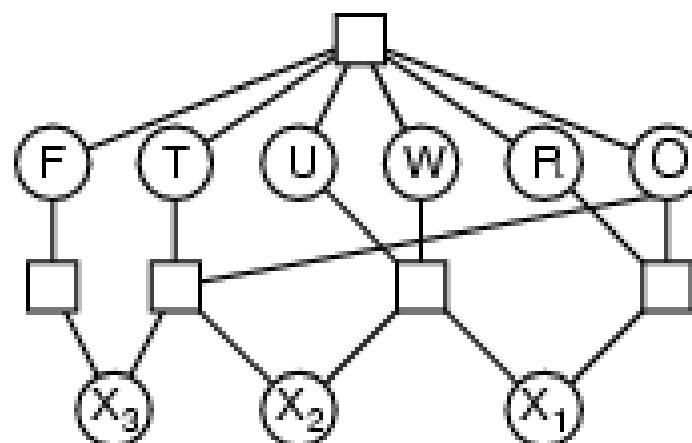
- Дъгата $\langle X, Y \rangle$ е съвместима, ако за всяка стойност на X от D_X съществува стойност на Y от D_Y , удовлетворяваща ограничението $p(X, Y)$
- Редуциране: ако дъгата $\langle X, Y \rangle$ не е съвместима, тогава всички стойности от D_X , за които няма съответна стойност от D_Y , могат да бъдат изтрети от D_X и това би направило дъгата $\langle X, Y \rangle$ съвместима

Примерен граф на ограниченията (constraint graph):



Пример: криптоаритметика (решаване на криптограма)

$$\begin{array}{r} \text{ T W O} \\ + \text{ T W O} \\ \hline \text{ F O U R} \end{array}$$



- Променливи: $F, T, U, W, R, O, X_1, X_2, X_3$
- Области на допустимите стойности: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ за променливите, които участват явно във формулировката на криптограмата, и $\{0, 1\}$ за X_i ($i=1, 2, 3$).
- Ограничения: $Alldiff(F, T, U, W, R, O)$

$$O + O = R + 10 * X_1$$

$$X_1 + W + W = U + 10 * X_2$$

$$X_2 + T + T = O + 10 * X_3$$

$$X_3 = F, T \neq 0, F \neq 0$$

Алгоритми за решаване на задачи за търсене на цел при спазване на ограничителни условия

- **генериране и тестване (*generate and test*)**
 - Генерира се множеството $D = DV_1 \times DV_2 \times \dots \times DV_n$
 - Тества се всяко свързване на променливите с елемент на D за това, дали удовлетворява ограниченията

- **търсене с възврат (*backtracking*)**

- Свързване на променливите със стойности в определен ред
- След свързването на всяка поредна променлива със стойност се извършва проверка дали свързаните до момента променливи удовлетворяват ограниченията:
 - ✓ ако ограниченията се удовлетворяват, се продължава с избора на стойност за следващата несвързана променлива
 - ✓ ако ограниченията не се удовлетворяват, се избира нова стойност за последната свързана променлива

```

function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING( $\{\}$ , csp)

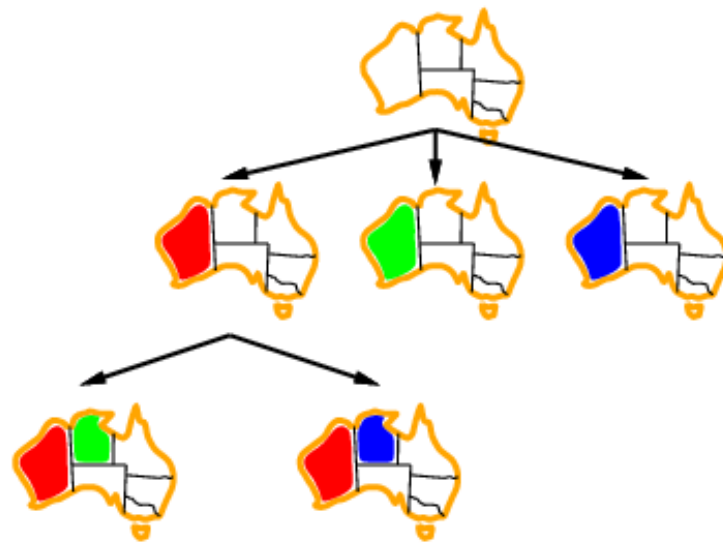
function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or
failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove { var = value } from assignment
    return failure

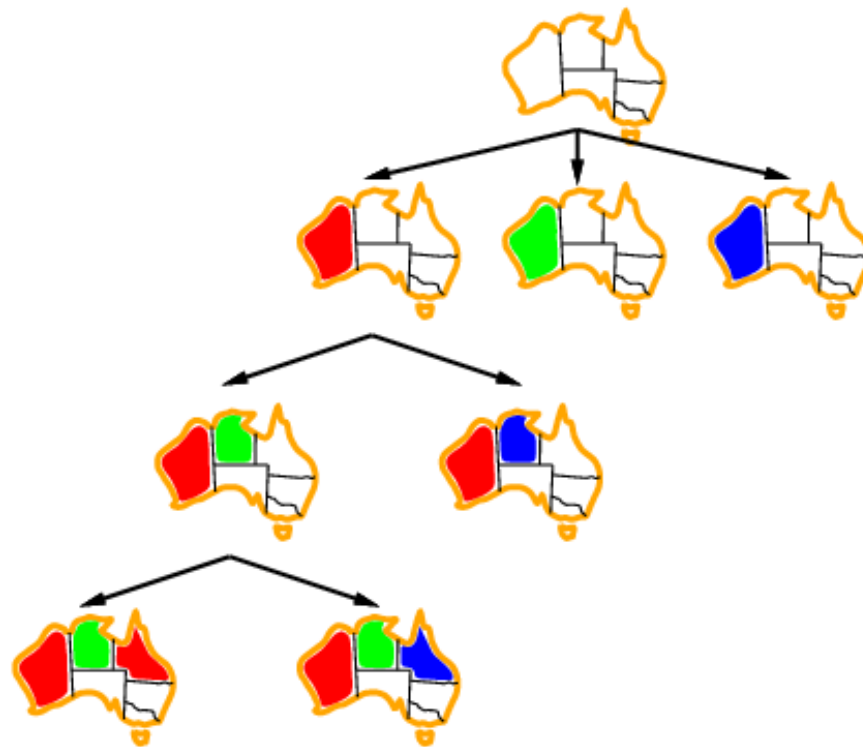
```

Пример за търсене с възврат









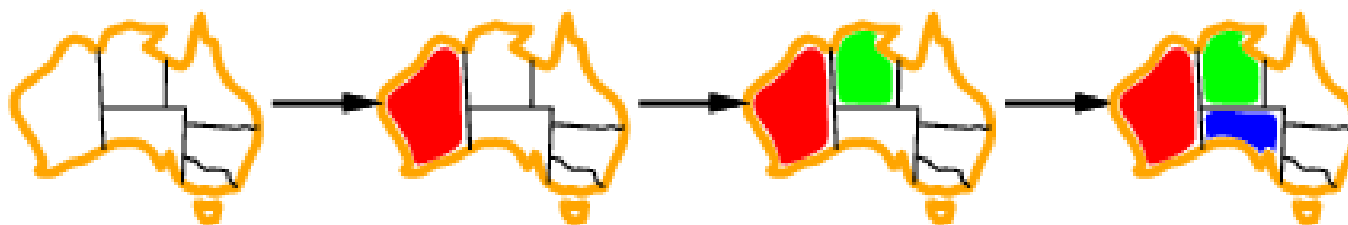
Подобряване на ефективността на търсенето с възврат

Използването на общи методи (каквото е текущо разглежданият) поставя редица въпроси, свързани с ефективността на търсенето:

- коя променлива трябва да бъде свързана най-напред?
- в какъв ред трябва да бъдат пробвани различните допустими стойности на избраната променлива?
- може ли да се прецени предварително дали дадено свързване ще доведе до неуспех?

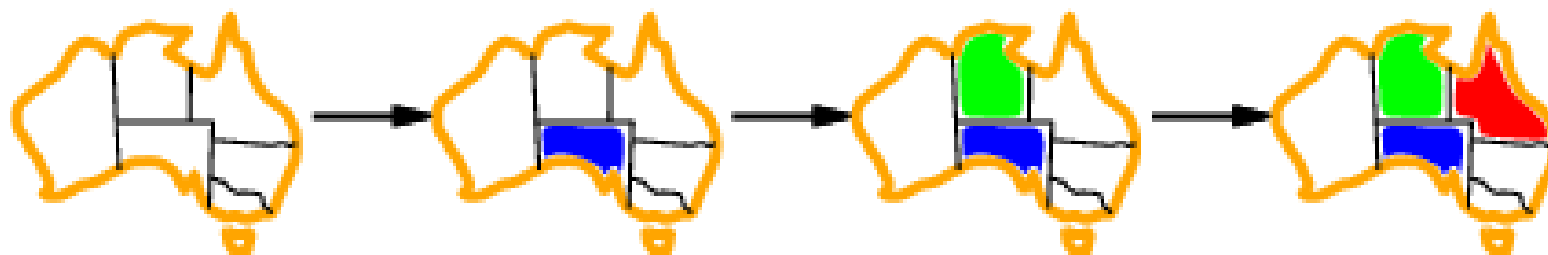
✓ **Евристика:** избор на най-ограничената променлива

Най-напред се избира за свързване онази променлива, която има най-малък брой допустими стойности



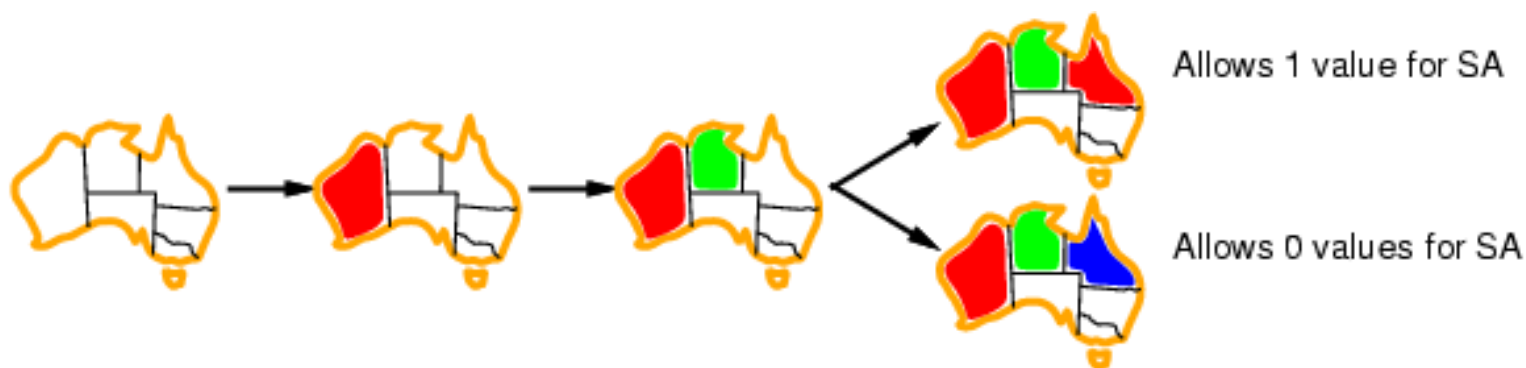
✓ **Евристика:** избор на най-ограничаващата променлива

Най-напред се избира за свързване онази променлива, която ще наложи най-много ограничения върху оставащите несвързани променливи



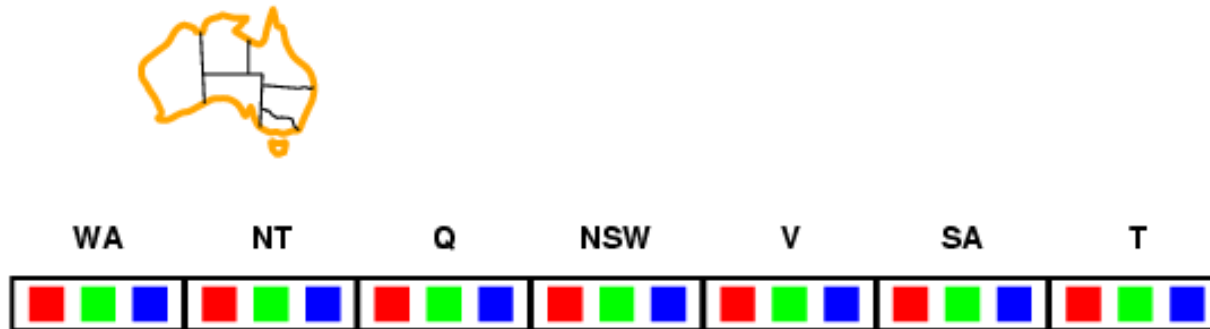
✓ **Евристика:** избор на най-малко ограничаващата променлива

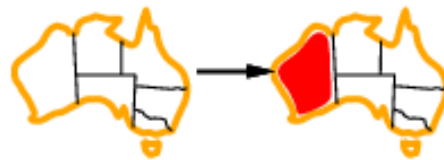
Най-напред се избира за свързване онази променлива, която ще наложи най-малко ограничения върху оставащите несвързани променливи



- **разпространяване на ограниченията (*forward checking*)**

- Следи се за оставащите допустими стойности за несвързаните променливи. Търсенето се прекратява, ако на дадена стъпка се окаже, че няма допустими стойности за някоя променлива.





WA	NT	Q	NSW	V	SA	T



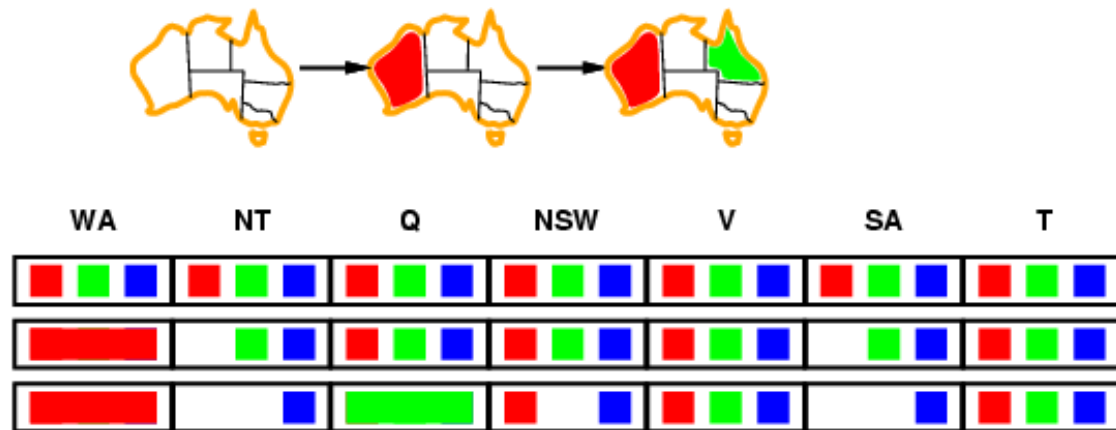
WA	NT	Q	NSW	V	SA	T



WA	NT	Q	NSW	V	SA	T

- **разпространяване на ограниченията (constraint propagation)**

- *Forward checking* разпространява информация (ограничения) от свързаните към несвързаните променливи, но не открива всички случаи, които ще доведат до неуспех:

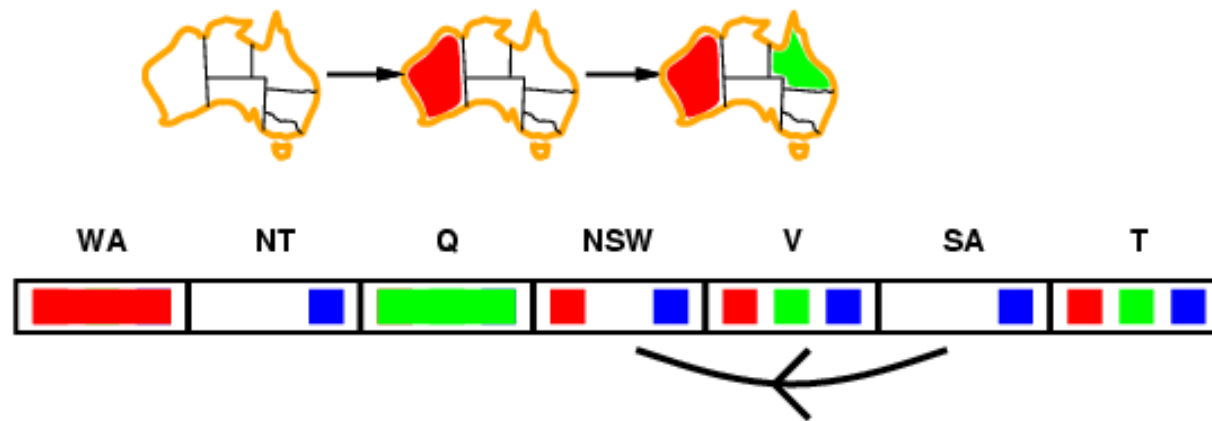


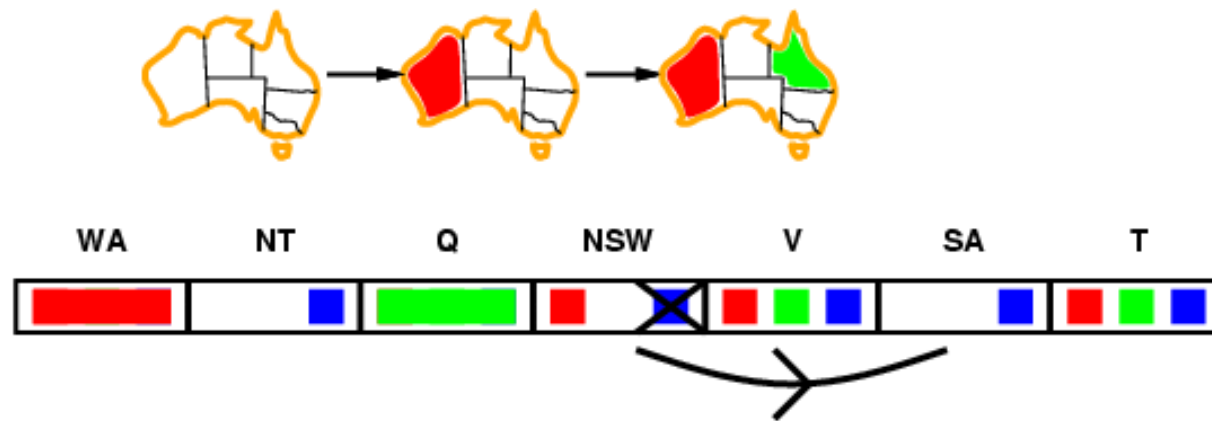
!!! *NT* и *SA* не могат едновременно да имат стойност *blue* (т.к. *NT* и *SA* не могат да имат еднакви стойности).

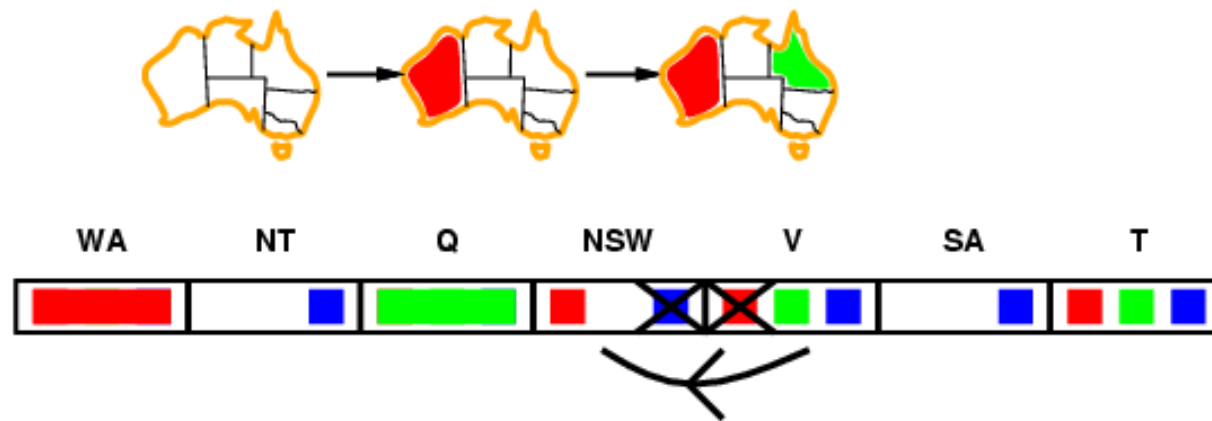
- *Constraint propagation* е общото понятие за разпространяване на ограничения от една променлива към други.
- *Constraint propagation* изисква *многократно засилване на ограниченията на локално равнище* (предполага генериране на логически следствия от текущите ограничения).

Съвместимост на дъгите

Проверката за съвместимост и осигуряването на съвместимост на дъгите чрез *редукция* е бърз метод за разпространяване на ограниченията (при задачи с бинарни ограничения), който е по-силен от *forward checking*.







Алгоритъм AC-3 за установяване на съвместимост на дъгите

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$ 
    if RM-INCONSISTENT-VALUES( $X_i, X_j$ ) then
      for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
        add  $(X_k, X_i)$  to queue



---


function RM-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff remove a value
  removed  $\leftarrow$  false
  for each  $x$  in DOMAIN[ $X_i$ ] do
    if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy constraint( $X_i, X_j$ )
      then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
  return removed
```