Der α–β-Algorithmus

… gibt für den Spieler am Zug den maximalen Wert für ALPHABETAMAX(n,$-\infty$,$+\infty$) zurück (entspricht einem optimalen Zug)

```
ALPHABETAMAX(Knoten, α, β)

If TiefenschrankeErreicht(Knoten) Return(Bewertung(Knoten))
NeueKnoten = Nachfolger(Knoten)
While NeueKnoten ≠ ∅
    α = Maximum(α, ALPHABETAMIN(Erster(NeueKnoten), α, β)
    If α ≥ β Return(β)
    NeueKnoten = Rest(NeueKnoten)
Return(α)
```

```
ALPHABETAMIN(Knoten, α, β)

If TiefenschrankeErreicht(Knoten) Return(Bewertung(Knoten))
NeueKnoten = Nachfolger(Knoten)
While NeueKnoten ≠ ∅
    β = Minimum(β, ALPHABETAMAX(Erster(NeueKnoten), α, β)
    If β ≤ α Return(α)
    NeueKnoten = Rest(NeueKnoten)
Return(β)
```

s. auch (für Spielkinder) www-i1.informatik.rwth-aachen.de/~algorithmus/algo19.php

Praktische Spielprogramme

- Optimiere Bewertungsberechnung (wenn möglich in Hardware!)
- Verwende große Eröffnungs- und Endspielbibliotheken
- Beispiel **Schach**: Deep Blue **www.chess.ibm.com/**
  - Kasparow 1997 in 6-Spiel-Turnier geschlagen
  - Analysierte 200 Mill. Zustände/sec (1997!)
  - Horizont ~12 Halbzüge; in Extremfällen bis 40 Halbzüge
- Beispiel **Dame**: Chinook **www.cs.ualberta.ca/~chinook/**
  - Praktisch amtierender Weltmeister seit 1994
  - Stand ca 2005: Komplette(!) Endspielbibliothek für ≤8 Steine: ~450.000.000.000 Zustände
  - 2007: Konstruktiver Beweis(!): Damespiel zwischen optimal spielenden Gegnern endet unentschieden
    (J. Schaeffer & al.: Checkers is Solved. Science 317:1518-1522, 2007)

Nichtdeterminismus in Spielen

… zum Beispiel: Würfeln

- Verwende Variante von MINIMAX, in der die Bewertungen mit der Wahrscheinlichkeit ihres Auftretens gewichtet werden
- Es gibt α–β-Variante

Games are to AI
as grand prix racing is to automobile design

Russell/Norvig

General Game Playing

- Seit 2005 Wettbewerb auf großer KI-Konferenz (AAAI)
- Motivation: KI-Ziel eigentlich nicht Weltklasse-Schach/ Dame, sondern Algorithmen für „clevere“ Spieler von Schach <u>und</u> Dame <u>und</u> Skat <u>und</u> Poker <u>und</u> Kniffel <u>und</u> …
  („Deep Blue has no clue how to play checkers!“ M. Genesereth)
- Voraussetzung: Sprache zur Beschreibung beliebiger Spielregeln: **GDL** (*Game Description Language*)
  games.stanford.edu/language/spec/gdl_spec_2008_03.pdf
  (GDL ist eine Datalog-Variante – entscheidbare Untermenge von PL1)
- Herausforderung: Allein aus Spielregeln erkenne Strategien und Bewertungsfunktionen für konkrete Spiele
- Unterschied zu „Alltagsintelligenz“: Spiele haben feste Regeln und klare Erfolgs/Gewinn-Kriterien

3.4 Constraint Satisfaction

Eliminate all other factors,
and the one which remains,
must be the truth.

A. Conan Doyle

Constraints sind ein Wissensrepräsentationsformat

- für das es spezielle Suchverfahren gibt
- das zur „Vorbehandlung von Suchbereichen"
  für allgemeine Suchverfahren verwendet werden kann

Website mit Tutorial und Java Tool zu CSPs:
aispace.org/constraint/

Definitionen zu Constraints

Ausgangspunkt: **Variablen** $\{X_1, \ldots, X_n\}$, je mit Werten
aus **Wertebereichen** (*domains*) $D_i\,(i{=}1, \ldots, n)$

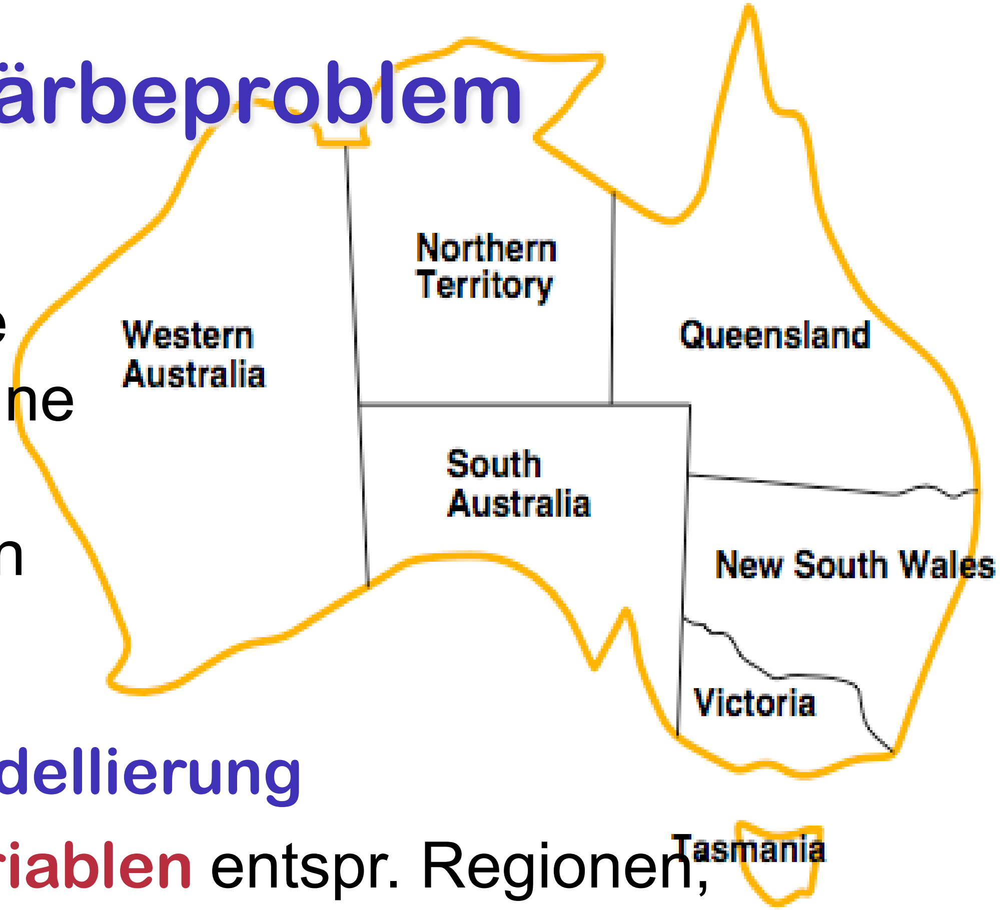
Ein k-stelliger **Constraint** C besteht aus einer Menge $\{X_1, \ldots, X_k\}$ von Variablen und einer k-stelligen Relation R_C auf den Variablen.

k Werte $\langle v_1, \ldots, v_k\rangle$ **erfüllen** C, wenn $v_i \in D_i$ u. $\langle v_1, \ldots, v_k\rangle \in R_C$

Ein **Constraintnetz** (*Constraint Satisfaction Problem*, **CSP**) besteht aus einer Menge $\{X_1, \ldots, X_n\}$ von Variablen und e. Menge $\{C_1, \ldots, C_m\}$ von Constraints auf Teilmengen dieser Variablen.

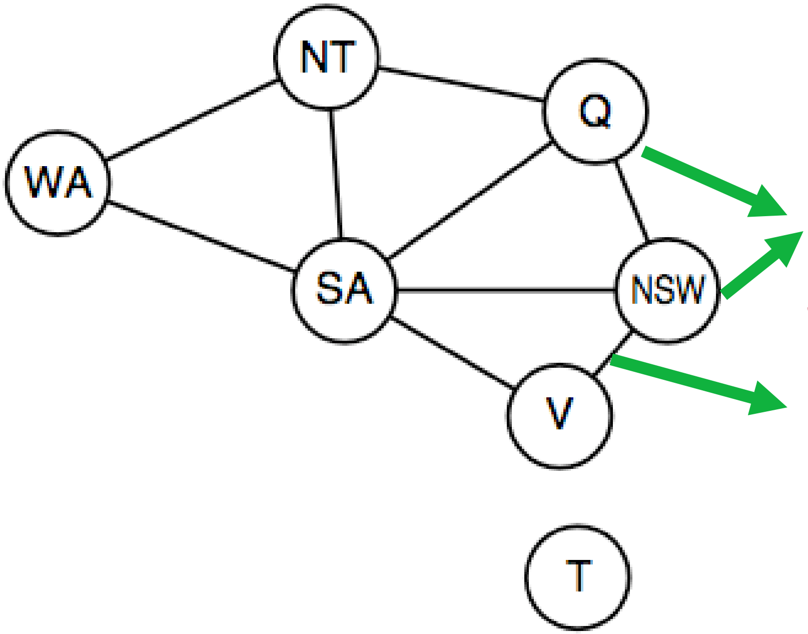
Eine **Lösung** eines CSP ist eine Belegung $\{X_1 = v_1, \ldots, X_n = v_n\}$, sodass $v_i \in D_i$ und alle Constraints aus $\{C_1, \ldots, C_m\}$ erfüllt sind.

Beispiel: Färbeproblem



Problem

Färbe die Regionen einer Karte mit drei Farben so ein, dass keine zwei aneinandergrenzende Regionen dieselbe Farbe haben



Modellierung

Variablen entspr. Regionen;

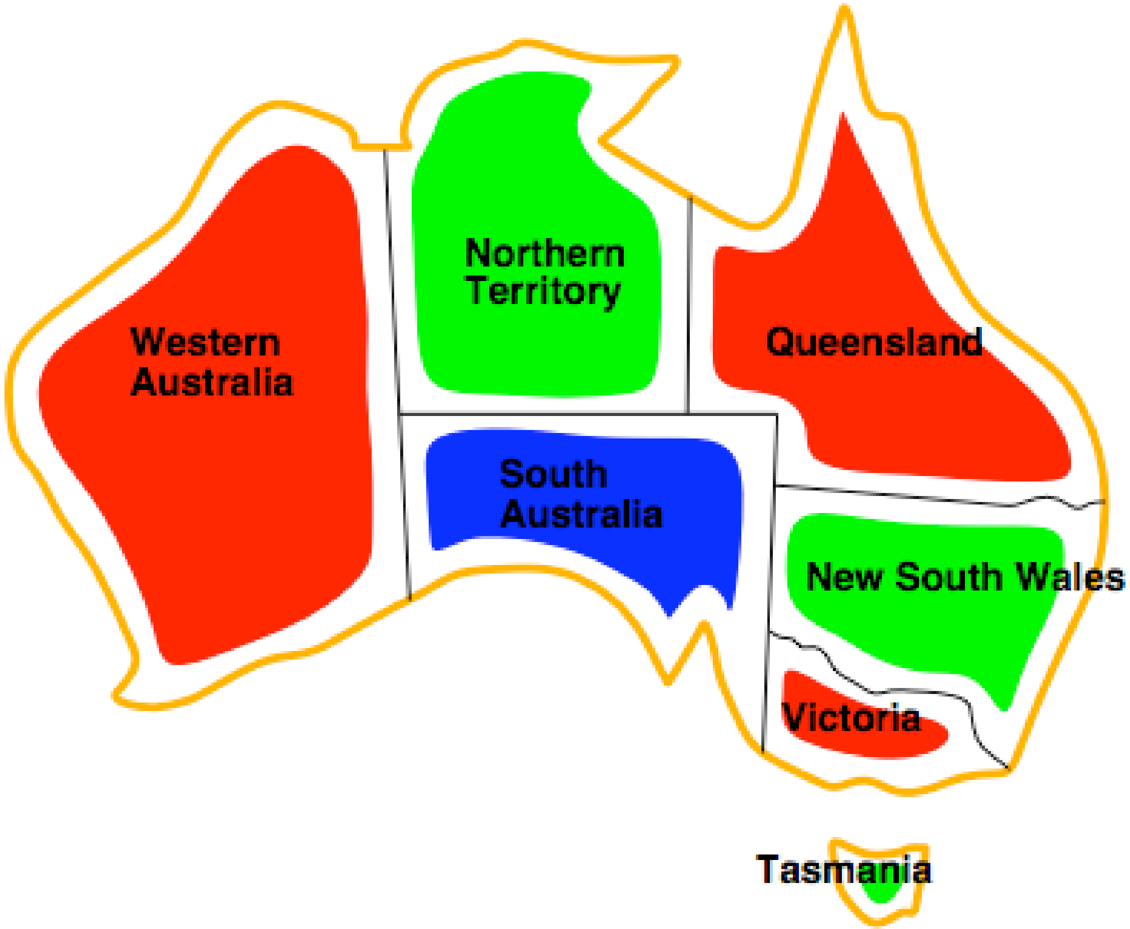
Wertebereiche jeweils {R, G, B};

Relation „ungleiche_Farbe" zw. je 2 angrenzenden Regionen

(Hier als Kante zw. Variablen skizziert)

Eine Färbung Australiens

ungleiche_Farbe={⟨R,G⟩, ⟨R,B⟩, ⟨G,R⟩, ⟨G,B⟩, ⟨B,R⟩, ⟨B,G⟩}



Eine Lösung des CSP:

{WA=R,
NT=G,
Q=R,
NSW=G,
V=R,
SA=B,
T=G }

Klassen von Constraintproblemen

Diskrete vs. kontinuierliche Wertebereiche

- diskrete endliche Wertebereiche
  ↪ $O(d^n)$ Belegungen bei max. Wertebereichgröße d
  - z.B. Färben, Boolesche Erfüllbarkeit (einschl. 3SAT)
- diskrete unendliche Wertebereiche
  - z.B. Scheduling-Probleme über Zeiteinheiten
- kontinuierlich
  - z.B. Scheduling-Probleme (Temporal C. S. Ps., TCSPs)

Unäre vs. Binäre vs. k-äre CSPs entspr. Relationen-Stelligkeit

- binär z.B. Färbeprobleme; „universelle“ Darstellung!

Strikte vs. Präferenz CSPs

- Strikte Relationen (z.B. Färbepr.) oder Nutzen/Strafwerte

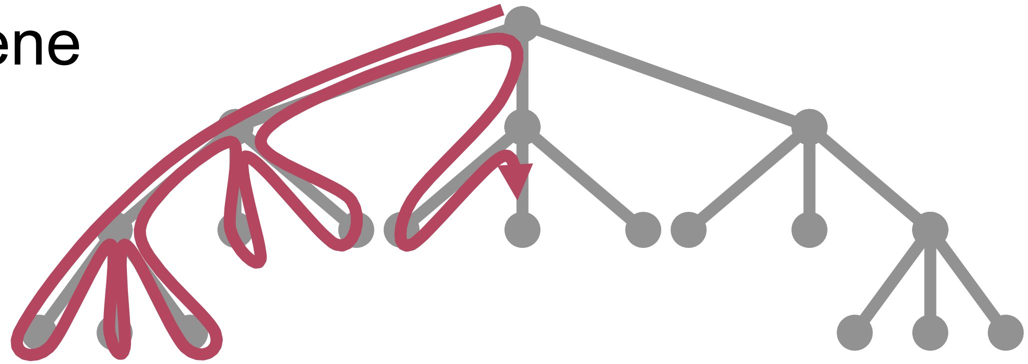
Backtracking zum Lösen von CSPs

<u>Uninformierter</u> Basisalgorithmus für diskrete endliche CSPs

Entsprechend Backtracking-Suche (Folie Nr. 125):

Starte mit leerer Variablenbelegung und durchlaufe:

1. Wähle für einzelne(!) Variablen nacheinander Zuweisungen, sodass deren Constraints erfüllt sind;
2. wenn alle Variablen belegt, Lösung gefunden!
3. wenn keine mögliche Belegung aktuellen Constraint erfüllt, nimm <u>zuletzt</u> vorgenommene Belegung zurück (➥ „**Chronologisches**“ Rücksetzen/Backtracking)



Ein Backtracking-CSP-Algorithmus

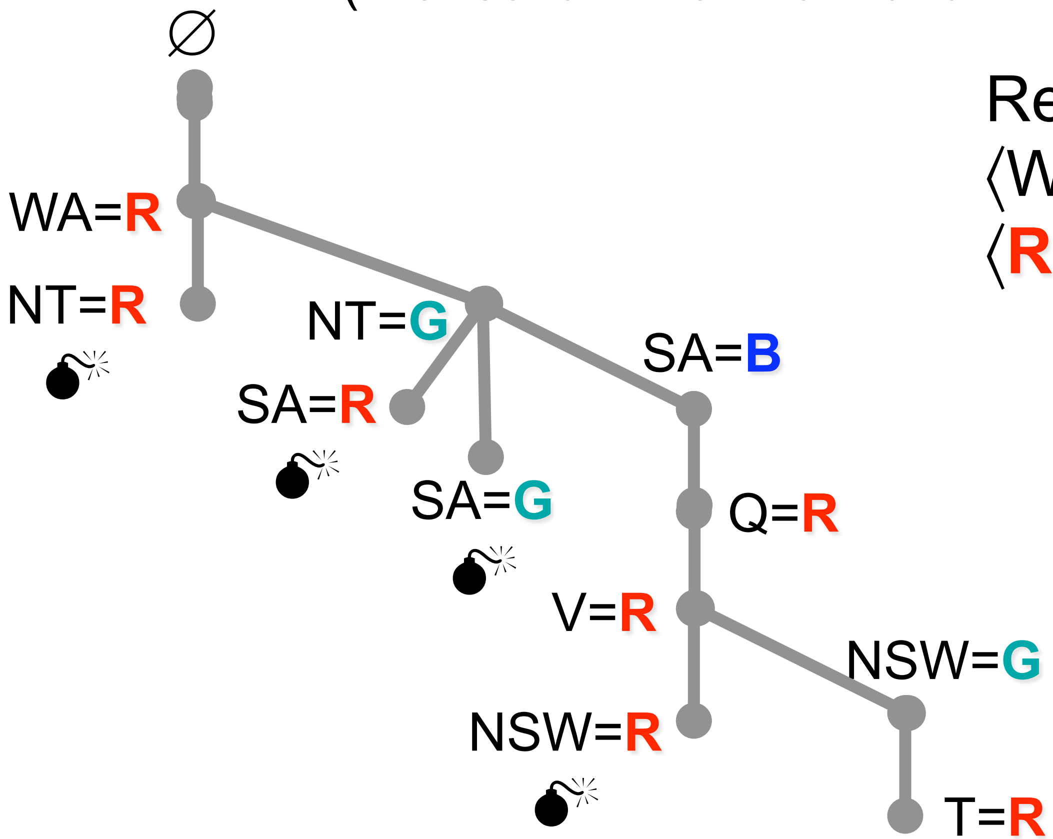
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING([], csp)

function RECURSIVE-BACKTRACKING(assigned, csp) returns solution/failure
    if assigned is complete then return assigned
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assigned, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assigned, csp) do
        if value is consistent with assigned according to CONSTRAINTS[csp] then
            result ← RECURSIVE-BACKTRACKING([var = value|assigned], csp)
            if result ≠ failure then return result
    end
    return failure
```

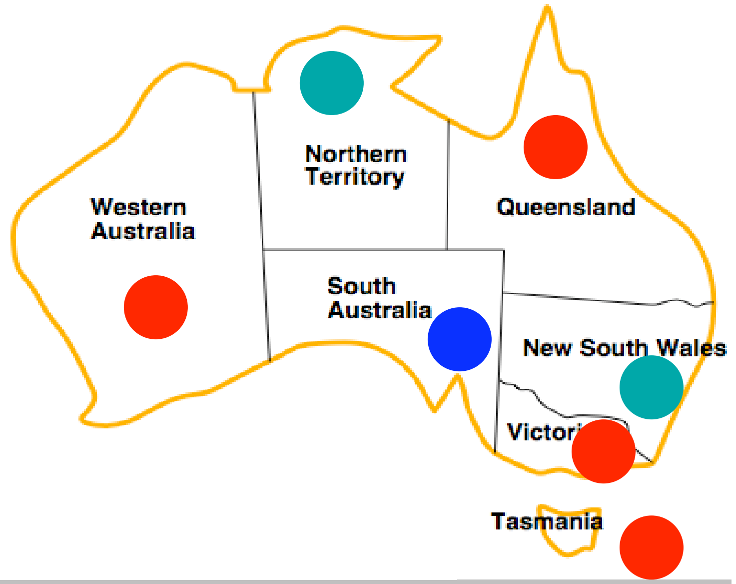
➥ Wie würden Sie das in Prolog implementieren?

Beispiel: Australien per Backtracking

Wichtige Entscheidung: Reihenfolge, in der Variablen und ihre Werte versucht werden!
(Hier schummle Information ins uninformierte Backtracking ein!)



Reihenfolge hier:
⟨WA, NT, SA, Q, V, NSW, T⟩;
⟨R, G, B⟩ für alle Variablen



Eigenschaften des Backtracking-CSP-Lösers

☺ Vollständig für diskrete endliche Wertebereiche

☹ Laufzeit $O(d^n)$ bei n Variablen und max. Wertebereichgröße d

☺ Speicher: Constraintnetz der Größe $O(n^2d^2)$ (für Binärconstraints!)

- für reale Probleme in der Regel zu ineffizient
- geeignet als Referenzverfahren

Backtracking – jetzt mit Verstand

Heuristik der ***Minimum Remaining Values*** (MRV, *fail-first*) in Funktion SELECT-UNASSIGNED-VARIABLE :

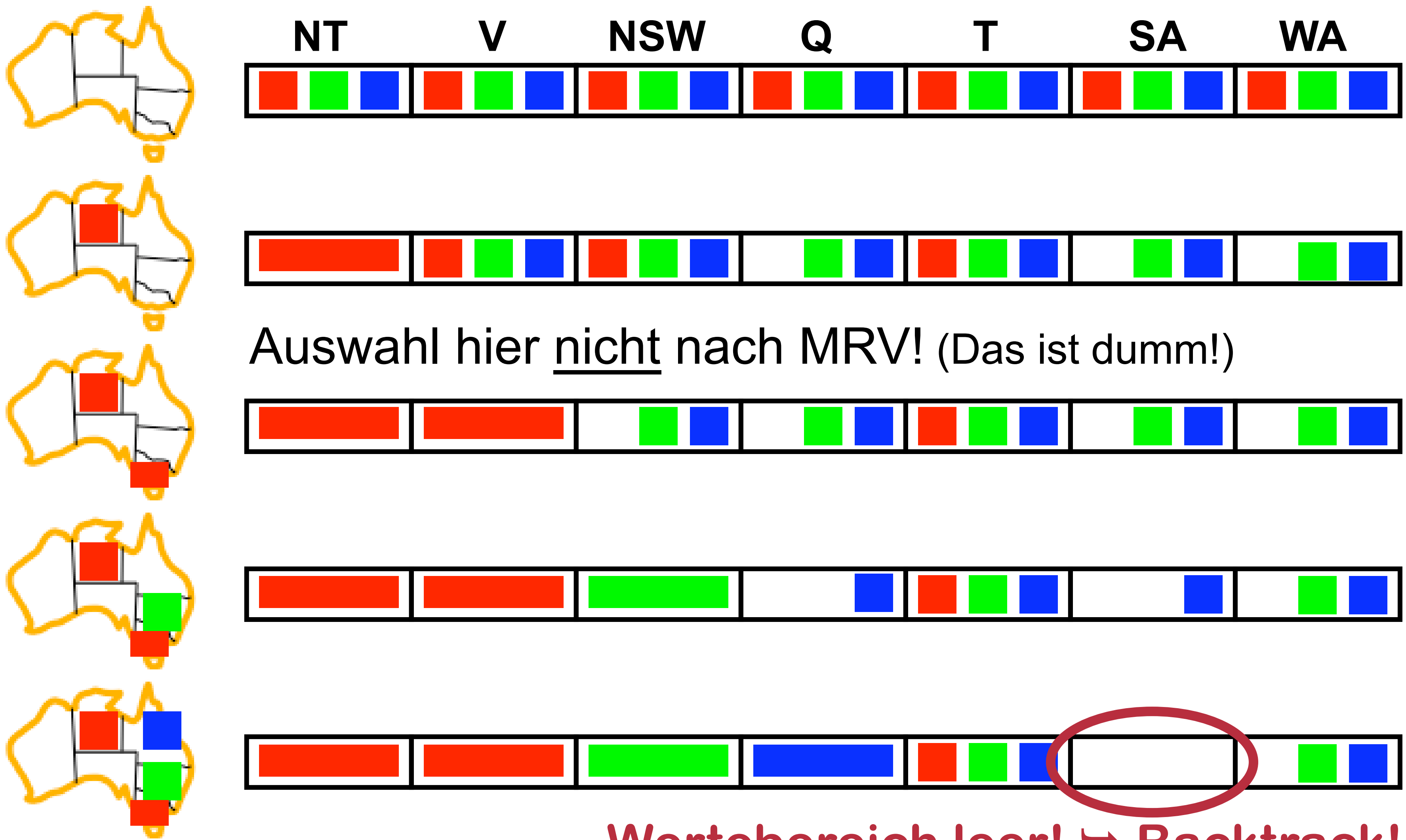
Wähle Variable mit kleinstem aktuellem/verbleibendem Wertebereich!

Forward Checking (einfache Propagierung):

Wann immer ein Wert v an X zugewiesen wird, **maskiere** in allen Variablen, mit denen X durch einen Constraint C verbunden ist, alle mit v inkonsistenten Werte!

(Offensichtlich kombinierbar mit MRV)

Forward Checking in Australien



Auswahl hier <u>nicht</u> nach MRV! (Das ist dumm!)

Wertebereich leer! ↪ Backtrack!

Lokale Konsistenz (Kantenkonsistenz)

Sei C ein k-stelliger Constraint ($k \geq 2$) und seien X, Y zwei der k Variablen von C. Die **Kante** im CSP von X über C nach Y ist **konsistent**, gdw. es für jede mögliche Belegung x von X einen Wert y von Y und ein k-Tupel $T \in R_C$ gibt, sodass (x,y) die Projektion von T auf (X,Y) ist.

„Für jeden Wert x gibt es einen erlaubten Wert y"

Ein CSP über den Variablen $\{X_1, \ldots, X_n\}$ ist **lokal konsistent** (**kantenkonsistent**), gdw. es bezüglich aller seiner Kanten konsistent ist und für alle X_i mit Wertebereich D_i gilt $D_i \neq \emptyset$.

„Das Netz enthält lokal plausible Werte, und nur diese"