Backtracking-Search Pseudocode Explanation (CSP)

Let's break down and analyze BACKTRACKING-SEARCH pseudocode as shown in your textbook. We have two functions there. First is the main one (think: startTheCSPSearch):

function BACKTRACKING-SEARCH(*csp*) **returns** a solution or *failure*
  return BACKTRACK(*csp*, { })

The argument for this function is *csp* (for constraint satisfaction problem). This is going to be some sort of a programming language representation of the entire constraint satisfaction problem that you want the BACKTRACKING-SEARCH algorithm to solve. How would you represent *csp* in your program is up to you, but it will need to include

CSP:
Variables
Domains
Constraints

(Note that domains and constraints may change as we go).

BACKTRACKING-SEARCH can return one of two things:

- a *solution*: it went through the search tree and found such assignments of values (from corresponding variable domains) to variables that all constraints are satisfied, or
- a *failure*: it went through the search tree and decided that an assignment of values (chosen from corresponding variable domains) to variables that all constraints are satisfied does NOT exist.

The phrase "went through the search tree" is reflected in the line:

return BACKTRACK(*csp*, *{}*)

which means BACKTRACKING-SEARCH will initiate a RECURSIVE tree search with FIRST call to the BACKTRACK function. Backtracking-Search will wait for the result (a *solution* or a *failure*) reported by the BACKTRACK() function and return it.

BACKTRACKING-SEARCH will PASS two arguments to the INITIAL (think:it will start at the tree root) call to BACKTRACK:

- *csp*: the constraint satisfaction problem representation it received
- INITIAL assignment, which is going to be empty (no variables have values). *{}* is meant to represent an empty set / variables without values YET. For the Australia map, it would be

Assignment:
WA=?? NT=??

Q=?? NSW=??
V=?? SA=??
T=??

?? - no value assigned

This gets us to the second function, BACKTRACK:

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

This is a RECURSIVE function, because it can call ITSELF:

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

It accepts two arguments:

- *csp*: the constraint satisfaction problem representation,
- *assignment*: CURRENT assignment of values to variables (think: at each subsequent level of the tree, another variable will get it value assigned).

When BACKTRACK function arrives at the tree LEAF, assignment is COMPLETE (all variables have values). This is the end of this particular search tree branch (even if it is a dead end). It is time to go back a level ("Backtrack") instead of going deeper into the tree and to report the current assignment:

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

If the assignment is INCOMPLETE (there is at least one variable left to assign a value to), we need to pick the next unassigned variable *var* (this is decided by the SELECT-UNASSIGND-VARIABLE function; could simply be using static ordering and pick next variable on the list):

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

Once the variable to be assigned a value (*var*) is selected, we iterate over its domain (all possible values that can be assigned) to try and pick a value that works (makes the assignment consistent):

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

If our consistency check is satisfied, we can move on and add the *{var = value}* pair to the assignment.

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

If not, we iterate and try the next possible value for variable *var*.

Once the loop is over and there are no more values to try, current call to the BACKTRACK function will return *failure* (it is a dead end).

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
      if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
          add inferences to csp
          result ← BACKTRACK(csp, assignment)
          if result ≠ failure then return result
          remove inferences from csp
        remove {var = value} from assignment
  return failure
```

However, if that *{var = value}* pair is consistent we can continue down the tree to try another UNASSIGNED variable. INFERENCE function will try to look ahead (for example by using forward checking) to decide if it is worth continuing down the tree structure:

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
      if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
          add inferences to csp
          result ← BACKTRACK(csp, assignment)
          if result ≠ failure then return result
          remove inferences from csp
        remove {var = value} from assignment
  return failure
```

If yes, **update/modify** the *csp* representation (for example reduce domains):

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

Now, remembering that trees are recursive structures (subtrees are also trees), a recursive search call is made to BACKTRACK itself (with UPDATED *csp* and CURRENT *assignment* as arguments). You can think of it as starting a new search on a subtree:

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

This call will return a result (either a *failure* or a new *assignment*).

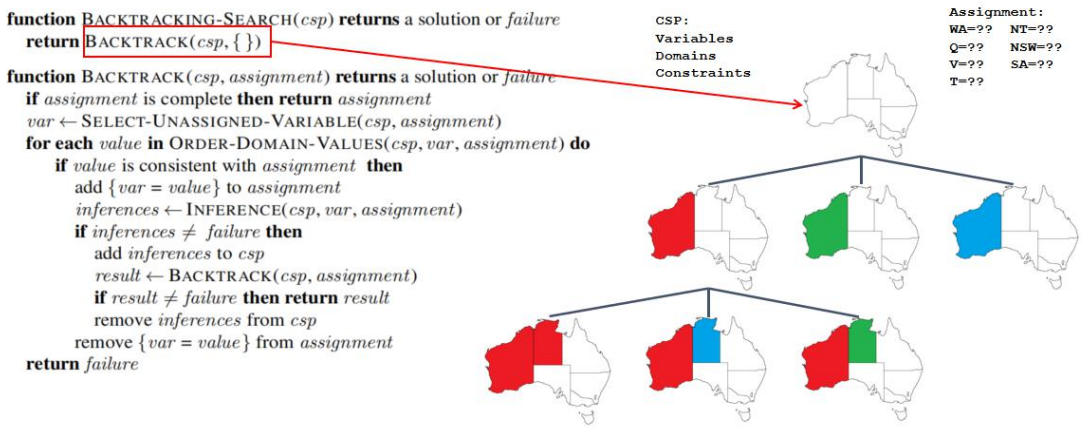
If it is not a *failure*, we can return the *assignment* (at this point we are at the leaf!) back to the function that called this instance of BACKTRACK (in most cases it will be another instance of BACKTRACK, up a level, but eventually, the assignment, will bubble up all the way to BACKTRACK-SEARCH).

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

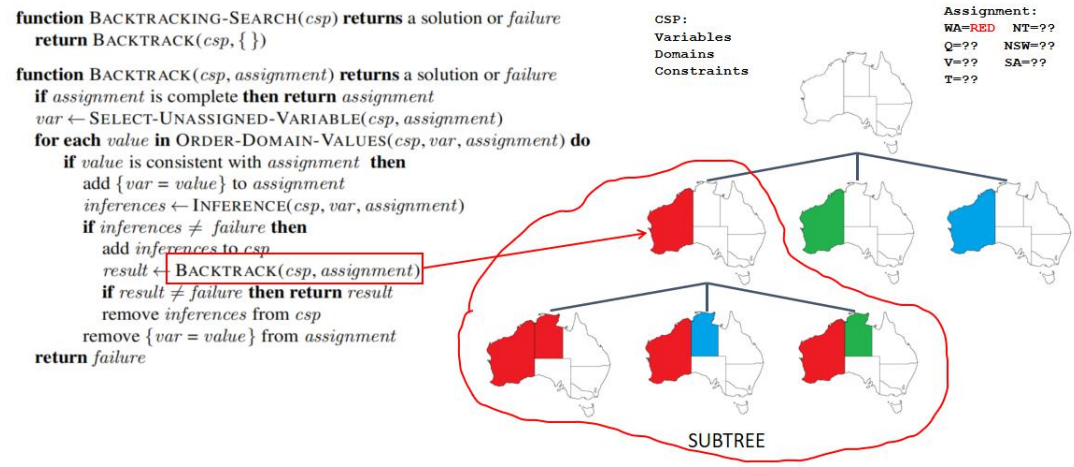
If it is a *failure*, we hit a dead end, and we need to stop exploring the search tree and go back one level. Before we do that, we need to reverse all changes made at this node / in this BACKTRACK call:

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

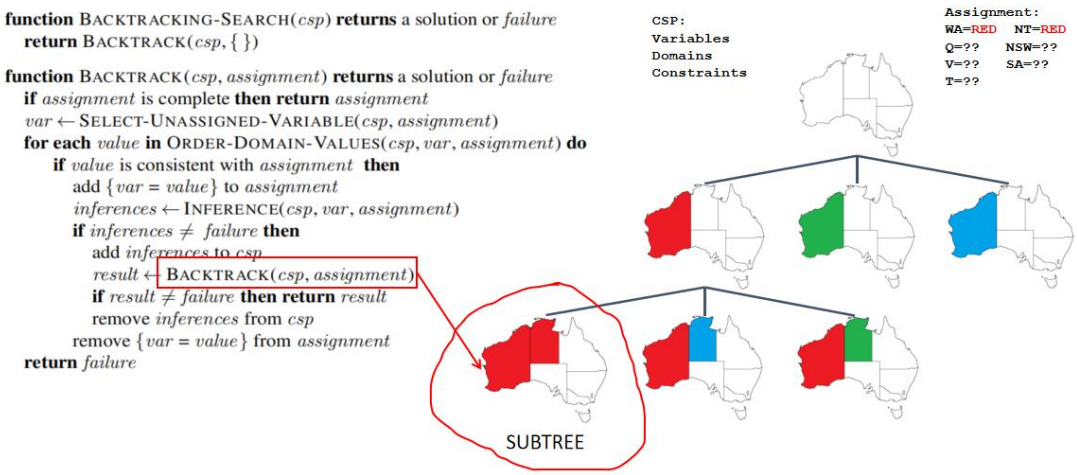
For the Australia Map problem, the initial call to BACKTRACK would be made here (note: I picked specific variable and value orderings to illustrate the point):



with initial "empty" *assignment*. This first call would pick WA as the first VARIABLE to assign the VALUE to. Once WA is selected as VARIABLE, the for loop will start with the pairing WA = RED and add it to the *assignment*, then it will make a RECURSIVE call to BACKTRACK using CURRENT *assignment*:



The story repeats. Eventually, another call to BACKTRACK is made and another variable is assigned a value (NT = RED):



This assignment violates a constraint (neighboring regions cannot be colored with the same color) and it is a dead end. BACKTRACK function will reverse changes and... backtrack one level to try NT=BLUE.

And so on...