Definition: Arc consistency

• A constraint C_xy is said to be arc consistent wrt x if for each value v of x there is an allowed value of y
• Similarly, we define that C_xy is arc consistent wrt y
• A binary CSP is arc consistent iff every constraint C_xy is arc consistent wrt x as well as y
• When a CSP is not arc consistent, we can make it arc consistent, e.g. by using AC3
  – This is also called “enforcing arc consistency”
Let’s define the 4-queens problem as a CSP with the variable $X_i$ denoting the position (row) of the queen on column $i$.

Remember the constraints: two queens attack each other when they are in the same row, the same column or on the same diagonal. We want to place $n=4$ queens on the board so no queen is attacking another.
Suppose we set $X_1 = 1$

Show the effect of forward checking on the domains of the remaining variables
(I suggest crossing off values in the lists below:)

- Domain $X_2 = \{1, 2, 3, 4\}$
- Domain $X_3 = \{1, 2, 3, 4\}$
- Domain $X_4 = \{1, 2, 3, 4\}$
Forward checking will delete values from the domains of all other variables, as shown.
Question 2

Is this CSP now arc consistent?

(for the purposes of this question – assume that there is one constraint between each pair of queens that rules out all attacks)
No, the constraint between X2 and X3 is not consistent with respect to X2. There exists a value in the domain of X2 (specifically X2=3) such that NO value for X3 will work. Furthermore, the constraint between X4 and X3 is not consistent with respect to X4, because X4=3 also leaves X3 with no legal values.
Simulate the behavior of AC3 to make the CSP arc consistent
First subquestion, what goes on the queue?
For each pair of variables, you need to put a directed constraint. I’ll write $X_2 \rightarrow X_3$ to mean the constraint wrt $X_2$ (ie $X_2$ is the tail). For this example, let’s ignore constraints with $X_1$ because those constraints are consistent (as a result of forward checking) and can’t become inconsistent because we’ve chose a single value for $X_1$. So the queue might be $<X_2 \rightarrow X_3, X_2 \rightarrow X_4, X_3 \rightarrow X_2, X_3 \rightarrow X_4, X_4 \rightarrow X_2, X_4 \rightarrow X_3>$ We’ve already established that $X_2 \rightarrow X_3$ is inconsistent. **What does AC-3 do to fix this?**
AC-3 deletes from the domain of ... X2...
So now Domain(X2) = {4}
AC-3 also adds some more constraints onto the queue, X3→X2 and X4→X2, but since they are already there there is no change. So the queue is <X2→X4, X3→X2, X3→X4, X4→X2, X4→X3>

Is X2→X4 consistent?
Answer 6

Yep. Now the queue is
<X3→X2, X3→X4, X4→X2, X4→X3>

Is X3→X2 consistent?
Nope. We need to delete $X_3=4$.
That means we need to add some stuff to the queue.
**So what’s the queue become?**
We add $X_2 \rightarrow X_3$ and $X_4 \rightarrow X_3$ but the latter was already there so we get $<X_3 \rightarrow X_4, X_4 \rightarrow X_2, X_4 \rightarrow X_3, X_2 \rightarrow X_3>$

Now what happens when we process the next constraint?
X₃→X₄ is inconsistent so we need to remove X₃=2, but now X₃’s domain is empty, which means that the CSP is unsolvable. So the very first decision to Assign X₁=1 was a mistake.

In fact, following the pseudocode, AC3 will keep running and remove some more stuff – a bit pointless. But I’ll stop here.
Part II – Tree structured CSPs

Let’s color this!
Tree-Structured CSPs

- Algorithm for tree-structured CSPs:
  1. Order: Choose a root variable, order variables so that parents precede children
  2. Remove backward: For $i = n : 2$, apply RemoveInconsistent(Parent($X_i$),$X_i$)
  3. Assign forward: For $i = 1 : n$, assign $X_i$ consistently with Parent($X_i$)
Tree-Structured CSPs

- **Algorithm for tree-structured CSPs:**
  1. **Order:** Choose a root variable, order variables so that parents precede children

My choice to start with A as the root is arbitrary – could have started with anything else. It also doesn’t matter if B comes before C in the ordering etc.
Question 10

- **Algorithm for tree-structured CSPs:**
  1. Order: Choose a root variable, order variables so that parents precede children
  2. Remove backward: For $i = n : 2$, apply RemoveInconsistent(\text{Parent}(X_i), X_i)

Suppose that the initial legal colors are as I show above

**Simulate step 2 of the algorithm** (I suggest cross off colors in the diagram above)
Algorithm for tree-structured CSPs:

1. Order: Choose a root variable, order variables so that parents precede children
2. Remove backward: For $i = n : 2$, apply $\text{RemoveInconsistent}(\text{Parent}(X_i), X_i)$

When processing $D \rightarrow F$, we need to remove blue from the domain of $D$

What about when we process $D \rightarrow E$?
Algorithm for tree-structured CSPs:

1. Order: Choose a root variable, order variables so that parents precede children
2. Remove backward: For $i = n : 2$, apply RemoveInconsistent(Parent($X_i$),$X_i$)

When processing $D \rightarrow E$, we don’t do anything. We would only remove something from the parent, $D$, but red is consistent, because we can make $E$ green. So we just leave it as is.

What about $B \rightarrow D$?
Algorithm for tree-structured CSPs:
1. Order: Choose a root variable, order variables so that parents precede children
2. Remove backward: For $i = n : 2$, apply RemoveInconsistent(\text{Parent}(X_i),X_i)

When processing $B \rightarrow C$, we don't do anything.
What about $A \rightarrow B$?
Algorithm for tree-structured CSPs:

1. Order: Choose a root variable, order variables so that parents precede children
2. Remove backward: For \( i = n : 2 \), apply \( \text{RemoveInconsistent} (\text{Parent}(X_i), X_i) \)
3. Assign forward: For \( i = 1 : n \), assign \( X_i \) consistently with \( \text{Parent}(X_i) \)

Right, we delete blue from A.

**Now simulate step 3.**

Any choice for A is ok. B will be blue. C can be red or green. D is red, E will be green… It all works!!