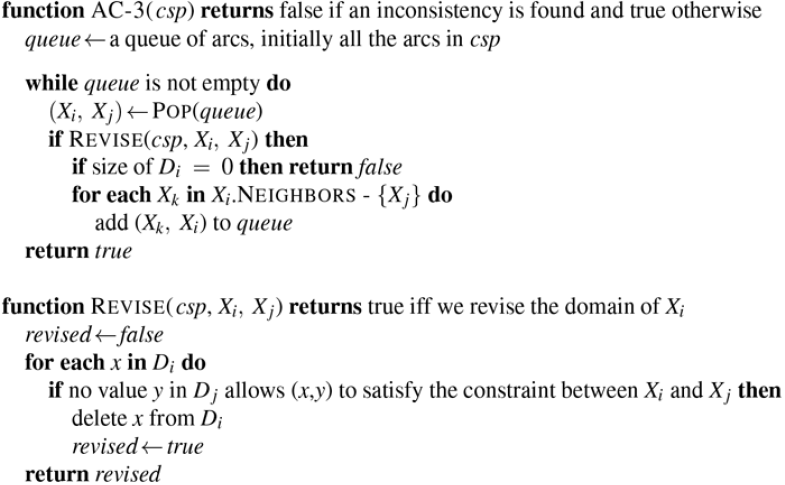
* **Factored representation** - representing a state as a set of variables each of which has a value
  + **Constraint satisfaction problem (CSP)** is solved when each variable has value that satisfies all constraints on the variable
  + Can take advantage of the structure of states and use general rather than domain specific heuristics to enable the solution of complex problems
  + **Eliminate large portions of search space all at once by identifying variable/value combos that violate the constraints**
  + Actions and transition model can be deduced from the problem description
* Constraint satisfaction problem consists of
  + *X* is set of variables
  + *D* is a set of domains, one for each variable
    - Domain *Di* consists of a set of allowable values *v* for variable *X*i
      * E.g. Boolean variable would have domain {true, false}
      * Different variables can have different domains of different sizes
  + *C* is a set of constraints that specify allowable combinations of values
    - Each constraint *Cj* consists of pair <*scope, rel*>
      * *scope* is a tuple of variables that participate in the constraint
      * *rel* is a relation that defines the values that those variables can take oin
        + Relation can be represented as explicit set of tuples of values that satisfy the constraint
        + Or function that compute whether a tuple is a member of that relation
  + An assignment of values to variables that do not violate any constraints is **consistent** or **legal**
  + Assignment in which every variable is assigned a value is **complete**
  + Solution to CSP is a consistent and complete assignment
    - Partial assignment is one that leaves variables unassigned
    - Partial solution is a partial assignment that is consistent
* **Constraint graph -** nodes of the graph correspond to variables of the problem, and edge connects any 2 variables that participate in a constraint
* Why formulate problem as CSP?
  + CSPs yield natural representation for wide variety of problems
  + Lot of work gone into making CSP solvers fast
  + CSP can help quickly prune large state spaces that atomic state space searcher can’t do
    - When partial assignment violates constraint can discard further refinements
    - Can see why the assignment is not a solution
* **Absolute constraints** - constraints which when violated, rule out a potential solution
* **Precedence constraints** - Whenever a task *T1* must occur before *T2* and *T1* takes duration *d1* to complete
  + *T1* + *d1* *T2*
* **Disjunctive constraint** - When tasks *T1* and *T2* must not overlap in time. Either one comes first or the other
  + *T1* + *d1* *T2* or *T2* + *d2* *T1*
* **Unary constraint** - restricts the value of a single variable
  + *X1*  *v1*
* **Binary constraint** - relates two variables
  + E.g. *X1*  *X2*
* **Binary CSP -** CSP with only unary and binary constraints
* **Global constraint** - constraint involving arbitrary number of variables
  + E.g. *Alldiff* - all variables involved in constraint must have different variables
* Simplest kind of CSP involves variables that have discrete and finite domains
  + Discrete domain can be infinite (set of integers or strings)
    - With infinite domains must use implicit constraints rather than explicit tuples of values
      * E.g. *T1* + *d1* *T2*
    - Special solution algorithms exist for linear constraints on integer variables
    - No algorithm exists for solving general nonlinear constraints on integer variables
* **Constraint hypergraph -** consists of ordinary nodes and hypernodes which represent constraints involving *n* variables
* Every finite domain constraint can be reduced to a set of binary constraints if enough auxiliary variables are introduced
  + Can transform any CSP into one with only binary constraints
  + **Dual graph transformation** - creating a new graph in which where there will be 1 variable for each constraint in the original graph and one binary constraint for each pair of constraints in the original graph that share variables
* Might prefer global constraint rather than set of binary constraints
  + Less error prone to write the problem description
  + Possible to design special purpose inference algorithm for global constraints that are more efficient than operating with primitive constraints
* **Preference constraints** - indicate which solutions are preferred
  + Common in real world CSPs
  + Can be encoded as costs on individual variable assignments
  + **Constrained Optimization Problem (COP) -** CSPs with preferences can be solved with optimization search methods (path-based or local)
* CSP algorithm can generate successors by
  + Choosing a new variable assignment
  + Or by **constraint propagation** - using constraints to reduce number of legal values for a variable
    - Which in turn reduces legal values for another variable
    - May be intertwined with search or preprocessing step
      * Sometimes preprocessing can solve the whole problem! No search required
* **Local consistency** - If we treat each variable as a node in a graph and each binary constraint as edge, eliminate inconsistent values throughout the graph
  + **Node-consistent** - A variable is node consistent if all the values in the variable’s domain satisfy the variable’s unary constraints
    - A graph is node consistent if every variable in graph is node consistent
  + **Arc-consistent** - A variable is arc consistent if every value in its domain satisfies the variable’s binary constraints
    - *Xi* is arc-consistent to *Xj* if for every value in the domain *Di*there is some value in domain *Dj* that satisfies the binary constraint on the arc/edge (*Xi*, *Xj)*
    - Graph is arc consistent if every variable is arc consistent with every other variable
    - AC-3 - Popular algorithm for enforcing arc consistency
    - Arc-consistent CSP will be faster to search because variables have smaller domains
    - Worst-case time: *O(cd3)*

* + **Path consistency -** tightens binary constraints by using implicit constraints that are inferred by looking at triples of variables
    - Two variable set {*Xi, Xj*} is path consistent to *Xm* if for every assignment {*Xi = a, Xj = b*} consistent with the constraints on {*Xi, Xj}* there is an assignment to *Xm* that satisfies the constraints on {*Xi, Xm*} and {*Xm, Xj*}
  + **k-consistency -** A CSP is *k*-consistent if for any set of *k* - 1 variables and for any consistent assignment to those variables a consistent value can always be assigned to any *k*th variable.
    - 1-consistency: given empty set we can make any set of one variable consistent (node consistency)
    - 2-consistency: arc consistency
    - For binary constraint graphs 3-consistency is path consistency
    - **Strongly k-consistent:** if CSP is *k* consistent and is also (*k-*1) consistent, (*k* - 2) consistent, all the way down to 1-consistent
* **Global constraints** - one involving an arbitrary number of variables but not necessarily all variables
  + Simple consistency procedure for higher-order constraint is sometimes more effective than applying arc consistency to an equivalent set of binary constraints
  + **Resource (*Atmost)* constraint**
    - E.g. scheduling problem *P1…P4* denotes number of personnel assigned to 4 task
    - Atmost(10,*P1, P2, P3, P4*): Constraint that no more than 10 personnel are assigned in total
      * Can detect inconsistency by checking sum of minimum values of current domains
    - Can enforce consistency by deleting maximum value of any domain if it’s not consistent with minimum values of other domains
* For resource-limited problems its not possible to represent domain of each variable and gradually reduce that set by consistency checking methods
  + Domains are managed by **bounds propagation**: upper and lower bounds
  + **Bounds consistent:** CSP is bounds-consistent if for every variable *X* and for lower and upper bound values of *X* there exists some value of *Y* that satisfies the constraint between *X* and *Y* for every variable *Y*
* **Sometimes we can finish constraint propagation process and still have variables with multiple possible values, and we have to search for a solution**
  + **Commutativity:** problem is commutative if the order of application of any given set of actions doesn’t matter
    - Crucial property of CSPs
    - Only need to consider a single variable at each node in search tree
    - With this number of leaves is *dn* instead of *n*! \* *d*n
* Backtracking-search
  + Only keeps single representation of a state and alters that representation rather than creating new ones
  + Can be improved using domain independent heuristics that take advantage of the factored representations of CSPs
    - **Which variable should be assigned next and in what order should its values be tried?**
      * Simplest strategy is static ordering, or randomly but neither optimal
      * **Minimum remaining values** (MRV) heuristic: Choose the variable with the fewest legal values
        + Picks a variable that is most likely to cause a failure soon, thereby pruning search tree
        + If variable *X* has no legal values, MRV will select it and failure will be detected immediately avoiding pointless searching through other variables
      * **Degree heuristic:** attempts to reduce branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables
        + MRV is usually more powerful, but degree heuristic can be a good tiebreaker
      * What order should the algorithm examine its values?
        + **Least-constraining value heuristic**: prefers the value that rules out the feast choices for neighboring variables in constraint graph

Tries to leave maximum flexibility for subsequent variable assignments

We only need one solution, makes sense to look for the most likely values first

* + - **What inferences should be performed at each step in the search?**
      * **Forward checking**: Whenever a variable *X* is assigned, establish arc consistency for it
        + For each unassigned variable *Y* that is connected to *X* by a constraint, delete from *Y*’s domain any value that is inconsistent with the value chosen for *X*
        + Search will be effective if combine the MRV heuristic with forward checking
        + Doesn’t detect all inconsistencies because it doesn’t look far ahead enough
      * **Maintaining Arc Consistency (MAC):** After a variable *Xi* is assigned a value, the INFERENCE procedure calls AC-3 but instead of a queue of all arcs in CSP, only start with arcs (*Xj*, *Xi*) for all *Xj* that are unassigned variables that are neighbors of *Xi*
        + AC-3 does constraint propagation and if any variable has its domain reduced to empty set, AC-3 fails and we can backtrack
        + Forward checking doesn’t recursively propagate constraints when changes are made to domains of variables
    - **Can we backtrack more than one step when appropriate?**
      * **Chronological backtracking**: back up to the preceding variable try a different value
        + Simplest. We can do better
      * **Chronological backjumping:** backtracks to most recent assignment in the conflict set
        + When we reach a contradiction backjumping can tell us how far to back up so we don’t waste time changing variables that won’t fix the problem
        + Backtrack to the variable that might fix the problem because it was responsible for making one of the possible values of *Xi*impossible
        + Keep track of a set of assignments that are in conflict with some value for *Xi*
        + Accumulates conflict set while checking for a legal value to assign

If no legal value is found, algorithm should return most recent element of conflict set along with failure indicator

* + - * + Forward checking can supply conflict set with no extra work

Whenever forward checking based on assignment *X = x* deletes value from *Y*’s domain it should add *X = x* to *Y*’s conflict set

If the last value is deleted from *Y*’s domain the assignments in the conflict set of *Y* are added to the conflict set of *X*

Now we know *X = x* leads to a contradiction in *Y* and a different assignment should be tried

* + - * + Every branch pruned by backjumping is also pruned by forward checking

Simple backjumping is redundant in a forward checking search or a search that uses MAC

ONLY DO ONE OR THE OTHER

* + - * **Conflict-directed backjumping**
        + Backtrack based on the reasons for failure

Sometimes failure is because of a set of preceding variables together with any subsequent variables to have no consistent solution

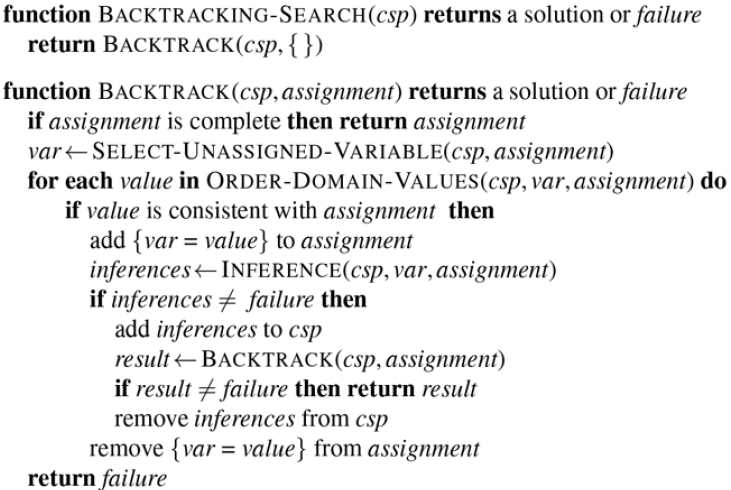
* + - * + *Xj*is current variable and *conf*(*Xj)* is its conflict set
        + If every possible value for *Xj* fails, backjump to the most recent variable *Xi* in *conf(Xj)* and recompute the conflict set for *X*i
    - **Can we save and reuse partial results from the search?**
      * **Constraint learning** - finding a minimum set of variables from the conflict set that causes the problem/contradiction
        + Set of variables and their values are called **no-good**
        + Record no-good by

adding new constraint to CSP

forbidding combination of assignments

caching no-goods

* + - * No-goods can be used by forward checking or backjumping



* **Local search for CSPs** are effective in solving CSPs using complete-state formulation
  + Each state assigns a value to every variable and the search changes the value of one variable at a time
  + Given a variable, change value to something that brings us closer to a solution
    - **Min-conflicts heuristic**: selecting value that results in minimum number of conflicts with other variables
      * Run time is roughly independent of problem size
  + All local search techniques are candidates for application to CSPs
  + Landscape of CSP under min conflicts heuristic has series of plateaus
    - **Plateau search**: allowing sideways moves to another state with the same score (helping local search find way off plateau
    - **Tabu search**: keeping small list of recently visited states and forbidding algorithm to return to those states
  + **Constraint weighting** - concentrate search on important constraints
    - Each constraint is given a numeric weight (initially 1)
    - Each step of search algorithm chooses variable/value pair to change that will result in lowest total weight of all violated constraints
    - Weights then adjusted by incrementing weight of each constraint that is violated by current assignment
      * Adds topography to plateaus
      * Adds learning
        + Difficult constraints are assigned higher weights over time
  + Local search can be used in an online setting
* **How can the structure of the problem, represented by the constraint graph, be used to find oslutions quickly?**
  + Need to decompose problems in the real world into **independent subproblems**
    - Any solution for the larger problem combined with the solution of the independent subproblem yields a solution
  + Independence can be ascertained by finding **connected components** of the constraint graph
    - Each component corresponds to a subproblem CSPi
    - If assignment *Si* is a solution of CSPi then *Si* is a solution of *CSPi*
      * Each *CSPi* has *c* variables from total of *n* variables where *c is a constant*
      * Then there are *n/c* subproblems which takes at most *dc* work to solve
        + *d -* size of domain
      * Total work is O(*dcn/c)* which is linear
      * Without decomposition total work is O(*dn*) which is exponential
  + Constraint graph is a tree when any two variables are connected by only one path
  + Any tree structured CSP can be solved in time linear by the number of variables
    - **Directional Arc Consistency (DAC)**
      * CSP is defined to be DAC under a ordering of variables *X1…Xn* if every *Xi* is arc-consistent with each *Xj* for *j* > *i*
    - Solving a tree-structured CSP
      * Pick any variable to be root of tree
      * **Topological sort:** Choose ordering of variable such that each variable appears after its parent
      * Any tree with *n* nodes has *n -* 1 edges so graph can be DAC in O(*n*) steps
        + Each of which must compare up to *d* possible domain values for two variables for time O(*nd2*)
      * With DAC graph can just go down list of variables and choose any remaining value
        + Each edge from a parent to its child is arc-consistent

Know that for any value we choose for parent, there will be valid value left to choose for child

No need to backtrack, can move linearly

* How can we reduce general constraint graph to trees?
  + **Cutset conditioning** - assigning values to some variable sos that remaining variables form a tree. Can solve remaining tree with TREE-CSP-SOLVER
    - General algorithm
      * Choose a subset *S* of the CSP’s variables such that the constraint graph becomes a tree after removal of *S. S* is called **cycle cutset**
      * For each possible assignment to the variables in *S* that satisfies all constraints on *S*
        + Remove from the domains of remaining variables any values that are inconsistent with assignment for *S*
        + If remaining CSP has solution return it together with assignment for *S*
  + **Tree Decomposition** - transformation of original graph into a tree where each node in tree consists of a set of variables
    - Must satisfy these requirements
      * Every variable in the original problem appears in at least one of the tree nodes
      * If two variables are connected by constraint in original problem, they must appear together with constraint in at least one of the tree does
      * If variable appears in two nodes in tree, it must appear in every node along the path connecting those nodes
    - WIth a tree-structured graph, can apply TREE-CSP-SOLVER and get a solution in O(*nd2*) time
      * *n* number of nodes, *d* is size of largest domain
      * In the tree domain is a set of tuples of values, not just individual values
    - Aim is to make subproblems as small as possible
    - **Tree width** of a tree decomposition graph is one less than the size of the largest node
      * Tree width of the graph itself is defined to be the minimum width among all its tree decompositions
    - CSPs with constraint graphs of bounded tree width are solvable in polynomial time
  + Time consideration favors tree decomposition but cycle-cutset cab be executed in linear memory
* **Value symmetry -** reduce search space by a factor of *d!* by breaking symmetry in assignments
  + **Symmetry-breaking constraint**
    - E.g. arbitrary ordering constraint that ensures only one of the *d!* solutions is possible