Constraint Satisfaction Problem [Var, Dom, Con]

- a list $\mathrm{Var} = [X_1, \ldots, X_n]$ of *variables* X_i
- a list $\mathrm{Dom} = [D_1, \ldots, D_n]$ of finite sets D_i of size s_i
- a finite set Con of *constraints* that may or may not be satisfied by (a node) instantiating X_i with a value in D_i (search space size $\prod_{i=1}^{n} s_i$)

Constraint Satisfaction Problem [Var, Dom, Con]

- a list $\text{Var} = [X_1, \ldots, X_n]$ of *variables* X_i
- a list $\text{Dom} = [D_1, \ldots, D_n]$ of finite sets D_i of size s_i
- a finite set Con of *constraints* that may or may not be satisfied by (a node) instantiating X_i with a value in D_i (search space size $\prod_{i=1}^{n} s_i$)

E.g. SAT: $D_i = \{0, 1\}$, $s_i = 2$ for search space of size 2^n

Constraint Satisfaction Problem [Var, Dom, Con]

- a list Var $= [X_1, \ldots, X_n]$ of *variables* X_i
- a list Dom $= [D_1, \ldots, D_n]$ of finite sets D_i of size s_i
- a finite set Con of *constraints* that may or may not be satisfied by (a node) instantiating X_i with a value in D_i (search space size $\prod_{i=1}^{n} s_i$)

E.g. SAT: $D_i = \{0, 1\}$, $s_i = 2$ for search space of size 2^n

Problem: satisfy all constraints in Con, instantiating variables if necessary/convenient

Constraint Satisfaction Problem [Var, Dom, Con]

- a list $\text{Var} = [X_1, \ldots, X_n]$ of *variables* X_i
- a list $\text{Dom} = [D_1, \ldots, D_n]$ of finite sets D_i of size s_i
- a finite set Con of *constraints* that may or may not be satisfied by (a node) instantiating X_i with a value in D_i (search space size $\prod_{i=1}^{n} s_i$)

E.g. SAT: $D_i = \{0, 1\}$, $s_i = 2$ for search space of size 2^n

Problem: satisfy all constraints in Con, instantiating variables if necessary/convenient

```
| ?- X\=Y, X=a, Y=b.
no
```

Constraint Satisfaction Problem [Var, Dom, Con]

- a list Var $= [X_1, \ldots, X_n]$ of *variables* X_i
- a list Dom $= [D_1, \ldots, D_n]$ of finite sets D_i of size s_i
- a finite set Con of *constraints* that may or may not be satisfied by (a node) instantiating X_i with a value in D_i (search space size $\prod_{i=1}^{n} s_i$)

E.g. SAT: $D_i = \{0, 1\}$, $s_i = 2$ for search space of size 2^n

Problem: satisfy all constraints in Con, instantiating variables if necessary/convenient

```
| ?- X\=Y, X=a, Y=b.
no
```

```
| ?- X=a, Y=b, X\=Y.
X = a, Y = b
```

Order-independent unification (Martelli-Montanari)

Input: set $\mathcal{E}$ of pairs $[t, t']$
Output: substitution $[[X1, t1], \ldots, [Xk, tk]]$ unifying pairs in $\mathcal{E}$

Order-independent unification (Martelli-Montanari)

Input: set $\mathcal{E}$ of pairs $[t, t']$
Output: substitution $[[X1, t1], \ldots, [Xk, tk]]$ unifying pairs in $\mathcal{E}$

Simplify $\mathcal{E}$ non-deterministically until no longer possible

1. $[f(s1, \ldots, sk), f(t1, \ldots, tk)]$ (allowing $k = 0$)
   $\Longrightarrow$ replace by pairs $[s1, t1], \ldots, [sk, tk]$
2. $[f(s1, \ldots, sk), g(t1, \ldots, tm)]$ where $f \neq g$ or $k \neq m$
   $\Longrightarrow$ halt with failure
3. $[X, X] \Longrightarrow$ delete
4. $[t, X]$ where t is not a var $\Longrightarrow$ replace by $[X, t]$
5. $[X, t]$ where $X \notin Var(t)$ and X occurs elsewhere
   $\Longrightarrow$ apply $[X, t]$ to all other pairs
6. $[X, t]$ where $X \in Var(t)$ and $X \neq t \Longrightarrow$ halt with failure

Order-independent unification (Martelli-Montanari)

Input: set $\mathcal{E}$ of pairs $[t, t']$
Output: substitution $[[X1, t1], \ldots, [Xk, tk]]$ unifying pairs in $\mathcal{E}$

Simplify $\mathcal{E}$ non-deterministically until no longer possible

1. $[f(s1, \ldots, sk), f(t1, \ldots, tk)]$ (allowing $k = 0$)
   $\Longrightarrow$ replace by pairs $[s1, t1], \ldots, [sk, tk]$
2. $[f(s1, \ldots, sk), g(t1, \ldots, tm)]$ where $f \neq g$ or $k \neq m$
   $\Longrightarrow$ halt with failure
3. $[X, X] \Longrightarrow$ delete
4. $[t, X]$ where t is not a var $\Longrightarrow$ replace by $[X, t]$
5. $[X, t]$ where $X \notin Var(t)$ and X occurs elsewhere
   $\Longrightarrow$ apply $[X, t]$ to all other pairs
6. $[X, t]$ where $X \in Var(t)$ and $X \neq t \Longrightarrow$ halt with failure

N.B. Prolog omits *occurs check* $X \in Var(t)$ in 5, 6 for speed-up

Instantiate before negating (as failure)

```
% \+p :- (p,!,fail); true.


p(X) :- \+q(X), r(X).
q(a).  q(b).
r(a).  r(c).
--------------
| ?- p(X).                          % contra ?- p(c).
```

Instantiate before negating (as failure)

```
% \+p :- (p,!,fail); true.
```

```
p(X) :- \+q(X), r(X).
q(a).  q(b).
r(a).  r(c).
---------------
| ?- p(X).                          % contra ?- p(c).
```

```
p(X) :- \+q(X).
q(X) :- \+r(X).
r(c).
---------------
| ?- p(X).                          % contra ?- p(a).
```

Generate-and-test

brute force: instantiate all variables before testing constraints

```
genTest(D1...Dn) :- node(X1...Xn,D1...Dn),
                    constraint(X1...Xn).
node(X1...Xn,D1...Dn) :- member(X1,D1),...,
                         member(Xn,Dn).
```

Generate-and-test

brute force: instantiate all variables before testing constraints

```
genTest(D1...Dn) :- node(X1...Xn,D1...Dn),
                    constraint(X1...Xn).
node(X1...Xn,D1...Dn) :- member(X1,D1),...,
                         member(Xn,Dn).
```

For each of the $\prod_{i=1}^{n} s_i$-choices of `X1...Xn` such that

`node(X1...Xn,D1...Dn)`

(with `Di` of size `si`), assume

`constraint(X1...Xn)`

can be checked within a polynomial of `X1...Xn`.

Generate-and-test

brute force: instantiate all variables before testing constraints

```
genTest(D1...Dn) :- node(X1...Xn,D1...Dn),
                    constraint(X1...Xn).
node(X1...Xn,D1...Dn) :- member(X1,D1),...,
                         member(Xn,Dn).
```

For each of the $\prod_{i=1}^{n} s_i$-choices of `X1...Xn` such that

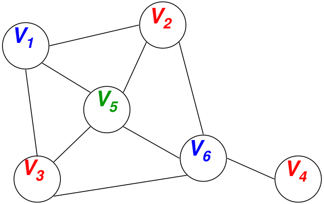
`node(X1...Xn,D1...Dn)`

(with `Di` of size `si`), assume

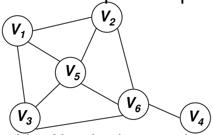
`constraint(X1...Xn)`

can be checked within a polynomial of `X1...Xn`.

Nodes are generated in lexicographic order with*out* regard to constraints.



Canonical Example: Graph Coloring



- Consider *N* nodes in a graph
- Assign values $V_1,..,V_N$ to each of the *N* nodes
- The values are taken in {*R*,*G*,*B*}
- Constraints: If there is an edge between *i* and *j*, then V_i must be different of V_j

Cryptarithmetic

```
  S E N D
+ M O R E
---------
M O N E Y
```

Prolog code

Inferring changes

Horn-SAT by minimal changes to $00 \cdots 0$ (all variables 0/false)

CSAT	definite clause	list encoding
$\overline{u} \vee x \vee \overline{z}$	x :- u, z.	$[x, u, z]$
$\overline{u} \vee \overline{v}$	*false* :- u, v.	$[\mathit{false}, u, v]$

Inferring changes

Horn-SAT by minimal changes to $00 \cdots 0$ (all variables 0/false)

CSAT	definite clause	list encoding
$\overline{u} \vee x \vee \overline{z}$	x :- $u, z.$	$[x, u, z]$
$\overline{u} \vee \overline{v}$	*false* :- $u, v.$	$[\mathit{false}, u, v]$

For each stage i, collect the variables set at stage i to 1/true in A_i

$$
\begin{aligned}
A_0 &:= \emptyset \qquad \text{(all variables false)} \\
A_{i+1} &:= \{x \mid \underbrace{\text{member}([x|T], KB)}_{x \text{ :- } t_1 \ldots t_k \text{ in } KB} \text{ and } \underbrace{\text{all}(T, A_i)}_{\{t_1 \ldots t_k\} \subseteq A_i}\}
\end{aligned}
$$

check: $\mathit{false} \notin A_n$

Inferring changes

Horn-SAT by minimal changes to $00 \cdots 0$ (all variables 0/false)

CSAT	definite clause	list encoding
$\overline{u} \vee x \vee \overline{z}$	x :- u, z.	$[x, u, z]$
$\overline{u} \vee \overline{v}$	$false$:- u, v.	$[false, u, v]$

For each stage i, collect the variables set at stage i to 1/true in A_i

$$
\begin{aligned}
A_0 &:= \emptyset \qquad \text{(all variables false)} \\
A_{i+1} &:= \{x \mid \underbrace{\text{member}([x|T], KB)}_{x \text{ :- } t_1 \ldots t_k \text{ in } KB} \text{ and } \underbrace{\text{all}(T, A_i)}_{\{t_1 \ldots t_k\} \subseteq A_i}\}
\end{aligned}
$$

check: $false \notin A_n$

No minimal set for non-Horn $x \vee y$ (or xor).

Instantiate one variable at a time

allow node to map X_i to ?, raising search space size from

$$\prod_{i=1}^{n} s_i \text{ to } \prod_{i=1}^{n}(s_i + 1) \text{ from adding ? to } D_i$$

PAY-OFF: search tree of depth n and branching factor $\max_i s_i$
with start node instantiating no variable, and
an arc instantiating least uninstantiated variable

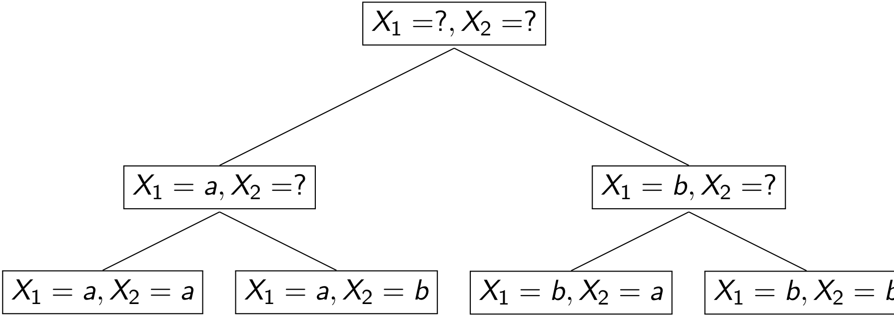
Instantiate one variable at a time

allow node to map X_i to ?, raising search space size from

$$\prod_{i=1}^{n} s_i \quad \text{to} \quad \prod_{i=1}^{n}(s_i + 1) \quad \text{from adding ? to } D_i$$

PAY-OFF: search tree of depth n and branching factor $\max_i s_i$
with start node instantiating no variable, and
an arc instantiating least uninstantiated variable

E.g. $n = 2, \quad D_1 = D_2 = \{a, b\}$



Interleave generation with testing + backtracking

whenever $arc(N0, N1)$,

- $N1$ instantiates one more variable than $N0$, and
- $N1$ satisfies every constraint on instantiated variables

Interleave generation with testing + backtracking

whenever $arc(N0, N1)$,

$N1$ instantiates one more variable than $N0$, and

$N1$ satisfies every constraint on instantiated variables

Reduce domains of un-instantiated variables via constraints

Constraint Graph: node = variable (e.g. 3-Color)

$$arc(X_i, X_j) \iff \text{Con}[X_i, X_j] \neq \emptyset$$

Arc Consistency: for $arc(X_i, X_j)$ and $i < j$,

$$(\forall d \in D(X_i))(\exists d' \in D(X_j))\ d, d' \text{ satisfy Con}[X_i, X_j]$$

removing d from $D(X_i)$ when no such d' exists

Interleave generation with testing + backtracking

whenever $arc(N0, N1)$,

$N1$ instantiates one more variable than $N0$, and

$N1$ satisfies every constraint on instantiated variables

Reduce domains of un-instantiated variables via constraints

Constraint Graph: node = variable (e.g. 3-Color)

$$arc(X_i, X_j) \iff \text{Con}[X_i, X_j] \neq \emptyset$$

Arc Consistency: for $arc(X_i, X_j)$ and $i < j$,

$$(\forall d \in D(X_i))(\exists d' \in D(X_j))\ d, d' \text{ satisfy Con}[X_i, X_j]$$

removing d from $D(X_i)$ when no such d' exists

Optimizing the backtracking search

- MRV: instantiate variable with minimum remaining values (to minimize branching/cases)
- LCV: assign value that is least constraining (for greatest chance of success)