Constraint System *FD*

Domain

The set $\mathcal{Z}$ of integers.

Signature

- Function symbols.
  - Constant 0 and unary successor function s
  - Lists (used for enumeration domains)
  - Binary infix operators + and .. (for interval domains)
- Constraint symbols.
  - Nullary symbols *true*, *false*
  - Binary symbols $=, <, \leq, >, \geq, \neq$, and *in* (for domains)

Interpretation maps arithmetic expressions to integers.

Constraint System *FD* (2)

Allowed atomic constraints

Linear equations and inequations:

$C ::= \mathit{true} \mid \mathit{false} \mid X \texttt{ in } n..m \mid X \texttt{ in } [k_1, \ldots, k_l] \mid X \odot Y \mid X{+}Y{=}Z$

$n,\ m,\ k_1, \ldots, k_l$: integers ($l \geq 0$)
$\odot \in \{=, <, >, \leq, \geq, \neq\}$
$X,\ Y$ and Z: pairwise distinct variables

domain constraint X `in` D: $X \in D$ (D: given finite domain)

- *enumeration domain constraint* X `in` $[k_1, \ldots, k_l]$:
  values of X explicitly enumerated
- *interval domain constraint* X `in` $n..m$:
  values of X in interval $n..m$ (bounds included)

Constraint System *FD* (3)

Constraint theory: Presburger Arithmetic

$0 = s(X) \rightarrow \bot$

$X = X$

$X = Y \rightarrow s(X) = s(Y)$

$s(X) = s(Y) \rightarrow X = Y$

$X = Y \wedge Y = Z \rightarrow X = Z$

$X + 0 = X$

$X + s(Y) = s(X + Y)$

Decidable and complete CT for integer linear arithmetic.

Peano arithmetic adds $*$ and induction principle – incomplete.

Gödel: Any consistent extension of Peano arithmetic is incomplete.

Constraint System *FD* (4)

Constraint theory

Presburger's arithmetic extended by

- $0 \leq s(X)$, $s(X) \leq s(Y) \leftrightarrow X \leq Y$, ...
- X `in` $n..m \leftrightarrow n \leq X \wedge X \leq m$
- X `in` $[k_1, \ldots, k_l] \leftrightarrow X{=}k_1 \vee \ldots \vee X{=}k_l$

Empty domain X `in` [] or X `in` $n..m$ with $n > m$ is unsatisfiable.

FD Interval vs. Enumeration Domain

Different implementations:

- interval domain
  - constraint simplification performed only on interval bounds
- enumeration domain
  - each element in the enumeration considered
  - allow more simplification (tighter domains)
  - only tractable for sufficiently small enumerations

Example:

- X *in* $[1,2,3] \wedge X \neq 2$ yields tighter domain constraint X *in* $[1,3]$
- X *in* $1..3 \wedge X \neq 2$: no propagation since interval bounds do not change

FD Flat Normal Form, Linear Polynomial

- allowed atomic constraints in *flat normal form* and integers are not allowed in the place of variables
- determined variable ($X{=}v$) is expressed by a domain constraint X *in* $[v]$ or X *in* $v..v$.
- linear polynomial equation can be expressed as a conjunction of allowed constraints
  - multiply coefficients of polynomial s.t. they are all integers
  - rewrite multiplications as sums, e.g., $3X$ becomes $X + X + X$
  - flatten the resulting expression, e.g., $X{+}X{+}Y{>}5$ becomes $W{>}F \wedge X{+}V{=}W \wedge X{+}Y{=}V \wedge F$ *in* $[5]$

Constraint Networks

Binary constraint network:

- Variables
- Binary constraints between variables

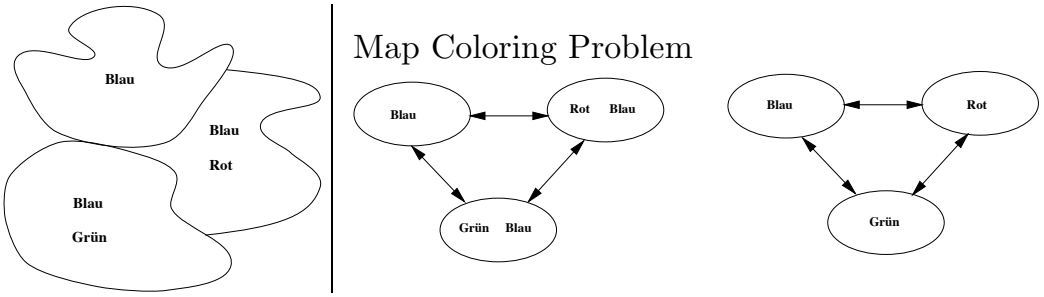
The network can be represented by a *directed constraint graph* where

- Nodes: variables
- Arcs: binary constraints

Solution of a constraint network:

Assignment of values to variables that satisfies all constraints

FD – Example Consistency



FD – Arc Consistency

Atomic constraint $c(X_1, \ldots, X_n)$ *(hyper-)arc consistent* with respect to a conjunction of enumeration domain constraints
X_1 *in* $D_1 \wedge \ldots \wedge X_n$ *in* D_n, if for all $i \in \{1, \ldots, n\}$ and for all values v_i in D_i ($v_i \in \{k_{1i}, \ldots, k_{li}\}$) constraint
$\exists(X_1$ *in* $D_1 \wedge \ldots \wedge X_i{=}v_i \wedge \ldots \wedge X_n$ *in* $D_n \wedge c(X_1, \ldots, X_n))$ is satisfiable, i.e. if for each variable in the constraint and for each value in the domain of the variable, there exist values in the domains of the other variables such that the constraint is satisfied. Every value of every domain takes part in a solution

($X_1, \ldots, X_n$: pairwise distinct variables)

FD – Arc Consistency (2)

Logically, each domain is the projection of the constraints.

$$\exists_{-X_i} (c(X_1, \ldots, X_n) \wedge X_1 \text{ in } D_1 \wedge \cdots \wedge X_n \text{ in } D_n) \rightarrow X_i \text{ in } D_i$$

A conjunction of constraints is arc consistent if each atomic constraint in it is arc consistent.

FD – Arc Consistency (3)

Examples:

- X *in* $[1,2,3] \wedge X \neq 2$ not arc consistent
  But: X *in* $[1,3] \wedge X \neq 2$ arc consistent
- X *in* $[1,2,3] \wedge Y$ *in* $[1,2,3] \wedge X < Y$ not arc consistent
  But: X *in* $[1,2] \wedge Y$ *in* $[2,3] \wedge X < Y$ arc consistent

FD – Arc Consistency (4)

Limitations:

- must rename apart multiple occurrences of same variable
  $X \neq X$ represented by $X \neq Y \wedge X = Y$
- arc consistency does not imply satisfiability (global consistency):
  X *in* $D \wedge Y$ *in* $D \wedge X \neq Y \wedge X = Y$ arc consistent for all $|D| > 1$
- arc consistency sensitive to flattening:
  X *in* $[1,2] \wedge Z$ *in* $[2,3,4] \wedge 2X = Z$ not arc consistent
  But:
  X *in* $[1,2] \wedge Y$ *in* $[1,2] \wedge Z$ *in* $[2,3,4] \wedge X = Y \wedge X + Y = Z$
  arc consistent

FD – Arc Consistency (5)

Local Consistency Methods

Look at a small, fixed number of variables and constraints:

- Use them to propagate new, redundant constraints,
- Simplify the new constraints with old constraints
- Reconsider the changed old constraints until no more change (fixpoint reached)

Local consistency (Propagation) + Search (Labeling, Enumeration) = Complete Solver (Global consistency)

FD – Arc Consistency (6)

local-consistency algorithm = local propagation algorithm

- make atomic constraint arc consistent by deleting values from domain of its variables that do not participate in any solution
- make conjunction of constraints arc consistent by making each atomic constraint arc consistent
- worst case time complexity $O(cd^n)$ (c: number of at most n-ary constraints, d: size of largest domain)

FD – Arc Consistency (7)

Achieving Arc Consistency

An algorithm based on the logical formulation:

$X_1 :: D_1 \wedge \ldots \wedge X_i :: D_i \wedge \ldots \wedge X_n :: D_n$
$\wedge\ c(X_1, \ldots, X_n) \rightarrow X_i :: D_i'$
Then $D_i' \cap D_i$ is the new domain.

Example:

Domain constraints: $X :: [1,2,3], Y :: [1,2,3], Z :: [1,2,3]$
Binary constraints: $X < Y \wedge Y < Z \wedge Z \leq 2$

1. $X < Y$ implies $X :: [1,2]$ and $Y :: [2,3]$
2. $Z \leq 2$ implies $Z :: [1,2]$
3. $Y < Z$ implies $Y :: []$ and $Z :: []$

FD – Global Constraints

Take arbitrary number of variables as arguments

- *alldifferent* $(X_1, \ldots, X_n)$ logically equivalent to $\bigwedge_{1 \leq i < j \leq n} X_i \neq X_j$
- arc consistency does not detect unsatisfiability of $X_1 \neq X_2 \wedge X_1 \neq X_3 \wedge X_2 \neq X_3$ when the variables are constrained to the same domain of two values
- sophisticated algorithms for global constraints achieve more propagation than arc consistency and detect unsatisfiability in this case

FD – Bounds Consistency

For interval domains, a weaker but analogous form of arc consistency proves useful.

Atomic constraint $c(X_1, \ldots, X_n)$ is *bounds (or: box) consistent* with respect to a conjunction of interval domain constraints X_1 *in* $D_1 \wedge \ldots \wedge X_n$ *in* D_n, if for all $i \in \{1, \ldots, n\}$ and for all bounds v_i in D_i ($D_i = n_i..m_i$, $v_i \in \{n_i, m_i\}$) the constraint $\exists(X_1$ *in* $D_1 \wedge \ldots \wedge X_i{=}v_i \wedge \ldots \wedge X_n$ *in* $D_n \wedge c(X_1, \ldots, X_n))$ is satisfiable.

($X_1, \ldots, X_n$: pairwise distinct variables)

FD – Bounds Consistency (2)

A conjunction of constraints is bounds consistent if each atomic constraint in it is bounds consistent.

Analogously to arc consistency enforcement, constraints can be made bounds consistent by tightening their interval domains.

Examples:

- X *in* $1..3 \wedge X{\neq}2$ is bounds consistent
- – X *in* $1..3 \wedge Y$ *in* $1..3 \wedge X{<}Y$ is not bounds consistent
  – X *in* $1..2 \wedge Y$ *in* $2..3 \wedge X{<}Y$ is bounds consistent
- X *in* $D \wedge Y$ *in* $D \wedge X{\neq}Y \wedge X{=}Y$ is bounds consistent for all $|D| > 1$

FD – Local-Propagation Constraint Solver for Interval Domains

- `in`, `le`, `eq`, `ne`, `add`: CHR constraints
- `<`, `>`, `=<`, `>=`,`\=`: built-in arithmetic constraints
- `min`, `max`, `+`, `-`: built-in arithmetic functions
- rules for bounds consistency affect interval in constraints only
- rules based on interval arithmetic

```
inconsistency @ X in A..B <=> A>B | false.
intersection  @ X in A..B, X in C..D <=>
                X in max(A,C)..min(B,D).
```


FD Interval Domains – Inequalities

Sample rules for inequalities:

```
le @ X le Y, X in A..B, Y in C..D <=> B>D |
     X le Y, X in A..D, Y in C..D.
le @ X le Y, X in A..B, Y in C..D <=> C<A |
     X le Y, X in A..B, Y in A..D.

eq @ X eq Y, X in A..B, Y in C..D <=> A\=C |
     X eq Y, X in max(A,C)..B, Y in max(C,A)..D.
eq @ X eq Y, X in A..B, Y in C..D <=> B\=D |
     X eq Y, X in A..min(B,D), Y in C..min(D,B).

ne @ X ne Y, X in A..B, Y in C..D <=> A=C,C=D |
     X ne Y, X in (A+1)..B, Y in C..D.
```

FD Interval Domains – Inequalities (2)

Example:

```
      A in 2..3, B in 1..2, A le B
↦le   B in 1..2, A le B, A in 2..2
↦le   A le B, A in 2..2, B in 2..2.
```


FD Interval Domains – Add

$X+Y=Z$ represented in relational form as `add(X,Y,Z)`:

```
add @ add(X,Y,Z), X in A..B, Y in C..D, Z in E..F <=>
      not (A>=E-D,B=<F-C,C>=E-B,D=<F-A,E>=A+C,F=<B+D) |
      add(X,Y,Z),
      X in max(A,E-D)..min(B,F-C),
      Y in max(C,E-B)..min(D,F-A),
      Z in max(E,A+C)..min(F,B+D).
```

Guard negates condition that describes arc consistency of `add`.

FD Interval Domains – Add (2)

Examples

- `A in 1..3, B in 2..4, C in 0..4, add(A,B,C)` $\mapsto_{\texttt{add}}$ `add(A,B,C), A in 1..2, B in 2..3, C in 3..4`
- `X in 1..1000, Y in 1..1000, Z in 1..1000, X eq Z, add(X,Y,Z)` $\mapsto^*$ `...`

FD Interval Domains – Termination

- rules `inconsistency` and `intersection` remove one interval constraint each
- assume that remaining rules deal with non-empty intervals only
- in each rule, at least one interval in the body is strictly smaller than the corresponding interval in the head, while the other intervals remain unaffected

FD Interval Domains – Confluence

The solver is confluent provided the intervals are given.

FD Interval Domains – Complexity (1)

- arithmetic built-in constraints take constant time to compute
- find domain of a variable in constant time using indexing
- each rule application or try takes constant time
- – $w = m - n + 1$: maximum *width (size)* of constraint X *in* $n..m$
  – v: number of different variables, c: number of constraints

  w, c and v do not increase during derivation

FD Interval Domains – Complexity (1)

- worst number of rule applications is $O(vw)$, not dependent on number of constraints (v can not exceed $O(c)$)
- there are at most $O(c)$ rule tries
- worst case time complexity is $O(cvw)$

Example:

```
  X in 1..1000, Y in 1..1000, X le Y, X eq Y
↦ X in 1..999, Y in 2..1000, X le Y, X eq Y
↦ ...
```

FD – Local-Propagation Constraint Solver for Enumeration Domains

```
inconsistency @ X in [] <=> false.
intersection  @ X in L1, X in L2 <=>
                intersection(L1,L2,L3), X in L3.

le @ X le Y, X in L1, Y in L2 <=> max(L1) > max(L2) |
     filter_max(L1,L2,L3),
     X le Y, X in L3, Y in L2.
...
```

(`filter_max` removes all values from a list that are larger than all values in another list)

FD Example – Enumeration Domains

- `X le Y, X in [4,6,7], Y in [3,7]` $\mapsto^*$
  `X le Y, X in [4,6,7], Y in [7]`
- `X le Y, X in [2,3,4,5], Y in [1,2,3]` $\mapsto^*$
  `X le Y, X in [2,3], Y in [2,3]`
- `X le Y, X in [2,3,4], Y in [0,1]` $\mapsto^*$
  `false`

FD Enumeration Domains – `eq` and `add`

```
eq @ X eq Y, X in L1, Y in L2 <=> diff(L1,L2) |
     intersection(L1,L2,L3),
     X eq Y, X in L3, Y in L3.

add @ add(X,Y,Z), X in L1, Y in L2, Z in L3 ==>
    all_substractions(L3,L2,L4),
    all_substractions(L3,L1,L5),
    all_additions(L1,L2,L6),
    not (L1 se L4, L2 se L5, L3 se L6),
    |
    X in L4, Y in L5, Z in L6.
```

(`se`: set equality)

FD Enumeration Domains – Termination, Confluence

Termination and confluence similar to interval domain solver.

FD Enumeration Domains – Complexity

- replace interval width w by maximum size of an enumeration domain d
- built-in constraints take up to $O(d^2)$ for operations on arbitrarily large enumeration domains
- worst time complexity is $O(cvd^3)$

FD – Search

Use search to achieve satisfaction-completeness.

```
enum([]) <=> true.
enum([X|Xs]) <=> indomain(X), enum(Xs).

indomain(X), X in [V|L] <=> L=[_|_] |
            (X in [V] ; X in L, indomain(X)).
```

For interval domains, search is usually done by splitting intervals in two halves until bounds of interval are the same

```
indomain(X), X in A..B <=> A<B |
             C is (A+B)//2,
            (X in A..C ; X in (C+1)..B), indomain(X).
```

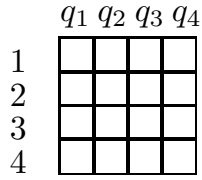
The guards ensure termination.

FD – Implementations

- linear polynomial equations are allowed as constraints
- hybrid, compact form of domains is used in implementations
- domain is list of intervals, so an interval can have holes since it can be split if a value inside the interval needs to be removed
- for small enumeration domains, bit vectors can be used

FD Application – n-Queens Problem

Place n queens $q_1, \ldots, q_n$ on an $n \times n$ chess board, such that they do not attack each other.



$$q_1, \ldots, q_n \in \{1, \ldots, n\}$$

$$\forall\ i \neq j.\ q_i \neq q_j \wedge |q_i - q_j| \neq |i - j|$$

- no two queens on same row, column or diagonal
  - each row and each column with exactly one queen
  - each diagonal at most one queen
- q_i: row position of the queen in the i-th column

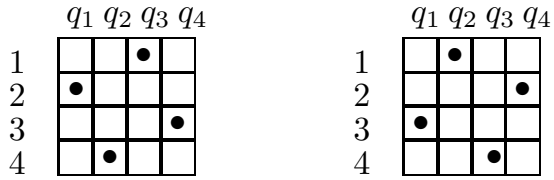
FD Application – n-Queens Problem (2)

```
constraints solve/2, queens/1, safe/3, no_attack/3.

solve(Qs,N)      <=> make_domains(Qs,N), queens(Qs), enum(Qs).

queens([])       <=> true.
queens([Q|Qs])   <=> safe(Q,Qs,1), queens(Qs).

safe(X,[],N)     <=> true.
safe(X,[Y|Qs],N) <=> no_attack(X,Y,N), NP1 is N+1, safe(X,Qs,NP

no_attack(X,Y,N) <=> X ne Y, VN in N..N,
                     add(X,VN,XPN), XPN ne Y,
                     add(Y,VN,YPN), YPN ne X.
```

Application – n-Queens Problem (3)

`solve(4,[Q1,Q2,Q3,Q4])`

- `make_domains` produces
  `Q1 in [1,2,3,4], Q2 in [1,2,3,4]`
  `Q3 in [1,2,3,4], Q4 in [1,2,3,4]`
- `safe` adds `noattack` producing `ne` constraints
- `label` called for labeling
- `[Q1,Q2,Q3,Q4] = [2,4,1,3]`, `[Q1,Q2,Q3,Q4] = [3,1,4,2]`



Application – Send More Money (1)

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

Replace distinct letters by distinct digits,
numbers have no leading zeros.

Application – Send More Money (2)

```
:- use_module(library(clpfd)).

send([S,E,N,D,M,O,R,Y]) :-
     gen_domains([S,E,N,D,M,O,R,Y],0..9),
     S #\= 0, M #\= 0,
     all_distinct([S,E,N,D,M,O,R,Y]),
                 1000*S + 100*E + 10*N + D
     +           1000*M + 100*O + 10*R + E
     #= 10000*M + 1000*O + 100*N + 10*E + Y,
     labeling([],[S,E,N,D,M,O,R,Y]).
```