NOPT042 Constraint programming: Tutorial 10 - Modeling with sets

What was in Lecture 7?

Symmetry breaking

- example: tournament scheduling (match symmetry, round symmetry)

Global constraints

- faster GAC/filtering algorithm, arbitrary arity, exploit semantics
- **all_different**: domain filtering based on matching in bipartite graphs (remove edges that do not belong to any maximum matching)
- **global_cardinality**: similar, based on network flows
- **lex**: filtering using two pointers)
- **regular:** filtering using "state DAG", rostering (scheduling with sequence constraints)
- **grammar:** sequence generated by CFG? filtering using CYK algorithm
- **slide:** generalizes lex, regular

Scheduling

- how to represent (resources, disjunctive, precedence), disjunction bad (almost no filtering)
- edge_finding, not_first filtering rules

In [1]:
```
%load_ext ipicat
```

```
Picat version 3.9
```

Modelling with sets

In Picat, the cp solver doesn't work natively with sets and set constraints (unlike e.g. MiniZinc). Instead, we can model a set as an array (or a list) representing its characteristic vector. For a collection of sets, we can use a matrix or a list of lists.

- A subset $S \subseteq \{1, \ldots, n\}$: `S = new_array(N), S :: 0..1`
- Fixed cardinality subset: `exactly(K, S, 1)`
- Bounded cardinality subset: `at_most(K, S, 1)`, `at_least(K, S, 1)` (or we could use `sum`)

Set operations can be computed bitwise, e.g.

```
SintersectT = [X : I in 1..N, X #= S[I] * T[I]]
```

Alternatively, we could use a strictly increasing list of elements:

```
S = new_list(Length),
S :: 1..N,
increasing_strict(S).
```

A partition of $\{1, \ldots, n\}$ with k classes can be modelled as a function $\{1, \ldots, n\} \rightarrow \{1, \ldots, k\}$:

```
Partition = new_array(N),
Partition :: 1..K
```

Do not forget about symmetry breaking, e.g. `Partition[1] #= 1` or

```
foreach(I in 1..K)
    Partition[I] #<= I
end.
```

Similarly for a collection of k pairwise disjoint subsets: using 0 to denote that an element is not covered by any subset.

Exercise: Finite projective plane

> A projective plane geometry is a nonempty set X (whose elements are called "points"), along with a nonempty collection L of subsets of X (whose elements are called "lines"), such that:
>
> - For every two distinct points, there is exactly one line that contains both points.
> - The intersection of any two distinct lines contains exactly one point.
> - There exists a set of four points, no three of which belong to the same line.

(from Wikipedia)

A projective plane of **order** `N` has $M = N^2 + N + 1$ points and the same number of lines, each line must have $K = N + 1$ points and each point must lie on K lines. A famous example is the Fano plane where $N = 2$, $M = 7$, and $K = 3$.

If the order N is a power of a prime power, it is easy to construct a projective plane of order N. It is conjectured otherwise, no projective plane exists. For

$N = 10$ this was famously proved by a computer-assisted proof (that finished in 1989). The case $N = 12$ remains open.

In [2]:
```
!picat projective/projective 2
```

```
*** error(existence_error([p,r,o,j,e,c,t,i,v,e,/,p,r,o,j,e,c,t,i,v,e]),pica
t)
```

Exercise: Ramsey's partition

Partition the integers 1 to n into three parts, such that for no part are there three different numbers with two adding to the third. For which n is it possible?

In [3]:
```
!picat ramsey/ramsey 23
```

```
*** error(existence_error([r,a,m,s,e,y,/,r,a,m,s,e,y]),picat)
```

Exercise: Kirkman's schoolgirl problem

> Fifteen young ladies in a school walk out three abreast for seven days in succession: it is required to arrange them daily so that no two shall walk twice abreast.

See Wikipedia.

Exercise: Word Design for DNA Computing on Surfaces

Problem 033 from CSPLib: Find as large a set S of strings (words) of length 8 over the alphabet $W = \{A, C, G, T\}$ with the following properties:

- Each word in S has 4 symbols from $\{C, G\}$.
- Each pair of distinct words in S differ in at least 4 positions.
- Each pair of words x and y in S (where x and y may be identical) are such that x^R and y^C differ in at least 4 positions.

Here, $(x_1, \ldots, x_8)^R = (x_8, \ldots, x_1)$ is the reverse of $(x_1, \ldots, x_8)$ and $(y_1, \ldots, y_8)^C$ is the Watson-Crick complement of $(y_1, \ldots, y_8)$, i.e., the word where each A is replaced by a T and vice versa and each C is replaced by a G and vice versa.