Now we explain how CLP is implemented in Prolog. The pred. symbols =, true, fail are pre-defined in Prolog. But we have to tell Prolog which constraint theories we want to use. Prolog has a module-system of pre-defined libraries.

```prolog
use_module : predicate to import all predicates defined in a module
```

To import the library with the predicate symbols from the constraint theory $\Sigma_{FD}$, one can add the following directive to the Prolog-program:

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

Afterwards, $\Sigma_{FD}$ and $\Delta_{FD}$ are available and one can use a constraint solver for $\Sigma_{FD}$. However, entailment is undecidable for $\Sigma_{FD}$. Thus, one can't decide automatically whether

$$\Sigma_{FD} \lor \{ \forall X \ X=X, \text{true} \} \models \exists X \phi. \quad \text{(4)}$$

Even for theories where this is decidable, it could
be very inefficient. Therefore, Rolay only approximates (8). Instead of checking satisfiability of CO, it only checks path consistency of CO.

**Def 6.2.1 (Path Consistency)**

Let $CO = \psi_1 \land \ldots \land \psi_m$ be a conjunction of constraints with $\psi_i \in \text{AT}(\Sigma_{FD}, \Delta_{FD}, \upsilon)$. Let $X_1, \ldots, X_n$ be the variables of CO. Let $D_1, \ldots, D_n$ be subsets of $\mathbb{Z}$. We say that $D_1, \ldots, D_n$ are admissible domains for $X_1, \ldots, X_n$ w.r.t. CO iff for all $\psi_i$ with $1 \leq i \leq m$ and all $X_j$ with $1 \leq j \leq n$ we have: For all $a \in D_j$, there exist $a_i \in D_i$, $a_j \in D_j$, $a_{j+1} \in D_{j+1}$, $\ldots$, $a_n \in D_n$, such that $CT_{FD} \models \psi_i[X_1/a_1, \ldots, X_n/a_n].$

If there exist admissible domains $D_1, \ldots, D_n$ which are all non-empty, then $CO$ is path-consistent.

*Main difference between path-consistency and satisfiability is that for path-consistency one regards all constraints separately.*

**Ex. 6.2.2**
Typically, one first sets all \( D_j = \mathbb{Z} \). Afterwards one iterates through all constraints and all variables and reduces the domains \( D_j \). This is repeated until the domains \( D_j \) do not change anymore.

\[
X_1 \# > 5 \land X_1 \# < X_2 \land X_2 \# < 9
\]

We start with \( D_1 = \mathbb{Z} \) and \( D_2 = \mathbb{Z} \).

Now we regard the constraint \( X_1 \# > 5 \) and variable \( X_1 \): We have to remove all elements from \( D_1 \) where the constraint does not hold.

\[
D_1 = \{ 6, 7, \ldots \} \quad \text{and} \quad D_2 = \mathbb{Z}
\]

Now we regard constraint \( X_1 \# < X_2 \) and var. \( X_1 \):

Is it true that for every \( a_1 \in D_1 \) there is an \( a_2 \in D_2 \) with \( a_1 \# \leq a_2 \)? Yes.

Now constr. \( X_1 \# \leq X_2 \) and var. \( X_2 \):

Is it true that for every \( a_2 \in D_2 \) there is an \( a_1 \in D_1 \) with \( a_1 \# \leq a_2 \)? No.

\[
D_1 = \{ 6, 7, \ldots \} \quad \text{and} \quad D_2 = \{ 7, 8, \ldots \}
\]

Now we regard \( X_2 \# < 9 \) and var. \( X_2 \):

\[
D_1 = \{ 6, 7, \ldots \} \quad \text{and} \quad D_2 = \{ 7, 8 \}
\]

Now we again regard \( X_1 \# > 5 \). \( D_1, D_2 \) do not change.
Then we regard $X_1 \not\leq X_2$ and var. $X_1$:

$D_1 = \{6, 7\}$  \quad $D_2 = \{7, 8\}$

Afterwards, $D_1$ and $D_2$ do not change any more $\Rightarrow$ admissible domains.
Since $D_1 \neq \emptyset$ and $D_2 \neq \emptyset$, our conjunction of constraints is path-consistent (i.e., "true" in Prolog).

$X_1$ in $6..7$ $\leftarrow$ pre-defined predicate "in"

which abbreviates

$X_1 \not\geq 6, \quad X_1 \not\leq 7$.

The predicate "in" can also be used by the programmer.

?- $X_1$ in $6 .. sup, X_1 \not\leq X_2, X_2$ in inf .. 8.

\[ \infty \quad \infty \]

There exists a pre-defined pred. label in clpfd which forces Prolog to enumerate solutions.

?- $X_1 \not\geq 5, X_1 \not\leq X_2, X_2 \not\leq 9, \text{label}([X_1, X_2])$.

$X_1 = 6, \quad X_2 = 7$.
\[ x_1 = 6, \ x_2 = 8; \]
\[ x_1 = 7, \ x_2 = 8 \]

In this way, one can print out answer substitutions instead of answer constraints.

However, this only works if the admissible domains are finite.

?- \( x_1 \#> x_2 \), \( x_1 \#=< x_2 \), label([x_1, x_2]).

error

Ex. 6.2.3 There are examples where \( C_0 \) are

partially-consistent, but unsatisfiable.

?- \( x_1 \#> x_2 \), \( x_1 \#=< x_2 \).

Prolog starts with \( D_1 = \mathbb{Z} \), \( D_2 = \mathbb{Z} \).

• Now we start with \( x_1 \#> x_2 \).
  For every \( a_n \in D_1 \) there is an \( a_2 \in D_2 \) with \( a_n \#> a_2 \).
  \[ \text{--} \ a_2 \in D_2 \text{--} \ a_n \in D_1 \text{ with } a_n \#> a_2. \]
  • Then we regard \( x_1 \#=< x_2 \).
    Again, \( D_1 \) and \( D_2 \) remain unchanged.

Typical programs that are easy to compute with CLP:

Ex. 6.2.4. \( n \)-queens problem:

\[
\begin{matrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{matrix}
\]
Exercise 6.2.4. $n$-queens problem

Chess board of size $n \times n$

Goal: place $n$ queens on the board such that the queens cannot take each other.

There must not be more than one queen in any row, any column, and any diagonal.

We represent the positions of the queens by a list $[x_1, \ldots, x_n]$ which means that the queen in column $i$ is in row $x_i$.

$\text{queens}(4, L)$

$L = [2, 4, 1, 3]$ 

Here, $L$ should be the possible positions of queens on a $4 \times 4$ chess board.

- $L$ must be a permutation of the numbers from $1$ to $N$.
- pre-def. predicate "ins":
  
  $[x_1, \ldots, x_n]$ ins $1 \ldots N$ iff
  
  $x_j$ in $1 \ldots N$ for all $1 \leq j \leq k$.

- pre-def. predicate all different

- safe $(L)$ is needed to ensure that no two queens are on the same diagonal.
safe(L) ensures that no queen in L can take a queen on the right of her.

- safe_between (X, L, M) is true iff the queen in row X cannot take any queen in the list L provided that M is the number of columns between the queen X and the first queen in L.

In the example we would first have the query:

safe_between (2, [4, 1, 33, 1])
safe_between (2, [4, 1, 33, 2])
safe_between (2, [33, 3])