# Prolog 6

## Wie könnte man mit Prolog ein Sudoku lösen?

- alle Anforderungen deklarieren (in Prädikaten modellieren)
- mit CLP-FD
- `sudoku/1` wird mit einer Liste (Puzzle) von Listen aufgerufen. Diese Liste enthält je eine Liste mit 9 Elementen (Feldern) für jede der 9 vorgegebenen Zeilen

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

% Anforderungen deklarieren
sudoku(Rows) :-

    % Alle Listen in Rows in Liste Vs zusammenfassen (append)
    % Wertebereich 1 bis 9 festlegen (finite Domäne, Vs ins 1...9)
    append(Rows, Vs), Vs ins 1..9,

    maplist(all_distinct, Rows), % jede Ziffer exakt einmal pro Zeile

    transpose(Rows, Columns), % transponiert Rows in neue Liste Columns

    maplist(all_distinct, Columns), % jede Ziffer exakt einmal pro Spalte

    % die 9 Zeilen in Rows den Variablen A bis I zuweisen
    Rows = [A, B, C, D, E, F, G, H, I],

    % überprüft ob für Zeilen Z1, Z2 und Z3 Block-Anforderungen
    % erfüllt sind
    blocks(A, B, C), blocks(D, E, F), blocks(G, H, I),
    maplist(label, Rows).

% Vorgehen rekursiv
blocks([], [], []). % Einfacher Fall: die drei Zeilen sind leer

% Allgemeiner Fall: Ersten 3 Elemente auf Verschiedenheit der 3 Zeilen
% prüfen. Danach rekursiv Rest prüfen (Listenschwänze Bs1, Bs2, Bs3).
blocks([A, B, C|Bs1], [D, E, F|Bs2], [G, H, I|Bs3]) :-
    all_distinct([A, B, C, D, E, F, G, H, I]),
    blocks(Bs1, Bs2, Bs3).
```

```
% Programm aufrufen, freie Felder als anonyme Variablen _
?- Puzzle = [
```

```
    [5, 3, _, _, 7, _, _, _, _],
    [6, _, _, 1, 9, 5, _, _, _],
    [_, 9, 8, _, _, _, _, 6, _],
    [8, _, _, _, 6, _, _, _, 3],
    [4, _, _, 8, _, 3, _, _, 1],
    [7, _, _, _, 2, _, _, _, 6],
    [_, 6, _, _, _, _, 2, 8, _],
    [_, _, _, 4, 1, 9, _, _, _],
    [_, _, _, _, 8, _, _, 7, 9]
    ],

    % um die Lösung schön zu formatieren
    Puzzle = [A, B, C, D, E, F, G, H, I],
    sudoku([A, B, C, D, E, F, G, H, I]).
```

## Was macht das Prädikat `label/1` im Sudoku-Beispiel?

- Weist den Variablen von `Rows` Werte zu,
- so dass die angegebenen Constraints alle erfüllt sind.

> Damit beginnt im Sudoku-Beispiel die eigentliche Lösungssuche an!

## Unterstützt SWI-Prolog HTTP?

- Ja, mit der `http/http_client` - Library
- Damit stehen u.a. Prädikate für http-Anfragen (GET, POST)
  - `http_get/3`
  - `http_post/4`

```
?- use_module(library(http/http_client)).
?- use_module(library(http/http_json)).

% ...

% GET
% library(http/http_client) compiled into http_client 0.11 sec,
% 1,744 clauses
?- http_get('http://wherever.ch/pcp.txt', Reply, []).

Reply = 'Hallo zusammen! Dies ist eine Test-Datei fuer das PCP-
        Modul, abgelegt unter http://wherever.ch/pcp.txt. - Prolog rockt!
        :-) MfG, Ruedi Arnold' .
```

```
% POST
?- http_post(
    'http://localhost:16316/test',
    json(['say hi to http post']),
    SolutionResponse,[]
).

SolutionResponse = 'PCPProblemProvider server up and running at
                    http://localhost:16316/, reached via HTTP POST -
POSTed data: ["say hi to http post" ]'.
```

## Unterstützt SWI-Prolog JSON?

- Ja, mit der Library `http/json` → `use_module(library(http/json))`
- JSON-Objekte werden in Prolog in Terme der Form `json([…])` abgepackt, wobei in dieser Liste jeweils Paare von Name=Value kommen.
- z.B. `json([institution=hslu, dept=t&a])`
- `json()` ist also praktisch ein "Markier"-Prädikat (marker predicate) und bietet sonst keine Funktionalität

## Wie ruft man Prädikate generisch auf?

- Mit dem eingebauten Prädikat `call/2`
- `?- call(is_bigger, horse, dog).`
- `call/2` verwendet VarArgs, also können beliebig viele Argumente mitgegeben werden.

## Wie kann ein Prädikat auf eine Liste angewandt und das Resultat in einer andern Liste abgelegt werden?

- mit Hilfe von `maplist/3`

```
?- maplist(sqrt, [4, 9, 16], X).
X = [2.0, 3.0, 4.0].
```

- Damit kann "funktional" programmiert werden, indem eine Funktion (hier: `sqrt/1` - Prädikat) auf alle Elemente einer Liste angewandt wird