Prof. Dr. Jürgen Giesl Jera Hensel

Exercise 1 (Procedural Semantics): (4 points)

Consider the following constraint logic program $\mathcal{P}$ for the computation of the *powers* function introduced in Exercise Sheet 7 together with the constraint theory CT_{FD} from Example 6.1.4 of the lecture (i.e., CT_{FD} consists of all true formulas on integers):

```
p(X,1,X) :- X #>= 0.
p(X,Y,Z) :- X #>= 0, Y #> 1, Ydec #= Y - 1, p(X,Ydec,Zpre), Z #= Zpre^X.
```

Please write down a successful computation of the query $G = \{\neg \mathtt{p}(2, \mathtt{A}, 4)\}$ w.r.t. $\mathcal{P}$ and CT_{FD}, i.e., a sequence of configurations of the form $(\mathtt{p}(2, \mathtt{A}, 4), \mathtt{true}) \vdash_{\mathcal{P}} \ldots \vdash_{\mathcal{P}} (\Box, CO)$. As in Example 6.1.13 from the lecture, you may leave out the negations in the queries and simplify your constraints after every computation step. Moreover, please give the computed answer $P[\![\mathcal{P}, CT_{FD}, G]\!]$.

Solution:

$$
\begin{aligned}
&(\mathtt{p}(2, \mathtt{A}, 4), \mathtt{true})\\
\vdash_{\mathcal{P}} &(\mathtt{X} \#>= 0, \mathtt{Y} \#> 1, \mathtt{Ydec} \#= \mathtt{Y} - 1, \mathtt{p}(\mathtt{X}, \mathtt{Ydec}, \mathtt{Zpre}), \mathtt{Z} \#= \mathtt{Zpre\hat{}X}, \quad \underbrace{\mathtt{true} \wedge \overline{\mathtt{p}(2, \mathtt{A}, 4) = \mathtt{p}(\mathtt{X}, \mathtt{Y}, \mathtt{Z})}}_{\mathtt{X}=2 \wedge \mathtt{A}=\mathtt{Y} \wedge \mathtt{Z}=4})\\
\vdash_{\mathcal{P}} &(\mathtt{Y} \#> 1, \mathtt{Ydec} \#= \mathtt{Y} - 1, \mathtt{p}(\mathtt{X}, \mathtt{Ydec}, \mathtt{Zpre}), \mathtt{Z} \#= \mathtt{Zpre\hat{}X}, \quad \mathtt{X} \#>= 0 \wedge \mathtt{X} = 2 \wedge \mathtt{A} = \mathtt{Y} \wedge \mathtt{Z} = 4)\\
\vdash_{\mathcal{P}} &(\mathtt{Ydec} \#= \mathtt{Y} - 1, \mathtt{p}(\mathtt{X}, \mathtt{Ydec}, \mathtt{Zpre}), \mathtt{Z} \#= \mathtt{Zpre\hat{}X}, \quad \mathtt{Y} \#> 1 \wedge \mathtt{X} \#>= 0 \wedge \mathtt{X} = 2 \wedge \mathtt{A} = \mathtt{Y} \wedge \mathtt{Z} = 4)\\
\vdash_{\mathcal{P}} &(\mathtt{p}(\mathtt{X}, \mathtt{Ydec}, \mathtt{Zpre}), \mathtt{Z} \#= \mathtt{Zpre\hat{}X}, \quad \mathtt{Ydec} \#= \mathtt{Y} - 1 \wedge \mathtt{Y} \#> 1 \wedge \mathtt{X} \#>= 0 \wedge \mathtt{X} = 2 \wedge \mathtt{A} = \mathtt{Y} \wedge \mathtt{Z} = 4)\\
\vdash_{\mathcal{P}} &(\mathtt{X} \#>= 0, \mathtt{Z} \#= \mathtt{Zpre\hat{}X},\\
&\underbrace{\overline{\mathtt{p}(\mathtt{X}, \mathtt{Ydec}, \mathtt{Zpre}) = \mathtt{p}(\mathtt{X1}, 1, \mathtt{X1})} \wedge \mathtt{Ydec} \#= \mathtt{Y} - 1 \wedge \mathtt{Y} \#> 1 \wedge \mathtt{X} \#>= 0 \wedge \mathtt{X} = 2 \wedge \mathtt{A} = \mathtt{Y} \wedge \mathtt{Z} = 4}_{\mathtt{Ydec}=1 \wedge \mathtt{Zpre}=2 \wedge \mathtt{X1}=2 \wedge \mathtt{X}=2 \wedge \mathtt{Y}=2 \wedge \mathtt{A}=2 \wedge \mathtt{Z}=4})\\
\vdash_{\mathcal{P}} &(\mathtt{Z} \#= \mathtt{Zpre\hat{}X}, \quad \mathtt{X} \#>= 0 \wedge \mathtt{Ydec} = 1 \wedge \mathtt{Zpre} = 2 \wedge \mathtt{X1} = 2 \wedge \mathtt{X} = 2 \wedge \mathtt{Y} = 2 \wedge \mathtt{A} = 2 \wedge \mathtt{Z} = 4)\\
\vdash_{\mathcal{P}} &(\Box, \quad \underbrace{\mathtt{Z} \#= \mathtt{Zpre\hat{}X} \wedge \mathtt{X} \#>= 0 \wedge \mathtt{Ydec} = 1 \wedge \mathtt{Zpre} = 2 \wedge \mathtt{X1} = 2 \wedge \mathtt{X} = 2 \wedge \mathtt{Y} = 2 \wedge \mathtt{A} = 2 \wedge \mathtt{Z} = 4}_{\mathtt{Ydec}=1 \wedge \mathtt{Zpre}=2 \wedge \mathtt{X1}=2 \wedge \mathtt{X}=2 \wedge \mathtt{Y}=2 \wedge \mathtt{A}=2 \wedge \mathtt{Z}=4})
\end{aligned}
$$

$P[\![\mathcal{P}, CT, G]\!] = \{\mathtt{p}(2, 2, 4)\}$

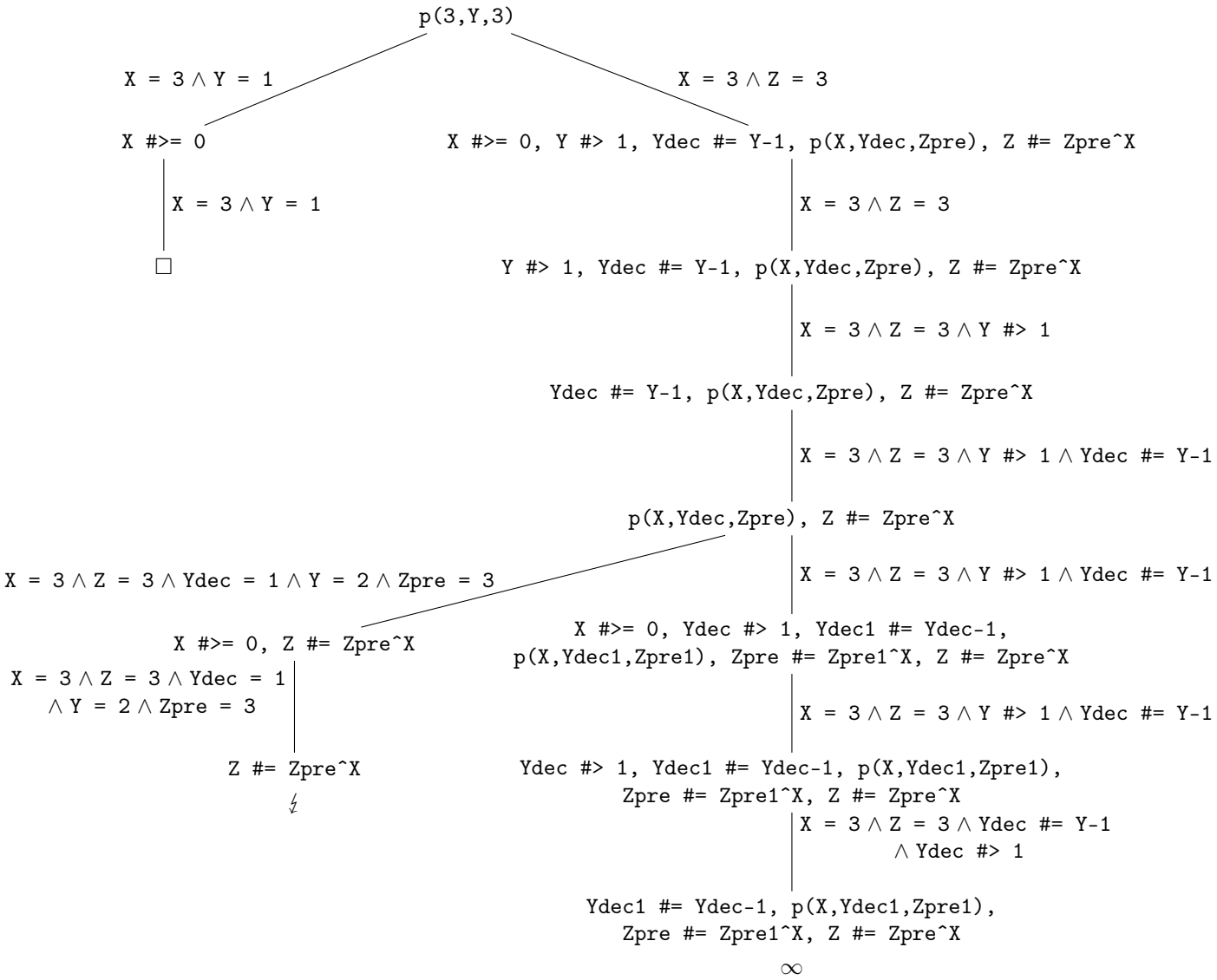
Exercise 2 (SLD Trees With CLP): (5 points)

Again, consider the program $\mathcal{P}$ from Exercise 1 with constraints from CT_{FD}:

```
p(X,1,X) :- X #>= 0.
p(X,Y,Z) :- X #>= 0, Y #> 1, Ydec #= Y - 1, p(X,Ydec,Zpre), Z #= Zpre^X.
```

Please give a graphical representation of the full SLD tree for the query `?- p(3,Y,3)` up to depth seven. Here, the query is at depth zero. You can restrict your representation to those parts of the tree where the conjunction of constraints is satisfiable (i.e., do not show the parts where the edges contain unsatisfiable constraints). Use ∞ to indicate infinite paths and $\lightning$ to mark nodes that cannot be evaluated further, e.g., due to unsatisfiable constraints. Also give all answer substitutions.

Solution:

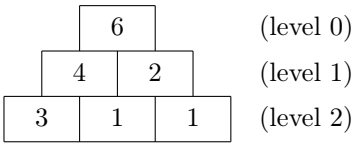


The only answer substitution is $\{\texttt{Y} = 1\}$.

Exercise 3 (Programming in CLP): (7 points)

Important: In addition to handing in the solution on paper, please also mail your solutions for this exercise to **lp17-hiwis@i2.informatik.rwth-aachen.de**. Make sure to send this email from an address that ends with "`rwth-aachen.de`" (otherwise we will not receive your email). Indicate your immatriculation numbers in the subject of the mail and inside the Prolog file.

A number pyramid is a pyramid of $k \geq 0$ levels, where level 0 has one element, level 1 has two elements, etc., with the property that for $0 \leq i < k-1$ and $0 \leq j \leq i$, the element at position j of level i is the sum of the element at position j of level $i+1$ and the element at position $j+1$ of level $i+1$. All numbers in the pyramid are natural numbers between 0 and a certain maximal number n. An example for a number pyramid is given below.



Your task is to implement a CLP predicate `pyramid/2` in the constraint theory CT_{FD}. Let a number pyramid be given by a list `Levels` of lists of numbers and let `N` be the maximal number to be used in the pyramid. The query `?- pyramid(Levels,N)` should be provable iff the variables used in `Levels` can be instantiated to a valid number pyramid in our representation. In addition, if `pyramid(Levels,N)` is provable, then the variable entries in the individual level lists of the pyramid should be instantiated by the answer substitution.
For example, the query `?- pyramid([[6],[4,X],[Y,1,Z]],6)` should have the result `X = 2, Y = 3, Z = 1`. The query `?- pyramid([[4,2],[Y,1,Z]],6)` should fail.

Solution:

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

pyramid([],_).
pyramid([[A]],N) :- !, A in 0..N.
pyramid([[A],L2|Levels],N) :- checkLevels([[A],L2|Levels],N).

% For each two consecutive levels, check if the sum property holds using the
% predicate sums.
checkLevels([L1,L2],N) :- !, sums(L1,L2,N).
checkLevels([L1,L2|Levels],N) :- sums(L1,L2,N), checkLevels([L2|Levels],N).

% For two lists representing the levels i and i+1, step through all positions
% and check if each number A of level i is the sum of the numbers B1 and B2
% that are directly below A in level i+1.
sums([],[_],_).
sums([A|L1],[B1,B2|L2],N) :- A in 0..N, B1 in 0..N, B2 in 0..N, A #= B1+B2,
                             sums(L1,[B2|L2],N).
```