A case study in verification of Prolog programs

C. Renso, S. Ruggieri

February 9, 1996

1 The case study

1.1 Problem

Given two positive integers R and C output in the screen a rectangle with width C and heigth R containing the numbers from 1 to $R \cdot C$ in a spiraling fashion. For example, for $R = 4$ and $C = 3$ the output is:

1	2	3
10	11	4
9	12	5
8	7	6

The spiral of R rows and C columns starting from $Start \in GaeN$ (denoted with $sp(R, C, Start)$ is formally defined as the lists of lists of GaeN

$$
\begin{array}{llr}
[[Start, & \ldots & Start + C - 1], \\
\;[Start + 2 \cdot C + 2 \cdot R - 5, & & Start + C], \\
\;[\;\;\vdots & & \vdots\;\;], \\
\;[Start + 2 \cdot C + R - 2, & & Start + C + R - 3], \\
\;[Start + 2 \cdot C + R - 3, & \ldots & Start + C + R - 2]]
\end{array}
$$

1.2 A solution

In the following we present a solution to the problem, together with its specification (pre- and post-conditions and level mappings) and a moding for each predicate.

We will use some abbreviations:

- $GaeN$ is the set of numeral ground arithmetic expressions, i.e. `0, 1, -1, 2, -2, 3, -3, ...`;
- with a small abuse of notation we will confuse a numeral gae N with the the natural number denoted by N (and formally denoted with $value(N)$);
- $list(\alpha)$ is the set of lists whose elements are in α; we define $listGae = list(Gae)$, and $listGaeN = list(GaeN)$;
- $\|xs\|$ is the length of the list xs;
- $*$ is the concatenion operator between two lists;
- for a list $xs = [x_1, \ldots, x_n]$, we denote with xs_i the i^{th} element of xs, i.e. x_i;

% spiral(+, + , -)
% $pre = \{spiral(R, C, S) |\ R, C \in GaeN \ \wedge \ R, C > 0\}$
% $post = \{spiral(R, C, S) |\ R, C \in GaeN \ \wedge \ R, C > 0 \ \wedge \ S = sp(R, C, 1)\}$
% $|spiral(R, C, S)| = R \cdot C$

```
spiral( R, C, S ) :-
        calc( 1, R, C, S).
```

% append(+, +, -)
% $Pre = \{append(Xs, Ys, Zs) |\ Xs, Ys \in list(\alpha)\}$
% $Post = \{append(Xs, Ys, Zs) |\ Xs, Ys, Zs \in list(\alpha) \ \wedge \ Zs = Xs * Ys\}$
% $|append(Xs, Ys, Zs)| = \|Xs\|$

```
append( [], L, L).
append( [X | Xs], L, [X | Zs]) :-
        append(Xs, L, Zs).
```

% attach(+, +, +, -)
% $Pre = \{attach(First, List, Last, Res) | List \in list(\alpha) \ \wedge \ First, Last \in \alpha\}$
% $Post = \{attach(First, List, Last, Res) | Res \in list(\alpha) \ \wedge$
% $\quad Res = [First] * List * [Last]\}$
% $|attach(First, List, Last, Res)| = \|List\|$

```
attach( First, List, Last, [First | Rest] ) :-
        append(List, [Last], Rest).
```

% sequence_r(+, +, -)
% $Pre = \{sequence\_r(Start, N, Res) | Start, N \in GaeN \ \wedge \ N > 0\}$
% $Post = \{sequence\_r(Start, N, Res) | Start, N \in GaeN \ \wedge \ N > 0 \ \wedge$
% $\quad Res = [Start, ..., Start + N - 1]\}$
% $|sequence\_r(Start, N, Res)| = N$

```
sequence_r( Start, 1, [Start]).
sequence_r( Start, N, [Start | List]) :-
        N > 1,
        Start1 is Start + 1, N1 is N - 1,
        sequence_r(Start1, N1, List).
```

% sequence_re(+, +, -)
% $Pre = \{sequence\_re(Start, N, Res) | Start, N \in GaeN \ \wedge \ N > 0\}$
% $Post = \{sequence\_re(Start, N, Res) | Start, N \in GaeN \ \wedge \ N > 0 \ \wedge$
% $\quad Res = [Start, ..., Start - N + 1]\}$
% $|sequence\_rr(Start, N, Res)| = N$

```
sequence_re( Start, 1, [Start]).
sequence_re( Start, N, [Start | List]) :-
        N > 1,
        Start1 is Start - 1, N1 is N - 1,
        sequence_re(Start1, N1, List).
```

% sequence_c(+, +, -)
% $Pre = \{sequence\_c(Start, N, Res) | Start, N \in GaeN \ \wedge \ N > 0\}$
% $Post = \{sequence\_c(Start, N, Res) | Start, N \in GaeN \ \wedge \ N > 0 \ \wedge$
% $\quad Res = [[Start], ..., [Start + N - 1]]\}$
% $|sequence\_c(Start, N, Res)| = N$

```
sequence_c( Start, 1, [[Start]]).
sequence_c( Start, N, [[Start] | List]) :-
       N > 1,
       Start1 is Start + 1, N1 is N - 1,
       sequence_c(Start1, N1, List).

% sorround(+, +, +, -)
%   Pre = {sorround(Ls, L, R, Res)|L, R ∈ GaeN, Ls ∈ list(listGaeN)}
%   Post = {sorround(Ls, L, R, Res)|L, R ∈ GaeN, Ls ∈ list(listGaeN)
%            Res = [[L] * Ls_1 * [R], [L − 1] * Ls_2 * [R + 1], ..., [L − N + 1] * Ls_N * [R + N − 1]]
%              where N = ||Ls|| }
%   |sorround(Ls, L, R, Res)| = Σ_{X∈Ls} ||X||

sorround([], _, _, []).
sorround([ Row | List ], L, R, [ RowS | ListS ]) :-
       attach( L, Row, R, RowS),
       L1 is L - 1 , R1 is R + 1,
       sorround(List, L1, R1, ListS).

% sorround1(+, +, +, -)
%   Pre = {sorround1(N, L, R, Res)|N, L, R ∈ GaeN ∧ N ≥ 0}
%   Post = {sorround1(N, L, R, Res)|N, L, R ∈ GaeN ∧ N ≥ 0
%       N = 0 ⇒ Res = [] ∧ N > 0 ⇒ Res = [[L, R], [L − 1, R + 1], ..., [L − N + 1, R + N − 1]]}
%   |sorround1(N, L, R, Res)| = N

sorround1(0, _, _, []).
sorround1(N, L, R, [ [L,R] | ListS ]) :-
       N > 0,
       L1 is L - 1 , R1 is R + 1, N1 is N - 1,
       sorround1(N1, L1, R1, ListS).

% calc( +, +, +, -)
%   pre = {calc(Start, R, C, S)|Start, R, C ∈ GaeN ∧ R, C > 0}
%   post = {calc(Start, R, C, S)|Start, R, C ∈ GaeN ∧ R, C > 0 ∧ S = sp(R, C, Start)}
%   |calc(Start, R, C, S)| = R · C,

calc( Start, 1, C, [ Row ]) :-
       sequence_r( Start, C, Row).

calc( Start, R, 1, Spiral ):-
       sequence_c( Start, R, Spiral).

calc( Start, 2, C, [Row1, Row2]) :-
       C > 1,
       sequence_r( Start, C, Row1 ),
       Start1 is Start + 2*C - 1,
       sequence_re( Start1, C, Row2).

calc( Start, R, 2, Spiral ) :-
       R > 1,
       Left is Start + 2*R - 1,
       Right is Start + 2,
       R1 is R - 2,
       sorround1(R1, Left, Right, SpiralS),
       Start1 is Start + 1,
```

```
        StartL is Start + R,
        StartL1 is StartL + 1,
        attach( [Start,Start1], SpiralS, [StartL1,StartL], Spiral).

calc( Start, R, C, Spiral ) :-
        R > 2, C > 2,
        sequence_r( Start, C, FirstRow),
        Start1 is Start + 2*R + 2*C - 4,
        R1 is R - 2, C1 is C - 2,
        calc(Start1, R1, C1, SpiralR),
        Left is Start1 - 1, Right is Start + C ,
        sorround(SpiralR, Left, Right, SpiralS),
        Start2 is Start + 2*C + R - 3,
        sequence_re( Start2, C, LastRow),
        attach( FirstRow, SpiralS, LastRow, Spiral).
```

% > (+, +)
% $Pre = \{N > M | N, M \in Gae\}$
% $Post = \{N > M | N, M \in Gae \ \wedge \ value(N) > value(M)\}$
% $|N > M| = 0$

% is (-, +)
% $Pre = \{X \ \ is \ \ M | M \in Gae\}$
% $Post = \{X \ \ is \ \ M | N \in GaeN, M \in Gae \ \wedge \ value(X) = value(M)\}$
% $|N \mathtt{is} M| = 0$

2 Proof outlines

2.1 The triple $\vdash_t \{Pre\}\ P\ \{Post\}$

According to [3], we define the notion of triples $\vdash_t \{Pre\}\ P\ \{Post\}$ and $\vdash_t \{Pre\}\ Q\ \{Post\}$.

A proof outline $\mathcal{PO}$ for a clause $A \leftarrow A_1, \ldots, A_n$ and $|\ |$, Pre, $Post$ is a construct of the form

$$
\begin{array}{llll}
\quad\{g_0\} & & & \\
A_0 & \leftarrow & & \{t_0\} \\
 & & \{g_1\} & \\
 & A_1, & & \{t_1\} \\
 & & \{f_1\} & \\
 & & \{g_2\} & \\
 & \vdots & & \\
 & A_{n-1}, & & \{t_{n-1}\} \\
 & & \{f_{n-1}\} & \\
 & & \{g_n\} & \\
 & A_n. & & \{t_n\} \\
 & & \{f_n\} & \\
\quad\{f_0\} & & &
\end{array}
$$

where t_i and f_i, g_i, for $i \in [0, n]$ are respectively integer expressions and assertions (in some formal logic), such that every ground instance of the following *proof obligations* is satisfied:

(i) for $i \in [0, n]$: $t_i = |A_i|$,

(ii) for $i \in [0, n]$: $g_i \Leftrightarrow A_i \in Pre$,

(iii) for $i \in [0, n]$: $f_i \Leftrightarrow A_i \in Post$,

(iv) for $i \in [1, n]$

$$g_0 \wedge f_1 \wedge \ldots \wedge f_{i-1} \Rightarrow g_i \wedge \begin{cases} t_0 > t_i & \text{if } rel(A_0) \simeq rel(A_i) \\ t_0 \geq t_i & \text{otherwise} \end{cases}$$

(v) $g_0 \wedge f_1 \wedge \ldots \wedge f_n \Rightarrow f_0$.

(We recall that $\simeq$ relates mutually recursive predicates.) The intuition is that the assertion g_i specifies the conditions under which the atom A_i of the clause is in Pre, the assertion f_i specifies the conditions under which A_i is in $Post$, and the expressions t_i represent the level mapping of A_i.

We say that $\vdash_t \{Pre\}\ P\ \{Post\}$ holds if and only if there exists a level mapping $|\ |$ and a proof outline for each clause of P and $|\ |$, Pre, $Post$.

In the sequel, we shall also make use of a weakened form of proof outlines, which enable us to use f_i's and g_i's which do not coincide with post and pre-conditions, but rather represent strengthenings and weakenings of $Post$ and Pre, according to the following equivalent interpretation of the assertions used in proof outlines. We denote with h_i for $i \in [1, n]$ the assertion $g_0 \wedge f_1 \wedge \ldots \wedge f_{i-1}$.

(i) for $i \in [0, n]$: $h_i \Rightarrow t_i = |A_i|$,

(ii) $g_0 \Leftarrow A_0 \in Pre$, and for $i \in [1, n]$: $h_i \wedge g_i \Rightarrow A_i \in Pre$,

(iii) for $i \in [1, n]$: $f_i \Leftarrow A_i \in Post \wedge h_i$, and $h_n \wedge f_0 \Rightarrow A_0 \in Post$,

(iv) for $i \in [1, n]$

$$g_0 \wedge f_1 \wedge \ldots \wedge f_{i-1} \Rightarrow g_i \wedge \begin{cases} t_0 > t_i \text{ if } rel(A_0) \simeq rel(A_i) \\ t_0 \geq t_i \quad\quad\quad \text{otherwise} \end{cases}$$

(v) $g_0 \wedge f_1 \wedge \ldots \wedge f_n \Rightarrow f_0$,

With this more liberal interpretation, simpler proof outlines can be constructed.

For a query Q, we say that $\vdash_t \{Pre\}\ Q\ \{Post\}$ holds if and only if there exists a natural number k such that for every ground instance $A_1, \ldots, A_n$ of Q

- for $i \in [1, n]$

$$Post \models A_1, \ldots, A_{i-1} \Rightarrow Pre \models A_i \quad \wedge \quad k > |A_i|$$

In the following we will show that $\vdash_t \{Pre\}$ `Spiral` $\{Post\}$ holds by presenting a proof outline for each clause of `Spiral`. We will make use of some syntactic sugar. We omit reporting the level mapping of arithmetic atoms, since the related proof obligations are trivially satisfied, and we group pre- and post-conditions for arithmetic atoms in the body of a clause.

$\{R, C \in GaeN \ \wedge \ R, C > 0 \ \}$
`spiral( R, C, S ) :-` $\quad\quad \{ \ R \cdot C \ \}$
$\quad\quad \{ \ R, C \in GaeN \ \wedge \ R, C > 0\}$
$\quad$ `calc( 1, R, C, S).` $\quad\quad \{ \ R \cdot C \ \}$
$\quad\quad \{ \ S = sp(R, C, 1) \ \}$
$\{ \ S = sp(R, C, 1) \ \}$

$\{L \in list(\alpha)\}$
`append([], L, L)`
$\{ \ L = [\,] * L, L \in list(\alpha)\}$

$\{[X|Xs], Ys \in list(\alpha)\}$
`append( [X|Xs], Ys, [X|Zs] )` $\leftarrow \quad\quad \{ \ \|Xs\| + 1 \ \}$
$\quad\quad \{ \ Xs, Ys \in list(\alpha)\}$
$\quad$ `append( Xs, Ys, Zs ).` $\quad\quad \{ \ \|Xs\| \ \}$
$\quad\quad \{ \ Zs = Xs * Ys, Z \in list(\alpha)\}$
$\{ \ [X|Zs] = [X|Xs] * Ys, [X|Zs] \in list(\alpha) \ \}$

Actually, the last two proof outlines (and the next one) are schemata of proof outlines, one for each instantiation of the parameter α.

$\{ \ List \in list(\alpha), First, Last \in \alpha \ \}$
`attach(First, List, Last, [First|Rest])` $\leftarrow \quad \{ \ \|List\| \ \}$
$\quad \{List, [Last] \in list(\alpha) \ \}$
$\quad$ `append(List, [Last], Rest),` $\quad\quad \{ \ \|List\| \ \}$
$\quad \{Rest = List * [Last], Rest \in list(\alpha) \ \}$
$\{ \ [First|Rest] = [First] * List * [Last] \in list(\alpha) \ \}$

$\{ \ Start \in GaeN\}$
`sequence_r( Start, 1, [Start]).`
$\{ \ true\}$

$\{\ Start, N \in GaeN, N > 0\ \}$
```
sequence_r( Start, N, [Start | List]) ←                    { N }
```
$\{\ N, Start \in GaeN\}$
```
N > 1,
Start1 is Start + 1, N1 is N - 1,
```
$\{\ Start1, N1 \in GaeN, N > 1, Start1 = Start + 1, N1 = N - 1\ \}$

$\{\ Start1, N1 \in GaeN, N1 > 0\ \}$
```
sequence_r(Start1, N1, List)                                { N - 1 }
```
$\{\ List = [Start1, \ldots, Start1 + N1 - 1]\}$

$\{\ [Start|List] = [Start, \ldots, Start + N - 1]\}$

Note that $[Start|List] = [Start, \ldots, Start + N - 1]$ since $[Start|List] = [Start, Start1, \ldots, Start1 + N1 - 1]$ and $Start1 = Start + 1, Start1 + N1 - 1 = (Start + 1) + (N - 1) - 1$.

$\{\ Start \in GaeN\ \}$
```
sequence_re( Start, 1, [Start]).
```
$\{\ true\}$

$\{\ Start, N \in GaeN, N > 0\ \}$
```
sequence_re( Start, N, [Start | List]) ←                   { N }
```
$\{\ Start, N \in GaeN\}$
```
N > 1,
Start1 is Start + 1, N1 is N - 1,
```
$\{\quad Start1, N1 \in GaeN, N > 1, Start1 = Start - 1, N1 = N - 1\}$

$\{\quad Start1, N1, \in GaeN, N1 > 0\ \}$
```
sequence_re(Start1, N1, List)                               { N - 1 }
```
$\{\ List = [Start1, \ldots, Start1 - N1 + 1]\}$

$\{\ [Start|List] = [Start, \ldots, Start - N + 1]\}$

$\{\ Start \in GaeN\ \}$
```
sequence_c( Start, 0, [[Start]]).
```
$\{\ true\}$

$\{\ Start, N \in GaeN, N > 0\ \}$
```
sequence_c( Start, N, [[Start] | List]) ←                  { N }
```
$\{\ N, Start \in GaeN\}$
```
N > 1,
Start1 is Start + 1, N1 is N - 1,
```
$\{\ N1, Start1 \in GaeN, N > 1, Start1 = Start - 1, N1 = N - 1\}$

$\{\ Start1, N1 \in GaeN, N1 > 0\ \}$
```
sequence_c(Start1, N1, List)                                { N - 1 }
```
$\{\ List = [[Start1], \ldots, [Start1 + N1 - 1]]\}$

$\{\ [Start|List] = [[Start], \ldots, [Start + N - 1]]\}$

$\{\ L, R \in GaeN\ \}$
```
sorround([], L, R,[]).
```
$\{\ true\}$

$\{\ L, R \in GaeN, [Row|List] \in list(ListGaeN)\ \}$
`sorround([Row|List], L, R, [RowS|ListS])` $\leftarrow$ $\{\ \|Row\| + \Sigma_{X \in List}\|X\|\ \}$
  $\{\ Row \in listGaeN, L, R \in GaeN\ \}$
  `attach( L, Row, R, RowS),` $\{\|Row\|\}$
  $\{\ RowS \in listGaeN, RowS = [L] * Row * [R]$
  $\{\ L, R \in GaeN\}$
  `L1 is L - 1, R1 is R + 1`
  $\{\ L1, R1 \in GaeN, L1 = L - 1, R1 = R + 1\ \}$
  $\{\ List \in list(listGaeN), L1, R1 \in GaeN\ \}$
  `sorround ( List, L1, R1, ListS )` $\{\Sigma_{X \in List}\|X\|\ \}$
  $\{\ ListS \in listGaeN, ListS = [[L1] * List_1 * [R1], \ldots, [L1 - N + 1] * List_N * [R1 + N - 1]]$
   where $N = \|List\|\}$
  $\{\ [Rows|ListS] = [[L] * Row * [R], \ldots, [L - N + 1] * List_N * [R + N - 1]]$
    where $N = \|[Row|List]\|$
   $[Rows|ListS] \in list(listGaeN)\ \}$

$\{\ L, R \in GaeN\ \}$
`sorround1(0, L, R,[]).`
$\{\ true\}$

$\{\ N, L, R \in GaeN, N \geq 0\ \}$
`sorround1(N,L,R, [[L,R]|ListS] )` $\leftarrow$ $\{\ N\ \}$
  $\{\ N, L, R \in GaeN\ \}$
  `N > 0,L1 is L-1, R1 is R + 1, N1 is N - 1`
  $\{\ N1, L1, R1 \in GaeN, N > 0, L1 = L - 1, R1 = R + 1, N1 = N - 1\}$
  $\{\ N1, L1, R1 \in GaeN, N1 \geq 0\}$
  `sorround1(N1,L1,R1,ListS)` $\{\ N - 1\ \}$
  $\{\ N1 = 0 \Rightarrow ListS = [], N1 > 0 \Rightarrow ListS = [[L1, R1], \ldots [L1 - N1 + 1, R1 + N1 - 1]]\ \}$
$\{\ [[L, R]|ListS] = [[L, R], \ldots [L - N + 1, R + N - 1]]\}$

$\{\ Start, C \in GaeN, C > 0\ \}$
`calc(Start,1,C,[Row])` $\leftarrow$ $\{\ C\ \}$
  $\{\ Start, C \in GaeN, C > 0\ \}$
  `sequence_r(Start, C, Row)` $\{\ C\ \}$
  $\{\ Row = [Start, Start + 1, \ldots, Start + C - 1]\}$
$\{\ [Row] = sp(1, C, Start)\ \ \}$

$\{\ Start, R \in GaeN, R > 0\ \}$
`calc(Start,R,1,Spiral)` $\leftarrow$ $\{\ R\ \}$
  $\{\ Start, R \in GaeN, R > 0\}$
  `sequence_c(Start, R, Spiral)` $\{\ R\ \}$
  $\{\ Spiral = [[Start], \ldots, [Start + R - 1]]\}$
$\{\ Spiral = sp(R, 1, Start)\}$

$\{\ Start, C \in GaeN, C > 0\ \}$
`calc(Start, 2, C, [Row1, Row2])` $\leftarrow$ $\{\ 2 \cdot C\ \}$
  $\{\ C \in GaeN\}$
  `C > 1`
  $\{\ C > 1\ \}$
  $\{\ Start, C \in GaeN, C > 0\ \}$

`sequence_r(Start, C, Row1)` $\{\ C\ \}$

$\{\ Row1 = [Start, Start+1, \ldots, Start+C-1]\}$

$\{\ Start, C \in GaeN\ \}$

`Start1 is Start + 2 * C - 1`

$\{\ Start1 \in GaeN, Start1 = Start + 2 \cdot C - 1\ \}$

$\{\ Start1, C \in GaeN, C > 0\ \}$

`sequence_re(Start1, C, Row2)` $\{\ C\ \}$

$\{\ Row2 = [Start1, Start1-1, \ldots, Start-C+1]\}$

$\{\ [Row1, Row2] = sp(2, C, Start)\ \}$

Note that $Row2 = [Start1, Start1-1, \ldots, Start-C+1]$ since $Start1 = Start + 2 \cdot C - 1$.

$\{\ Start, R \in GaeN, R > 0\ \}$

`calc(Start, R, 2, Spiral)` $\leftarrow$ $\{\ R \cdot 2\ \}$

$\{\ R \in GaeN\}$

`R > 1`

$\{\ R > 1\ \}$

$\{\ Start, R \in GaeN\ \}$

```
Left is Start+ 2 * R - 1
Right is Start + 2
R1 is R -2
```

$\{\ Left, Right, R1 \in GaeN, Left = Start + 2 \cdot R - 1, Right = Start + 2, R1 = R - 2\ \}$

$\{\ R1, Left, Right \in GaeN, R1 \geq 0\ \}$

`sorround1(R1, Left, Right, SpiralS)` $\{\ R - 2\ \}$

$\{\ R1 = 0 \Rightarrow SpiralS = [],$
$R1 > 0 \Rightarrow SpiralS = [[Left, Right], \ldots, [Left - R1 + 1, Right + R1 - 1]]\ \}$

$\{\ Start, R \in GaeN\ \}$

`Start1 is Start + 1, StartL is Start + R`

$\{\ Start1, StartL \in GaeN, Start1 = Start + 1, StartL = Start + R\ \}$

$\{\ StartL \in GaeN\ \}$

`StartL1 is StartL + 1`

$\{\ StartL1 \in GaeN, StartL1 = StartL + 1\ \}$

$\{\ SpiralS \in list(listGaeN), [Start, Start1], [StartL1, StartL] \in listGaeN\ \}$

`attach([Start, Start1], SpiralS, [StartL1, StartL], Spiral)` $\{\|SpiralS\|\ \}$

$\{\ Spiral = [[Start, Start1]] * SpiralS * [[StartL1, StartL]]\ \}$

$\{\ Spiral = sp(R, 2, Start)\ \}$

Note that $R \cdot 2 > \|SpiralS\|$ since $SpiralS$ is a list of lenght $R1$ and $R1$ is $R - 2$. Moreover, $Spiral = sp(R, 2, Start)$ since $Spiral = [[Start, Start1], [Left, Right], \ldots, [Left - R1 + 1, Right + R1 - 1], [StartL1, StartL]] = [[Start, Start+1], [Start + 2 \cdot R - 1, Start + 2], \ldots, [Start + R + 1, Start + R - 1], [Start + R + 1, Start + R]]$

$\{\ Start, C, R \in GaeN, C, R > 0\ \}$

`calc(Start, R, C, Spiral)` $\leftarrow$ $\{\ R \cdot C\ \}$

$\{\ R, C \in GaeN\ \}$

`R > 2, C > 2`

$\{\ R > 2, C > 2\ \}$

$\{\ Start, C \in GaeN, C \geq 0\ \}$

`sequence_r(Start, C, FirstRow),` $\{\ C\ \}$

$\{\ FirstRow = [Start, ..., Start + C - 1]\ \}$

$\{\ Start, R, C \in\ \ GaeN\ \}$

```
Start1 is Start + 2 * R + 2 * C -4
R1 is R - 2, C1 is C - 2
```
$\{\ Start1, R1, C1 \in GaeN,$
$\quad Start1 = Start + 2 \cdot R + 2 \cdot C - 4, R1 = R - 2, C1 = C - 2\ \}$
$\{\ Start1, R1, C1 \in GaeN, R1, C1 > 0\ \}$
`calc(Start1, R1, C1, SpiralR)` $\{\ (R-2) \cdot (C-2)\ \}$
$\{\ SpiralR = sp(R1, C1, Start1)\ \}$
$\{\ Start1, Start, C, \in GaeN\ \}$
`Left is Start1 - 1, Right is Start + C`
$\{\ Left, Right, \in GaeN, Left = Start1 - 1, Right = Start + C\ \}$
$\{\ Left, Right \in GaeN, SpiralR \in listGae\ \}$
`sorround(SpiralR, Left, Right, SpiralS)` $\{\ \Sigma_{X \in SpiralR} \|X\|\ \}$
$\{\ SpiralR \in list(listGae)$
$\quad SpiralS = [[Left] * SpiralR_1 * [Right], \ldots, [Left - K + 1] * SpiralR_{R1} * [Right + R1 - 1]]$
$\quad \{\ \text{where } R1 = \|SpiralR\|\ \}$
$\{\ Start, C, R \in GaeN\ \}$
`Start2 is Start + 2 * C + R - 3`
$\{\ Start2 \in GaeN, Start2 = Start + 2 \cdot C + R - 3\ \}$
$\{\ Start2, C \in GaeN, C > 0\ \}$
`sequence_re(Start2, C , LastRow)` $\{\ C\ \}$
$\{\ LastRow = [Start2, \ldots, Start2 - C + 1\ \}$
$\{\ SpiralS \in list(listGaeN), FirstRow, LastRow \in listGaeN\ \}$
`attach(FirstRow, SpiralS, LastRow, Spiral)` $\{\ \|SpiralS\|\ \}$
$\{\ Spiral = [FirstRow] * SpiralS * [LastRow]\ \}$
$\{\ Spiral = sp(R, C, Start)\ \}$

Note that $R \cdot C > \Sigma_{X \in SpiralR} \|X\| = (R-2) \cdot (C-2)$. Finally, $R \cdot C > \|SpiralS\| = R - 2$.

2.2 Well-supportedness

Following [3], we introduce well-supported post-conditions. They will allow us to state total correctness of Prolog programs.

A proof outline (to prove well-supportdness) $\mathcal{PO}$ for a clause $A \leftarrow B_1, \ldots, B_n$, and *Pre*, *Post*, is a construct of the form

$$\begin{array}{lll} & \{g\} & \\ A & & \leftarrow \\ & & B_1, \\ & & \quad \{f_1\} \\ & & \vdots \\ & & B_{n-1}, \\ & & \quad \{f_{n-1}\} \\ & & B_n. \\ & & \quad \{f_n\} \end{array}$$

where f_i, g, for $i \in [0, n]$ are assertions (in some formal logic), such that every ground instance of the following *proof obligations* holds:

(i) for $i \in [1, n]$: $g \wedge f_i \Rightarrow B_i \in Post$,

(ii) for $i \in [1, n]$: $g \Rightarrow f_i$.

The assertion g is used to instantiate the local variables of the clause to the end of satisfying the proof obligations.

We say that *Post* is well supported w.r.t. P and *Pre* if and only if there exist a number of proof outlines for (instances of) clauses from P, such that for every atom A in $Post \cap Pre$ a ground instance of a proof outline exists, such that the head of the clause is A, and the assertion associated with the clause holds.

Intuitively, *Post* is well-supported when there is a *justification* for every $A \in Post$.

Let us show the proof outlines for the clauses of `Spiral`.

$\{\ R, C \in GaeN, R, C > 0, S = sp(R, C, 1)\ \}$
```
spiral( R, C, S ) :-
    calc( 1, R, C, S).
```
$\{\ R, C \in GaeN, R, C > 0, S = sp(R, C, 1)\ \}$

$\{\ Ys \in list(\alpha)\ \}$
```
append([], Ys, Ys).
```

$\{\ [X|Xs], Ys \in list(\alpha), [X|Zs] = [X|Xs] * Ys\ \}$
```
append([X | Xs], Ys, [X, | Zs]) :-
    append(Xs, Ys, Zs).
```
$\{\ Xs, Ys \in list(\alpha), Zs = Xs * Ys\ \}$

We point out that for every $append(Xs, Ys, Zs) \in Post \cap Pre$, either $Xs = []$ or $Xs = [X|X1s]$ for some $X, X1s$. Therefore, the two proof outlines provide a justification for every case.

$\{\ [First|Rest] \in list(\alpha), First, Last \in \alpha, [First|Rest] = [First] * List * [Last]\ \}$
```
attach( First, List, Last, [First | Rest] ) :-
    append(List, [Last], Rest).
```
$\{\ List, [Last], Rest \in list(\alpha), Rest = List * [Last]\ \}$

$\{\ Start \in GaeN\ \}$
```
sequence_r( Start, 1, [Start]).
```

$\{\ Start, N, Start1, N1 \in GaeN, N > 1\ \wedge\ [Start|List] = [Start, ..., Start + N - 1]$
$\wedge\ Start1 = Start + 1, N1 = N - 1\ \}$
```
sequence_r( Start, N, [Start | List]) :-
    N > 1,
    Start1 is Start + 1,
    N1 is N - 1,
```
$\{\ N, Start1, N1, Start \in GaeN$
$\wedge\ N1 = N - 1, N > 1, Start1 = Start + 1\ \}$
```
    sequence_r(Start1, N1, List).
```
$\{\ Start1, N1 \in GaeN, List = [Start1, ..., Start1 + N1 - 1], N1 > 0\ \}$

As for `append`, we point out that for every $sequence\_r(Start, N, Res) \in Post \cap Pre$, either $N = 1$ or $N > 1$. Therefore, the two proof outlines provide a justification for every case. The same reasoning applies to `sequence_re` and `sequence_c`.

$\{\ Start \in GaeN\ \}$
```
sequence_re( Start, 1, [Start]).
```

$\{\ Start, N \in GaeN, N > 1, [Start|List] = [Start, ..., Start - N + 1]$
$\wedge\ Start1 = Start - 1\ \wedge\ N1 = N - 1\ \}$

```
sequence_re( Start, N, [Start | List]) :-
   N > 1,
   Start1 is Start - 1,
   N1 is N - 1,
```

$\{\ N, N1, Start1, Start \in GaeN$
$\wedge\ N1 = N - 1, N > 1, Start1 = Start - 1\ \}$

```
   sequence_re(Start1, N1, List).
```

$\{\ Start1, N \in GaeN, List = [Start1, ..., Start1 - N1 + 1], N1 > 0\ \}$

$\{\ Start \in GaeN\ \}$

```
sequence_c( Start, 0, [[Start]]).
```

$\{\ Start, N \in, N > 1, [Start|List] = [[Start], ..., [Start + N - 1]]$
$\wedge\ Start1 = Start + 1, N1 = N - 1\ \}$

```
sequence_c( Start, N, [[Start] | List]) :-
   N > 1,
   Start1 is Start + 1,
   N1 is N - 1,
```

$\{\ N, N1, Start, Start1 \in Gae$
$\wedge\ N1 = N - 1, N > 1, Start1 = Start + 1\ \}$

```
   sequence_c(Start1, N1, List).
```

$\{\ Start1, N1 \in GaeN, List = [[Start1], ..., [Start1 + N1 - 1]], N1 > 0\ \}$

$\{\ L, R \in GaeN\ \}$

```
sorround([], L, R, []).
```

$\{\ L1, R1, L, R \in GaeN\ \wedge\ [Row|List] \in list(listGaeN), L1 = L - 1\ \wedge\ R1 = R + 1\ \wedge$
$[RowS|ListS] = [[L] * Row * [R], [L - 1] * List_1 * [R + 1], ..., [L - n + 1] * List_n * [R + n - 1]]$
where $n = \|List\|\ \}$

```
sorround([ Row | List ], L, R, [ RowS | ListS ]) :-
   attach( L, Row, R, RowS),
```

$\{\ RowS \in listGaeN, RowS = [L] * Row * [R]\ \}$

```
   L1 is L - 1 ,
   R1 is R + 1,
```

$\{\ L, R, L1, R1 \in GaeN, L1 = L - 1\ \wedge\ R1 = R + 1\ \}$

```
   sorround(List, L1, R1, ListS).
```

$\{\ L1, R1 \in GaeN, ListS = [[L1] * List_1 * [R1], ..., [L - n + 1] * List_n * [R + n - 1]]$
where $n = \|List\|\ \}$

As above, for every $sorround(List, L, R, Res) \in Pre \cap Post$, either $List = []$ or $List = [X|Xs]$ for some X, Xs. Therefore, we covered both cases. A similar reasoning applies to `sorround1`, considering the cases $N = 0$ and $N > 0$.

$\{\ L, R \in GaeN\ \}$

```
sorround1(0, L, R, []).
```

$\{\ N, L, R, N1, L1, R1 \in GaeN, N > 0, [[L, R]|ListS] = [[L, R], ..., [L - N + 1, R + N - 1]]$
$\{\ L1 = L - 1, R1 = R + 1, N1 = N - 1\ \}$

```
sorround1(N, L, R, [ [L,R] | ListS ]) :-
   N > 0,
```

`L1 is L - 1 ,`

$\{ \ N, L, L1 \in GaeN, N > 0, L1 = L - 1 \ \}$

`R1 is R + 1,`

`N1 is N - 1,`

$\{ \ R, N, R1, N1 \in GaeN, R1 = R + 1, N1 = N - 1 \ \}$

`sorround1(N1, L1, R1, ListS).`

$\{ \ N1, L1, R1 \in GaeN, N1 \geq 0,$
$N1 = 0 \Rightarrow ListS = [], N1 > 0 \Rightarrow ListS = [[L1, R1], ..., [L1 - N1 + 1, R1 + N1 - 1]]$

$\{ \ Start, C \in GaeN, C > 0, S = sp(1, C, Start) \quad \}$

```
calc( Start, 1, C, [ Row ]) :-
   sequence_r( Start, C, Row).
```

$\{ \ Start, C \in GaeN, C > 0, Row = [Start, ..., Start + C - 1] \ \}$

$\{ \ Start, R \in GaeN, R > 0, S = sp(R, 1, Start) \quad \}$

```
calc( Start, R, 1, Spiral) :-
   sequence_c( Start, R, Spiral).
```

$\{ \ Start, R \in GaeN, R > 0, Spiral = [[Start], ..., [Start + R - 1]] \ \}$

$\{ \ Start, Start1, C \in GaeN, C > 1, Start1 = Start + 2 \cdot C - 1$
$S = [Row1, Row2] = sp(2, C, Start) \ \}$

```
calc( Start, 2, C, [Row1, Row2]) :-
   C > 1,
```

$\{ \ C \in GaeN, C > 1 \ \}$

`sequence_r( Start, C, Row1 ),`

$\{ \ Start, C \in GaeN, C > 0, Row1 = [Start, ..., Start + C - 1] \ \}$

`Start1 is Start + 2*C - 1,`

$\{ \ Start1, Start \in GaeN, Start1 = Start + 2 \cdot C - 1 \ \}$

`sequence_re( Start1, C, Row2 ),`

$\{ \ Start1, C \in GaeN, C > 0, Row2 = [Start1, ..., Start1 - C + 1] =$
$[Start + 2 \cdot C - 1, ..., Start + C] \ \}$

$\{ \ Start, C, Left, Right, Start1, StartL, R1 \in GaeN$
$Spiral = [[Start, Start1]] * SpiralS * [[StartL1, StartL]] = sp(R, 2, Start), R > 1$
$Left = Start + 2 \cdot R - 1, Right = Start + 2, R1 = R - 2$
$StartL1 = StartL + 1, Start1 = Start + 1, StartL = Start + R \ \}$

```
calc( Start, R, 2, Spiral ) :-
   R > 1,
```

$\{ \ R \in GaeN \ \wedge \ R > 1 \ \}$

`Left is Start + 2*R - 1,`

$\{ \ Left, Start \in GaeN \ \wedge \ Left = Start + 2 \cdot R - 1 \ \}$

`Right is Start + 2,`

$\{ \ Right, Start \in GaeN \ \wedge \ Right = Start + 2 \ \}$

`R1 is R - 2,`

$\{ \ R1, R \in GaeN \ \wedge \ R1 = R - 2 \ \}$

`sorround1(R1, Left, Right, SpiralS),`

$\{ \ R1, Left, Right \in GaeN, R1 \geq 0 \ \wedge$
$Spirals = [[Left, Right], ..., [Left - R1 + 1, Right + R1 - 1]] \ \}$

`Start1 is Start + 1,`

$\{ \ Start1, Start \in GaeN, Start1 = Start + 1 \ \}$

`StartL is Start + R,`

$\{ \ StartL, Start \in GaeN, StartL = Start + R \ \}$

`StartL1 is StartL + 1,`

$\{ \; StartL1, StartL \in GaeN, StartL1 = StartL + 1 \; \}$

`attach( [Start,Start1], SpiralS, [StartL1,StartL], Spiral).`

$\{ \; Spiral = [[Start, Start1]]] * SpiralS * [[StartL1, StartL]] \in list(listGaeN) \; \}$

$$\{ \; Spiral = \begin{array}{lcr} [[Start, & \ldots & Start + C - 1], \\ [Start + 2 \cdot C + 2 \cdot R - 5, & & Start + C], \\ [\;\; \vdots & SpiralR & \vdots \;\;], \\ [Start + 2 \cdot C + R - 2, & & Start + C + R - 3], \\ [Start + 2 \cdot C + R - 3, & \ldots & Start + C + R - 2]] \end{array} =$$

$$= [FirstRow] * SpiralS * [LastRow] = sp(R, c, Start)$$

$Start, Start1, Start2, R, C, R1, C1, Left, Right \in GaeN, Start1 = Start + 2 \cdot R + 2 \cdot C - 4$
$R1 = R - 2, C1 = C - 2, Left = Start1 - 1, Right = Start + C$
$Start2 = Start + 2 \cdot C + R - 3 \; \}$

`calc( Start, R, C, Spiral ) :-`

`R > 2, C > 2,`

$\{ \; R, C \in GaeN, R > 2, C > 2 \; \}$

`sequence_r( Start, C, FirstRow),`

$\{ \; Start, C \in GaeN, FirstRow = [Start, ..., Start + C - 1] \; \}$

`Start1 is Start + 2*R + 2*C - 4,`

$\{ \; Start1, Start \in GaeN, Start1 = Start + 2 \cdot R + 2 \cdot C - 4 \; \}$

`R1 is R - 2, C1 is C - 2,`

$\{ \; R1, R, C1, C \in GaeN, R1 = R - 2, C1 = C - 2 \; \}$

`calc(Start1, R1, C1, SpiralR),`

$\{ \; Start1, R1, C1 \in GaeN, R1, C1 > 0, SpiralR = sp(R1, C1, Start1) \quad \}$

`Left is Start1 - 1,`

$\{ \; Left, Start1 \in GaeN, Left = Start1 - 1 \; \}$

`Right is Start + C ,`

$\{ \; Right, Start \in GaeN, Right = Start + C \; \}$

`sorround(SpiralR, Left, Right, SpiralS),`

$\{ \; Left, Right \in GaeN, SpiralS = [[Left] * SpiralR_1 * [Right], ...,$
$..., [Left - (R - 2) + 1] SpiralR_{R-2} [Right + (R - 2) + 1]$
where $R - 2 = R1 = \|SpiralR\| \quad \}$

`Start2 is Start + 2*C + R - 3,`

$\{ \; Start2, Start \in GaeN, Start2 = Start + 2 \cdot C + R - 3 \; \}$

`sequence_re( Start2, C, LastRow),`

$\{ \; Start2, C \in GaeN, C > 0, LastRow = [Start2, ..., Start2 - C + 1] \; \}$

`attach( FirstRow, SpiralS, LastRow, Spiral).`

$\{ \; Spiral = [FirstRow] * SpiralS * [LastRow] \in list(listGae) \; \}$

Defining `calc`, we distinguished five cases: $R = 1$, $C = 1$, $R = 2 \wedge C > 1$, $R > 1 \wedge C = 2$, $R > 2 \wedge C > 2$. They cover all possibles cases in the precondition $R, C > 0$. Proving that for an instance of a proof outline such that the head $A \in Post \cap Pre$ the formula g holds is a simple exercise. For instance, consider an instance of the last proof outline, where $R, C > 2$ and $Spiral$ is a spiral of R rows and C columns starting from $Start$. Then by constraining $Start1, Start2, R1, C1, Left, Right$ - i.e. the local variables of that clause - as required, one simply notes that all the other relations are trivially satisfied.

3 Correctness

In this section we show a number of properties of `Spiral`, including termination, absence from arithmetic errors, correctness, computed instance characterization, safe omission of the occur check. They are all immediate consequences of the proof that $\vdash_t \{Pre\}$ `Spiral` $\{Post\}$, with *Post* well-supported w.r.t. `Spiral` and *Pre*. We recall the following results from [3].

THEOREM 3.1 *Let P be a program such that $\vdash_t \{Pre\}\ P\ \{Post\}$, Post well-supported w.r.t. P and Pre, and Q a query such that $\vdash_t \{Pre\}\ Q\ \{Post\}$. Then*

(i) $Post \cap Pre = M_P \cap Pre$,

(ii) every LD-tree for P and Q is finite,

(iii) no LD-derivation for P and Q ends in an error,

(iv) if $\mathcal{Q} = \{\ Q' \mid Q' \in [Q] \wedge Post \models Q'\ \}$ is finite, then $\mathcal{Q}$ is the set of computed instances of P and Q,

(v) let us suppose that for every atom A, if $Pre \models A$ then A is ground in its input positions. Suppose also that the head of every clause of P is output-linear. Then every LD-derivation for P and Q is NSTO.

□

We showed that $\vdash_t \{Pre\}$ `Spiral` $\{Post\}$, with *Post* well-supported w.r.t. `Spiral` and *Pre*. Consider now the query Q = `spiral(`R`,`C`,S)`, with $R, C \in GaeN$. It is immediate to verify that $\vdash_t \{Pre\}\ Q\ \{Post\}$, since the level of any instance of Q is bounded (equal) to $R \cdot C$, and $Q \in Pre$. Therefore, we are in the hypothesis of the Theorem. We conclude the following facts:

(i) $Post \cap Pre = M_{\texttt{Spiral}} \cap Pre$,

(ii) every LD-tree for `Spiral` and Q is finite,

(iii) no LD-derivation for `Spiral` and Q ends in an error,

(iv) the set $\mathcal{Q} = \{\ Q' \mid Q' \in [Q] \wedge Post \models Q'\ \}$ is finite (since $sp(R, C, Start)$ is unique), and equal to

$$\{\ spiral(R, C, S) \mid S \text{ spiral of R rows and C columns starting from 1 }\}$$

Therefore $\mathcal{Q}$ is the set of computed instances of `Spiral` and Q,

(v) it is immediate to verify that for every atom A, if $Pre \models A$ then A is ground in its input positions. In addition, every head of a clause of `Spiral` is output-linear. We conclude that every LD-derivation for `Spiral` and Q is NSTO.

Delays

We present the following theorem, as a corollary of the results presented in [1]. It is concerned with safe omission of the occur check and deadlock freedom of logic programs with delays.

THEOREM 3.2 *Let P be a program such that $\vdash \{Pre\}\ P\ \{Post\}$, Post well-supported w.r.t. P and Pre, and Q a query such that $\vdash \{Pre\}\ Q\ \{Post\}$. Then*

(i) assuming that the head of every clause of P is output linear and the delay declarations imply the moding, then the occur check can be safely omitted in any SLD-derivation for P and Q respecting the delays,

(ii) assuming a set of delay declaration for P, if for every atom A

$$Pre \models A \quad \textit{implies } A \textit{ satisfies its d.d.'s}$$

then no SLD-derivation for P and Q respecting the delays deadlocks,

(iii) assuming that P is simply moded and the delay declarations imply matching, then no SLD-derivation for P and Q respecting the delays ends in an error.

Proof.
(i). Theorem 9 from [1].
(ii). The proof is a simple extension of Theorem 26 from [1].
(ii). The proof is a simple extension of Theorem 33 from [1].

□

Concerning *(i)*, we already know that the head of every clause of `Spiral` is output linear. Assuming the delay declarations

```
delay p(X+,Y-) until ground(X+)                (1)
```

where `X+` are the variables occurring in input positions, for every predicate `p` defined in `Spiral`, it is easy to verify that they imply the moding. Therefore, we conclude that the occur check can be safely omitted in any SLD-derivation for `Spiral` and Q respecting the delays (1).

Concerning *(ii)*, we already know that for every atom A, if $Pre \models A$ then A is ground in its input positions. Therefore, assuming the delays (1), we have

$$Pre \models A \quad \text{implies } A \text{ satisfies its d.d.'s}$$

and then the conclusion that no SLD-derivation for `Spiral` and Q respecting the delays deadlocks.

Concerning *(iii)*, we point out that `Spiral` is simply moded and that the delay declarations (1) imply matching. Therefore, no SLD-derivation for `Spiral` and Q respecting the delays ends in an error.

Determinacy does not hold for SLD-derivations of `Spiral` (this is due to the non-determinism in the definition of `calc`.) Therefore, we cannot use Theorem 37 from [1] to show termination via determinacy. However, with some small changes, one can derive a regular guarded program whose SLDG-trees respecting the delay declarations are finite.

We conclude by noting that the delay declarations (1) allow only for a small parallelization of the program `Spiral`, since they are rather constraining. A better choice - always working when reasoning on lists - should be the use of delay declarations such as

```
delay append(Xs,Ys,Zs) until nonground(Xs)
```

However, in this case other properties may be lost.

4 Conclusions

In this paper, we applied a verification method to the case study of the program `Spiral`. We proved that $\vdash_t$ $\{Pre\}$ `Spiral` $\{Post\}$, with *Post* well-supported w.r.t. `Spiral` and *Pre*. As a result, the proof method allow us to conclude several properties of `Spiral`, including termination, absence from arithmetic errors, correctness, computed instance characterization, safe omission of the occur check, and properties concerning computations respecting some delay declarations (such as in the language Gödel [2].

We were able to reason on those properties by means of only one (declarative) proof method. This fact facilitated the development of the program and its proof. We found that several steps in the verification process are automatic (and, hopefully, automatizable) and intuitively correct.

References

[1] K. R. Apt and I. Luitjes. Verification of logic programs with delay declarations. In *Proceedings of the Fourth International Conference on Algebraic Methodology and Software Technology, (AMAST'95)*, Lecture Notes in Computer Science, Berlin, 1995. Springer-Verlag. Invited Lecture. In press.

[2] P.M. Hill and J.W. Lloyd. The logic programming language Gödel. Technical report, University of Bristol, 1991.

[3] S. Ruggieri. Metodi formali per lo sviluppo di programmi logici. Master's thesis, Dipartimento di Informatica, Università di Pisa, 1994. Tesi di Laurea.