# The factorial example

**Line 1**
Link server page to logic module exporting *factorial*/2. *factorial*(*N*,*F*) computes *F* = *N*!

**Line 5**
Just as Prolog predicates, functions in expression language expressions may fail, evaluating no result. Default values help display meaningful content in such situations.

```
 1 <html logic-module="factorial"
 2       xmlns:psp="http://prosper.iit.bme.hu">
 3 <h1>Factorial example</h1>
 4 <psp:assign var="E"
 5             expr="{atom_number(http_get_default(number, '1'))}">
 6  <p>The factorials from 1! to <psp:insert var="E" />! are</p>
 7  <ul>
 8    <psp:for-all function="between(1, E)" iterator="N">
 9      <li>
10        <psp:insert expr="{N}" />! =
11        <psp:insert function="factorial(N)" />
12      </li>
13    </psp:fo
14  </ul>
15 </psp:assig
16 </html>
```

**Lines 10 and 11**
*psp:insert* embeds the value of an expression or variable into the output. Writing *var="N"* instead of *expr="{N}"* would have the same effect.

**Line 8**
*between*/3 can return multiple results on backtracking. *psp:for-all* invokes *bagof*/3 to fetch all solutions. Each solution is accessible via the local variable *N*.

## Intermediate term

**Line 2 (below)**
Ground terms are stored separately and accessed by index.

```
 1 element(html, [], [
 2   ground_element(1),
 3   extension_predicate(prosper_builtin:assign_expression(E, ...)), [
 4     element(p, [], [
 5        'The factorials from 1! to',
 6        extension_predicate(prosper_builtin:insert_atomic(E), []),
 7        '! are'
 8     ]),
 9     element(ul, [], [
10       extension_predicate(prosper_builtin:for_all(
11         (factorial:between(1, _E))-['E'=_E], 'N', term
12       ), [
13          element(li, [], [
14            extension_predicate(prosper_builtin:insert_expression(
15              expression(_N, ['N'=_N], ...), []
16            ), '! ='
17            extension_predicate(prosper_builtin:insert_function(
18              (factorial:factorial(_N))-['N'=_N]), []
19            )
20         ])
21       ])
22     ])
23   ])
24 ])
```

**Lines 3, 6, 10, 14, 17: Special elements are translated into Prolog transformation rules.**

**Lines 11 and 15**
Prolog variables in functions, goals and expressions are identified with local variables. During evaluation, the proper bindings are made based on association lists. Expression language terms also have an execution plan, which is omitted.