Mechanical Program Verification – Part 3

Jürgen F H Winkler
Institute of Informatics
Friedrich Schiller University
Jena, Germany

ELTE, Budapest, 08 – 12 Oct 2007

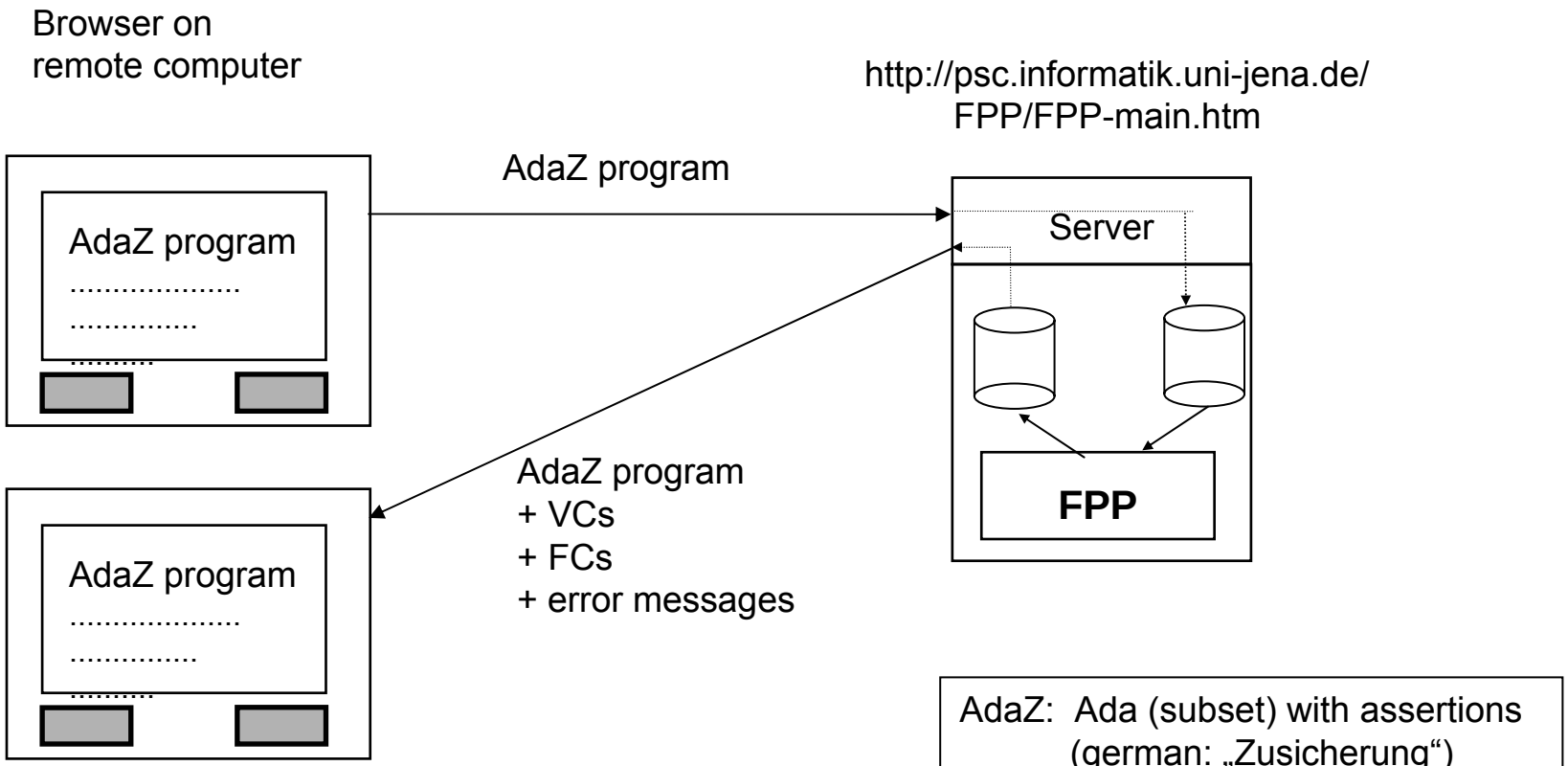
Overview

- Historical Overview, Basic Concepts, Realistic Progr Verific
- Mechanical Program Verification (MPV)
- Comparison of 3 Automatic Program Provers (APP)
- The Frege Program Prover (FPP) in More Detail
- Mechanical Generation of Invariants for FOR-Loops
- Problems of FPP (and others)
- Towards Realistic Verification Conditions (VC)
- Summary

Part 3

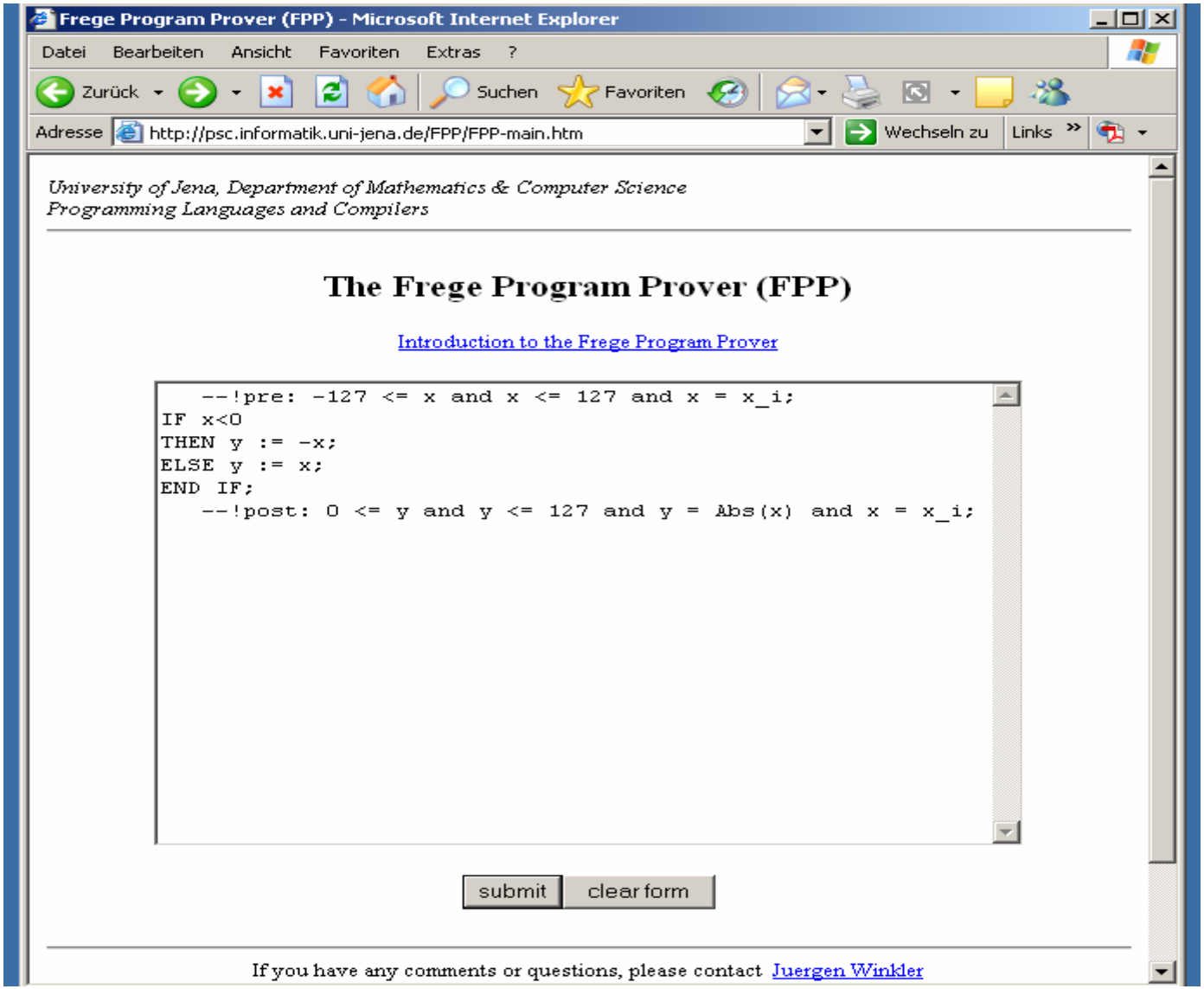
- FPP in more detail

Use of FPP over the WWW



AdaZ: Ada (subset) with assertions
(german: „Zusicherung“)

Gregor Weske: gregor@minet.uni-jena.de
„please reboot the FPP server, thanks“
+49 3641 946 333

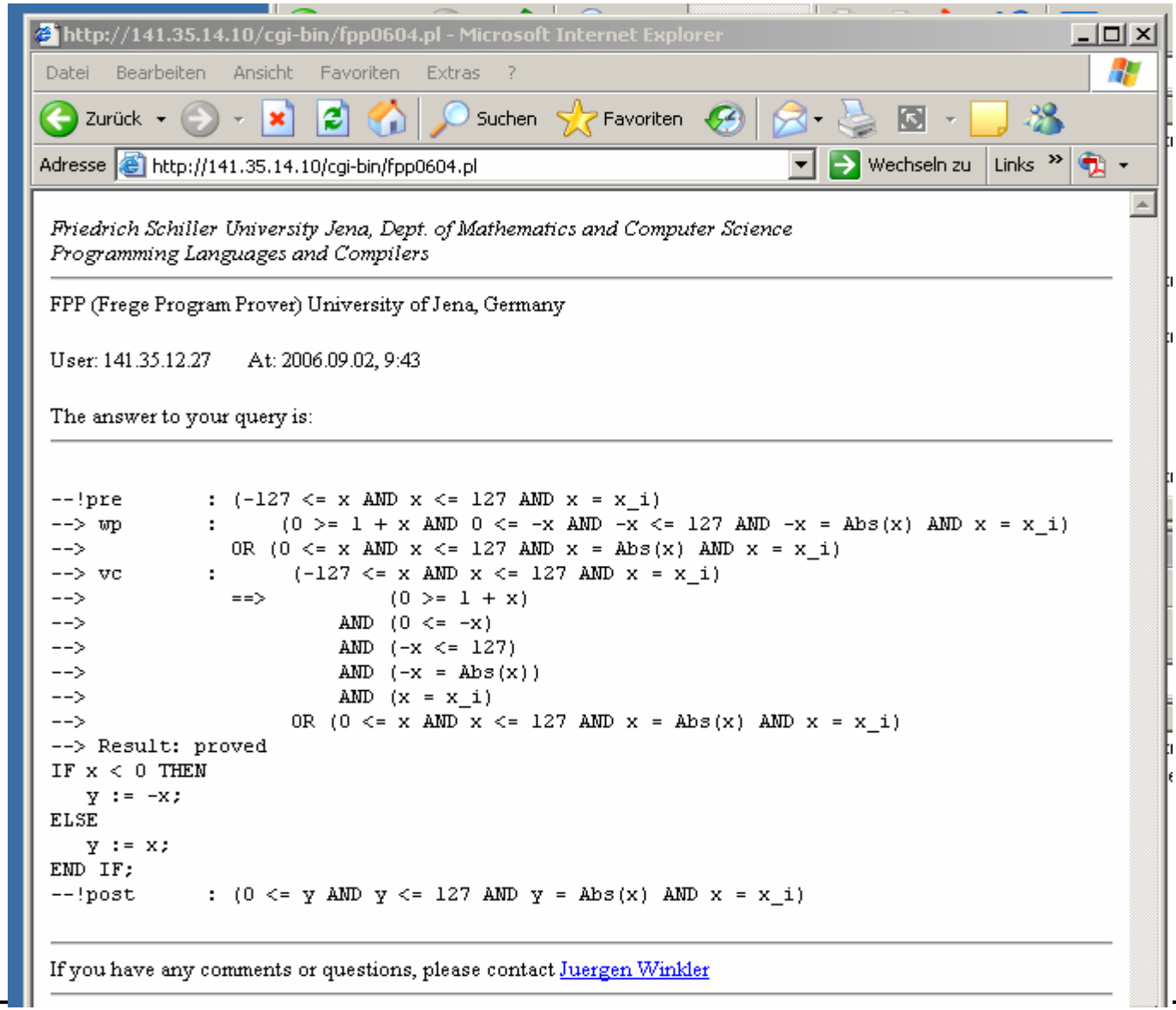


Example: Input

```
  --!pre: -127 <= x and x <= 127 and x = x_i;
IF x<0
THEN y := -x;
ELSE y := x;
END IF;
  --!post: 0 <= y and y <= 127 and y = Abs(x) and x = x_i;
```

x, y: program variables

x_i: specification variable

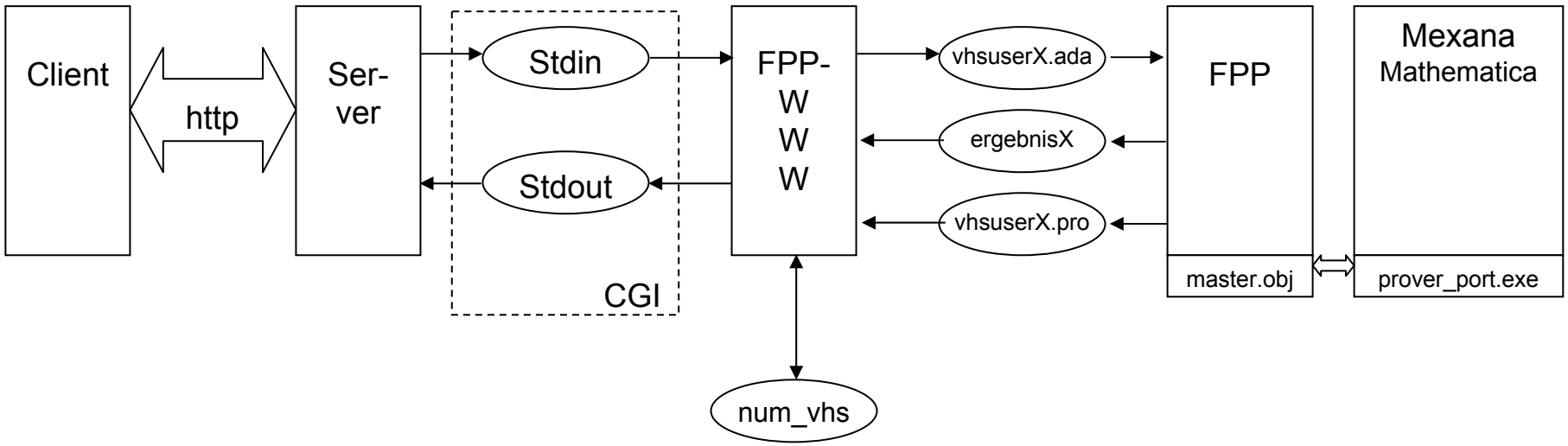


Example: Output

```
--!pre      : (-127 <= x AND x <= 127 AND x = x_i)
--> wp      :    (0 >= 1 + x AND 0 <= -x AND -x <= 127 AND -x = Abs(x) AND x = x_i)
-->          OR (0 <= x AND x <= 127 AND x = Abs(x) AND x = x_i)
--> vc      :    (-127 <= x AND x <= 127 AND x = x_i)
-->          ==>        (0 >= 1 + x)
-->                  AND (0 <= -x)
-->                  AND (-x <= 127)
-->                  AND (-x = Abs(x))
-->                  AND (x = x_i)
-->               OR (0 <= x AND x <= 127 AND x = Abs(x) AND x = x_i)
--> Result: proved
IF x < 0 THEN
  y := -x;
ELSE
  y := x;
END IF;

--!post     : (0 <= y AND y <= 127 AND y = Abs(x) AND x = x_i)
```

Structure of FPP



FPP : compute VCs

Mexana : try to prove the VCs mechanically
(extension of the automatic prover Analytica (Clarke/Zhao, CMU 1992)
written in Mathematica, executed interpretatively)

The Prover: Analytica (with some extensions)

written in Mathematica

executed interpretatively by the Mathematica system

intended application area: analysis

variant of natural deduction

quantifiers are eliminated by Sklolemization

results is a quantifier-free first order formula F

try to reduce F to True

Problem: Analytica is not sound (due to Mathematica) (details come later)

FPP: AdaZ

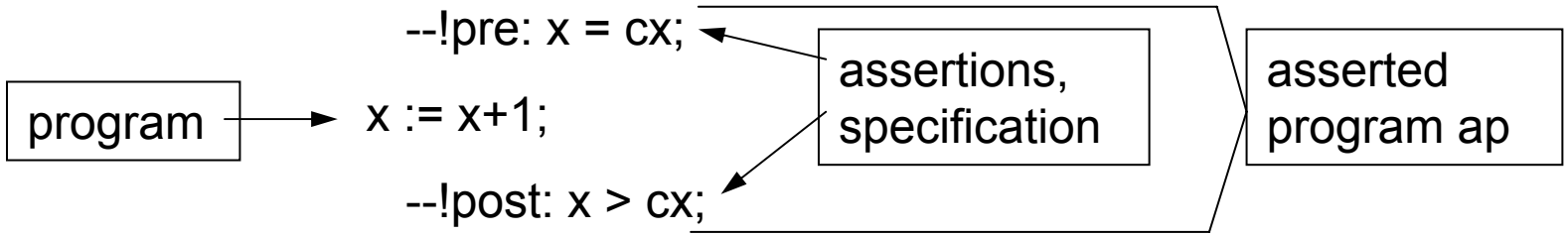
(small) subset of Ada + assertions as special comments

=> AdaZ program = Ada program fragment
(FPP assumes the fragment is legal)

Program p = AdaZ-program without assertions
(proper program or just "program")

Specification q = assertion part of the AdaZ-program
("the assertions" or "the specification")

AdaZ program = the "asserted program ap"



FPP: AdaZ

Statements	:	null | assg | if | case | while | for | sequence
Assertions	:	--!pre: | --!post: | --!inv: | --!term:
Types	:	integer (=$\mathbb{Z}$!) | Boolean
Variables	:	program variables (occur in the program proper): x specification variables (occur in assertions only): cx
Assertion lang	:	Boolean expressions + Implication + Quantification => (forAll …, (exists …

```
--!pre: x = cx;
x := x+1;
--!post: x > cx;
```

FPP: AdaZ: Example 21

```
   --!pre: x = x_i and y = y_i;
temp := x ;
x := y ;
y := temp;
   --!post: x = y_i and y = x_i;


--!pre       : (x = x_i AND y = y_i)
--> wp       : (y = y_i AND x = x_i)
--> vc       : (x = x_i AND y = y_i => y = y_i AND x = x_i)
--> Result: proved
temp := x;
x := y;
y := temp;

--!post      : (x = y_i AND y = x_i)
```

FPP: AdaZ: Example 21b

```
   --!pre: x = x_i and y = y_i;
temp := x ;
   --!post: x = x_i and y = y_i and temp=x;
x := y ;
y := temp;
   --!post: x = y_i and y = x_i;
```

```
--!pre      : (x = x_i AND y = y_i)                    At: 2006.09.05, 8:19
--> wp      : (x = x_i AND y = y_i)
--> vc      : (True)
--> Result: proved
temp := x;
--!post     : (x = x_i AND y = y_i AND temp = x)
--> wp      : (y = y_i AND temp = x_i)
--> vc      : (x = x_i AND y = y_i AND temp = x => y = y_i AND temp = x_i)
--> Result: proved
x := y;
y := temp;
--!post     : (x = y_i AND y = x_i)
```

FPP: AdaZ: Example 21c

```
   --!pre: x = x_i and y = y_i;
temp := x ;
   --!pre: x = x_i and y = y_i and temp=x;
x := y ;
y := temp;
   --!post: x = y_i and y = x_i;

--!pre      : (x = x_i AND y = y_i)                    At: 2006.09.05, 8:25
--> wp      : (x = x_i AND y = y_i)
--> vc      : (True)
--> Result: proved
temp := x;
--!pre      : (x = x_i AND y = y_i AND temp = x)
--> wp      : (y = y_i AND temp = x_i)
--> vc      : (x = x_i AND y = y_i AND temp = x => y = y_i AND temp = x_i)
--> Result: proved
x := y;
y := temp;
--!post     : (x = y_i AND y = x_i)
```

FPP: AdaZ: Example 21d

```
   --!pre: x = x_i and y = y_i;
temp := x ;
   --!post: x = x_i and y = y_i and temp=x;
   --!pre: x = x_i and y = y_i and temp=x;
x := y ;
y := temp;
   --!post: x = y_i and y = x_i;


--!pre       : (x = x_i AND y = y_i)                    At: 2006.09.05, 8:29
--> wp       : (x = x_i AND y = y_i)
--> vc       : (True)
--> Result: proved
temp := x;
--!post      : (x = x_i AND y = y_i AND temp = x)
--> vc       : (True)
--> Result: proved
--!pre       : (x = x_i AND y = y_i AND temp = x)
--> wp       : (y = y_i AND temp = x_i)
--> vc       : (x = x_i AND y = y_i AND temp = x => y = y_i AND temp = x_i)
--> Result: proved
```

FPP: AdaZ: Example 21e

```
   --!pre: x = x_i and y = y_i;
temp := x ;
   --!pre: x = x_i and y = y_i and temp=x;
   --!post: x = x_i and y = y_i and temp=x;
x := y ;
y := temp;
   --!post: x = y_i and y = x_i;
```

```
 2      --!pre: x = x_i and y = y_i;                    At: 2006.09.05, 8:41
 3   temp := x ;
 4      --!pre: x = x_i and y = y_i and temp=x;
 5      --!post: x = x_i and y = y_i and temp=x;
 6   x := y ;
Error^
Skipping...
```

```
Ayacc.YYParse :  1 syntax error found.
```

FPP: AdaZ: FOR-Loop

```
   --!pre: n >= 0 and n <= 7 and n = n_i;
prod := 1;

   --!pre :prod = 1 and n >= 0 and n <= 7 and n = n_i;          Precondition
   --!post: prod = factorial(n) and prod <= 32767 and n = n_i;  Postcondition
   --!inv : prod = factorial(i) and 1 <= i+1 and n <= 7 and n = n_i;  Invariant
FOR i IN 1 .. n LOOP
   prod := prod * i;
END LOOP;
```

Realistic factorial( )

FPP: AdaZ: FOR-Loop

```
--!pre       : (n >= 0 AND n <= 7 AND n = n_i)          At: 2006.09.05, 8:56
--> wp       : (n >= 0 AND n <= 7 AND n = n_i)
--> vc       : (True)
--> Result: proved
prod := 1;

--!pre       : (prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
```

FPP: AdaZ: FOR-Loop

```
--!pre       : (prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
--!post      : (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--!inv       : (prod = Factorial(i) AND 1 <= 1 + i AND n <= 7 AND n = n_i)

-->functionality -------------------------
-->func      : (initial AND induction AND final AND null loop)
-->initial   :
--> Result    : proved
-->induction :
--> Result    : proved
-->final     :
--> Result    : proved
-->null loop :
--> Result    : proved
```

```
FOR i IN 1 .. n  LOOP
   prod := prod * i;
END LOOP;
```

FPP: AdaZ: FOR-Loop

```
--!pre       : (prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
--!post      : (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--!inv       : (prod = Factorial(i) AND 1 <= 1 + i AND n <= 7 AND n = n_i)
-->functionality -------------------------
-->func      : (initial AND induction AND final AND null loop)

-->initial   :     (1 <= n AND prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
-->           ==> (prod = 1 AND n <= 7 AND n = n_i)
--> Result    : proved

-->induction :          (1 <= n)
-->               AND (prod = Factorial(-1 + i))  AND  (1 <= i)
-->               AND (n <= 7)  AND (n = n_i)
-->           ==> (i*prod = Factorial(i) AND 1 <= 1 + i AND n <= 7 AND n = n_i)
--> Result    : proved
```

```
FOR i IN 1 .. n  LOOP
   prod := prod * i;
END LOOP;
```

FPP: AdaZ: FOR-Loop

```
--!pre       : (prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
--!post      : (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--!inv       : (prod = Factorial(i) AND 1 <= 1 + i AND n <= 7 AND n = n_i)

-->functionality -------------------------
-->func      : (initial AND induction AND final AND null loop)

-->final     :          (1 <= n)
-->                AND (prod = Factorial(n)) AND (1 <= 1 + n)
-->                AND (n <= 7) AND (n = n_i)
-->           ==> (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--> Result    : proved

-->null loop :     (1 >= 1 + n AND prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
-->           ==> (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--> Result    : proved

FOR i IN 1 .. n  LOOP
   prod := prod * i;
END LOOP;
```

FPP: AdaZ: WHILE-Loop

```
   --!pre: n >= 0 and n <= 7 and n = n_i;
prod := 1;
i := 0;

   --!pre : prod = 1 and n >= 0 and n <= 7 and n = n_i and i=0;
   --!post: prod = factorial(n) and prod <= 32767 and n = n_i;
   --!inv : prod = factorial(i) and 0 <= i and i<=n and n <= 7 and n = n_i;
   --!term: n-i;                                          Termination function

WHILE i < n LOOP
   i := i+1;
   prod := prod * i;
END LOOP;
```

FPP: AdaZ: WHILE-Loop

```
--!pre       : (prod = 1 AND n >= 0 AND n <= 7 AND n = n_i AND i = 0)
--!post      : (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--!inv       : (prod = Factorial(i) AND 0 <= i AND i <= n AND n <= 7 AND n = n_i)
--!term      : (-i + n)
-->functionality -------------------------
-->initial
--> Result    : proved
-->induction :
--> Result    : proved
-->final
--> Result    : proved
-->termination ------------------------
-->initial
--> Result    : proved
-->induction
--> Result    : proved
```

```
WHILE i < n LOOP
   i := i + 1;
   prod := prod * i;
END LOOP;
```

FPP: Modi

a) Compute wp

Input: statement
assertion

Output: wp(statement, assertion) -- asg, if, case, null, sequ
statement
assertion

b) Try to verify -- asg, if, case, null, sequ

Input: assertion-1
statement
assertion-2

Output: assertion-1
wp(statement, assertion-2)
VC $\equiv$ [assertion-1 $\Rightarrow$ wp(statement, assertion-2)]
Result -- (proved | not proved + FC)
statement
assertion-2

FPP: AdaZ: wp

a) null

wp(“null;”, post) ≡ post

b) assignment

wp(“v := expr;”, post) ≡ $post^{v}_{expr}$ -- OK ?

c) sequence

wp(“s1 s2”, post) ≡ wp(“s1”, wp(“s2”, post))

FPP: AdaZ: wp

d) if-statement

wp("if Cond Then S1 Else S2 End If;", post) ≡

Cond ∧ wp("S1", post) ∨ ¬Cond ∧ wp("S2", post)

```
If Cond
Then S
End IF;
```

⇓

```
If Cond
Then S
Else null;
End IF;
```

```
If Cond1
Then S1
Elsif  Cond2
Then S2
Else S3
End IF;
```

⇒

```
If Cond1
Then S1
Else  If Cond2
        Then S2
        Else S3
        End If;
End IF;
```

=> reduction to canonical If Then Else End If

FPP: AdaZ: wp

e) case-statement

CASE =

```
Case  Expr is
   When V1.1 | … | V1.n1 => S1
   . . .
   When Vm.1 | … | Vm.nm => Sm
   When Others  => So
End Case;
```

$0 \le m$

$V_{i.j}$ ::= const
| const1 .. const2

$C_{i.j} \equiv Expr = const$
$C_{i.j} \equiv const1 \le Expr \wedge Expr \le const2$
$1 \le i \le m,\ 1 \le j \le n_i$

$C_i \equiv C_{i.1} \vee \ldots \vee C_{i.n_i} \qquad 1 \le i \le m$
$C \equiv C_1 \vee \ldots \vee C_m$
$C_o \equiv \neg C$

C_i must be pairwise disjoint

Type(Expr) must be covered

FPP: AdaZ: wp

e) case-statement

$$wp(CASE, post) \equiv C1 \wedge wp(S1, post) \vee \ldots Cm \wedge wp(Sm, post) \vee Co \wedge wp(So, post)$$

```
Case  Expr is
   When Sel.1 => S1
   . . .
   When Sel.m => Sm
   When Others  => So
End Case;
```

```
--!pre: c>0;
case x is
    when -c..-1 => z := -x;
    when 1..c   => z := x;
    when others => z := 1;
end case;
--!post: z>=0;
```

FPP: AdaZ: VC for loops

f) FOR-loop

For loops it is not so easy to compute the weakest precondition,

especially if the number of repetitions is not statically known.

Kauer has developed an algorithm which works in certain cases
[Kau 99; KW 2007]

This algorithm is not (yet) built into FPP.

FPP uses the traditional approximation based on

- loop invariant (FOR, WHILE)
- termination function (WHILE)

In Ada a FOR-loop always terminates if

- the computation of the bounds terminates and
- the execution of the body terminates

FOR-loops in the C-world do not have this property

FPP: AdaZ: VC for loops

f) FOR-loop

FOR id IN [REVERSE] simple-expr1 .. simple-expr2
LOOP statm-sequence
END LOOP;

declaration of id , id is a *constant* in the statm-sequence

simple-expr1 > simple-expr2 $\Rightarrow$ null-loop

let v1 = value of simple-expr1, v2 = value of simple-expr2

id IN v1 .. v2 : statm-sequence$^{id}_{v1}$
statm-sequence$^{id}_{succ(v1)}$
. . .
statm-sequence$^{id}_{v2}$

FPP: AdaZ: VC for loops

f) FOR-loop: VC (based on [Hoa 72])

```
    -- pre

FOR id IN e1 .. e2
LOOP
        -- e1≤e2 ∧ inv^id_pred(id)
    statm-sequence (ss)
        -- inv
END LOOP;
    -- post
```

$$\text{init} \equiv e1 \le e2 \wedge \text{pre} \Rightarrow \text{inv}^{id}_{pred(e1)}$$

$$\text{null} \equiv e1 > e2 \wedge \text{pre} \Rightarrow \text{post}$$

$$\text{ind} \equiv e1 \le e2 \wedge \text{inv}^{id}_{pred(id)} \Rightarrow \text{wp(ss, inv)}$$

$$\text{final} \equiv e1 \le e2 \wedge \text{inv}^{id}_{e2} \Rightarrow \text{post}$$

$$VC \equiv \text{init} \wedge \text{null} \wedge \text{ind} \wedge \text{final}$$

FPP: FOR-Loop: VC

```
--!pre :prod = 1 and n >= 0 and n <= 7 and n = n_i;          Precondition
--!post: prod = factorial(n) and prod <= 32767 and n = n_i;  Postcondition
--!inv : prod = factorial(i) and 1 <= i+1 and n <= 7 and n = n_i;  Invariant
FOR i IN 1 .. n LOOP
  prod := prod * i;
END LOOP;
```

init $\equiv$ $e1 \leq e2 \wedge pre \Rightarrow inv^{id}_{pred(e1)}$

$\equiv$ $1 \leq n \wedge prod = 1 \wedge n >= 0 \wedge n <= 7 \wedge n = n\_i$
$\Rightarrow$ $(prod = factorial(i) \wedge 1 <= i+1 \wedge n <= 7 \wedge n = n\_i)^{i}_{0}$

$\equiv$ $1 \leq n \wedge prod = 1 \wedge n >= 0 \wedge n <= 7 \wedge n = n\_i$
$\Rightarrow$ $prod = factorial(0) \wedge 1 <= 0+1 \wedge n <= 7 \wedge n = n\_i$

$\equiv$ $1 \leq n \wedge prod = 1 \wedge n >= 0 \wedge n <= 7 \wedge n = n\_i \Rightarrow prod = 1 \wedge 1 <= 1$

$\equiv$ True

```
-->initial   :    (1 <= n AND prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
-->           ==> (prod = 1 AND n <= 7 AND n = n_i)
--> Result    : proved
```

FPP: FOR-Loop: VC

```
  --!pre :prod = 1 and n >= 0 and n <= 7 and n = n_i;          Precondition
  --!post: prod = factorial(n) and prod <= 32767 and n = n_i;  Postcondition
  --!inv : prod = factorial(i) and 1 <= i+1 and n <= 7 and n = n_i;  Invariant
FOR i IN 1 .. n LOOP
  prod := prod * i;
END LOOP;
```

null ≡ e1>e2 ∧ pre ⇒ post

≡ 1>n ∧ prod = 1 ∧ n >= 0 ∧ n <= 7 ∧ n = n_i
⇒ prod = factorial(n) ∧ prod <= 32767 ∧ n = n_i

≡ n<=0 ∧ prod = 1 ∧ n >= 0 ∧ n <= 7 ∧ n = n_i
⇒ prod = factorial(n) ∧ prod <= 32767

≡ n=0 ∧ prod = 1 ∧ 0 <= 7 ∧ n = n_i ⇒ prod = factorial(0) ∧ 1 <= 32767

≡ n=0 ∧ prod = 1 ∧ 0 <= 7 ∧ n = n_i ⇒ 1 = 1 ∧ 1 <= 32767

≡ True

```
-->null loop :    (1 >= 1 + n AND prod = 1 AND n >= 0 AND n <= 7 AND n = n_i)
-->          ==> (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--> Result    : proved
```

FPP: FOR-Loop: VC

```
  --!pre :prod = 1 and n >= 0 and n <= 7 and n = n_i;          Precondition
  --!post: prod = factorial(n) and prod <= 32767 and n = n_i;  Postcondition
  --!inv : prod = factorial(i) and 1 <= i+1 and n <= 7 and n = n_i;  Invariant
FOR i IN 1 .. n LOOP
  prod := prod * i;
END LOOP;
```

ind $\equiv$ $e1 \leq e2 \wedge inv^{id}_{pred(id)} \Rightarrow wp(ss, inv)$

$\equiv$ $1 \leq n \wedge (prod = factorial(i) \wedge 1 <= i+1 \wedge n <= 7 \wedge n = n\_i)^{i}_{i-1}$
$\Rightarrow$ wp("prod := prod * i", prod = factorial(i) $\wedge$ 1 <= i+1 $\wedge$ n <= 7 $\wedge$ n = n_i)

$\equiv$ $1 \leq n \wedge prod = factorial(i-1) \wedge 1 <= i-1+1 \wedge n <= 7 \wedge n = n\_i$
$\Rightarrow$ prod * i = factorial(i) $\wedge$ 1 <= i+1 $\wedge$ n <= 7 $\wedge$ n = n_i

$\equiv$ $1 \leq n \wedge prod = factorial(i-1) \wedge 1 <= i \wedge n <= 7 \wedge n = n\_i$
$\Rightarrow$ factorial(i-1) * i = factorial(i) $\wedge$ 0 <= i

$\equiv$ $1 \leq n \wedge prod = factorial(i-1) \wedge 1 <= i \wedge n <= 7 \wedge n = n\_i \Rightarrow True \wedge 0 <= i$

```
-->induction : (1 <= n) AND (prod = Factorial(-1 + i))  AND  (1 <= i)
-->                AND (n <= 7)  AND (n = n_i)
-->           ==> (i*prod = Factorial(i) AND 1 <= 1 + i AND n <= 7 AND n = n_i)
--> Result    : proved
```

FPP: FOR-Loop: VC

```
  --!pre :prod = 1 and n >= 0 and n <= 7 and n = n_i;          Precondition
  --!post: prod = factorial(n) and prod <= 32767 and n = n_i;  Postcondition
  --!inv : prod = factorial(i) and 1 <= i+1 and n <= 7 and n = n_i;  Invariant
FOR i IN 1 .. n LOOP
  prod := prod * i;
END LOOP;
```

final $\equiv$ $e1 \leq e2 \wedge inv^{id}_{e2} \Rightarrow post$

$\equiv$ $1 \leq n \wedge (prod = factorial(i) \wedge 1 <= i+1 \wedge n <= 7 \wedge n = n\_i)^{i}_{n}$
$\Rightarrow$ prod = factorial(n) $\wedge$ prod <= 32767 $\wedge$ n = n_i

$\equiv$ $1 \leq n \wedge prod = factorial(n) \wedge 1 <= n+1 \wedge n <= 7 \wedge n = n\_i$
$\Rightarrow$ prod = factorial(n) $\wedge$ prod <= 32767 $\wedge$ n = n_i

$\equiv$ $1 \leq n \wedge prod = factorial(n) \wedge 0 <= n \wedge n <= 7 \wedge n = n\_i \Rightarrow$ prod <= 32767

$\equiv$ $1 \leq n \wedge prod = factorial(n) \wedge n <= 7 \wedge n = n\_i \Rightarrow$ prod <= 32767 $\equiv$ True

```
-->final    :    (1 <= n) AND (prod = Factorial(n)) AND (1 <= 1 + n)
-->              AND (n <= 7) AND (n = n_i)
-->          ==> (prod = Factorial(n) AND prod <= 32767 AND n = n_i)
--> Result    : proved
```

FPP: AdaZ: VC for loops

g) WHILE-loop

For loops it is not so easy to compute the weakest precondition, especially if the number of repetitions is not statically known.

Kauer has developed an algorithm which works in certain cases [Kau 99]

This algorithm is not (yet) built into FPP.

FPP uses the traditional approximation based on

- loop invariant (FOR, WHILE)
- termination function (WHILE)

FPP: AdaZ: VC for loops

g) WHILE-loop: VC

```
   -- pre
   -- inv
WHILE  cond
LOOP
      -- inv ∧ term>0 ∧ term=T
   statm-sequence (ss)
      -- inv ∧ term<T
END LOOP;
   -- post
```

T is a fresh identifier

$$init_f \equiv pre \Rightarrow inv$$
$$null_f \equiv inv \wedge \neg cond \Rightarrow post$$

$$init_t \equiv cond \wedge inv \Rightarrow term>0$$
$$ind_f \equiv cond \wedge inv \Rightarrow wp(ss, inv)$$
$$ind_t \equiv cond \wedge inv \Rightarrow [wp(ss, term<T)]^T_{term}$$

$$final_f \equiv inv \wedge \neg cond \Rightarrow post$$

$$VC \equiv init_f \wedge ind_f \wedge final_f \wedge init_t \wedge ind_t$$

FPP: AdaZ: WHILE-Loop

```
--!pre : prod = 1 and n >= 0 and n <= 7 and n = n_i and i=0;
--!post: prod = factorial(n) and prod <= 32767 and n = n_i;
--!inv : prod = factorial(i) and 0 <= i and i<=n and n <= 7 and n = n_i;
--!term: n-i;                                          Termination funct
WHILE i < n LOOP
  i := i+1;  prod := prod * i;
END LOOP;
```

$init_f \equiv pre \Rightarrow inv$

$\equiv$ prod = 1 $\wedge$ n >= 0 $\wedge$ n <= 7 $\wedge$ n = n_i $\wedge$ i=0
$\Rightarrow$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i

$\equiv$ prod = 1 $\wedge$ n >= 0 $\wedge$ n <= 7 $\wedge$ n = n_i $\wedge$ i=0
$\Rightarrow$ 1 = factorial(0) $\wedge$ 0 <= 0 $\wedge$ 0<=n

$\equiv$ prod = 1 $\wedge$ n >= 0 $\wedge$ n <= 7 $\wedge$ n = n_i $\wedge$ i=0
$\Rightarrow$ True $\wedge$ True

$\equiv$ True

```
-->initial   :    (prod = 1 AND n >= 0 AND n <= 7 AND n = n_i AND i = 0)
-->           ==> (prod = Factorial(i) AND 0 <= i AND i <= n AND n <= 7 AND n = n_i)
--> Result    : proved
```

FPP: AdaZ: WHILE-Loop

```
--!inv : prod = factorial(i) and 0 <= i and i<=n and n <= 7 and n = n_i;
WHILE i < n LOOP
  i := i+1;  prod := prod * i;
END LOOP;
```

$ind_f \equiv$ cond $\wedge$ inv $\Rightarrow$ wp(ss, inv)

$\equiv$ i < n $\wedge$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i

$\Rightarrow$ wp("i := i+1; prod := prod * i;", prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i)

$\equiv$ i < n $\wedge$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i

$\Rightarrow$ wp("i := i+1;", prod*i = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i)

$\equiv$ i < n $\wedge$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i

$\Rightarrow$ prod*(i+1) = factorial(i+1) $\wedge$ 0 <= i+1 $\wedge$ i+1<=n $\wedge$ n <= 7 $\wedge$ n = n_i

$\equiv$ i < n $\wedge$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i

$\Rightarrow$ factorial(i)*(i+1) = factorial(i+1) $\wedge$ -1 <= i $\wedge$ 0 <= i $\wedge$ i<n

$\equiv$ i < n $\wedge$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i

$\Rightarrow$ factorial(i)*(i+1) = factorial(i+1) $\wedge$ 0 <= i $\equiv$ True

induction : (n >= 1 + i) $\wedge$ (prod = Factorial(i)) $\wedge$ (0 <= i) $\wedge$ (i <= n) $\wedge$ (n <= 7) $\wedge$ (n = n_i)

--> ==> ((1 + i)*prod = Factorial(1 + i)) $\wedge$ (0 <= 1 + i) $\wedge$ (1 + i <= n) $\wedge$ (n <= 7) $\wedge$ (n = n_i)

--> Result : proved

FPP: AdaZ: WHILE-Loop

```
  --!inv : prod = factorial(i) and 0 <= i and i<=n and n <= 7 and n = n_i;
  --!post: prod = factorial(n) and prod <= 32767 and n = n_i;
WHILE i < n LOOP
  i := i+1;  prod := prod * i;
END LOOP;
```

$final_f \equiv$ $inv \wedge \neg cond \Rightarrow post$

$\equiv$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i $\wedge$ $\neg$(i < n)
$\Rightarrow$ prod = factorial(n) $\wedge$ prod <= 32767 $\wedge$ n = n_i

$\equiv$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i<=n $\wedge$ n <= 7 $\wedge$ n = n_i $\wedge$ i $\geq$ n
$\Rightarrow$ prod = factorial(n) $\wedge$ prod <= 32767

$\equiv$ prod = factorial(i) $\wedge$ 0 <= i $\wedge$ i=n $\wedge$ n <= 7 $\wedge$ n = n_i
$\Rightarrow$ prod = factorial(n) $\wedge$ prod <= 32767

$\equiv$ prod = factorial(n) $\wedge$ 0 <= n $\wedge$ i=n $\wedge$ n <= 7 $\wedge$ n = n_i
$\Rightarrow$ prod = factorial(n) $\wedge$ prod <= 32767 $\equiv$ True

final : (n <= i) $\wedge$ (prod = Factorial(i)) $\wedge$ (0 <= i) $\wedge$ (i <= n) $\wedge$ (n <= 7) $\wedge$ (n = n_i)

--> ==> (prod = Factorial(n) $\wedge$ prod <= 32767 $\wedge$ n = n_i)

--> Result : proved

FPP: AdaZ: WHILE-Loop

```
  --!inv : prod = factorial(i) and 0 <= i and i<=n and n <= 7 and n = n_i;
  --!term: n-i;                                          Termination funct
WHILE i < n LOOP
  i := i+1;  prod := prod * i;
END LOOP;
```

$init_t \equiv cond \wedge inv \Rightarrow term>0$

$\equiv i < n \wedge prod = factorial(i) \wedge 0 <= i \wedge i<=n \wedge n <= 7 \wedge n = n\_i \Rightarrow n-i>0$

$\equiv i < n \wedge prod = factorial(i) \wedge 0 <= i \wedge i<=n \wedge n <= 7 \wedge n = n\_i \Rightarrow n>i$

$\equiv$ True

initial : $(n >= 1 + i) \wedge (prod = Factorial(i)) \wedge (0 <= i) \wedge (i <= n) \wedge (n <= 7) \wedge (n = n\_i)$

--> ==> $(-i + n >= 1)$

--> Result : proved

FPP: AdaZ: WHILE-Loop

```
--!inv : prod = factorial(i) and 0 <= i and i<=n and n <= 7 and n = n_i;
--!term: n-i;                                              Termination funct
WHILE i < n LOOP
  i := i+1;  prod := prod * i;
END LOOP;
```

$ind_t \equiv cond \wedge inv \Rightarrow [wp(ss, term<T)]^T_{term}$

$\equiv i < n \wedge prod = factorial(i) \wedge 0 <= i \wedge i<=n \wedge n <= 7 \wedge n = n\_i$

$\Rightarrow [wp(\text{"i := i+1; prod := prod * i;"}, n-i<T)]^T_{n-i}$ <==========

$\equiv i < n \wedge prod = factorial(i) \wedge 0 <= i \wedge n <= 7 \wedge n = n\_i \Rightarrow [n-(i+1)<T]^T_{n-i}$

$\equiv i < n \wedge prod = factorial(i) \wedge 0 <= i \wedge n <= 7 \wedge n = n\_i \Rightarrow n-i-1 < n-i$

$\equiv i < n \wedge prod = factorial(i) \wedge 0 <= i \wedge n <= 7 \wedge n = n\_i \Rightarrow -1 < 0$

$\equiv$ True

induction : $(n >= 1 + i) \wedge (prod = Factorial(i)) \wedge (0 <= i) \wedge (i <= n) \wedge (n <= 7) \wedge (n = n\_i)$

--> ==> $(-i + n >= -i + n)$

--> Result : proved

Rest

Mechanical Program Verification – Part 3

Jürgen F H Winkler
Institute of Informatics
Friedrich Schiller University
Jena, Germany

ELTE, Budapest, 25 – 29 Sep 2006

Kauer: Algorithm for computing an invariant of a FOR-loop

Basic idea: $inv \equiv post^{up}_{loop\text{-}variable}$ and further techniques

Example [Kau 99*: 101..102]:

```
   -- s = 0 ∧ n ≥ 0
for i in m .. m + n loop
   s := s + i;
end loop;
   -- n ≥ 0 ∧ s = sum(j, 1 .. n, j) + m*(n+1)
```

Algorithm computes mechanically the following invariant:

$inv \equiv n \geq 0 \wedge s = sum(j, 1 .. i, j) + m*(i+1)$

*): Kauer, Stefan: Automatische Erzeugung von Verifikations- und Falsifikationsbedingungen sequentieller Programme. Dissertation, Friedrich Schiller University, 1999.Jan.27

Kauer: Algorithm for computing an invariant of a FOR-loop

With this method the task of the program developer becomes easier:

he needs only to provide pre and post

In FPP style:

```
   --!pre:  s = 0 ∧ n ≥ 0;
   --!post:  n ≥ 0 ∧ s = sum(j, 1 .. n, j) + m*(n+1)
for i in m .. m + n loop
   s := s + i;
end loop;
```

=> degree of mechanization is increased