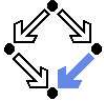
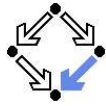


Verifying Concurrent Systems with PVS

Wolfgang Schreiner
 Wolfgang.Schreiner@risc.uni-linz.ac.at

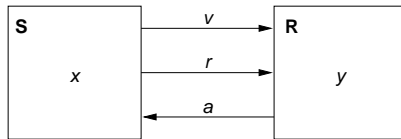
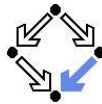
Research Institute for Symbolic Computation (RISC)
 Johannes Kepler University, Linz, Austria
<http://www.risc.uni-linz.ac.at>



1. A Bit Transmission Protocol

2. A Client/Server System

A Bit Transmission Protocol



var x, y
var v := 0, r := 0, a := 0

S: loop

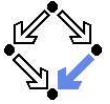
choose x ∈ {0, 1} ||
 1 : v, r := x, 1
 2 : **wait** a = 1
 r := 0
 3 : **wait** a = 0

R: loop

1 : **wait** r = 1
 y, a := v, 1
 2 : **wait** r = 0
 a := 0

Transmit a bit through a wire.

A (Simplified) Model of the Protocol



$State := PC^2 \times (\mathbb{N}_2)^5$

$I(p, q, x, y, v, r, a) :\Leftrightarrow p = q = 1 \wedge x \in \mathbb{N}_2 \wedge v = r = a = 0.$

$R(\langle p, q, x, y, v, r, a \rangle, \langle p', q', x', y', v', r', a' \rangle) :\Leftrightarrow$
 $S1(\dots) \vee S2(\dots) \vee S3(\dots) \vee R1(\dots) \vee R2(\dots).$

$S1(\langle p, q, x, y, v, r, a \rangle, \langle p', q', x', y', v', r', a' \rangle) :\Leftrightarrow$

$p = 1 \wedge p' = 2 \wedge v' = x \wedge r' = 1 \wedge$
 $q' = q \wedge x' = x \wedge y' = y \wedge v' = v \wedge a' = a.$

$S2(\langle p, q, x, y, v, r, a \rangle, \langle p', q', x', y', v', r', a' \rangle) :\Leftrightarrow$

$p = 2 \wedge p' = 3 \wedge a = 1 \wedge r' = 0 \wedge$
 $q' = q \wedge x' = x \wedge y' = y \wedge v' = v \wedge a' = a.$

$R1(\langle p, q, x, y, v, r, a \rangle, \langle p', q', x', y', v', r', a' \rangle) :\Leftrightarrow$

$p = 3 \wedge p' = 1 \wedge a = 0 \wedge x' \in \mathbb{N}_2 \wedge$
 $q' = q \wedge y' = y \wedge v' = v \wedge r' = r \wedge a' = a.$

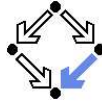
$R2(\langle p, q, x, y, v, r, a \rangle, \langle p', q', x', y', v', r', a' \rangle) :\Leftrightarrow$

$q = 1 \wedge q' = 2 \wedge r = 1 \wedge y' = v \wedge a' = 1 \wedge$
 $p' = p \wedge x' = x \wedge v' = v \wedge r' = r.$

$R3(\langle p, q, x, y, v, r, a \rangle, \langle p', q', x', y', v', r', a' \rangle) :\Leftrightarrow$

$q = 2 \wedge q' = 1 \wedge r = 0 \wedge a' = 0 \wedge$
 $p' = p \wedge x' = x \wedge y' = y \wedge v' = v \wedge r' = r.$

A Verification Task



$\langle I, R \rangle \models \Box(q = 1 \Rightarrow y = x)$

$Invariant(p, \dots) \Rightarrow (q = 1 \Rightarrow y = x)$

$I(p, \dots) \Rightarrow Invariant(p, \dots)$

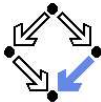
$R(\langle p, \dots \rangle, \langle p', \dots \rangle) \wedge Invariant(p, \dots) \Rightarrow Invariant(p', \dots)$

$Invariant(p, q, x, y, v, r, a) :\Leftrightarrow$

$(p = 1 \vee p = 2 \vee p = 3) \wedge (q = 1 \vee q = 2) \wedge$
 $(x = 0 \vee x = 1) \wedge (v = 0 \vee v = 1) \wedge (r = 0 \vee r = 1) \wedge (a = 0 \vee a = 1) \wedge$
 $(p = 1 \Rightarrow q = 1 \wedge r = 0 \wedge a = 0) \wedge$
 $(p = 2 \Rightarrow r = 1) \wedge$
 $(p = 3 \Rightarrow r = 0) \wedge$
 $(q = 1 \Rightarrow a = 0) \wedge$
 $(q = 2 \Rightarrow (p = 2 \vee p = 3) \wedge a = 1 \wedge y = x) \wedge$
 $(r = 1 \Rightarrow p = 2 \wedge v = x)$

The invariant captures the essence of the protocol.

The Verification Task in PVS



protocol: THEORY

BEGIN

p, q, x, y, v, r, a: nat

p0, q0, x0, y0, v0, r0, a0: nat

S1: bool =

p = 1 AND p0 = 2 AND v0 = x AND r0 = 1 AND
q0 = q AND x0 = x AND y0 = y AND v0 = v AND a0 = a

S2: bool =

p = 2 AND p0 = 3 AND a = 1 AND r0 = 0 AND
q0 = q AND x0 = x AND y0 = y AND v0 = v AND a0 = a

S3: bool =

p = 3 AND p0 = 1 AND a = 0 AND (x0 = 0 OR x0 = 1) AND
q0 = q AND y0 = y AND v0 = v AND r0 = r AND a0 = a

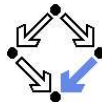
R1: bool =

q = 1 AND q0 = 2 AND r = 1 AND y0 = v AND a0 = 1 AND
p0 = p AND x0 = x AND v0 = v AND r0 = r

R2: bool =

q = 2 AND q0 = 1 AND r = 0 AND a0 = 0 AND
p0 = p AND x0 = x AND y0 = y AND v0 = v AND r0 = r

The Verification Task in PVS (Contd)



Init: bool =

p = 1 AND q = 1 AND (x = 0 OR x = 1) AND
v = 0 AND r = 0 AND a = 0

Step: bool =

S1 OR S2 OR S3 OR R1 OR R2

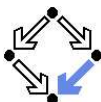
Property: bool =

q = 2 => y = x

Invariant(p, q, x, y, v, r, a: nat): bool =

(p = 1 OR p = 2 OR p = 3) AND
(q = 1 OR q = 2) AND
(x = 0 OR x = 1) AND
(v = 0 OR v = 1) AND
(r = 0 OR r = 1) AND
(a = 0 OR a = 1) AND
(p = 1 => q = 1 AND r = 0 AND a = 0) AND
(p = 2 => r = 1) AND
(p = 3 => r = 0) AND
(q = 1 => a = 0) AND
(q = 2 => (p = 2 OR p = 3) AND a = 1 AND y = x) AND
(r = 1 => (p = 2 AND v = x))

The Verification Task in PVS (Contd'2)



VC0: THEOREM

Invariant(p, q, x, y, v, r, a) => Property

VC1: THEOREM

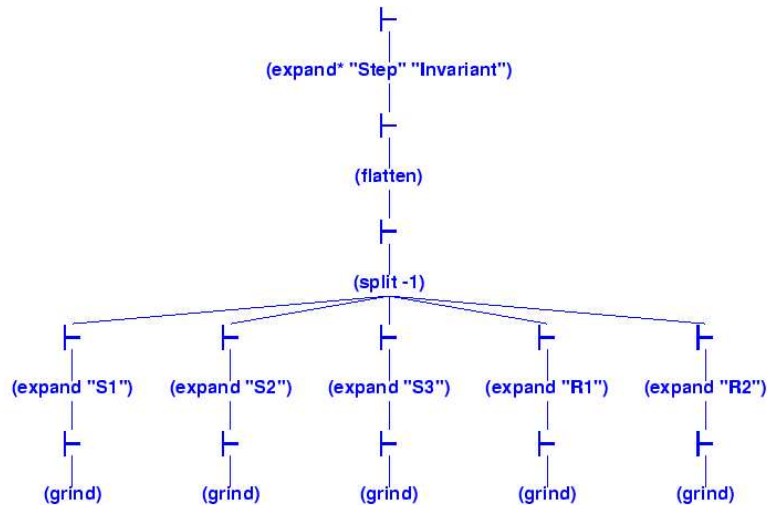
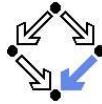
Init => Invariant(p, q, x, y, v, r, a)

VC2: THEOREM

Step AND Invariant(p, q, x, y, v, r, a) =>
Invariant(p0, q0, x0, y0, v0, r0, a0)

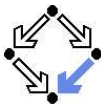
END protocol

The Proof in PVS

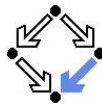


1. A Bit Transmission Protocol

2. A Client/Server System



The Client



Client system $C_i = \langle IC_i, RC_i \rangle$.

State := $PC \times \mathbb{N}_2 \times \mathbb{N}_2$.

Int := $\{R_i, S_i, C_i\}$.

$IC_i(pc, request, answer) \Leftrightarrow$

$pc = R \wedge request = 0 \wedge answer = 0$.

$RC_i(l, \langle pc, request, answer \rangle,$

$\langle pc', request', answer' \rangle) \Leftrightarrow$

$(l = R_i \wedge pc = R \wedge request = 0 \wedge$

$pc' = S \wedge request' = 1 \wedge answer' = answer) \vee$

$(l = S_i \wedge pc = S \wedge answer \neq 0 \wedge$

$pc' = C \wedge request' = request \wedge answer' = 0) \vee$

$(l = C_i \wedge pc = C \wedge request = 0 \wedge$

$pc' = R \wedge request' = 1 \wedge answer' = answer) \vee$

$(l = \overline{REQ}_i \wedge request \neq 0 \wedge$

$pc' = pc \wedge request' = 0 \wedge answer' = answer) \vee$

$(l = ANS_i \wedge$

$pc' = pc \wedge request' = request \wedge answer' = 1).$

Client(ident):

param ident

begin

loop

...

R: sendRequest()

S: receiveAnswer()

C: // critical region

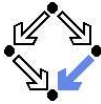
...

sendRequest()

endloop

end Client

The Server



Server system $S = \langle IS, RS \rangle$.

State := $(\mathbb{N}_3)^3 \times \{\{1, 2\} \rightarrow \mathbb{N}_2\}^2$.

Int := $\{D1, D2, F, A1, A2, W\}$.

$IS(given, waiting, sender, rbuffer, sbuffer) \Leftrightarrow$

$given = waiting = sender = 0 \wedge$

$rbuffer(1) = rbuffer(2) = sbuffer(1) = sbuffer(2) = 0$.

$RS(l, \langle given, waiting, sender, rbuffer, sbuffer \rangle,$

$\langle given', waiting', sender', rbuffer', sbuffer' \rangle) \Leftrightarrow$

$\exists i \in \{1, 2\} :$

$(l = D_i \wedge sender = 0 \wedge rbuffer(i) \neq 0 \wedge$

$sender' = i \wedge rbuffer'(i) = 0 \wedge$

$U(given, waiting, sbuffer) \wedge$

$\forall j \in \{1, 2\} \setminus \{i\} : U_j(rbuffer)) \vee$

...

$U(x_1, \dots, x_n) \Leftrightarrow x'_1 = x_1 \wedge \dots \wedge x'_n = x_n$.

$U_j(x_1, \dots, x_n) \Leftrightarrow x'_j(j) = x_j(j) \wedge \dots \wedge x'_n(j) = x_n(j)$.

Server:

local given, waiting, sender

begin

given := 0; waiting := 0

loop

D: sender := receiveRequest()

if sender = given then

if waiting = 0 then

given := 0

else

A1: given := waiting;

waiting := 0

sendAnswer(given)

endif

elsif given = 0 then

A2: given := sender

sendAnswer(given)

else

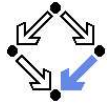
W: waiting := sender

endif

endloop

end Server

The Server (Contd)



```

...
( $I = F \wedge sender \neq 0 \wedge sender = given \wedge waiting = 0 \wedge$ 
 $given' = 0 \wedge sender' = 0 \wedge$ 
 $U(waiting, rbuffer, sbuffer)) \vee$ 

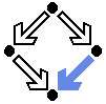
( $I = A1 \wedge sender \neq 0 \wedge sbuffer(waiting) = 0 \wedge$ 
 $sender = given \wedge waiting \neq 0 \wedge$ 
 $given' = waiting \wedge waiting' = 0 \wedge$ 
 $sbuffer'(waiting) = 1 \wedge sender' = 0 \wedge$ 
 $U(rbuffer) \wedge$ 
 $\forall j \in \{1, 2\} \setminus \{waiting\} : U_j(sbuffer)) \vee$ 

( $I = A2 \wedge sender \neq 0 \wedge sbuffer(sender) = 0 \wedge$ 
 $sender \neq given \wedge given = 0 \wedge$ 
 $given' = sender \wedge$ 
 $sbuffer'(sender) = 1 \wedge sender' = 0 \wedge$ 
 $U(waiting, rbuffer) \wedge$ 
 $\forall j \in \{1, 2\} \setminus \{sender\} : U_j(sbuffer)) \vee$ 
...

Server:
  local given, waiting, sender
  begin
    given := 0; waiting := 0
    loop
      D: sender := receiveRequest()
        if sender = given then
          if waiting = 0 then
            F: given := 0
              else
                A1: given := waiting;
                  waiting := 0
                    sendAnswer(given)
                      endif
                elsif given = 0 then
                  A2: given := sender
                    sendAnswer(given)
                      else
                W: waiting := sender
                  endif
                endif
            endloop
          end Server

```

The Server (Contd'2)



```

...
( $I = W \wedge sender \neq 0 \wedge sender \neq given \wedge given \neq 0 \wedge$ 
 $waiting' := sender \wedge sender' = 0 \wedge$ 
 $U(given, rbuffer, sbuffer)) \vee$ 

 $\exists i \in \{1, 2\} :$ 

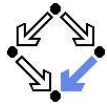
( $I = REQ_i \wedge rbuffer'(i) = 1 \wedge$ 
 $U(given, waiting, sender, sbuffer) \wedge$ 
 $\forall j \in \{1, 2\} \setminus \{i\} : U_j(rbuffer)) \vee$ 

( $I = \overline{ANS}_i \wedge sbuffer(i) \neq 0 \wedge$ 
 $sbuffer'(i) = 0 \wedge$ 
 $U(given, waiting, sender, rbuffer) \wedge$ 
 $\forall j \in \{1, 2\} \setminus \{i\} : U_j(sbuffer)).$ 

Server:
  local given, waiting, sender
  begin
    given := 0; waiting := 0
    loop
      D: sender := receiveRequest()
        if sender = given then
          if waiting = 0 then
            F: given := 0
              else
                A1: given := waiting;
                  waiting := 0
                    sendAnswer(given)
                      endif
                elsif given = 0 then
                  A2: given := sender
                    sendAnswer(given)
                      else
                W: waiting := sender
                  endif
                endif
            endloop
          end Server

```

The Composed System



$$State := (\{1, 2\} \rightarrow PC) \times (\{1, 2\} \rightarrow \mathbb{N}_2)^2 \times (\mathbb{N}_3)^2 \times (\{1, 2\} \rightarrow \mathbb{N}_2)^2$$

$$I(pc, request, answer, given, waiting, sender, rbuffer, sbuffer) :\Leftrightarrow$$

$$\forall i \in \{1, 2\} : IC(pc_i, request_i, answer_i) \wedge$$

$$IS(given, waiting, sender, rbuffer, sbuffer)$$

$$R(\langle pc, request, answer, given, waiting, sender, rbuffer, sbuffer \rangle,$$

$$\langle pc', request', answer', given', waiting', sender', rbuffer', sbuffer' \rangle) :\Leftrightarrow$$

$$(\exists i \in \{1, 2\} : RC_{local}(\langle pc_i, request_i, answer_i \rangle, \langle pc'_i, request'_i, answer'_i \rangle) \wedge$$

$$\langle given, waiting, sender, rbuffer, sbuffer \rangle =$$

$$\langle given', waiting', sender', rbuffer', sbuffer' \rangle) \vee$$

$$(RS_{local}(\langle given, waiting, sender, rbuffer, sbuffer \rangle,$$

$$\langle given', waiting', sender', rbuffer', sbuffer' \rangle) \wedge$$

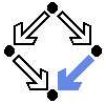
$$\forall i \in \{1, 2\} : \langle pc_i, request_i, answer_i \rangle = \langle pc'_i, request'_i, answer'_i \rangle) \vee$$

$$(\exists i \in \{1, 2\} : External(i, \langle request_i, answer_i, rbuffer, sbuffer \rangle,$$

$$\langle request'_i, answer'_i, rbuffer', sbuffer' \rangle) \wedge$$

$$pc = pc' \wedge \langle sender, waiting, given \rangle = \langle sender', waiting', given' \rangle)$$

The Verification Task



$$\langle I, R \rangle \models \Box \neg (pc_1 = C \wedge pc_2 = C)$$

$$Invariant(pc, request, answer, sender, given, waiting, rbuffer, sbuffer) :\Leftrightarrow$$

$$\forall i \in \{1, 2\} :$$

$$(pc(i) = C \vee sbuffer(i) = 1 \vee answer(i) = 1 \Rightarrow$$

$$given = i \wedge$$

$$\forall j : j \neq i \Rightarrow pc(j) \neq C \wedge sbuffer(j) = 0 \wedge answer(j) = 0) \wedge$$

$$(pc(i) = R \Rightarrow$$

$$sbuffer(i) = 0 \wedge answer(i) = 0 \wedge$$

$$(i = given \Leftrightarrow request(i) = 1 \vee rbuffer(i) = 1 \vee sender = i) \wedge$$

$$(request(i) = 0 \vee rbuffer(i) = 0)) \wedge$$

$$(pc(i) = S \Rightarrow$$

$$(sbuffer(i) = 1 \vee answer(i) = 1 \Rightarrow$$

$$request(i) = 0 \wedge rbuffer(i) = 0 \wedge sender \neq i) \wedge$$

$$(i \neq given \Rightarrow$$

$$request(i) = 0 \vee rbuffer(i) = 0)) \wedge$$

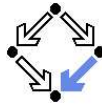
$$(pc(i) = C \Rightarrow$$

$$request(i) = 0 \wedge rbuffer(i) = 0 \wedge sender \neq i \wedge$$

$$sbuffer(i) = 0 \wedge answer(i) = 0) \wedge$$

...

The Verification Task (Contd)



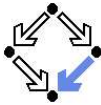
```

...
(sender = 0 ∧ (request(i) = 1 ∨ rbuffer(i) = 1) ⇒
  sbuffer(i) = 0 ∧ answer(i) = 0) ∧
(sender = i ⇒
  (waiting ≠ i) ∧
  (sender = given ∧ pc(i) = R ⇒
    request(i) = 0 ∧ rbuffer(i) = 0) ∧
  (pc(i) = S ∧ i ≠ given ⇒
    request(i) = 0 ∧ rbuffer(i) = 0) ∧
  (pc(i) = S ∧ i = given ⇒
    request(i) = 0 ∨ rbuffer(i) = 0)) ∧
(waiting = i ⇒
  given ≠ i ∧ pc_i = S ∧ request_i = 0 ∧ rbuffer(i) = 0 ∧
  sbuffer_i = 0 ∧ answer(i) = 0) ∧
(sbuffer(i) = 1 ⇒
  answer(i) = 0 ∧ request(i) = 0 ∧ rbuffer(i) = 0)

```

As usual, the invariant has been elaborated in the course of the proof.

The Verification Task in PVS



```

clientServer: THEORY
BEGIN

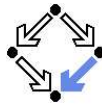
% client indices and program counter constants
Index : TYPE+ = { x: nat | x = 1 OR x = 2 } CONTAINING 1
Index0: TYPE+ = { x: nat | x < 3 } CONTAINING 0
PC: TYPE+ = { R, S, C }

% client states
pc, pc0: [ Index -> PC ]
request, request0: [ Index -> bool ]
answer, answer0: [ Index -> bool ]

% server states
given, given0: Index0
waiting, waiting0: Index0
sender, sender0: Index0
rbuffer, rbuffer0: [ Index -> bool ]
sbuffer, sbuffer0: [ Index -> bool ]

```

The Verification Task in PVS (Contd)



```

i, j: VAR Index

% -----
% initial state condition
% -----

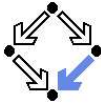
IC(pc: PC, request: bool, answer: bool): bool =
  pc = R AND request = FALSE AND answer = FALSE

IS(given: Index0, waiting: Index0, sender: Index0,
  rbuffer: [ Index -> bool ], sbuffer: [ Index -> bool ]): bool =
  given = 0 AND waiting = 0 AND sender = 0 AND
  (FORALL i: rbuffer(i) = FALSE AND sbuffer(i) = FALSE)

Initial: bool =
  (FORALL i: IC(pc(i), request(i), answer(i))) AND
  IS(given, waiting, sender, rbuffer, sbuffer)

```

The Verification Task in PVS (Contd'2)



```

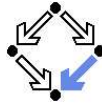
% -----
% transition relation
% -----

RC(pc: PC, request: bool, answer: bool,
  pc0: PC, request0: bool, answer0: bool): bool =
  (pc = R AND request = FALSE AND
   pc0 = S AND request0 = TRUE AND answer0 = answer) OR
  (pc = S AND answer = TRUE AND
   pc0 = C AND request0 = request AND answer0 = FALSE) OR
  (pc = C AND request = FALSE AND
   pc0 = R AND request0 = TRUE AND answer0 = answer)

RS(given: Index0, waiting: Index0, sender: Index0,
  rbuffer: [ Index -> bool ], sbuffer: [ Index -> bool ],
  given0: Index0, waiting0: Index0, sender0: Index0,
  rbuffer0: [ Index -> bool ], sbuffer0: [ Index -> bool ]): bool =
  (EXISTS i:
    sender = 0 AND rbuffer(i) = TRUE AND
    sender0 = i AND rbuffer0(i) = FALSE AND
    given = given0 AND waiting = waiting0 AND sbuffer = sbuffer0 AND
    (FORALL j: j /= i => rbuffer(j) = rbuffer0(j)) OR

```

The Verification Task in PVS (Contd'3)



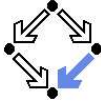
```
(sender /= 0 AND sender = given AND waiting = 0 AND
  given0 = 0 AND sender0 = 0 AND
  waiting = waiting0 AND rbuffer = rbuffer0 AND sbuffer = sbuffer0) OR

(sender /= 0 AND
  sender = given AND waiting /= 0 AND
  sbuffer(waiting) = FALSE AND % change order for type-checking
  given0 = waiting AND waiting0 = 0 AND
  sbuffer0(waiting) = TRUE AND sender0 = 0 AND
  rbuffer = rbuffer0 AND
  (FORALL j: j /= waiting => sbuffer(j) = sbuffer0(j))) OR

(sender /= 0 AND sbuffer(sender) = FALSE AND
  sender /= given AND given = 0 AND
  given0 = sender AND
  sbuffer0(sender) = TRUE AND sender0 = 0 AND
  waiting = waiting0 AND rbuffer = rbuffer0 AND
  (FORALL j: j /= sender => sbuffer(j) = sbuffer0(j))) OR

(sender /= 0 AND sender /= given AND given /= 0 AND
  waiting0 = sender AND sender0 = 0 AND
  given = given0 AND rbuffer = rbuffer0 AND sbuffer = sbuffer0)
```

The Verification Task in PVS (Contd'4)

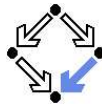


```
External(i: Index,
  pc: PC, request: bool, answer: bool,
  pc0: PC, request0: bool, answer0: bool,
  given: Index0, waiting: Index0, sender: Index0,
  rbuffer: [ Index -> bool ], sbuffer: [ Index -> bool ],
  given0: Index0, waiting0: Index0, sender0: Index0,
  rbuffer0: [ Index -> bool ], sbuffer0: [ Index -> bool ]): bool =

(request = TRUE AND
  pc0 = pc AND request0 = FALSE AND answer0 = answer AND
  rbuffer0(i) = TRUE AND
  given = given0 AND waiting = waiting0 AND sender = sender0 AND
  sbuffer = sbuffer0 AND
  (FORALL j: j /= i => rbuffer(j) = rbuffer0(j))) OR

(pc0 = pc AND request0 = request AND answer0 = TRUE AND
  sbuffer(i) = TRUE AND sbuffer0(i) = FALSE AND
  given = given0 AND waiting = waiting0 AND sender = sender0 AND
  rbuffer = rbuffer0 AND
  (FORALL j: j /= i => sbuffer(j) = sbuffer0(j)))
```

The Verification Task in PVS (Contd'5)

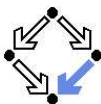


```
Next: bool =
  ((EXISTS i: RC(pc (i), request (i), answer (i),
    pc0(i), request0(i), answer0(i)) AND
    (FORALL j: j /= i =>
      pc(j) = pc0(j) AND request(j) = request0(j) AND
      answer(j) = answer0(j))) AND
    given = given0 AND waiting = waiting0 AND sender = sender0 AND
    rbuffer = rbuffer0 AND sbuffer = sbuffer0) OR

  (RS(given, waiting, sender, rbuffer, sbuffer,
    given0, waiting0, sender0, rbuffer0, sbuffer0) AND
  (FORALL j: pc(j) = pc0(j) AND request(j) = request0(j) AND
    answer(j) = answer0(j))) OR

  (EXISTS i:
    External(i, pc (i), request (i), answer (i),
      pc0(i), request0(i), answer0(i),
      given, waiting, sender, rbuffer, sbuffer,
      given0, waiting0, sender0, rbuffer0, sbuffer0) AND
    (FORALL j: j /= i =>
      pc(j) = pc0(j) AND request(j) = request0(j) AND
      answer(j) = answer0(j)))
```

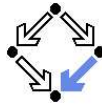
The Verification Task in PVS (Contd'6)



```
% -----
% invariant
% -----
Invariant(pc: [Index->PC], request: [Index -> bool],
  answer: [Index -> bool],
  given: Index0, waiting: Index0, sender: Index0,
  rbuffer: [Index -> bool], sbuffer: [Index->bool]): bool =

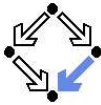
FORALL i:
  (pc(i) = C OR sbuffer(i) = TRUE OR answer(i) = TRUE =>
    given = i AND
    FORALL j: j /= i =>
      pc(j) /= C AND
      sbuffer(j) = FALSE AND answer(j) = FALSE) AND
  (pc(i) = R =>
    sbuffer(i) = FALSE AND answer(i) = FALSE AND
    (i /= given =>
      request(i) = FALSE AND rbuffer(i) = FALSE AND sender /= i)
    (i = given =>
      request(i) = TRUE OR rbuffer(i) = TRUE OR sender = i) AND
    (request(i) = FALSE OR rbuffer(i) = FALSE)) AND
```

The Verification Task in PVS (Contd'7)



```
(pc(i) = S =>
  (sbuffer(i) = TRUE OR answer(i) = TRUE =>
    request(i) = FALSE AND rbuffer(i) = FALSE AND sender /= i) AND
  (i /= given =>
    request(i) = FALSE OR rbuffer(i) = FALSE)) AND
(pc(i) = C =>
  request(i) = FALSE AND rbuffer(i) = FALSE AND sender /= i AND
  sbuffer(i) = FALSE AND answer(i) = FALSE) AND
(sender = 0 AND (request(i) = TRUE OR rbuffer(i) = TRUE) =>
  sbuffer(i) = FALSE AND answer(i) = FALSE) AND
(sender = i =>
  (sender = given AND pc(i) = R =>
    request(i) = FALSE AND rbuffer(i) = FALSE) AND
  (waiting /= i) AND
  (pc(i) = S AND i /= given =>
    request(i) = FALSE AND rbuffer(i) = FALSE) AND
  (pc(i) = S AND i = given =>
    request(i) = FALSE OR rbuffer(i) = FALSE)) AND
```

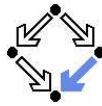
The Verification Task in PVS (Contd'8)



```
(waiting = i =>
  given /= i AND
  pc(waiting) = S AND
  request(waiting) = FALSE AND rbuffer(waiting) = FALSE AND
  sbuffer(waiting) = FALSE AND answer(waiting) = FALSE) AND
(sbuffer(i) = TRUE =>
  answer(i) = FALSE AND request(i) = FALSE AND rbuffer(i) = FALSE)

% -----
% mutual exclusion proof
% -----
MutEx: THEOREM
  Invariant(pc, request, answer,
    given, waiting, sender, rbuffer, sbuffer) =>
  NOT (pc(1) = C AND pc(2) = C)
```

The Verification Task in PVS (Contd'9)

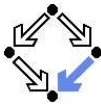


```
% -----
% invariance proof
% -----
Inv1: THEOREM
  Initial =>
  Invariant(pc, request, answer,
    given, waiting, sender, rbuffer, sbuffer)

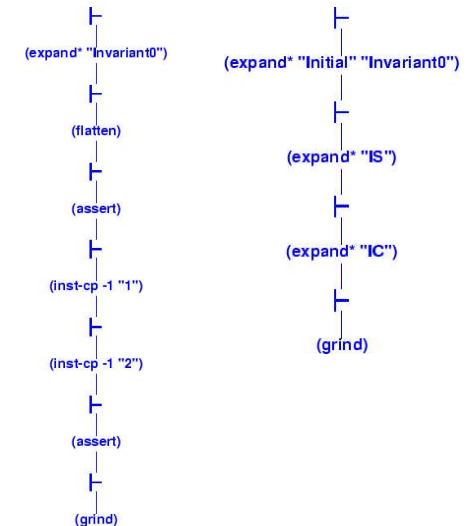
Inv2: THEOREM
  Invariant(pc, request, answer,
    given, waiting, sender, rbuffer, sbuffer) AND Next =>
  Invariant(pc0, request0, answer0,
    given0, waiting0, sender0, rbuffer0, sbuffer0)

END clientServer
```

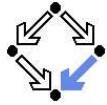
The Proof in PVS



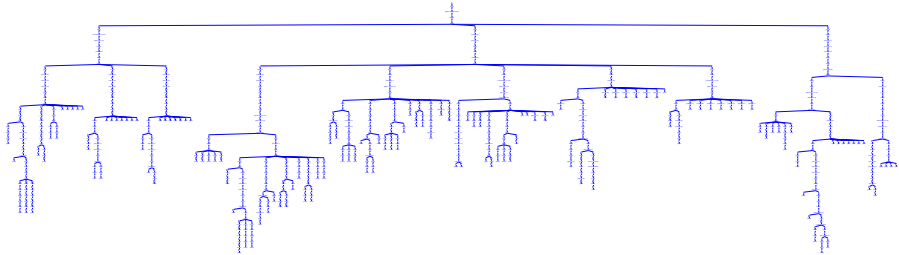
Proofs that the system invariant implies the mutual exclusion property and that the initial condition implies the invariant.



The Proof in PVS



Proof that every system transition preserves the invariant.



- 10 subproofs, one for each transition.
 - Three from client, five from server, two from communication system.
 - Download and investigate from course Web site.

Only with computer support, verification proofs become manageable.