NWPT'00, University of Bergen, Norway
October 11-13, 2000

http://www.ii.uib.no/~nwpt00/

Verification of a Simple Distributed Protocol in Erlang

Gennady Chugunov and Lars-åke Fredlund

Swedish Institute of Computer Science,
Stockholm

"Verification of Erlang Programs"
project funded by ASTEC

collaboration between

- **Ericsson Computer Science Laboratory:**

  (Thomas Arts, Clara Benac Earle)

- **Formal Design Techniques Group of Swedish Institute of Computer Science:**

  (Gennady Chugunov, Mads Dam,
  Lars-Åke Fredlund, Dilian Gurov,
  Thomas Noll (IT/KTH))

Outline of the presentation

1. Verification Framework

2. Erlang Verification Tool

3. Example: **Simple Distributed Protocol in Erlang**

   - Source code of Erlang module
   - Verification Problem
   - What are the challenges?
   - Sketch of a proof of the algorithm

4. Conclusions

Verification Framework

- Code Verification

- Erlang expressions

$$
\begin{array}{rcll}
e & ::= & op(e_1, \ldots, e_n) & n \geq 0 \\
 & | & \mathsf{V} & \\
 & | & \texttt{begin}\ e_1, \ldots, e_n\ \texttt{end} & n > 0 \\
 & | & e(e_1, \ldots, e_n) & n \geq 0 \\
 & | & \texttt{case}\ e\ \texttt{of}\ m\ \texttt{end} & \\
 & | & \texttt{exiting}\ e & \\
 & | & \texttt{try}\ e\ \texttt{catch}\ m\ \texttt{end} & \\
 & | & \texttt{receive}\ m\ [\texttt{after}\ e\ \texttt{->}\ e']\ \texttt{end} & \\
 & | & e_1 ! e_2 & \\
 & & & \\
v & ::= & op(v_1, \ldots, v_n) & n \geq 0 \\
p & ::= & op(p_1, \ldots, p_n) \mid V & n \geq 0 \\
m & ::= & p_1[\texttt{when}\ g_1]\ \texttt{->}\ e_1; \cdots ; p_n[\texttt{when}\ g_1]\ \texttt{->}\ e_n & n > 0 \\
g & ::= & eg_1, \ldots, eg_n & n > 0
\end{array}
$$

- Erlang process

$$< e, Pid, Q >$$

Verification Framework

- Erlang system

$$S =< e_1, Pid_1, Q_1 > ||...|| < e_n, Pid_n, Q_n >$$

- Semantics of Erlang
  - small-step structural operational semantics
  - expression and system level transition relations:
    - $*$ send$_0$ $\dfrac{e_1 \xrightarrow{\alpha} e_1{}'}{e_1!e_2 \xrightarrow{\alpha} e_1{}'!e_2}$
    - $*$ com $\dfrac{s_1 \xrightarrow{pid!s} s_1{}' \quad s_2 \xrightarrow{pid?s} s_2{}'}{s_1 \parallel s_2 \xrightarrow{\tau} s_1{}' \parallel s_2{}'}$

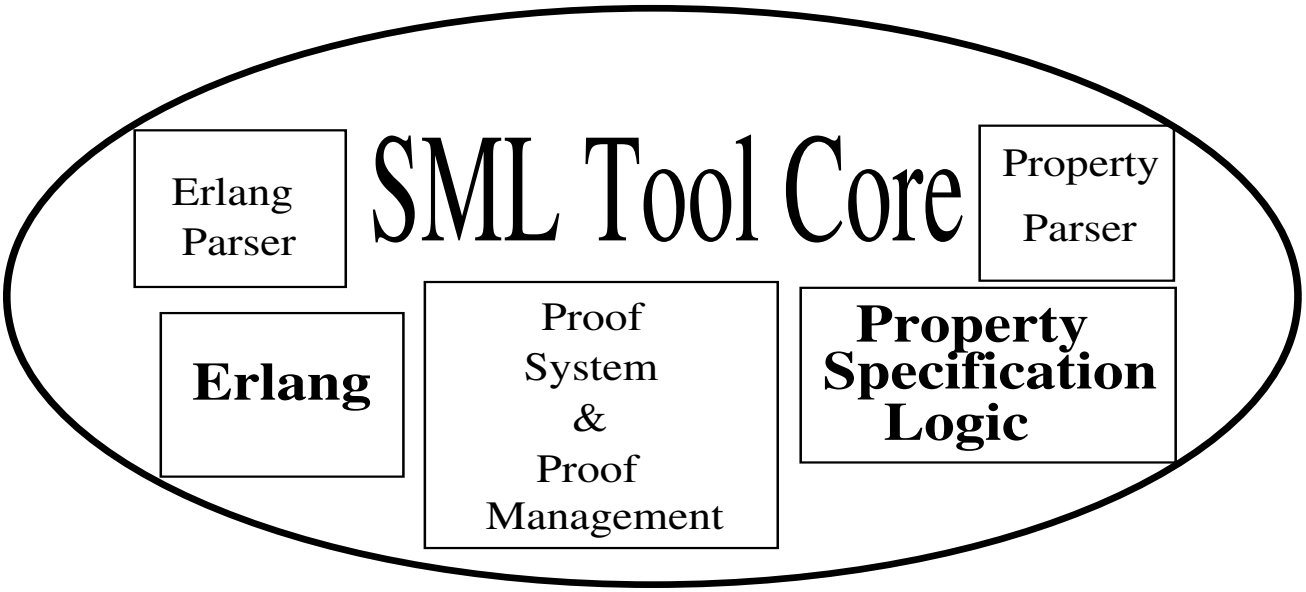
Verification Framework

- Property Specification Language
  - many-sorted first order modal logic with equality and fixpoints

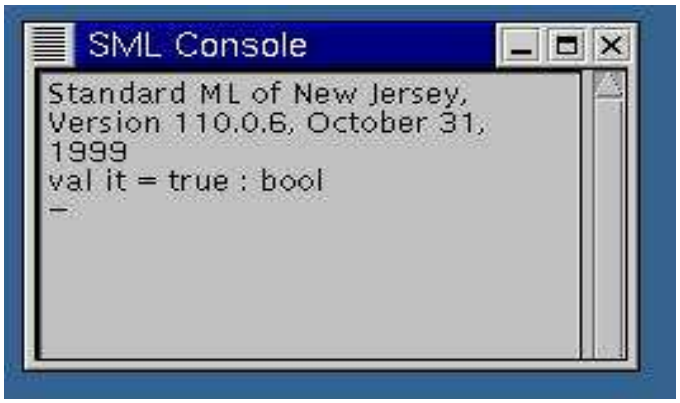
- Proof System
  - Gentzen-style proof system
    Proof judgement is a sequent: $\Gamma \vdash \Delta$,

    Γ, Δ are sets of formulas
  - Compositional reasoning
  - Inductive and Conductive reasoning through fixpoint approximation

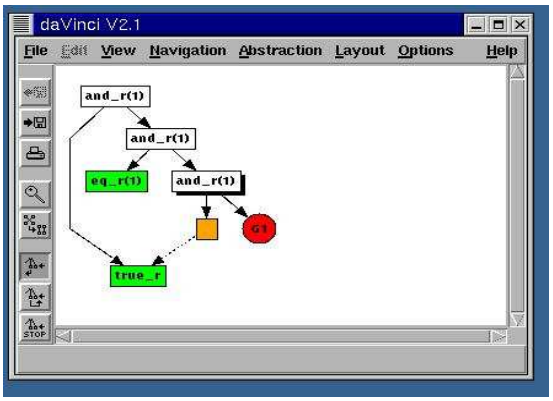
Erlang Verification Tool (EVT)



SML Console



daVinci Graph Visualization System



Java–based User–Interface

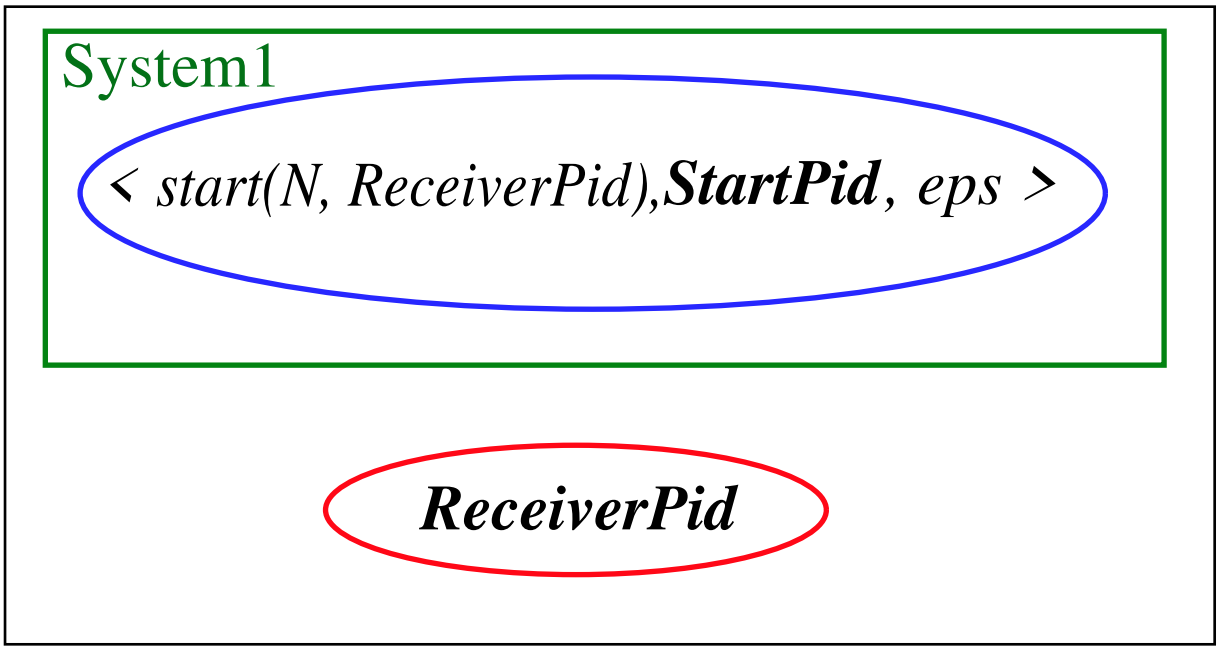
Example: Simple Distributed Protocol

(Source Code of Erlang module)

```
-module(simple_distributed_protocol).
-export([start/2, create_processes/2,
         pass_token/1].


start(N, ReceiverPid) ->
     (create_processes(N, ReceiverPid)) ! token.


create_processes(0, NextPid) ->
      NextPid;
create_processes(N, NextPid) ->
      spawn(simple_distributed_protocol,
            pass_token,
            [create_processes(N-1, NextPid)]).

pass_token(NextPid) ->
        receive
           token ->
               NextPid!token
        end.
```

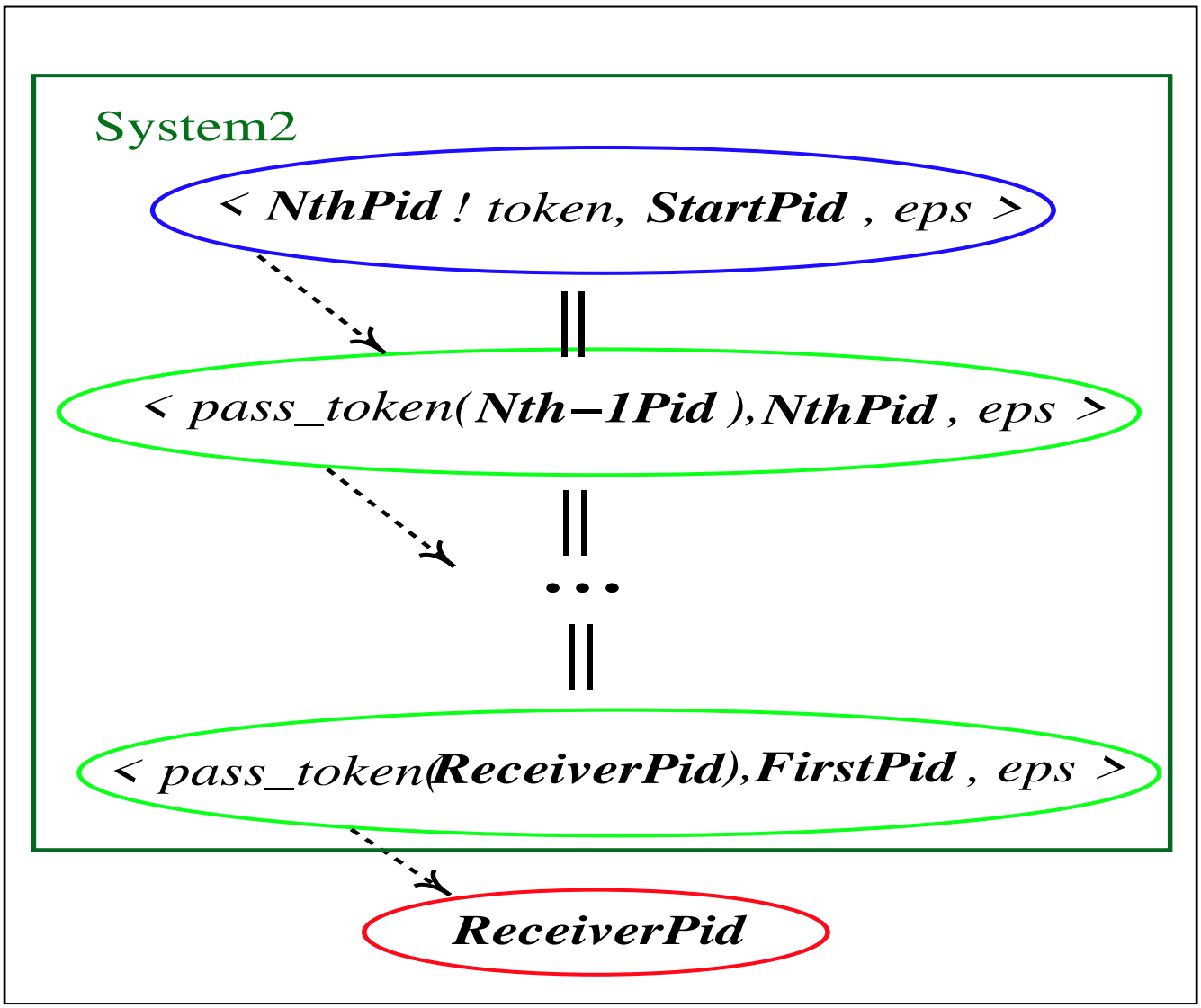
Example: Simple Distributed Protocol

System1 is the initial state of the Erlang system.



Example: Simple Distributed Protocol

System2 is the state of the system after a linked list of processes has been created.



Verification Problem

- Checking Satisfaction Relation between an Erlang system S and a property Φ

$$S : \Phi$$

- What is the property Φ we are going to prove for this Erlang system?
  - informally
    Φ = eventually the $token$ will be sent to the process identified by $ReceiverPid$
  - formally specified in the logic

$$\Phi \Leftarrow$$
$$\lambda OutPid : erlangPid.$$
$$< OutPid!token > tt$$
$$\vee$$
$$< \tau > (\Phi\ OutPid)$$

What are the challenges?

- Example is parametric on the number of processes being spawned

- Dynamically evolving structure of the system (process spawning)

- How to automate the reasoning?

- Specification does not exist

How to solve these problems?

- Compositional reasoning
  + Separate the reasoning between expression and process levels

- Induction on the number of processes

- Automation by means of tactics

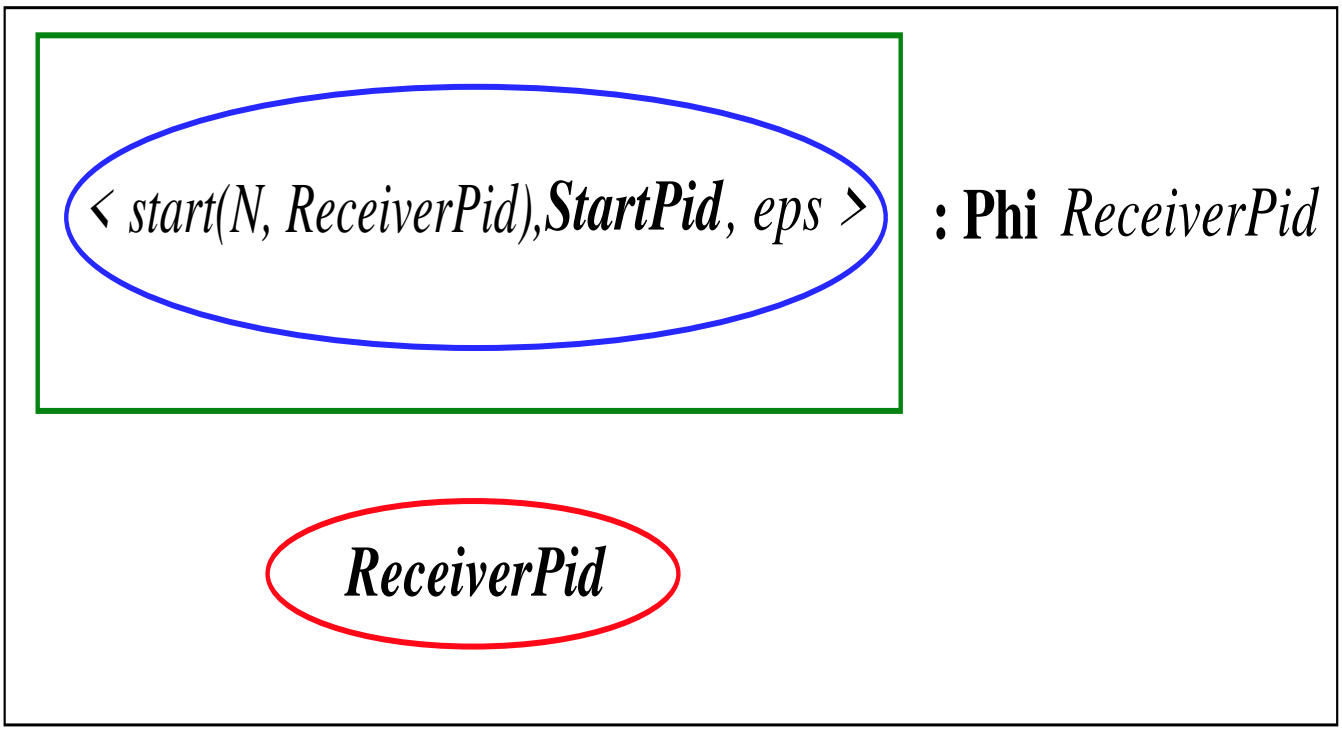
- Code Verification with respect to a set of properties

- Graphical User-Interface for the tool

Sketch of the Main proof

The original proof goal (1):

$declare\ StartPid, ReceiverPid : erlangPid, N : erlangNat\ in$
$\quad not(StartPid = ReceiverPid)$
$\vdash < start(N, ReceiverPid), StartPid, \varepsilon > : \Phi\ ReceiverPid$



After one computation step Goal (2):

$\quad not(StartPid = ReceiverPid)$
$\vdash < (create_processes(N, ReceiverPid))!token, StartPid, \varepsilon >$
$: \Phi\ ReceiverPid$

Sketch of the Main proof (continued)

Compositional reasoning step (Splitting expression and process level reasoning)

$$\text{(ExprCut)}\ \frac{\Gamma \vdash e : \Psi, \Delta \qquad \Gamma, X : \Psi \vdash < r[X], Pid, Q > : \Phi, \Delta}{\Gamma \vdash < r[e], Pid, Q > : \Phi, \Delta}$$

The proof splits into two subgoals: Goal (3), which is proved separately

$$\vdash create_processes(N, ReceiverPid) : \Phi_{cr_pr}\ ReceiverPid$$

and Goal (4):

$$not(StartPid = ReceiverPid),$$
$$X : \Phi_{cr_pr}\ ReceiverPid$$
$$\vdash < X!token, StartPid, \varepsilon > : \Phi\ ReceiverPid$$

Specification of the function create_processes

The source code of the function:

```
create_processes(0, NextPid) ->
      NextPid;
create_processes(N, NextPid) ->
      spawn(simple_distributed_protocol,
            pass_token,
            [create_processes(N-1, NextPid)]).
```

Formal specification in the logic:

$$\Phi_{cr_pr} \Leftarrow$$
$$\lambda OutPid : erlangPid.\ \lambda E : erlangExpression.$$
$$E = OutPid$$
$$\vee$$
$$E :< \tau > (\Phi_{cr_pr}\ OutPid)$$
$$\vee$$
$$\forall Pid : erlangPid.$$
$$E :< spawn(pass_token, [OutPid]) \rightsquigarrow Pid) > (\Phi_{cr_pr}\ Pid)$$

Sketch of the Main proof (continued)

We proceed by case analysis:

1. Goal (5):

$$not(StartPid = ReceiverPid),$$
$$\vdash < ReceiverPid!token, StartPid, \varepsilon > : \Phi\ ReceiverPid$$

The case when X has already been evaluated to a value $ReceiverPid$

2. Goal (6):

$$not(StartPid = ReceiverPid),$$
$$X :< \tau > \Phi_{cr_pr}\ ReceiverPid$$
$$\vdash < X!token, StartPid, \varepsilon > :< \tau > \Phi\ ReceiverPid$$

Proven by induction on the number of τ-steps

3. Goal (7):

$$\Gamma,\ X : < spawn(pass_token, [OutPid]) \rightsquigarrow FirstPid) > (\Phi_{cr_pr}\ FirstPid)$$
$$\vdash < X!token, StartPid, \varepsilon > :< \tau > \Phi\ ReceiverPid$$

where

$$\Gamma = \{not(StartPid = ReceiverPid),$$
$$not(FirstPid = ReceiverPid),$$
$$not(StartPid = FirstPid)\}$$

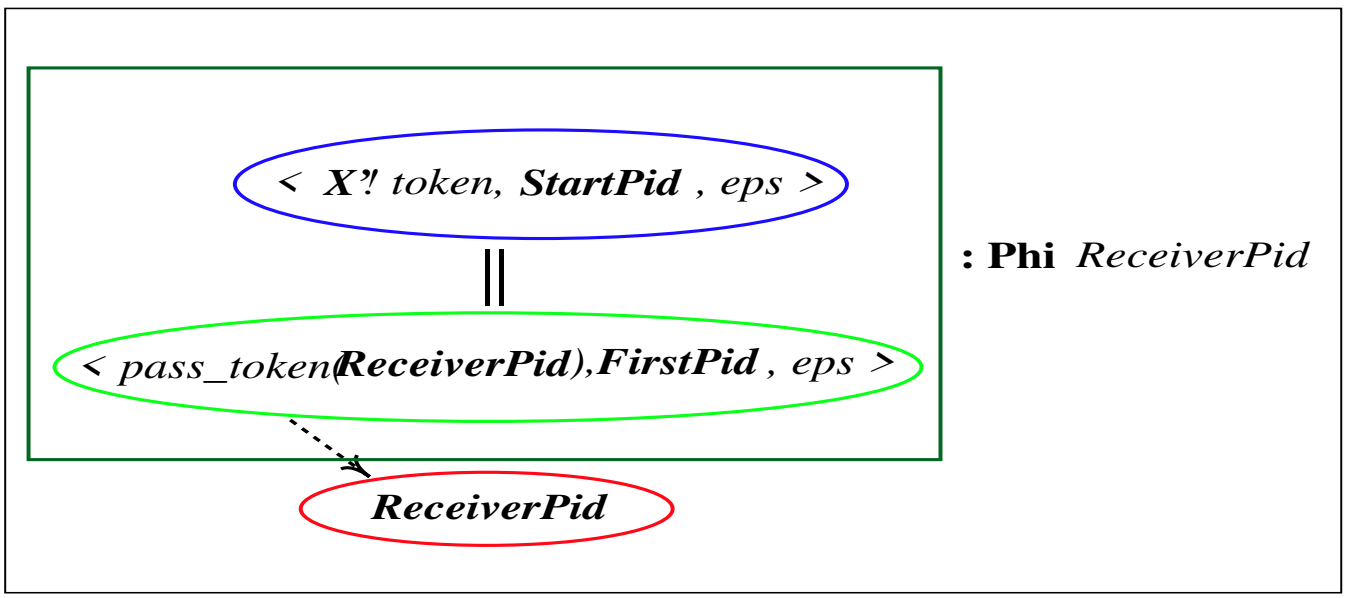
Sketch of the Main proof (continued)

Applying an operational semantics rule (Spawning a new process)

$$() \quad \frac{e \xrightarrow{spawn(m,f,[X_1,...X_n]) \rightsquigarrow Pid'} e'}{< r[e], Pid, Q > \xrightarrow{\tau} < r[e'], Pid, Q > \,||\, < f(X_1,...X_n), Pid', \varepsilon >}$$

Goal (8):

$$\Gamma, X' : \Phi_{cr_pr}\ FirstPid)$$
$$\vdash < X'!token, StartPid, \varepsilon > \ ||$$
$$< pass_token(ReceiverPid), FirstPid, \varepsilon > : \Phi\ ReceiverPid$$



Sketch of the Main proof (continued)

Compositional reasoning step (Splitting process and system level reasoning)

$$\text{(ProcessCut1)}\ \frac{\Gamma \vdash P_1 : \Psi_1, \Delta \qquad \Gamma, X_1 : \Psi_1 \vdash X_1 \mid\mid P_2 : \Phi, \Delta}{\Gamma \vdash P_1 \mid\mid P_2 : \Phi, \Delta}$$

Goal (9):

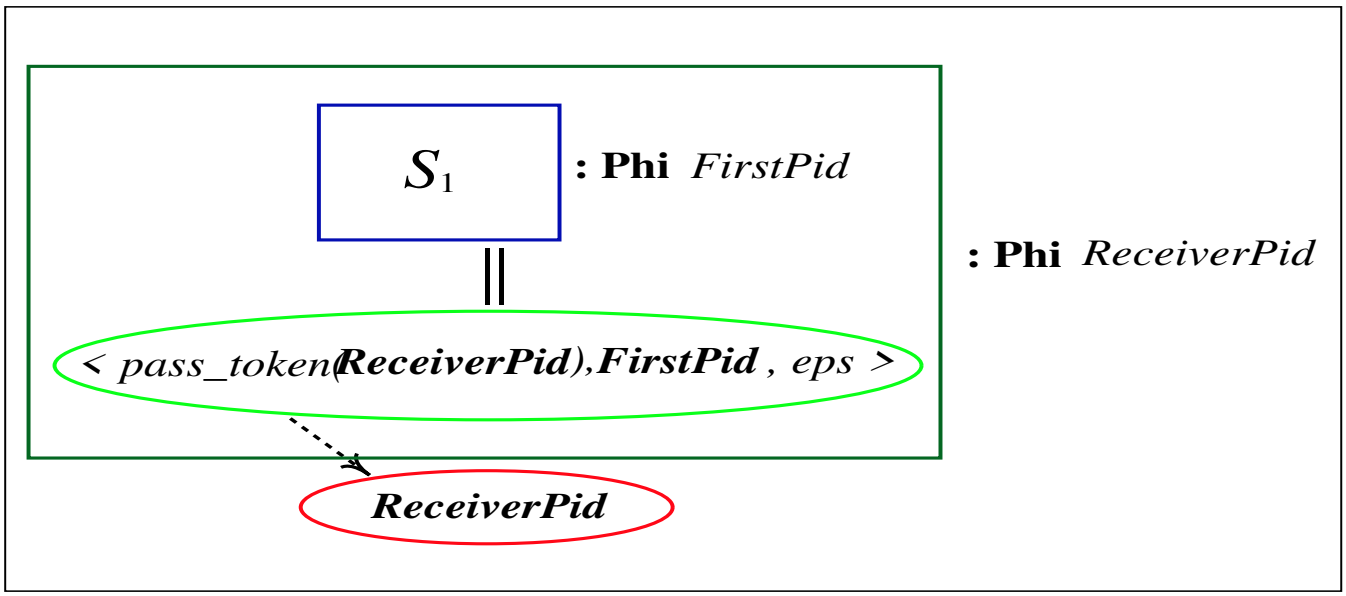
$$\Gamma, X' : \Phi_{cr_pr}\ FirstPid$$
$$\vdash < X'!token, StartPid, \varepsilon >: \Phi\ FirstPid$$

Goal (10):

$$\Gamma, X' : \Phi_{cr_pr}\ FirstPid,$$
$$S_1 : \Phi\ FirstPid$$
$$\vdash\ S_1 \mid\mid < pass_token(ReceiverPid), FirstPid, \varepsilon >$$
$$: \Phi\ ReceiverPid$$

Sketch of the Main proof (continued)

Induction on the number of processes being spawned



Goal (9): discharge against Goal (4) (using induction hypothesis)

$$\Gamma, X' : \Phi_{cr_pr}\ FirstPid$$
$$\vdash < X'!token, StartPid, \varepsilon >: \Phi\ FirstPid$$

Goal (4):

$$not(StartPid = ReceiverPid),$$
$$X : \Phi_{cr_pr}\ ReceiverPid$$
$$\vdash < X!token, StartPid, \varepsilon > : \Phi\ ReceiverPid$$

Sketch of the Main proof (continued)

Compositional reasoning step (Splitting process and system level reasoning)

$$\text{(ProcessCut2)}\ \frac{\Gamma \vdash P_2 : \Psi_2, \Delta \qquad \Gamma, X_2 : \Psi_2 \vdash P_1 \mid\mid X_2 : \Phi, \Delta}{\Gamma \vdash P_1 \mid\mid P_2 : \Phi, \Delta}$$

Goal (10):

$$\Gamma,\ S_1 : \Phi\ FirstPid \vdash\ S_1 \mid\mid < pass_token(ReceiverPid), FirstPid, \varepsilon > : \Phi\ ReceiverPid$$

Specification of the function pass_token

The source code of the function:

```
pass_token(NextPid) ->
        receive
           token ->
               NextPid!token
        end.
```

Formal specification in the logic:

$$\Phi_{pass_token} =$$
$$\lambda InPid : erlangPid.\ \lambda OutPidPid : erlangPid.$$
$$[InPid?token](\Phi\ OutPid) \wedge$$
$$< InPid?token > tt$$

Sketch of the Main proof (continued)

Remaining goals

1. Goal (11):

$$not(FirstPid = ReceiverPid),$$
$$\vdash \ < pass_token(ReceiverPid), FirstPid, \varepsilon > \ :$$
$$\Phi_{pass_token}\ FirstPid\ ReceiverPid$$

2. Goal (12):

$$\Gamma, S_1 : \Phi\ FirstPid,$$
$$S_2 : \Phi_{pass_token}\ FirstPid\ ReceiverPid$$
$$\vdash \ S_1 \ || \ S_2 \ : \Phi\ ReceiverPid$$

3. Goal (3):

$$\vdash create_processes(N, ReceiverPid) : \Phi_{cr_pr}\ ReceiverPid$$

Conclusions

- Within our verification framework we are able to verify temporal properties of Erlang systems with dynamically evolving structure.

- The effort of verification of this example using the Erlang Verification Tool was rather low. (between 1 and 2 hours)

- This example was automated to a very large extent by using tactics and model-checking like scripts. The non-automated parts of the proof consists mostly of setting up the induction hypothesis and decisions when to apply compositional reasoning rules.

- Verifying this protocol we used a recently developed graphical user interface for our proof assistant, that considerably eased the verification task.