CONPASU-tool: A Concurrent Process Analysis Support tool based on Symbolic Computation

Yoshinao Isobe (磯部祥尚)
Information Technology Research Institute AIST, Japan

CPA 2011 (21 June 2011)
CONAPSU is a static analysis tool of concurrent processes.

**Design**
- Structures of concurrent processes, Behaviors of component processes

**Input**
- CSP model (CSP\textsubscript{M} Script)

**Output**
- Analysis-results

**Analysis-results**
- Feedback
- Structure
  - SQ
  - REM
  - SUM
- Behavior

**Formalizing**
- CSP\textsubscript{M} : the machine readable dialect of CSP used in FDR

**Analysis-method**
- Sequentialization
- State-reduction
- Abstraction

**Application**
- Data transferring
- Analysis

**Introduction**
- Motivation
- CONPASU

**Related work**

**Summary**
Introduction

- Motivation
- CONPASU
Motivation

- How can we see behaviors of concurrent processes?

Implementation
- Concurrent process (CSP model)
  - SQ → REM → SUM

Specification
- Sequential process (CSP model)
  - SPEC

Requirements
- Writing: Difficult
- Reading: "Reading" is easier than "writing."

The goal is to develop it

Equality: \(=\): failures-equivalence
CAL: An example of concurrent process

- CAL: a concurrent process which consists of 3 processes with synchronous channels

“in” receives a value N times

“prt” prints each interim result.

“prts” prints the final sum.
The analysis method of CONPASU (outline)

[step 1] A transition graph is generated from a given CSP model (sequentialization).
[step 2] Needless internal-transitions are bypassed (state-reduction).

CAL(N) = (SQREM(N) |_| {{rem,end2}}) SUM(0)
\ {{rem,end2}}
SQREM(n) = (SQ(n) |_| {{sq,end1}}) REM
\ {{sq,end1}}

CAL = (SQREM(N) |_| {{rem,end2}}) SUM(0)
\ {{rem,end2}}
SQREM = (SQ(n) |_| {{sq,end1}}) REM
\ {{sq,end1}}

Structure

Behavior

Sequential process (CSP model)

SEQ(N) = SEQ0(N,0) \ {{tmp}}
SEQ0(n,y) = (n>0) & in?x1 -> SEQ6(n-1,x1*x1%10,y)
\ (n==0) & tmp!0 -> SEQ4(y)
SEQ4(y) = prts!(y) -> SEQ7
SEQ6(n,x3,y) = (n>0) & in?x1 -> SEQ11(n-1,x3,y,x1*x1)
\ (n==0) & tmp!0 -> SEQ9(x3,y,0)
\ prt!(x3) -> SEQ0(n,y+x3)
SEQ7 = STOP
SEQ9(x3,y,z1) = prt!(x3) -> SEQ4(y+x3)
SEQ11(n,x3,y,x2) = (n>0) & in?x1 -> SEQ12(x3,y,x2,n,x1)
\ (n==0) & prs!(y) -> SEQ6(n,x2%10,y+x3)
SEQ12(x3,y,x2,n,x1) = prs!(x3) -> SEQ11(n-1,x2%10,y+x3,x1*x1)

Sequential process (CSP model)

= F : (stable) failures-equivalence,

Graphviz is used for display graphs.
Analysis method

- Sequentialization
- State-reduction
- Abstraction
Sequentialization

- A **symbolic** operational semantics with data-assignments and locations is used.
- Variables are **not instantiated** to values in symbolic semantics.
  - Many values can be folded into a variable in symbolic labeled transition graphs.
  - State-minimization is difficult (often undecidable).

### The CSP model of CAL(N)

- **CAL(N)** = (SQREM(N) [[{rem, end2}]] SUM(0)) ¥ {{rem, end2}}
- SQREM(n) = (SQ(n) [[{sq, end1}]] REM) ¥ {{sq, end1}}

\[
\begin{align*}
\text{SQ}(n) &= ((n>0) \land \text{in?}x1 \rightarrow \text{sq!}(x1-x1) \rightarrow \text{SQ}(n-1)) \\
&\quad (n==0) \land \text{end1!}0 \rightarrow \text{STOP} \\
\text{REM} &= \text{sq?}x2 \rightarrow \text{rem!}(x2\%10) \rightarrow \text{REM} \\
&\quad \text{end1?}z1 \rightarrow \text{end2!}z1 \rightarrow \text{STOP} \\
\text{SUM}(y) &= \text{rem?}x3 \rightarrow \text{prt!}x3 \rightarrow \text{SUM}(y+x3) \\
&\quad \text{end2?}z2 \rightarrow \text{prts!}y \rightarrow \text{STOP}
\end{align*}
\]

**Symbolic** operational semantics

- \( \text{prt!}x3 / y:=y+x3 @ (01) \)
- \( \tau / x3:=x2\%10 @ ((01)1) \)

**Locations**

- @((10)0)
- @((01)0)
- @(01)

**τ**: Internal event
### State-reduction (internal-choice)

- **Needless internal transitions** are bypassed with preserving the **failures-equivalence** $=_{F}$.

  e.g. A removable state with **non-deterministic internal transitions**.
  (in fact, it is more complex because conditions and assignments are considered)

---

$$=_{F} : (stable) failures-equivalence$$
Needless internal transitions are bypassed with preserving failures-equivalence \( =_F \).

e.g. Removable states with interleaving.

In CONPASU, locations are used for checking the independency.
State-reduction (an example)

- The removable states in the transition graph of \( \text{CAL}(N) \) and the reduced graph.

By Corollary 2.1 (p.353) in Proceedings of CPA2011
Abstraction

- Analysis by focusing on interesting channels (e.g. in and prts)

ACAL(N) = CAL(N) \ {\{|prt|\}}

CAL(N) = (SQREM(N) \ {\{|rem,end2|\}} \ SUM(0)) \ {\{|rem,end2|\}}

SQREM(n) = (SQ(n) \ {\{|sq,end1|\}} \ REM) \ {\{|sq,end1|\}}

SQ(n) = ((n>0) & in?x1 -> sq!(x1*x1) -> SQ(n-1))
[] ((n==0) & end1!0 -> STOP)

REM = sq?x2 -> rem!(x2%10) -> REM
[] end1?z1 -> end2!z1 -> STOP

SUM(y) = rem?x3 -> prt!x3 -> SUM(y+x3)
[] end2?z2 -> prts!y -> STOP

Sequentialization, State-reduction

Hiding interim result on prt

3 states (12→3)
3 transitions (17→3)

17:10
Application

- Data-sequence transfer
- Analysis
The CSP model of TransferSys

- **TransferSys** is a concurrent process that consists of 3 processes: UI, Sender, and Receiver.
- **Sender** transfers data-sequences from UI to Receiver (it can be cancelled).

### Structure

- **TransferSys**
  
  ```
  TransferSys = (UI[{{input, quit0, succ, ok, ng}}] Transfer)
  \ {{input, quit0, succ, ok, ng}}
  
  Transfer = (Sender[{{start, net, term, quit1, ack}}] Receiver)
  \ {{start, net, term, quit1, ack}}
  ```

- **UI**
  
  ```
  UI = upload?ds -> input!ds -> (ok?a -> Wait [] ng?a -> UI)
  Wait = cancel?b -> quit0!0 -> UI [] succ?u -> complete!0 -> UI
  ```

- **Sender**
  
  ```
  Sender = input?ds0 -> Check(ds0)
  Check(ds0) = ((#ds0>0) & ok!0 -> start!0 -> Sending(ds0))
  \ ((not #ds0>0) & ng!0 -> Sender)
  Sending(ds0) = ((#ds0>0) & net!(head(ds0)) -> Sending(tail(ds0))]
  \ ((not #ds0>0) & term!0 -> Term)
  \ (quit0?x -> quit1!0 -> Sender)
  Term = ack?z -> succ!0 -> Sender
  ```

- **Receiver**
  
  ```
  Receiver = start?y -> Receiving(<>)
  Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
  \ (term?y -> output!ds1 -> ack!0 -> Receiver)
  \ (quit1?y -> Receiver)
  ```
The behaviors of the 3 components

- Sender **synchronously communicates** with UI or Receiver.

→ How does their composition behave?

Sender synchronously communicates with UI or Receiver.
The behavior of TransferSys

- The symbolic labeled transition graph generated by CONPASU from TransferSys

TransferSys \ \{|complete \} \n
- 8 states \(18 \rightarrow 8\)
- 14 transitions \(27 \rightarrow 14\)

\[ \text{ds0} : \text{the sequence-variable in Sender} \]
\[ \text{ds1} : \text{the sequence-variable in Receiver} \]
A revision of Sender

A transition is added in Sender for receiving the cancel signal after transfer completion.

Sender

SND

SND0

input?ds0
ngl0 [not#ds0>0]

SND1(ds0)

okl0 [#ds0>0]

SND2(ds0)

start0

SND3(ds0)

nethead(ds0)[ds0>0]/ds0:=tail(ds0)

term10 [not#ds0>0]

SND4

SND5

SND6

Sending

Competed

Cancel

Revision

SND

SND0

input?ds0
ngl0 [not#ds0>0]

SND1(ds0)

okl0 [#ds0>0]

SND2(ds0)

start0

SND3(ds0)

nethead(ds0)[ds0>0]/ds0:=tail(ds0)

term10 [not#ds0>0]

SND4

SND5

SND6

(A revised version)

It can receive the cancel signal after the completion.

Added

quit0?x
TransferSys = (UI \{|input, quit0, succ, ok, ng|\}) Transfer
\{|input, quit0, succ, ok, ng|\}
Transfer = (Sender \{|start,net,term,quit1,ack|\}) Receiver
\{|start,net,term,quit1,ack|\}
UI = upload?ds -> input!ds -> (ok?a -> Wait [] ng?a -> UI)
Wait = cancel?b -> quit0!0 -> UI [] succ?u -> complete10 -> UI
Sender = input?ds0 -> Check(ds0)
Check(ds0) = ((#ds0>0) & ok!0 -> start!0 -> Sending(ds0))
[] ((not #ds0>0) & ng!0 -> Sender)
Sending(ds0) = ((#ds0>0) & net!(head(ds0)) -> Sending(tail(ds0)))
[] ((not #ds0>0) & term0 -> Term)
[] (quit0?x -> quit10 -> Sender)
Term = ack7z -> (succ10 -> Sender [] quit07x -> Sender)
Receiver = start?y -> Receiving(<>)
Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
[] (term?y -> output?ds1 -> ack10 -> Receiver)
[] (quit1?y -> Receiver)

The behavior of the revised TransferSys

The transition graph of the revised TransferSys.

Sequentialization, State-reduction

7 states (18 → 8)
14 transitions (27 → 14)
Related works

- PAT
- LTSA
**PAT (Process Analysis Toolkit)**

- **PAT** can display transition graphs of CSP models.
- Standard (non-symbolic) semantics is used.
  (all variables are instantiated to possible values)

The standard transition graph of CAL(3)

<table>
<thead>
<tr>
<th>States</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

\[ \text{cf. CONPASU} \]

**by Simulator**

fix N=3 and finitize input in \{0,1\}

105 states
160 transitions

20:50
LTSA (LTS analyzer)

- LTSA can display minimized transition graphs.
- Standard (non-symbolic) semantics is used. (all variables are instantiated to possible values)

**LTSA (GUI)**

- Any $N$ and any input

The minimized standard transition graph of CAL(3)

- 42 states (102→42)
- 67 transitions (157→67)

by Draw

fix $N=3$ and finitize input in $\{0,1\}$

**cf. CONPASU**

- 8 states
- 12 transitions
Summary

- Advantages
- Future works
A **symbolic analysis method** and its **implementation** CONPASU have been presented.

The **advantages[A]** and **disadvantages[D]** of CONPASU compared with model-checkers:

- [A] Symbolic operational semantics is used (i.e. variables are **not instantiated**),
- [A] An equal sequential process (and the graph) can be **automatically generated**.
- [D] Symbolic labels are usually **more complex** than standard (instantiated) labels.
- [D] Generated sequential processes are **not necessarily optimized** (e.g. not minimized).

→ CONPASU and model checker will **complement** each other.

![Diagram showing symbolic and standard semantics](image)

**Future works:**

- Careful consideration about **livelocks**
- Symbolic computation of **data-expressions** \((1+2 \neq 2+1)\) in the prototype
- Improvement of CONPASU (Java, 6,000 lines) and evaluation of **performance**

CONPASU-website: [http://staff.aist.go.jp/y-isobe/conpasu/](http://staff.aist.go.jp/y-isobe/conpasu/)