Translating FSP into LOTOS and Networks of Automata

Jeff Kramer

Imperial College London

Jeff Magee

Gwen Salaün*

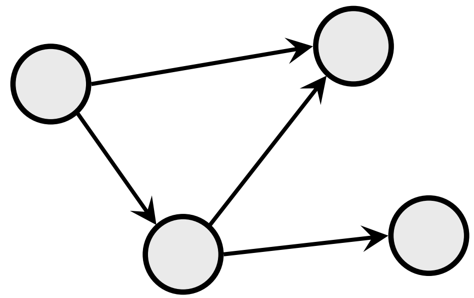
VASY INRIA RHÔNE-ALPES

Frédéric Lang

* New affiliation: Universidad de Málaga

Motivations

- Process algebras are abstract description languages to specify concurrent systems:
  - expressive and textual notations
  - compositional specifications
  - formal verification tools



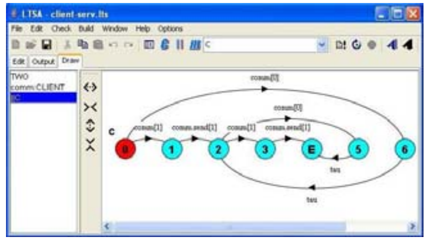
- Fragmentation of the process algebra community

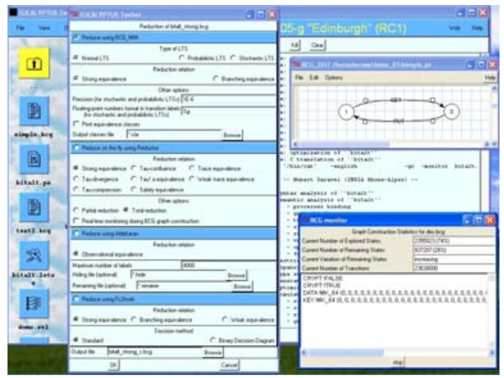
  $\Rightarrow$ languages incompatible in practice

- Our goal:
  - filling the gap between process algebras
  - making the joint use of existing tool-boxes possible

Motivations

- FSP is a popular process algebra
  - + concise, expressive, and easy-to-use notation
  - – basic verification means (LTSA)
    - ⇒ animation and LTL property checking





- LOTOS is an ISO standard
  - + rich verification toolbox CADP
  - – expressive notation, needs expertise
- Translating FSP into LOTOS:
  - – FSP is a simple yet expressive notation
  - – CADP is a rich toolbox to be used jointly with LTSA to analyse FSP specifications

Comparison

Criteria	FSP	LOTOS
Binary communication	Yes	Yes
N-ary communication	Yes	Yes
M among N comm.	No	Yes (E-LOTOS)
Name matching	Yes	Yes
Tools	Yes (-)	Yes (++)
Graphical notations	Yes (++)	Yes (-)
Data	Simple	Complex
Expressiveness	Yes (+)	Yes (+)
Compositionality	No	Yes
User-friendliness	Yes (+)	Yes (-)
Conciseness/readability	Yes (+)	Yes (-)

LOTOS + EXP.OPEN

- High-level translation between process calculi are preferred as often as possible:
  - Translation of behavioural operators easier
  - Mandatory to use some verification tools of CADP
  - Benefit from the Caesar.adt and Caesar compilers

- However, FSP composite processes are difficult to encode into LOTOS:
  - Synchronisations between complex labels
  - Priorities

  $\Rightarrow$ encoding into EXP.OPEN (EXP for short) which allows the description of networks of automata

Outline of the Talk

- FSP, LOTOS, and EXP
- Translating FSP basic processes into LOTOS
- Translating FSP composite processes into EXP
- Prototype and validation
- Conclusion and future work

Finite State Processes (FSP)

- Constants, ranges, sets

```
const C=3        range R=1..C        set S={ash,eat}
```

- Expressive notation to specify labels

```
comm[k:R]    ⇒    comm.1, comm.2, comm.3
order.m[S]   ⇒    order.mash, order.meat
```

- Prefix, choice, if, sequence, hiding, renaming

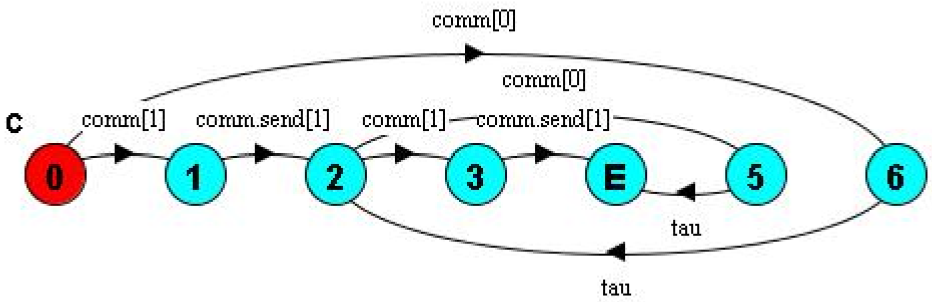
```
SERVER = ( request[id:0..1] -> LOC[id] ),
LOC[id:0..1] = ( when id==0  over -> END |
                 when id!=0  comm.send[id] -> END ).
TWO = SERVER; SERVER; END /{comm/request} \{over}.
```

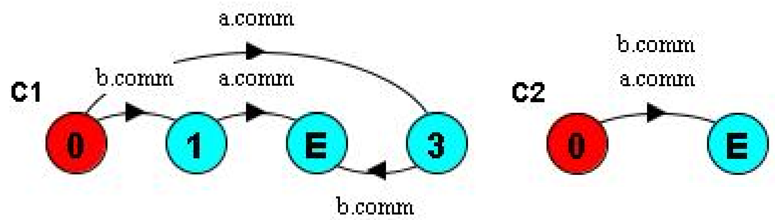
Finite State Processes (FSP)

- Parallel composition $C_1||C_2$ of processes
- Label priority: $>> \{l_1, \ldots, l_n\}$, $<< \{l_1, \ldots, l_n\}$
- Renaming $/\{l_1/l_1', \ldots, l_n/l_n'\}$, hiding $\backslash\{l_1, \ldots, l_n\}$
- Process labelling $\{l_1, \ldots, l_n\}:C$ and sharing $\{l_1, \ldots, l_n\}::C$

```
CLIENT = ( [1] -> send[1] -> CLIENT ).
||SYS = ( TWO || comm:CLIENT ).
```

```
P = (comm->END).
||C1 = {a,b}:P.
||C2 = {a,b}::P.
```





Language of Temporal Ordering Specification (LOTOS)

- Abstract datatypes:

  $\Rightarrow$ sorts, operations, generators, axioms

- Basic LOTOS (only behaviours)

```
aa; exit [] ( bb; comm; exit |[comm]| cc; comm; exit )
```

- Full LOTOS (behaviours + data terms)

```
aa; exit [] ( bb; comm!5; exit
                  |[comm]|
                cc; comm?x:Nat; ( [x>2] -> dd; exit ) )
```

Networks of Automata (EXP.OPEN)

- Parallel composition of automata (bcg format):
  - CCS, CSP, (E)LOTOS, MuCRL compositions, for instance

    label par l_1, …, l_m in B_1 || … || B_n end par

    B_1 ||| … ||| B_n *(interleaving)*
  - Synchronisation vectors

    label par v_1, …, v_m in B_1 || … || B_n end par
- Renaming, hiding, cutting, priority operators

  total rename $l_1 \rightarrow l_1'$, …, $l_n \rightarrow l_n'$ in B end rename

  total hide/cut l_1, …, l_n in B end hide/cut

  total prio l_1, …, l_n > all but l_1, …, l_n in B end prio

  total prio all but l_1, …, l_n > l_1, …, l_n in B end prio

Outline of the Talk

- FSP, LOTOS, and EXP
- Translating FSP basic processes into LOTOS
- Translating FSP composite processes into EXP
- Prototype and validation
- Conclusion and future work

Action Labels

- One FSP label may describe several LOTOS ones

  $\Rightarrow$ expansion of labels to make renaming and hiding possible

- Full expansion when renaming/hiding needed

lab[x:1..2]	$\Rightarrow$	EVENT!CONS(LAB,CONS(1,NIL))
	$\Rightarrow$	EVENT!CONS(LAB,CONS(2,NIL))

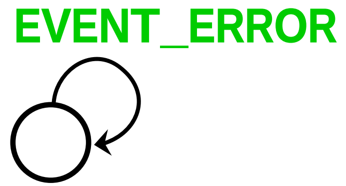
- Compact notation keeping variable otherwise

lab[x:1..2] $\Rightarrow$

choice X:Int[] … EVENT!CONS(LAB,CONS(X,NIL)) [X≥1 and X≤2]

Sequential Processes

- Terminations:
  - END $\Rightarrow$ exit
  - STOP $\Rightarrow$ stop
  - ERROR $\Rightarrow$ P_ERROR [EVENT_ERROR]



- Action prefix l->B $\Rightarrow$ (l_1; exit [] … [] l_n; exit) >> B

  $\rightarrow$ *l_i obtained by expansion, renaming, hiding*

- Choice: when G_1 B_1 | when G_2 B_2

  $\Rightarrow$ [G_1] -> B_1 [] [G_2] -> B_2

- Sequential composition: B_1;B_2 $\Rightarrow$ B_1>>B_2
- if G then B_1 else B_2 $\Rightarrow$ [G] -> B_1 [] [$\neg$G] -> B_2

Example (1)

```
SERVER = ( request[id:0..1] -> LOC[id] ),
LOC[id:0..1] = ( when id==0  over -> END |
                 when id!=0  comm.send[id] -> END ).
TWO = SERVER; SERVER; END /{comm/request} \{over}.
```

```
process SERVER_1 [EVENT] : exit :=
    (  EVENT !CONS (COMM, CONS (POS(0), NIL)) ; LOC_1 [EVENT] (0 of Int)
    [] EVENT !CONS (COMM, CONS (POS(1), NIL)) ; LOC_1 [EVENT] (1 of Int) )
where
    process LOC_1 [E
    [ID==POS(0)] ->
     ( [true] -> i; ex
       []
       [false] -> EVENT !CONS (COMM, CONS (SEND, CONS (POS(0), NIL))); exit )
    [] [ID==POS(1)] -> …
    endproc
 endproc
```

Code never accessed:
optim possible

Example (2)

```
SERVER = ( request[id:0..1] -> LOC[id] ),
LOC[id:0..1] = ( when id==0  over -> END |
                 when id!=0   comm.send[id] -> END ).
TWO = SERVER; SERVER; END /{comm/request} \{over}.
```

```
process SERVER_1 [EVENT] : exit :=
   (  EVENT !CONS (COMM, CONS (POS(0), NIL)) ; LOC_1 [EVENT] (0 of Int)
   [] EVENT !CONS (COMM, CONS (POS(1), NIL)) ; LOC_1 [EVENT] (1 of Int) )
where
   process LOC_1 [EVENT] (ID : Int) : exit :=
   [ID==POS(0)] -> i; exit
   []
   [ID==POS(1)] -> EVENT !CONS (COMM, CONS (SEND, CONS (POS(1), NIL))); exit
   endproc
 endproc
```

Outline of the Talk

- FSP, LOTOS, and EXP
- Translating FSP basic processes into LOTOS
- Translating FSP composite processes into EXP
- Prototype and validation
- Conclusion and future work

Composite Processes

- Process P is translated as "P.bcg" if sequential
- Parallel composition $C_1 || C_2 \Rightarrow$ label par $l_1, \ldots, l_m$ in $C_1 || C_2$ end par with l_i=alph(C_1) $\cap$ alph(C_2)
- Label priority, hiding: total prio, total hide
- Renaming using vectors (1-to-many renaming)

  $/\{l_1/l_1', \ldots, l_n/l_n'\} \Rightarrow$ label par $v_1, \ldots, v_m$ in … end par
- Process labelling and sharing:
  - $\{l_1,\ldots,l_n\}$:C $\Rightarrow$ prefixing with vectors + interleaving
  - $\{l_1,\ldots,l_n\}$::C $\Rightarrow$ prefixing with vectors
- if G then C_1 else $C_2 \Rightarrow$ [G] -> C_1 [] [¬G] -> C_2

Example

```
CLIENT = ( [1] -> send[1] -> CLIENT ).
||SYS = ( TWO || comm:CLIENT ).
```

```
label par "EVENT !CONS (COMM, CONS (POS(1), NIL))",
          "EVENT !CONS (COMM, CONS (SEND, CONS (POS(1), NIL)))" in
   total cut exit in "TWO.bcg" end cut
   ||
   (    label par
             "EVENT !CONS (POS(1), NIL)"
                  -> "EVENT !CONS (COMM, CONS (POS(1), NIL))", … in
        total cut exit in "CLIENT.bcg" end cut
        end par
   )
end par
```

Outline of the Talk

- FSP, LOTOS, and EXP
- Translating FSP basic processes into LOTOS
- Translating FSP composite processes into EXP
- Prototype and validation
- Conclusion and future work

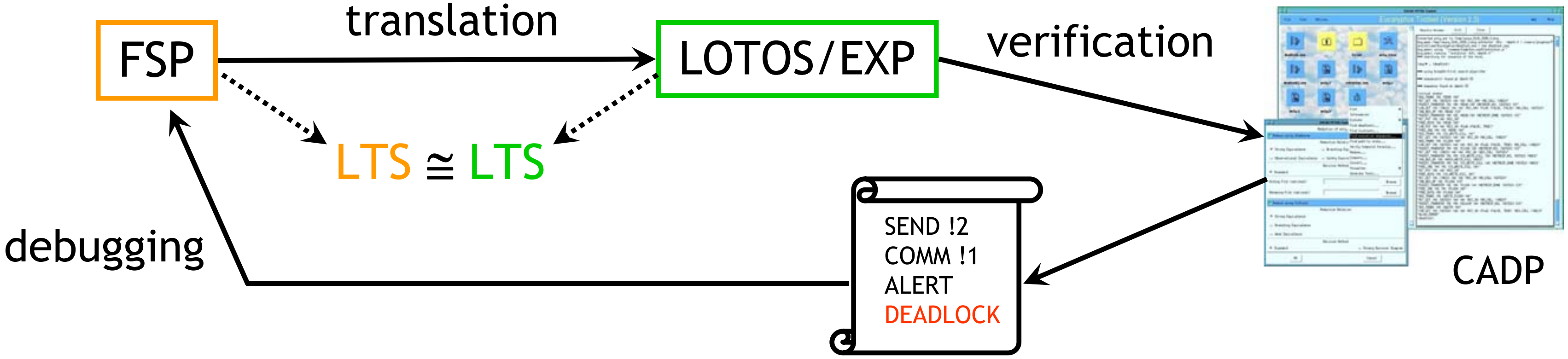
Prototype

- A prototype translator fsp2lotos:
  - total of 25,500 lines of SYNTAX, LOTOS NT, and C
  - validated on 10,500 lines of FSP specifications

    ➡ 72,000 l. LOTOS, 8,000 l. EXP, 2,000 l. SVL
- Translation in two steps:
  - parsing and building an abstract syntax tree
  - translating the tree into semantically equivalent LOTOS code
- In the paper, application to a semaphore example for which CADP is used to analyse FSP specifications

Semantics Preservation

- Essential to ensure that verification on the LOTOS specification is valid on the FSP one



- Conjecture: our translation preserves a branching equivalence relation
- Checked automatically on all the examples with Bisimulator (tool part of CADP)

Outline of the Talk

- FSP, LOTOS, and EXP
- Translating FSP basic processes into LOTOS
- Translating FSP composite processes into EXP
- Prototype and validation
- Conclusion and future work

Conclusion

- Translation from FSP to LOTOS and EXP

  $\Rightarrow$ makes the joint use of LTSA and CADP possible

Future Work

- $LTS_{FSP} \cong LTS_{LOTOS}$: equivalence to be proven
- Application to a complex system, *e.g.*, in web services, where CADP tools would be necessary
- Encoding FSP safety and progress properties into mu-calculus formulas, input format of Evaluator