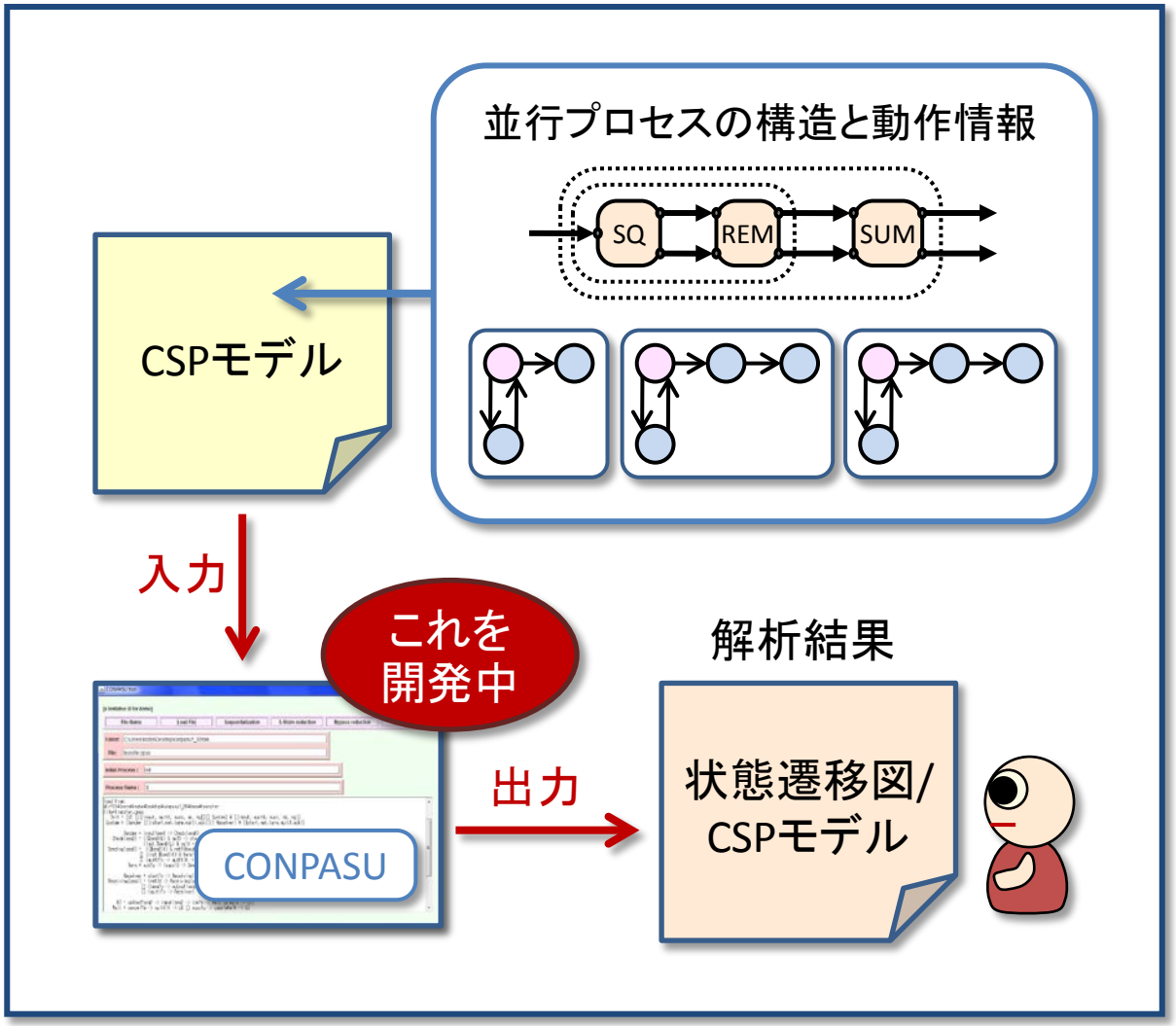
CONPASU (CONcurrent Process Analysis SUpport tool):
CSPモデルの解析支援ツール

産業技術総合研究所
情報技術研究部門
磯部祥尚

第7回CSP研究会 (2011/05/21)

発表内容

- **研究概要**
  - 目的
  - 背景
  - 動機
  - 本研究
  - 比較
- **解析法**
  - 逐次化
  - 状態数削減
  - 抽象化
- **解析例**
  - 概要
  - 解析
- **関連研究とまとめ**
  - 解析ツール
  - まとめ

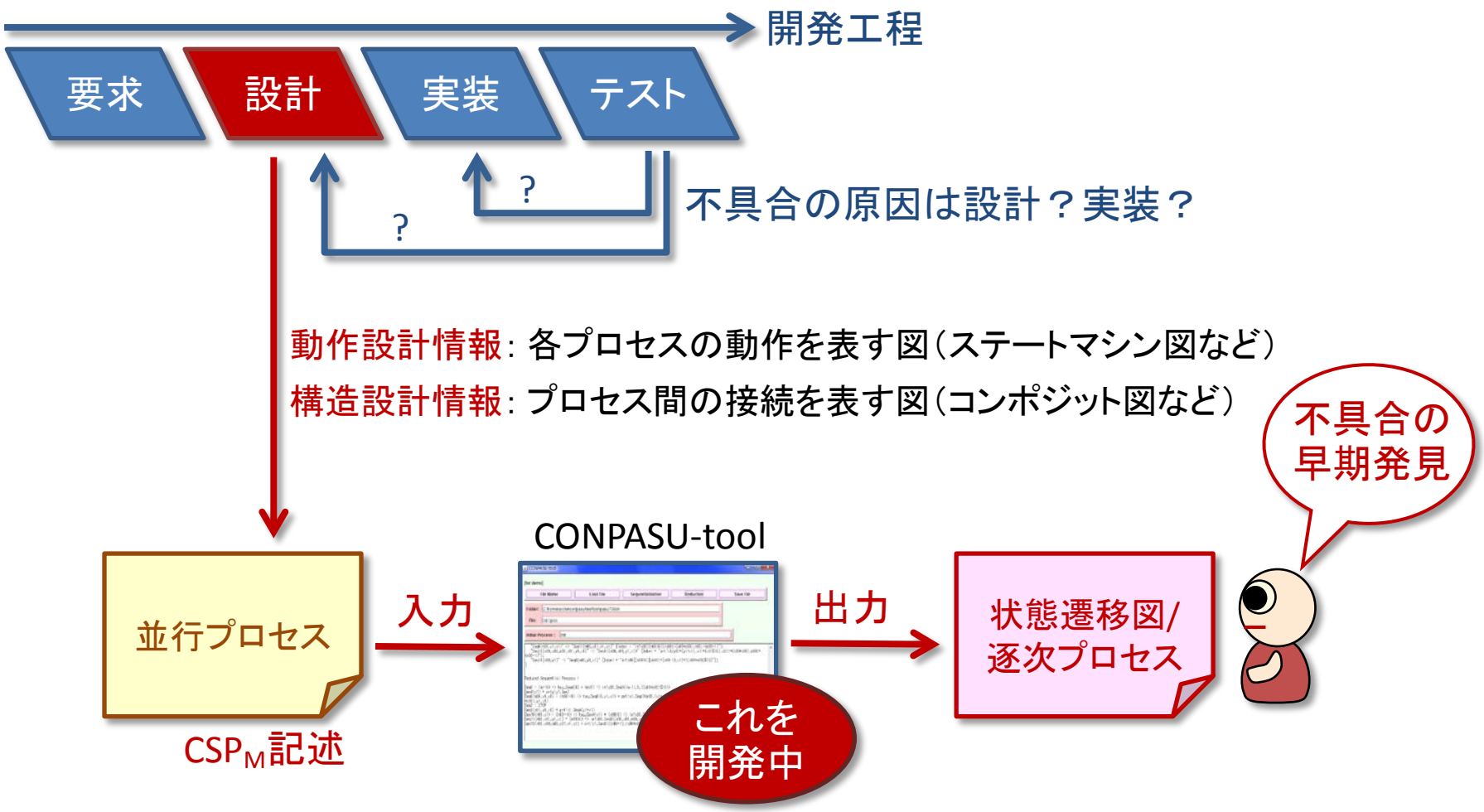


研究概要

- 目的
- 背景
- 動機
- 本研究
- 比較

目的：並行プロセスの設計支援

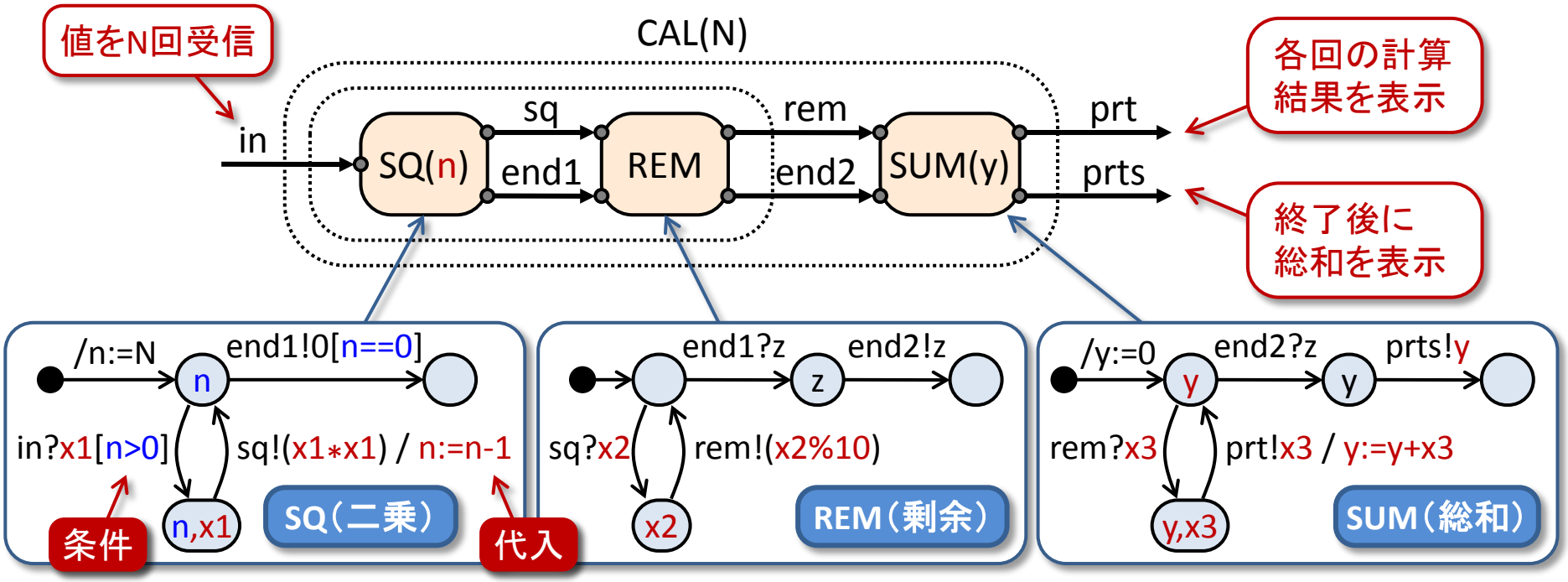
■ 目標： 設計段階で並行プロセスの動作を解析するツール（CONPASU）の開発。



CSPM（Machine Readable CSP）：FDRの記述言語

背景：並行プロセスの例

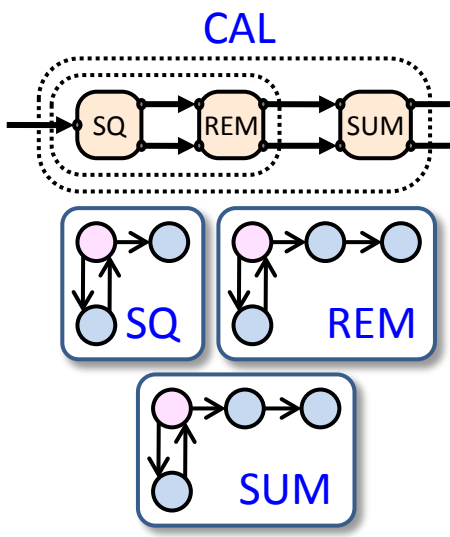
■ CAL(N)：3つのプロセスから構成される並行プロセス（チャネルは同期型通信）



■ 検証：入出力に着目した仕様 SPEC(N) の通りにCAL(N)が動作するか？

背景：プロセス代数（CSP）

■ プロセス代数を使うと、並行プロセスCAL(N)と仕様SPECの等しさを証明できる。

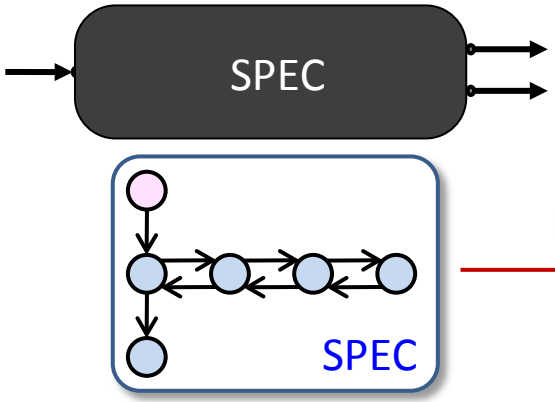


構造

動作

CAL(N): 実装/並行プロセス（CSP_M）

```
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
```



（in,prt,prtsの依存関係）

動作

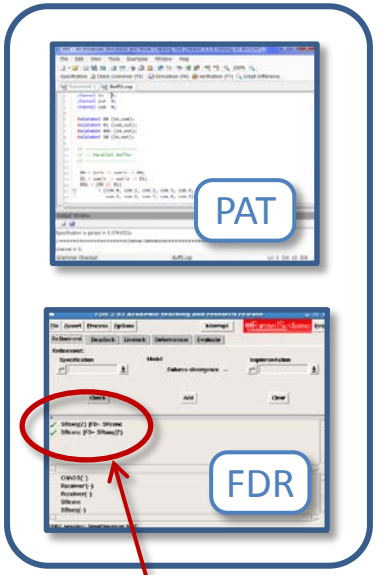
SPEC(N): 仕様/逐次プロセス（CSP_M）

```
SPEC(N) = SPEC0(N,0)
SPEC0(n,y) = (n>0) & in?x -> SPEC1((n-1),y,((x*x)%10))
             [] (n==0) & prts!y -> STOP
SPEC1(n,y,x1) = (n>0) & in?x -> SPEC2((n-1),(x*x),y,x1)
                [] prt!x1 -> SPEC0(n,(y+x1))
SPEC2(n,x0,y,x1) = (n>0) & in?x -> SPEC3(x,n,x0,y,x1)
                   [] prt!x1 -> SPEC1(n,(y+x1),(x0%10))
SPEC3(x,n,x0,y,x1) = prt!x1 -> SPEC2((n-1),(x*x),(y+x1),(x0%10))
```

背景：等価性自動判定（モデル検査器）

■ モデル検査器（FDR, PAT等）：CAL(N)とSPEC(N)の等しさを自動判定できる。

モデル検査器
（完全自動判定）



受信値制限、
反復回数固定

（例 in：{0..5}, N=3）

CAL(N): 実装/並行プロセス（CSP_M）

```
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
```

SPEC(N): 仕様/逐次プロセス（CSP_M）

```
SPEC(N) = SPEC0(N,0)
SPEC0(n,y) = (n>0) & in?x -> SPEC1((n-1),y,((x*x)%10))
             [] (n==0) & prts!y -> STOP
SPEC1(n,y,x1) = (n>0) & in?x -> SPEC2((n-1),(x*x),y,x1)
                [] prt!x1 -> SPEC0(n,(y+x1))
SPEC2(n,x0,y,x1) = (n>0) & in?x -> SPEC3(x,n,x0,y,x1)
                   [] prt!x1 -> SPEC1(n,(y+x1),(x0%10))
SPEC3(x,n,x0,y,x1) = prt!x1 -> SPEC2((n-1),(x*x),(y+x1),(x0%10))
```

✓ CAL(N) $=_F$ SPEC(N)

動機：自動生成

- 動機：並行プロセスの記述から自動的に逐次プロセスの記述を生成できないか？

各プロセスの動作図やその接続図から、比較的機械的に記述できる

CAL(N): 実装/並行プロセス（CSP_M）

```
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
```

CAL(N)からSPEC(N)を自動生成したい

SPEC(N): 仕様/逐次プロセス（CSP_M）

```
SPEC(N) = SPEC0(N,0)
SPEC0(n,y) = (n>0) & in?x -> SPEC1((n-1),y,((x*x)%10))
             [] (n==0) & prts!y -> STOP
SPEC1(n,y,x1) = (n>0) & in?x -> SPEC2((n-1),(x*x),y,x1)
                [] prt!x1 -> SPEC0(n,(y+x1))
SPEC2(n,x0,y,x1) = (n>0) & in?x -> SPEC3(x,n,x0,y,x1)
                   [] prt!x1 -> SPEC1(n,(y+x1),(x0%10))
SPEC3(x,n,x0,y,x1) = prt!x1 -> SPEC2((n-1),(x*x),(y+x1),(x0%10))
```

並列動作を考慮しながら全体の動作を適切に記述することは難しい

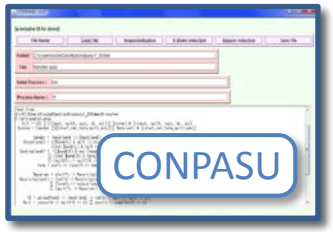
本研究：自動生成（CONPASU）

■ 逐次プロセスを自動生成するツールCONPASUを開発

CAL(N): 実装/並行プロセス（CSP_M）

```
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
```

CONPASU-tool
（完全自動生成）



受信値無制限、
反復回数任意

SEQ(N): 自動生成した逐次プロセス（CSP_M）

```
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)
                   [] prt!(x3) -> SEQ6(n,x2%10,y+x3)
SEQ12(x3,y,x2,n,x1) = prt!(x3) -> SEQ11(n-1,x2%10,y+x3,x1*x1)
```

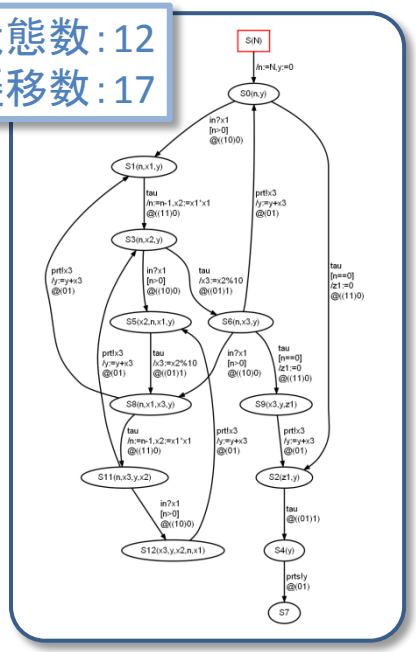
$CAL(N) =_F SEQ(N)$

失敗等価性を保証

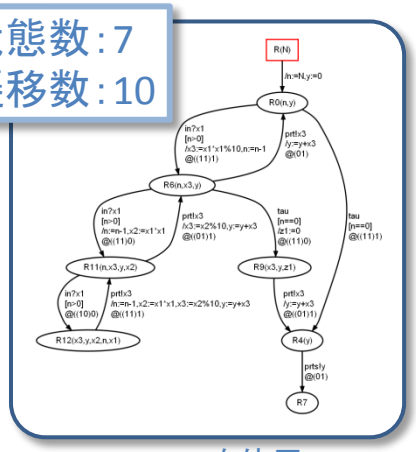
本研究：自動生成（CONPASU）

■ CONAPASUの主機能：①逐次化（状態遷移図作成）、②状態数削減

状態数：12
遷移数：17



状態数：7
遷移数：10



状態数削減

逐次化

（簡単）

CAL(N): 実装/並行プロセス（CSP$_M$）

```
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
```

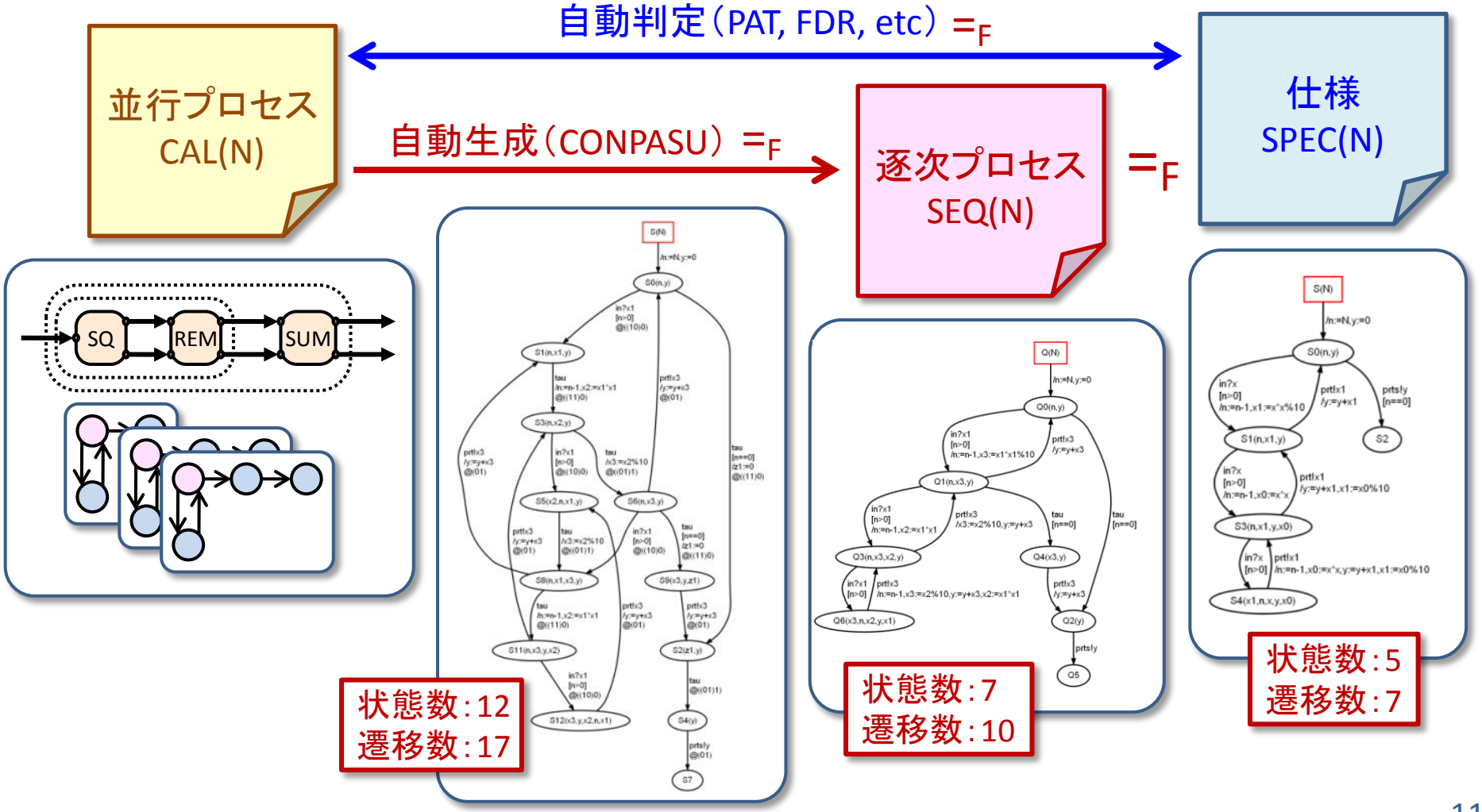
SEQ(N): 自動生成した逐次プロセス（CSP$_M$）

```
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)
            [] prt!(x3) -> SEQ6(n,x2%10,y+x3)
SEQ12(x3,y,x2,n,x1) = prt!(x3) -> SEQ11(n-1,x2%10,y+x3,x1*x1)
```

Graphvizを使用

比較：自動判定と自動生成

- PAT, FDR: CAL(N)とSPEC(N)の等価性を自動判定する。
- CONPASU: CAL(N)から等価なSEQ(N)を自動生成する。



解析法

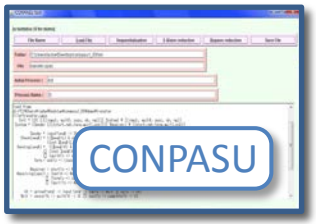
- 逐次化
- 状態数削減
- 抽象化

逐次化：記号的意味論

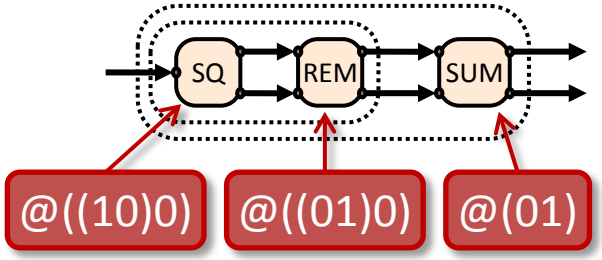
■ 本研究の記号的意味論：代入付記号的意味論[LC99]＋位置情報
（記号的意味論では変数を値で具体化しない）

CAL(N): 実装/並行プロセス（CSP_M）

```
    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
```



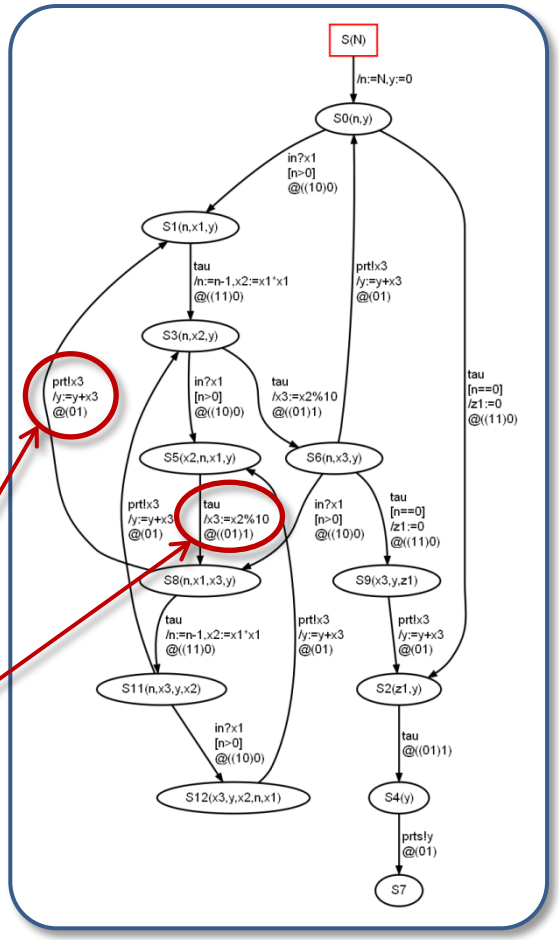
記号的意味論



prt!x3 / y:=y+x3 @ (01)

τ / x3:=x2%10 @ ((01)1)

τ：内部イベント

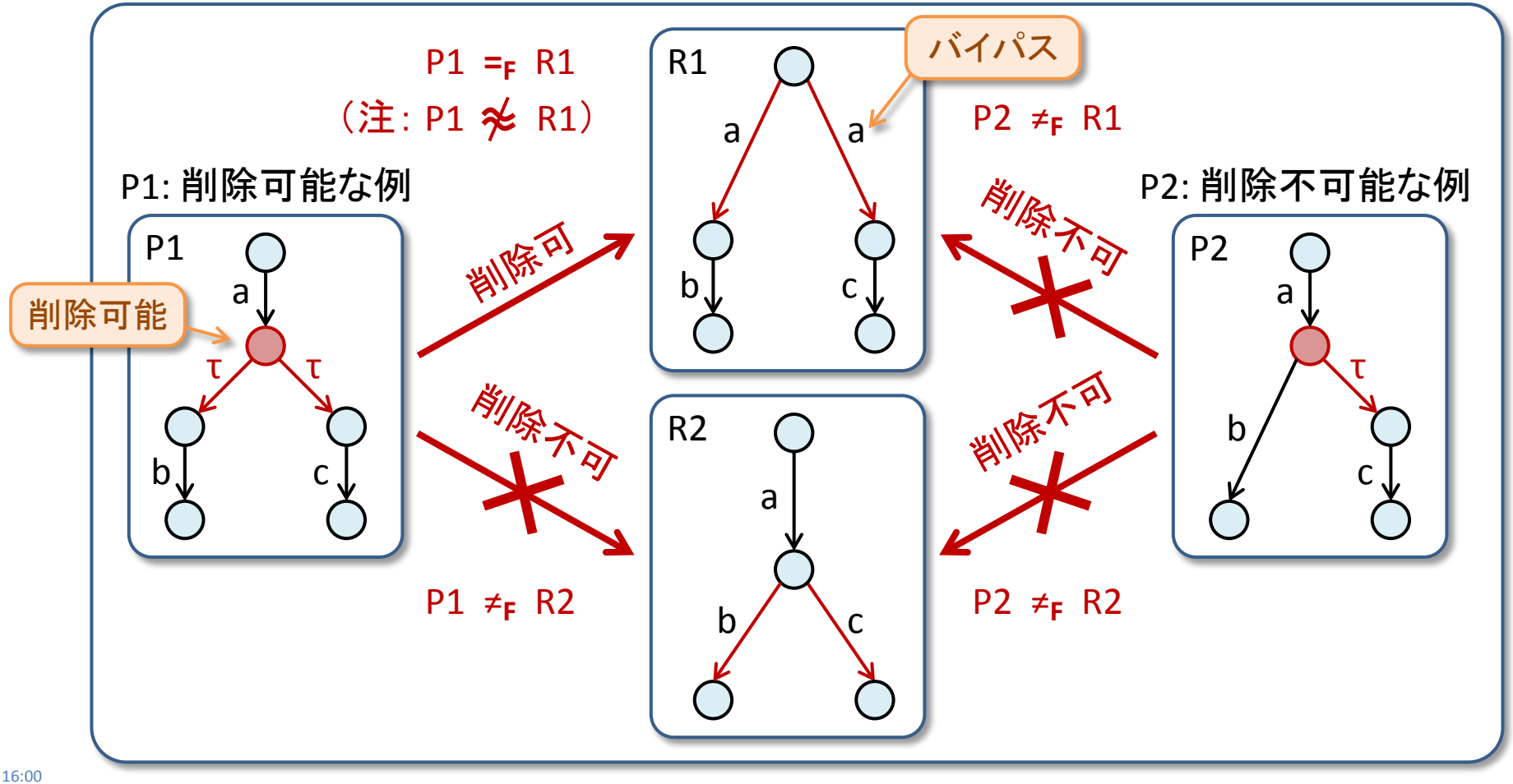


[LC99] Z. Li and H. Chen, Computing Strong/Weak Bisimulation Equivalences and Observation Congruence for Value-Passing Processes, TACAS 99, LNCS 1579, pp.300-314, 1999.

状態数削減:ポイント1(非決定性)

- 失敗等価 $=_F$ になるように、内部遷移に着目して削除可能な状態をバイパスする。
  (弱双模倣等価 $\approx$ になるとはかぎらない)

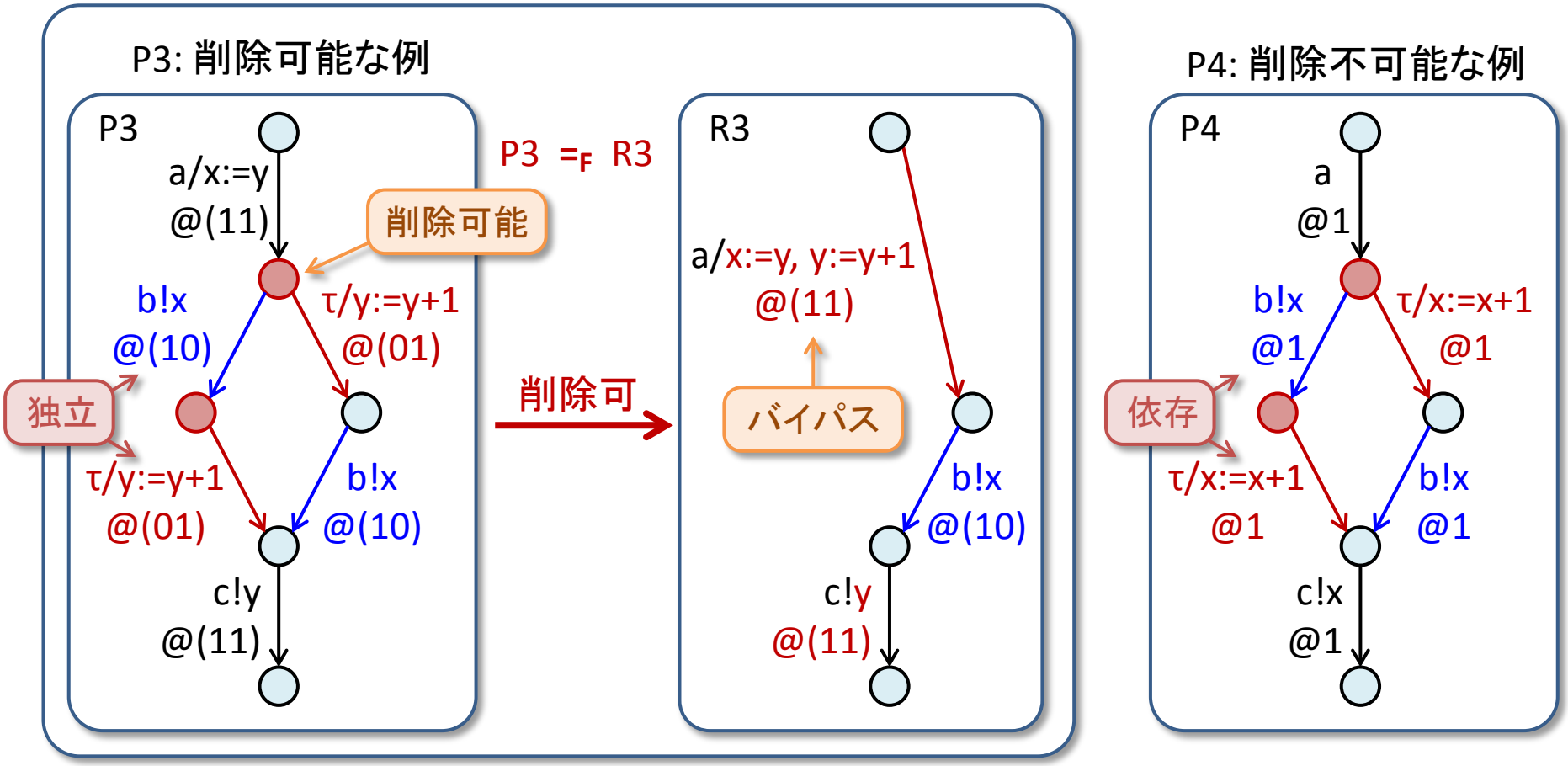
内部遷移をもつ状態(不安定状態)の削除の例 (実際には代入や条件を考慮)



状態数削減：ポイント2（インターリービング）

■ 失敗等価 $=_F$ になるように、内部遷移に着目して削除可能な状態をバイパスする。

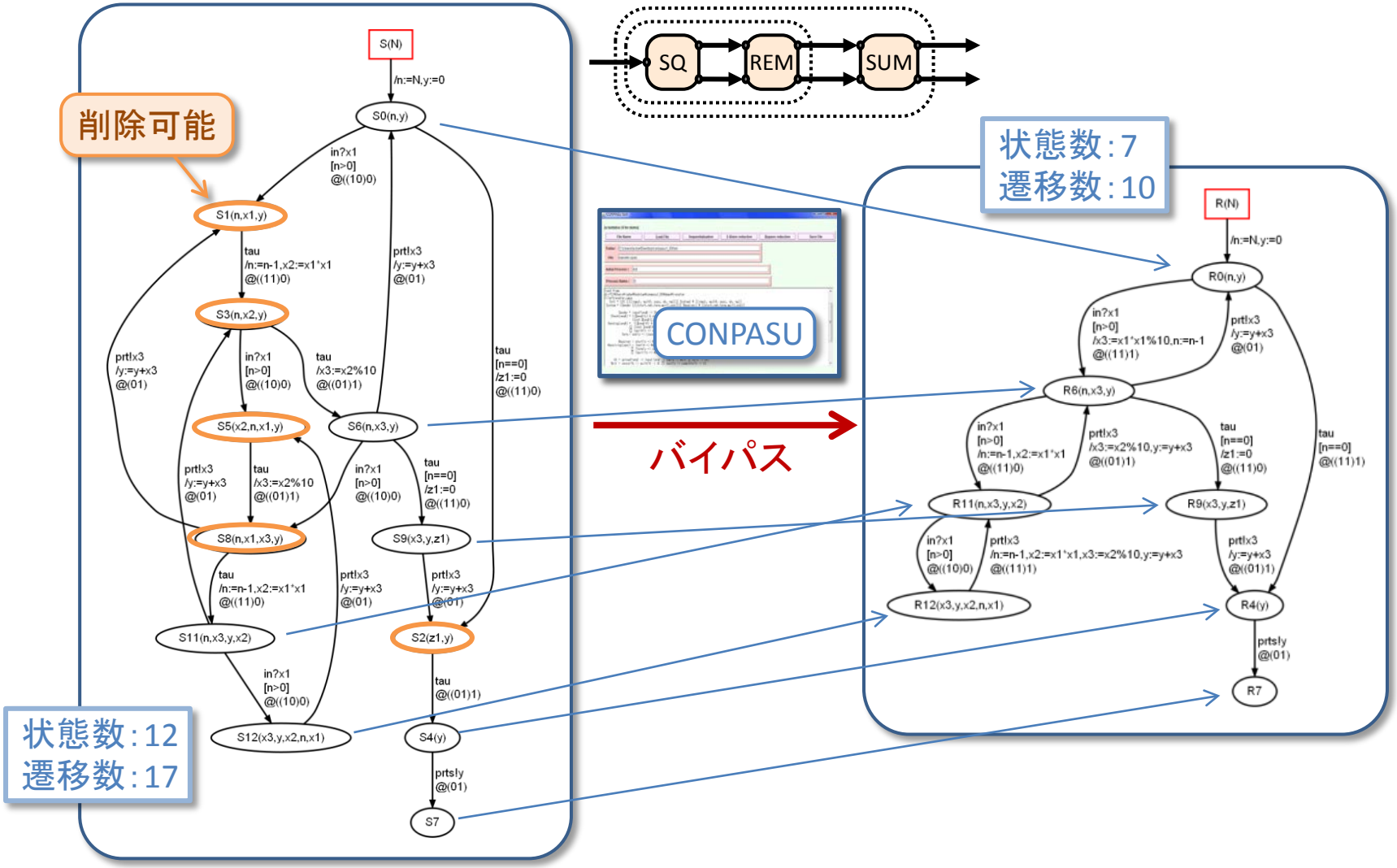
インターリービングの削除の例



位置情報を用いて遷移の独立性を判定

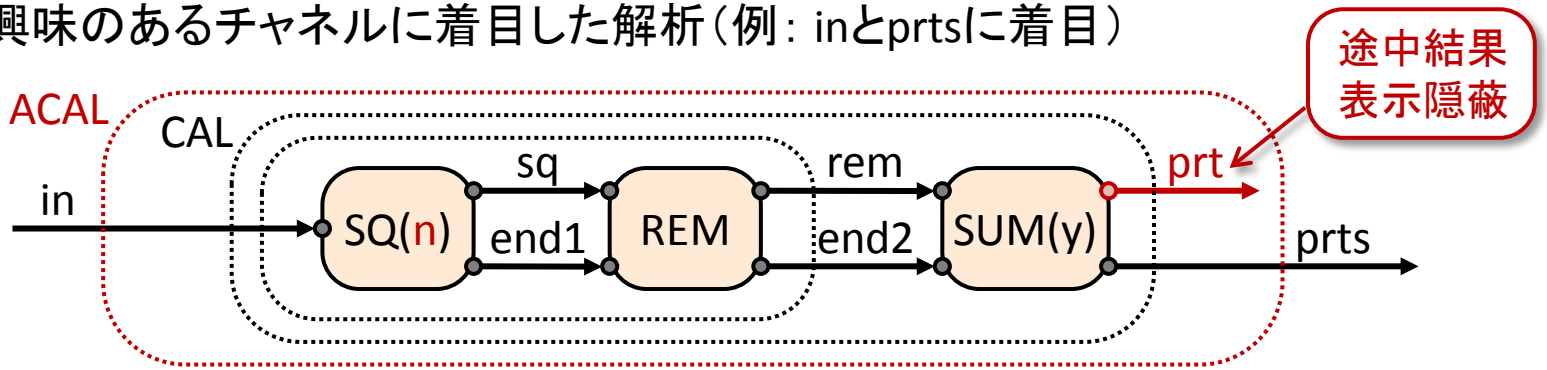
状態数削減:バイパスの例

■ 並行プロセスCAL(N)の状態遷移図中の削除可能状態とバイパス後の状態遷移図

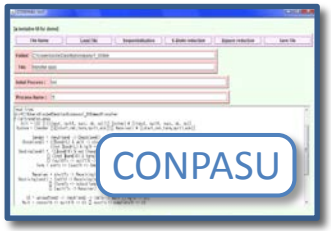


抽象化：部分仕様の抽出

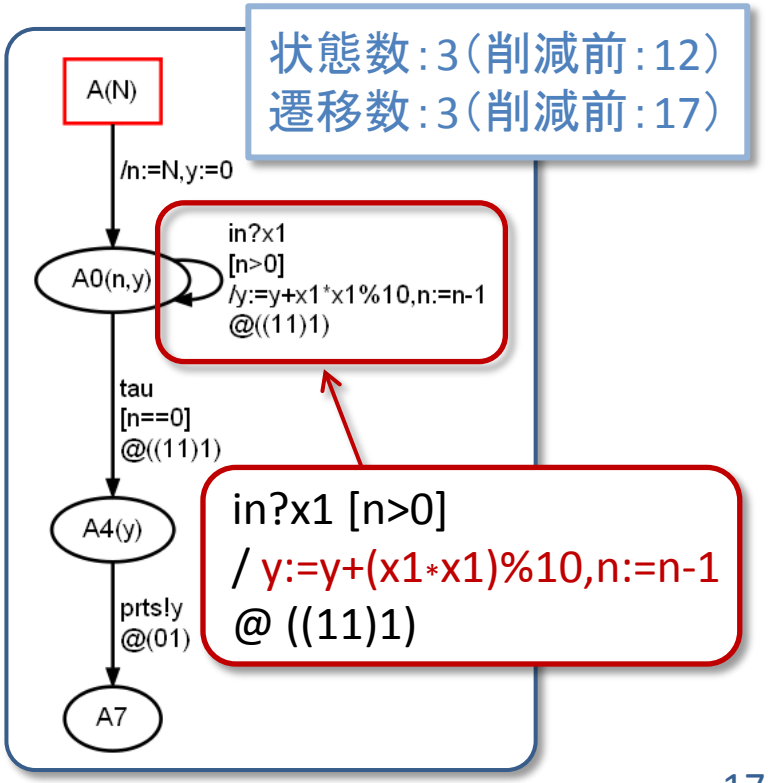
■ 興味のあるチャネルに着目した解析（例：inとprtsに着目）




```
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
```



逐次化、状態数削減



解析例

- 概要
- 解析

概要：並行プロセスTransferSys

- TransferSys: 3つのプロセスUI, Sender, Receiverから構成される並行プロセス
  SenderはUIから入力されたデータ列を順番にReceiverに転送する（途中でキャンセル可）

データ列の転送 TransferSys の記述例（CSP_M）

構造

```
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
```

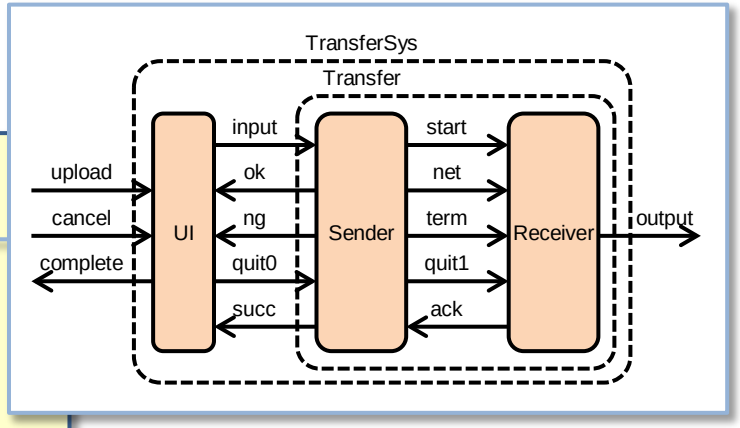
Sender

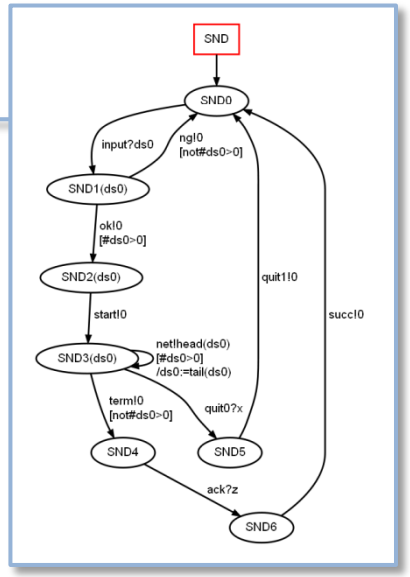
動作

```
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

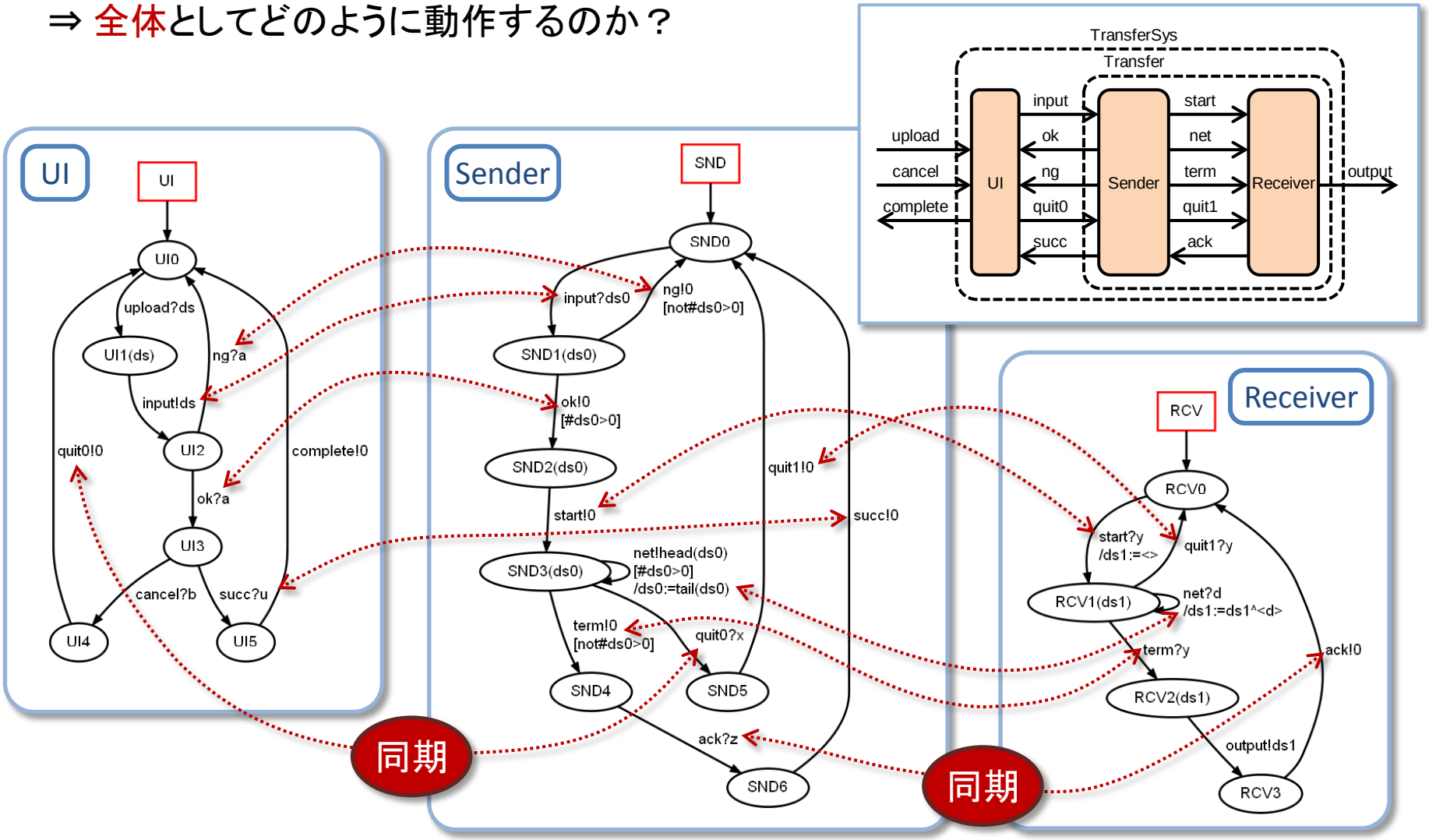
```
Receiver = start?y -> Receiving(<>)
Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
               [] (term?y -> output!ds1 -> ack!0 -> Receiver)
               [] (quit1?y -> Receiver)
```





概要:プロセス間同期通信

- UI, Sender, Receiverは互いに同期通信によって情報を伝達する
  ⇒ 全体としてどのように動作するのか？



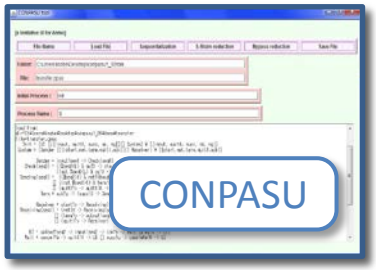
解析：結果1

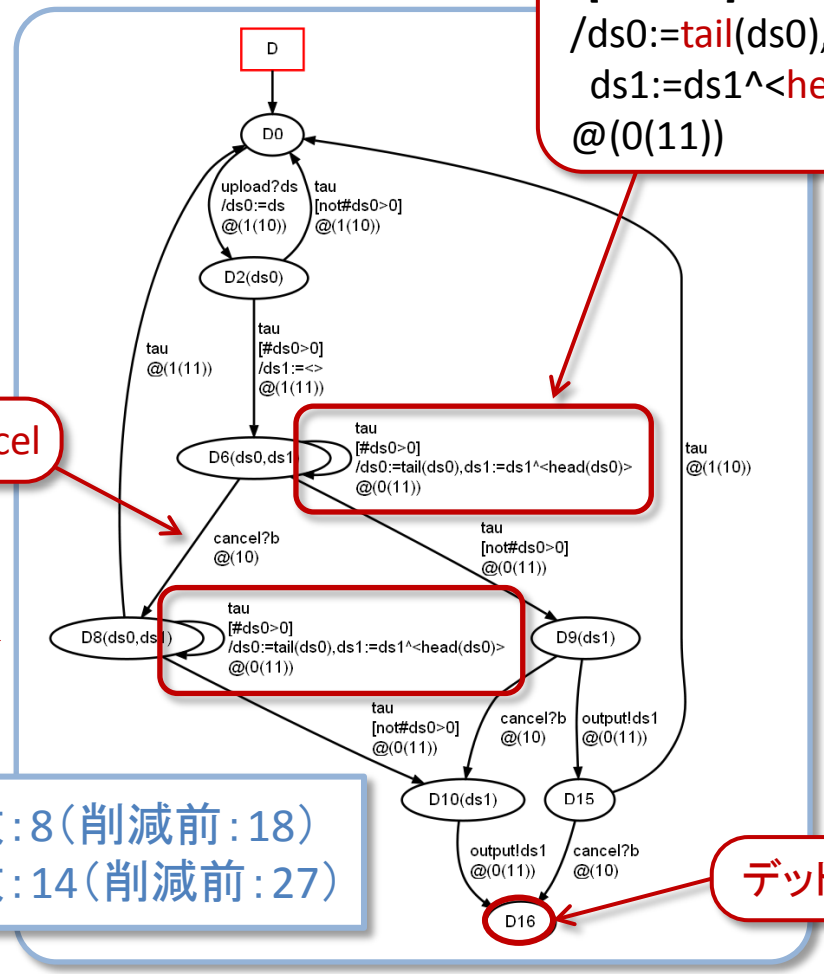
■ TransferSys から生成した状態遷移図（逐次化、状態数削減）

TransferSys ¥ {|complete |}

```
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 -> complete!0 -> 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) & term!0 -> Term)
               [] (quit0?x -> quit1!0 -> Sender)
Term = ack?z -> succ!0 -> Sender

Receiver = start?y -> Receiving(<>)
Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
                 [] (term?y -> output!ds1 -> ack!0 -> Receiver)
                 [] (quit1?y -> Receiver)
```

逐次化、状態数削減



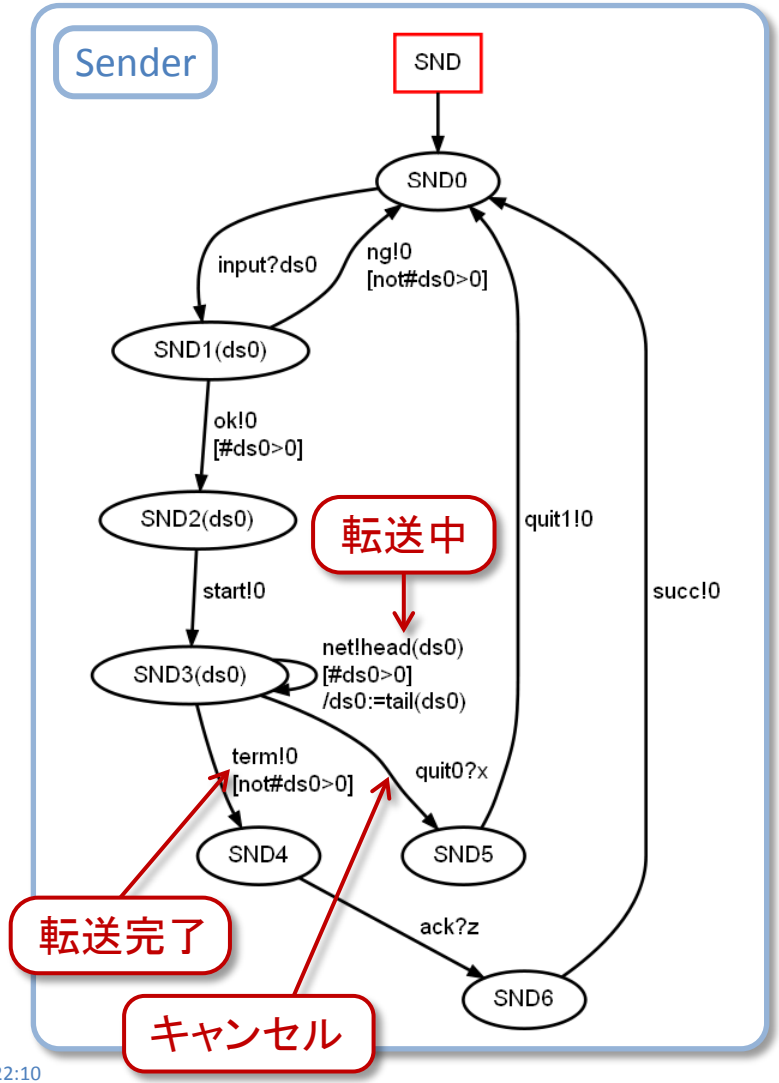


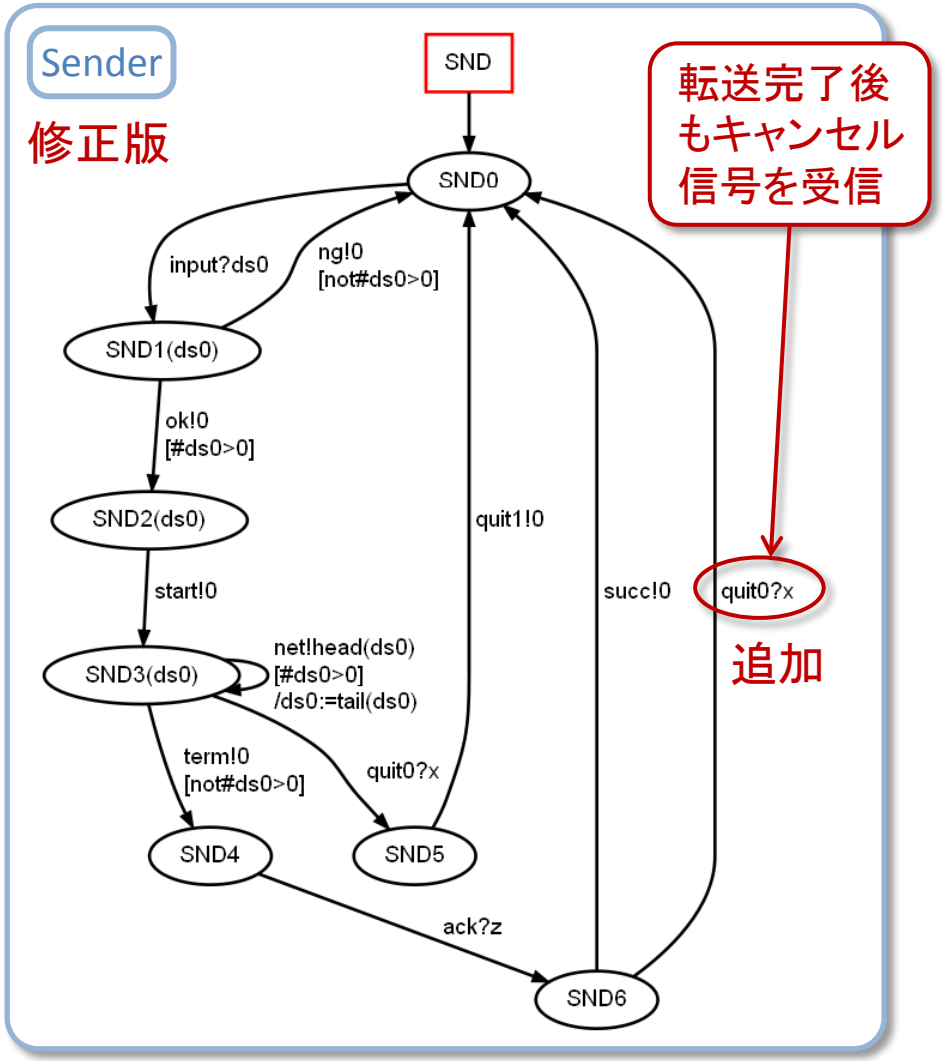
状態数：8（削減前：18）
遷移数：14（削減前：27）

グラフの表示にはGraphvizを使用

解析：デバッグ

■ Senderの修正：転送完了後もUIからのキャンセル信号を受信できるようにする





解析：結果2

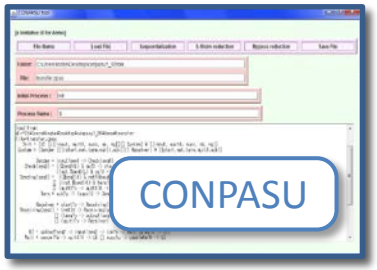
■ TransferSys（修正版）から生成した状態遷移図（逐次化、状態数削減）

TransferSys ¥ {|complete |}

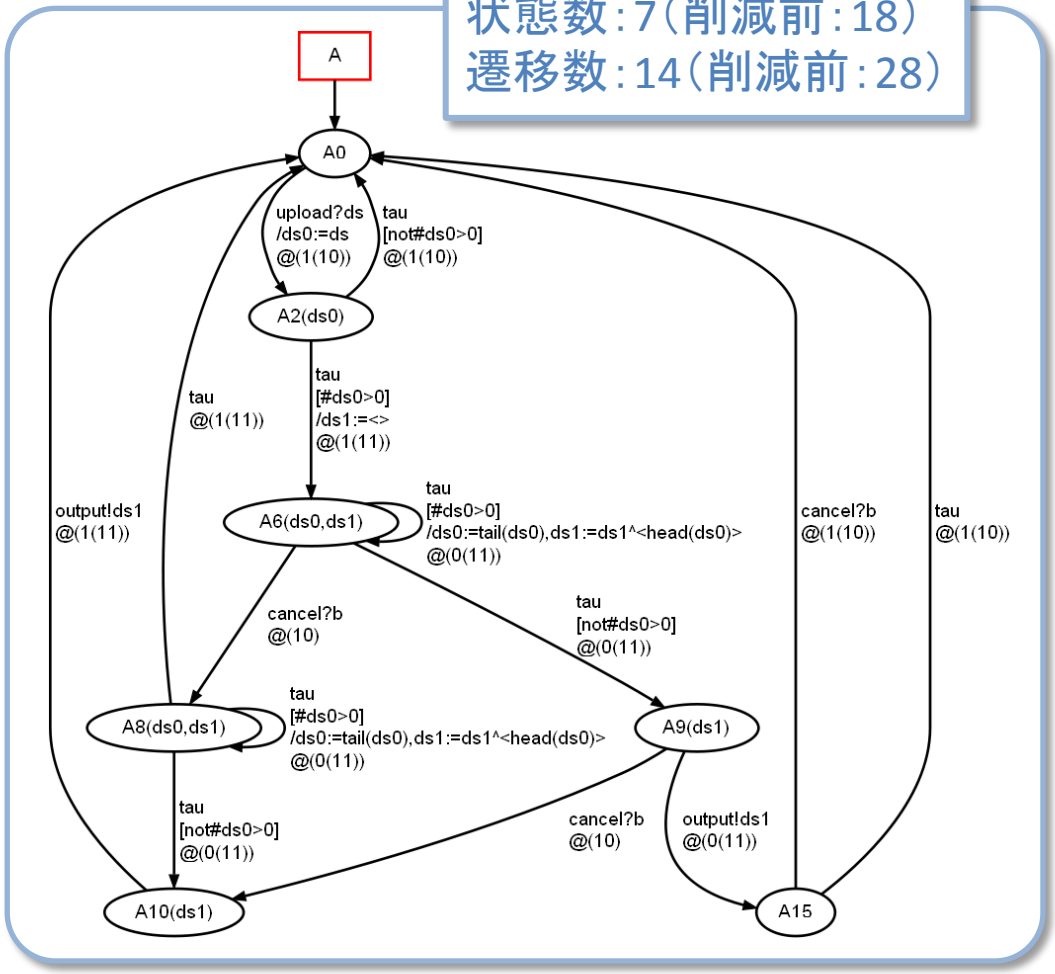
```
TransferSys = (UI [|{|input, quit0, succ, ok, ng|}|] Transfer)
                ¥ {|input, quit0, succ, ok, ng|}

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 -> complete!0 -> 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) & term!0 -> Term)
               [] (quit0?x -> quit1!0 -> Sender)
Term = ack?z -> (succ!0 -> Sender [] quit0?x -> Sender)

Receiver = start?y -> Receiving(<>)
Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
                 [] (term?y -> output!ds1 -> ack!0 -> Receiver)
                 [] (quit1?y -> Receiver)
```

修正済

逐次化、状態数削減



状態数：7（削減前：18）
遷移数：14（削減前：28）



グラフの表示にはGraphvizを使用

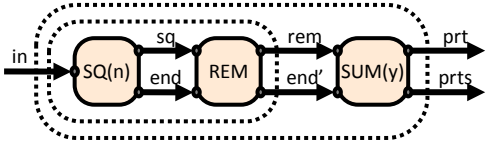
関連研究とまとめ

- 解析ツール
- まとめ

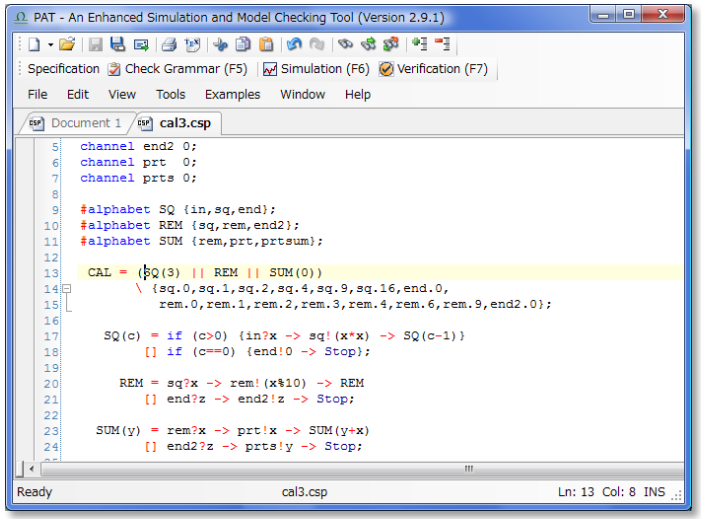
解析ツール:PAT

- モデル検査器PATでも状態遷移図を表示できる。
  変数は全て具体化されるため状態数が多くなる。

CAL



PATのGUI

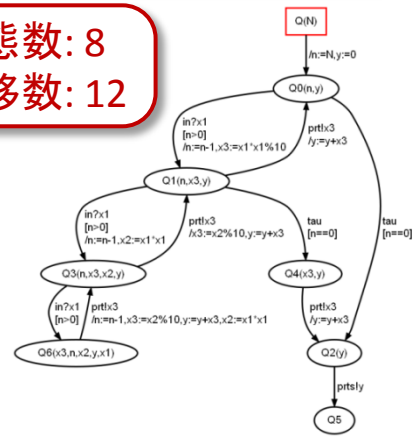


シミュレータの起動

N=3に固定, 入力値は{0,1}に有限化

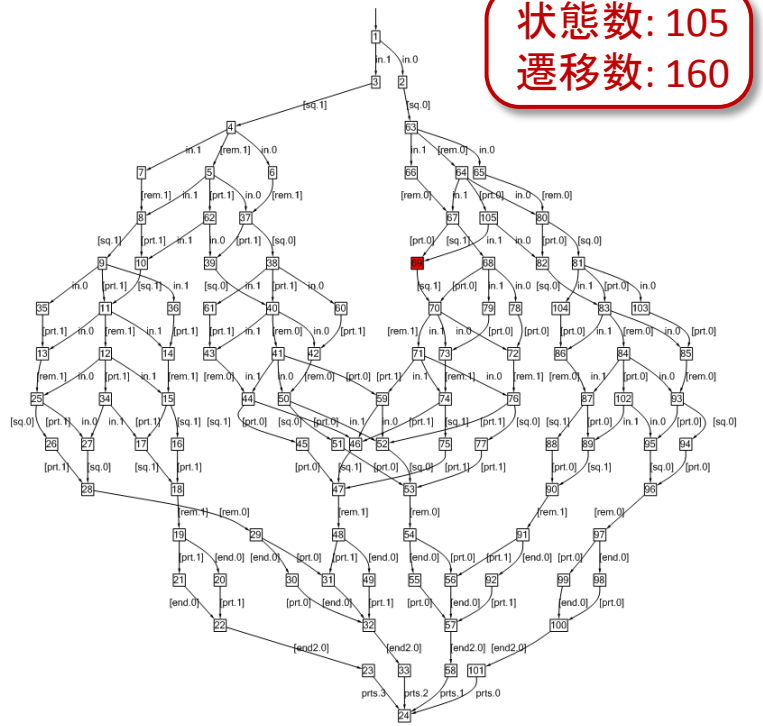
比較: CONPASU

状態数: 8
遷移数: 12



CAL(3)の状態遷移図

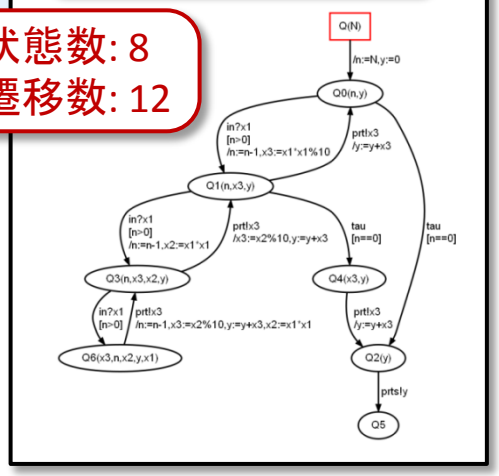
状態数: 105
遷移数: 160



解析ツール:LTSA

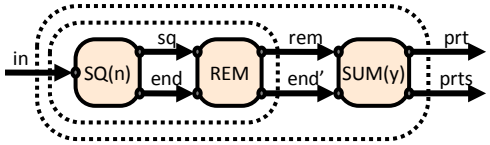
比較: CONPASU

状態数: 8
遷移数: 12

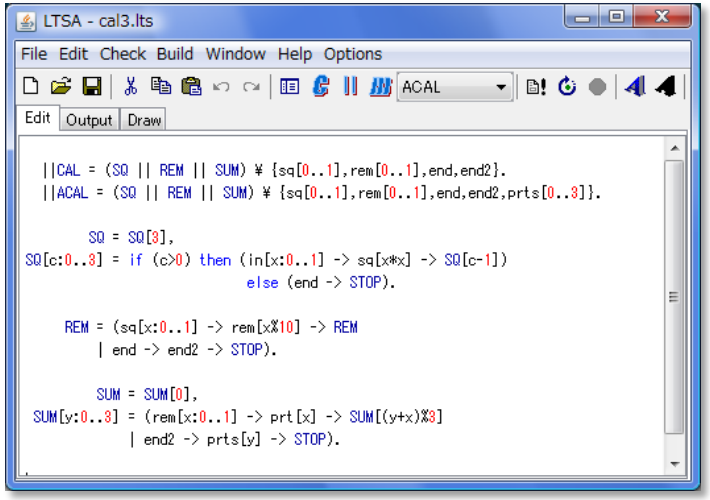


■ モデル検査器LTSAによる状態遷移図表示例

変数を具体化するため、入力値を0,1に制限し、
状態数最小化してもCAL(3)の状態数は42になる。



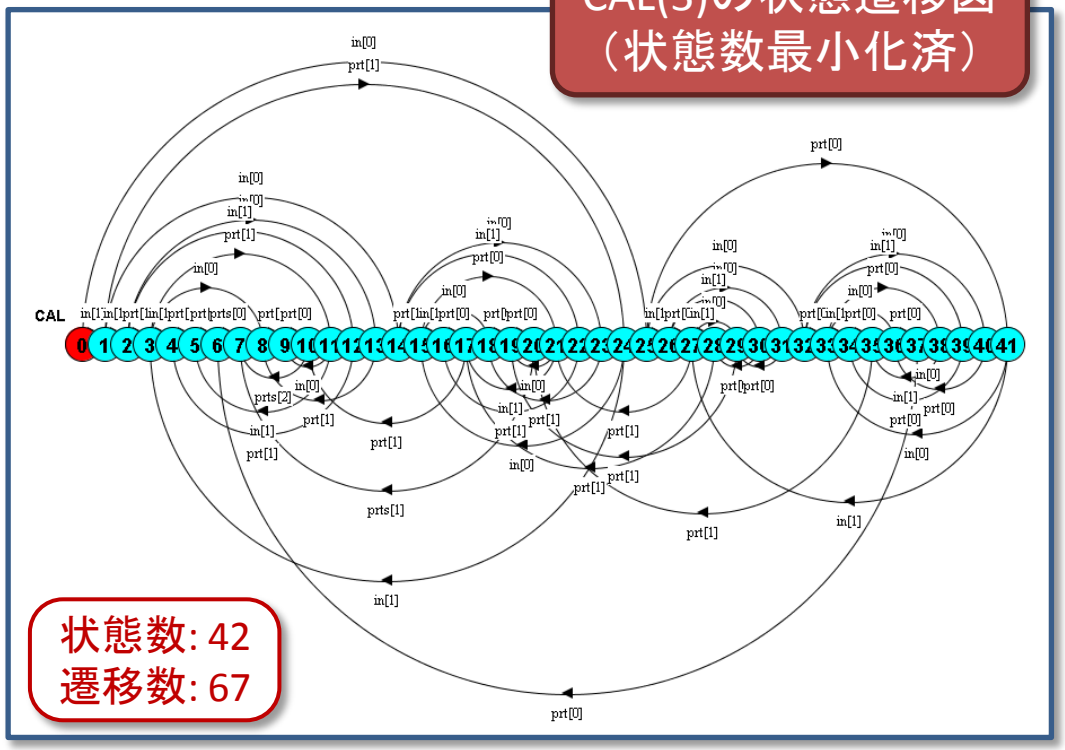
LTSAのGUI



Drawで表示

N=3に固定, 入力値は{0,1}に有限化

CAL(3)の状態遷移図
(状態数最小化済)



状態数: 42
遷移数: 67

まとめ：現状と今後の課題

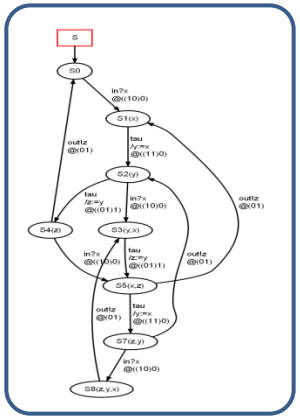
■ CONPASUの機能

- □ 逐次化機能：記号的意味論による並行プロセスの逐次化
- □ 状態数削減機能：失敗等価性による状態数削減

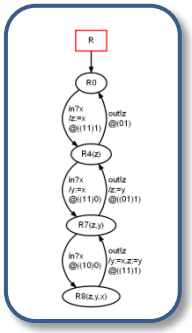
並行プロセス（CSP_M記述）

```
IMPL = PAR ¥{|com1,com2|}
 PAR = (IN [|{|com1|}|] MID) [|{|com2|}|] OUT
  IN = in?x -> com1!x -> IN
 MID = com1?y -> com2!y -> MID
 OUT = com2?z -> out!z -> OUT
```

逐次化 →



状態数削減 $=_F$ →



プロセス生成 →

逐次プロセス（CSP_M記述）

```
R = R0¥{|tmp|}
R0 = in?x -> R4(x)
R4(z) = in?x -> R7(z,x) [] out!(z) -> R0
R7(z,y) = in?x -> R8(z,y,x) [] out!(z) -> R4(y)
R8(z,y,x) = out!(z) -> R7(y,x)
```

■ CONPASUの開発方針

- □ 完全自動記号処理：状態数の半自動最小化よりも完全自動削減（最小でなくても可）
  （代入付状態遷移図の最小化は決定不能）

■ 今後の課題

- □ ライブロックの保存性についてより詳細な議論が必要
- □ データ式の記号処理の強化（試作版：1+2 ≠ 2+1）
- □ CONPASU試作版（Java, 6,000行程度）の改善、パフォーマンスの評価

conpasu-website: http://staff.aist.go.jp/y-isobe/conpasu/