CONPASU-tool (CONcurrent Process Analysis SUpport tool):
記号処理に基づく並行プロセス解析支援ツールの試作

情報技術研究部門
ミドルウェア基礎研究グループ
磯部祥尚

FOSE2010 2010/11/19

発表内容

背景と動機
- 並行プロセス設計の難しさ
- 本研究の目的
- プロセス代数記術
- CONAPSUの機能
- 既存ツールとの違い

解析ツール CONPASU
- CONPASUの概要
- 逐次化機能
- 状態数削減機能

CONPASU適用例
- 並列計算の解析例
- 解析時間

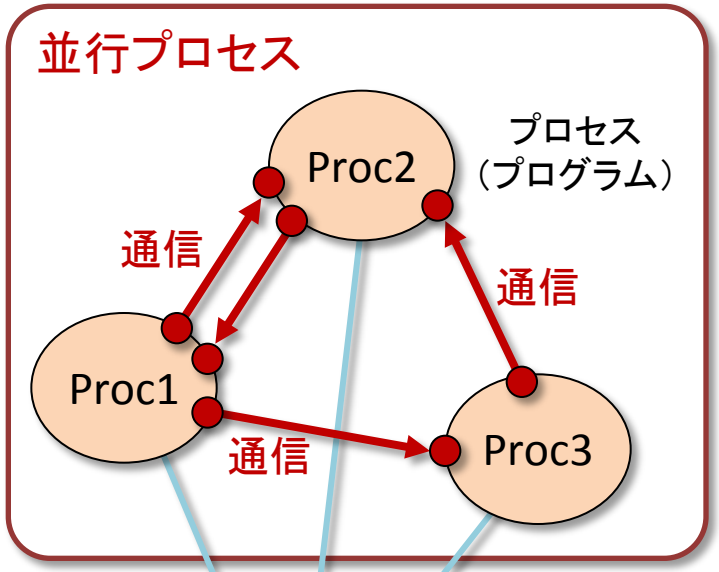
関連研究とまとめ
- 既存ツールとの違い
- まとめと今後の課題

本研究の概要

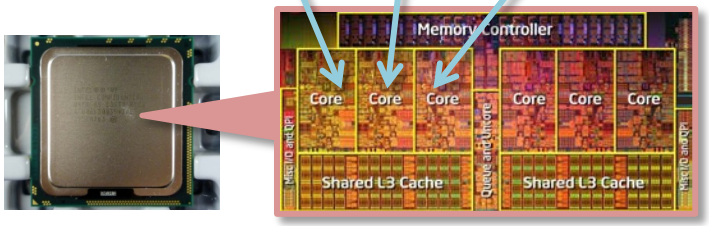
- 背景
- 目的
- プロセス代数
- CONPASU

背景：並列処理環境

- マルチコアCPU等の普及により、並列処理環境が身近なものになってきている。
  - 複数のプロセス（プログラム）を同時に実行可能。
  - 通信等により複数のプロセスが協調可能。
    ⇒ 並行プロセスの構築



- 並行プロセスの長所と短所
  - 長所：協調動作による処理の高速化
  - 短所：協調動作による設計の複雑化
    ⇒ 全体の動作の把握が難しい



例: 6コアCPU Core i7-980X（[1]より写真を転載）

[1] 大原雄介,「Core i7-980X Extreme Edition」徹底攻略!! 6コア“Gulftown”の全貌解明,
マイコミジャーナル, 2010/03/11, http://journal.mycom.co.jp/special/2010/gulftown/index.html

背景：並行処理の例

“プロセス”は人

“通信”はバケツ渡し

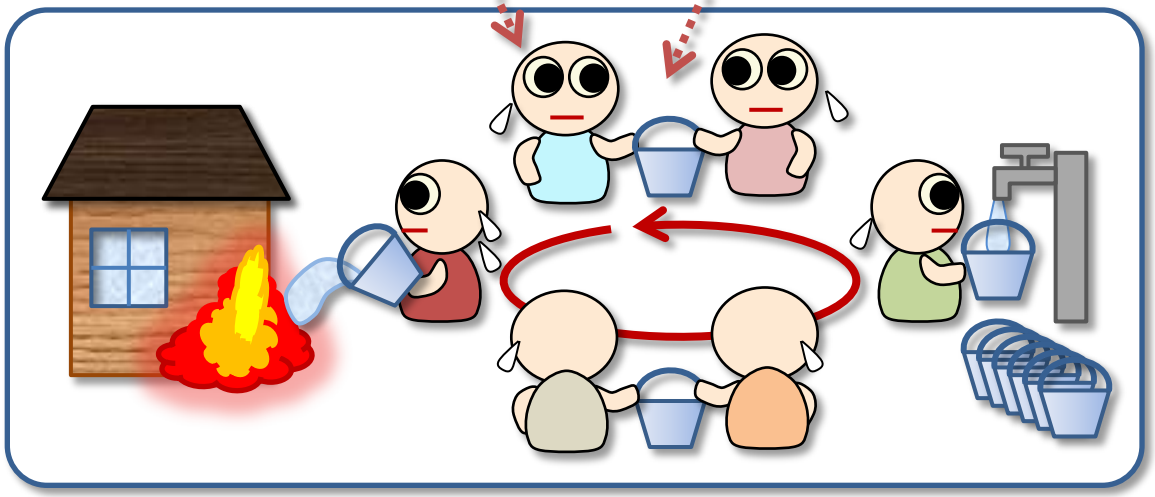
■ 並行処理の例：バケツリレーによる消火

長所

複数人による協力が可能

⇒消火作業を高速化できる。

（消火作業は各自の部分的な作業の相互作用の結果）

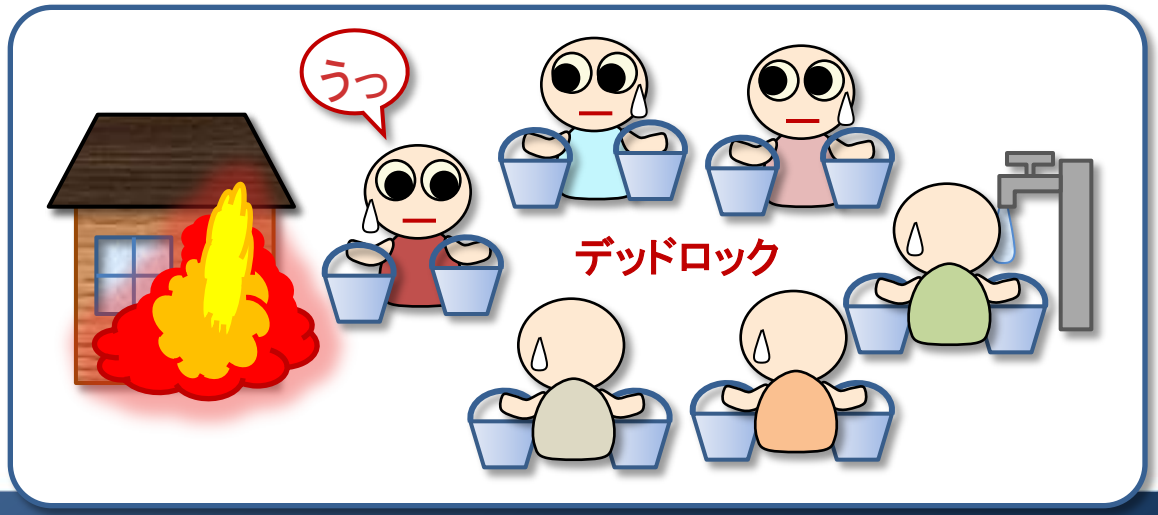


短所

全体の動作の把握が困難

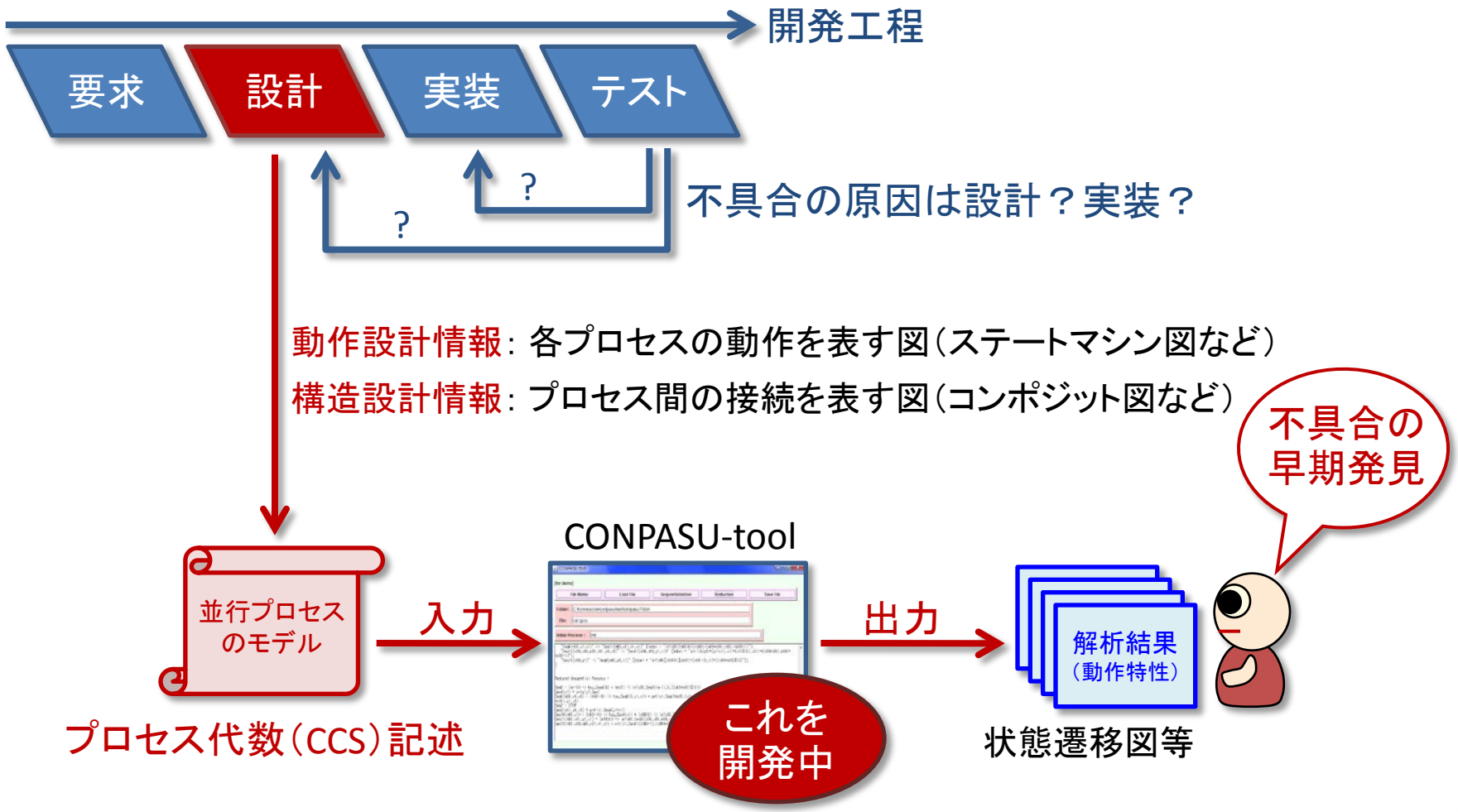
手順を適切に設定しないと…
⇒動けなくなることもある。

（CONPASUは各自の作業から全体の消火作業を導出する）



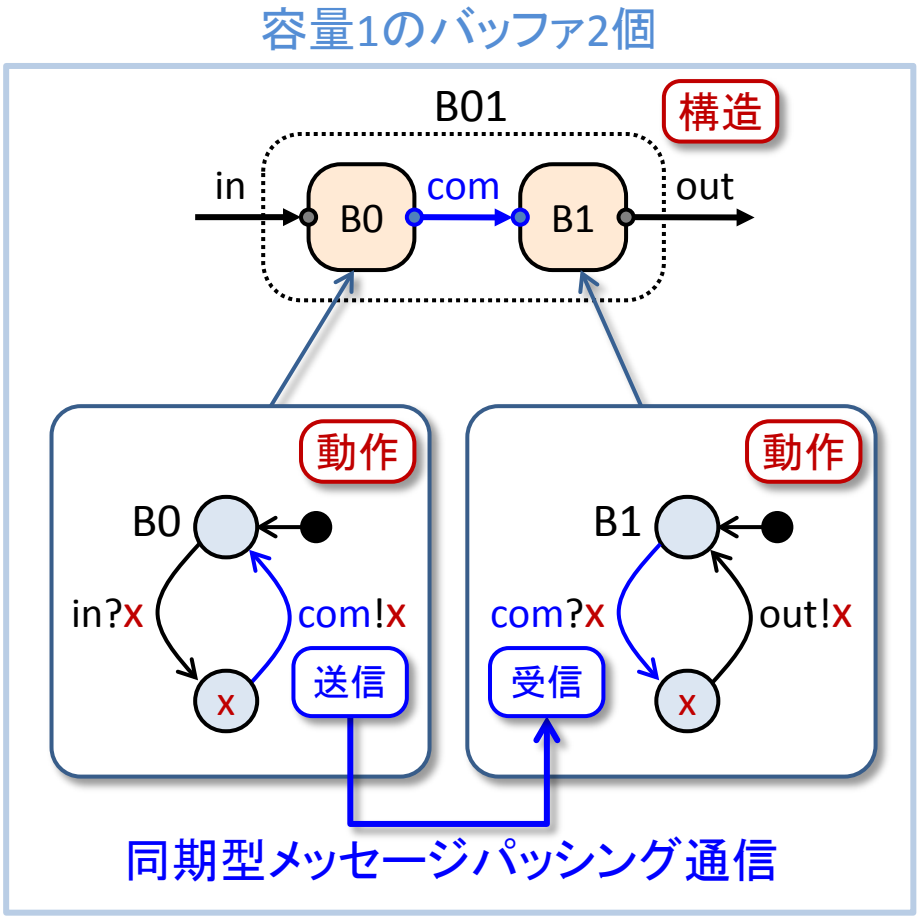
目的：並行プロセスの設計支援

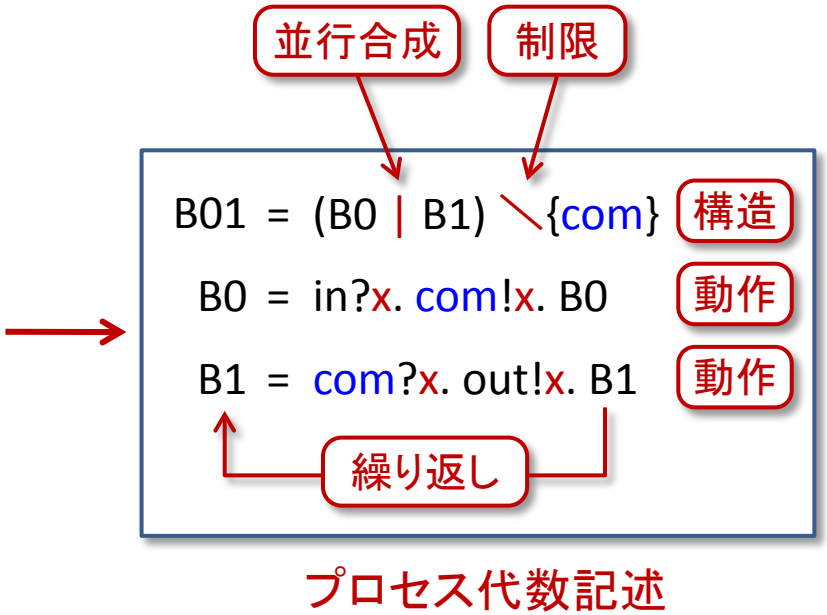
■ 目標：設計段階で並行プロセスの動作を解析するツール（CONPASU）の開発。



記述:プロセス代数？

- プロセス代数:並行プロセスの構造と動作を式の形で表現し、解析するための理論





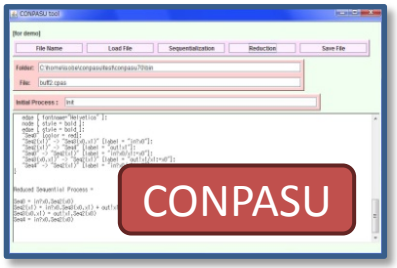
注: c!vはチャネルcへ値vを送信するアクション、c?xは チャネルcから受信した値をxへ代入するアクションを表す。

機能：CONPASUで何ができるのか？

- 並行プロセスからそれと弱等価な逐次プロセスとその状態遷移図を自動生成できる。
  （弱等価 ≈ は外部から観測して区別できない振舞いの等しさの一つ）

容量1のバッファ2個

$$B01 = (B0 \mid B1) \setminus \{com\}$$
$$B0 = in?x.\ com!x.\ B0$$
$$B1 = com?x.\ out!x.\ B1$$



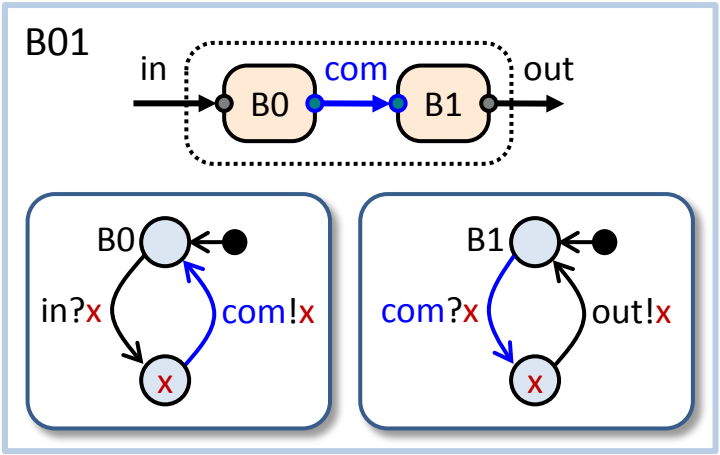
逐次化と
状態数削減

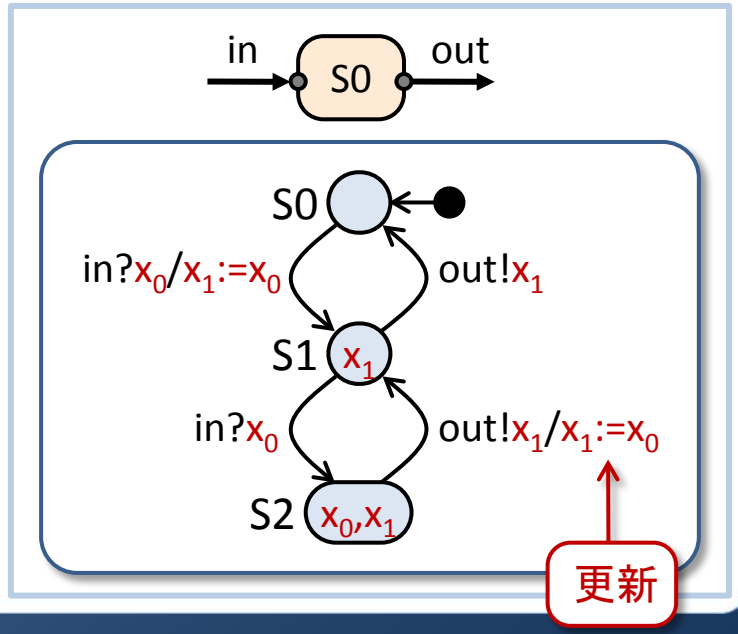
弱等価
B01 ≈ S0

容量2のバッファ

選択

$$S0 = in?x0.S1(x0)$$
$$S1(x1) = in?x0.S2(x0,x1) + out!x1.S0$$
$$S2(x0,x1) = out!x1.S1(x0)$$

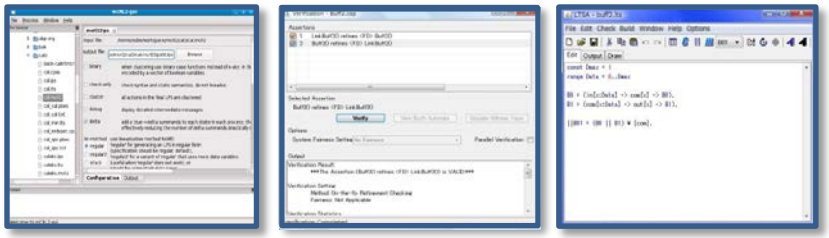




比較：既存のツールでは？

- 状態遷移図を表示する機能をもつツール（モデル検査器）はある。

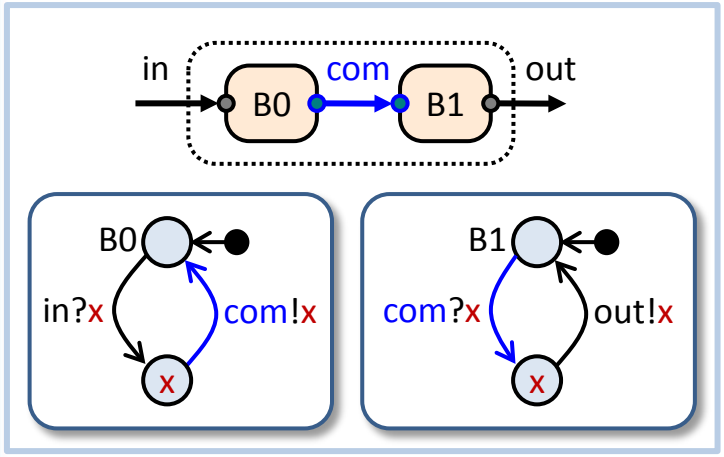
mCRL2 PAT LTSA

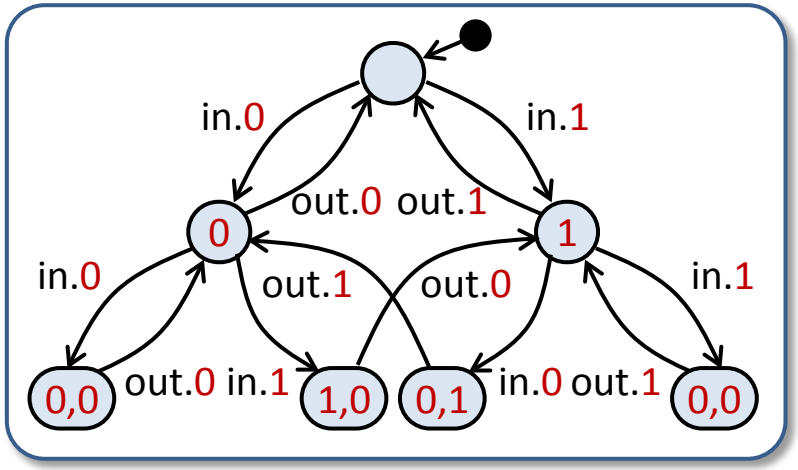


容量1のバッファ2個

```
B01 = (B0 | B1) \{com}
 B0 = in?x. com!x. B0
 B1 = com?x. out!x. B1
```

逐次化＆状態数削減
（入力値∈{0,1}）





比較：CONPASUと既存ツールとの違い

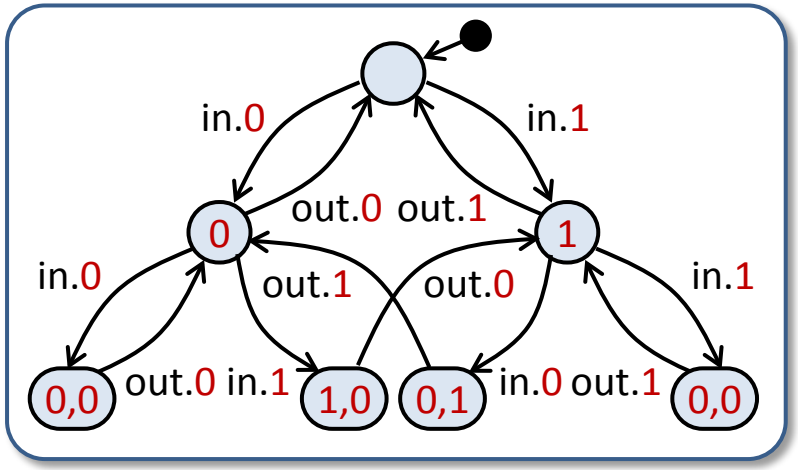
- CONPASUでは記号的意味論を採用
  - 基礎的意味論：変数を値で具体化して状態遷移図に変換する。
  - 記号的意味論：変数を具体化せずに状態遷移図に変換する。

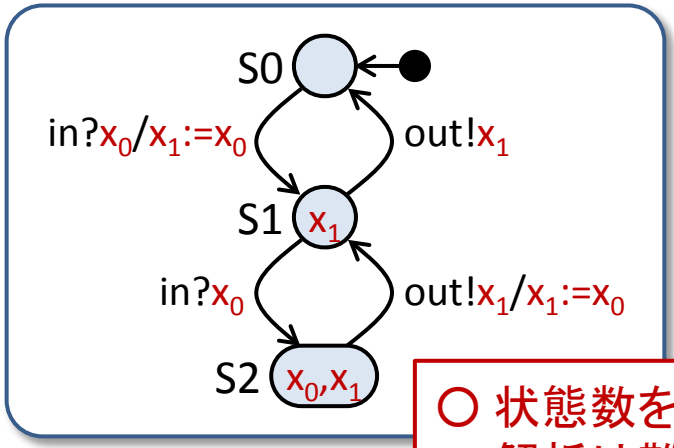
容量1のバッファ2個

```
B01 = (B0 | B1) ＼{com}
B0  = in?x. com!x. B0
B1  = com?x. out!x. B1
```

既存ツール
基礎的意味論
（入力値∈{0,1}）

CONPASU
記号的意味論
（入力値無制限）





○ 状態数を抑えられる
× 解析は難しい

- CONAPSUの方針：自動化できる範囲で解析をする

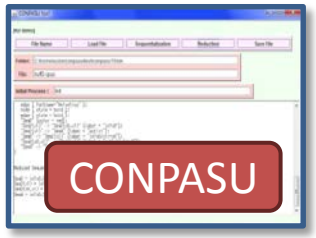
参考：状態遷移図の表示例

■ 既存のモデル検査器（PAT, LTSA）とCONPASUによる状態遷移図の表示例

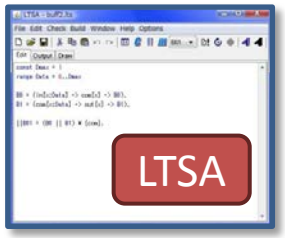
容量1のバッファ2個

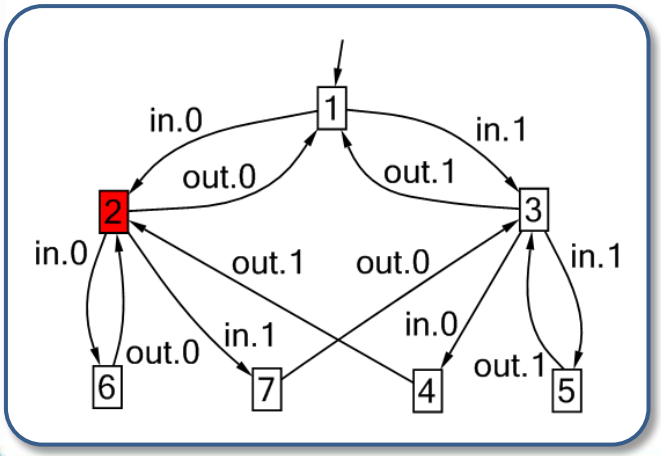
```
B01 = (B0 | B1) \{com}
B0  = in?x. com!x. B0
B1  = com?x. out!x. B1
```

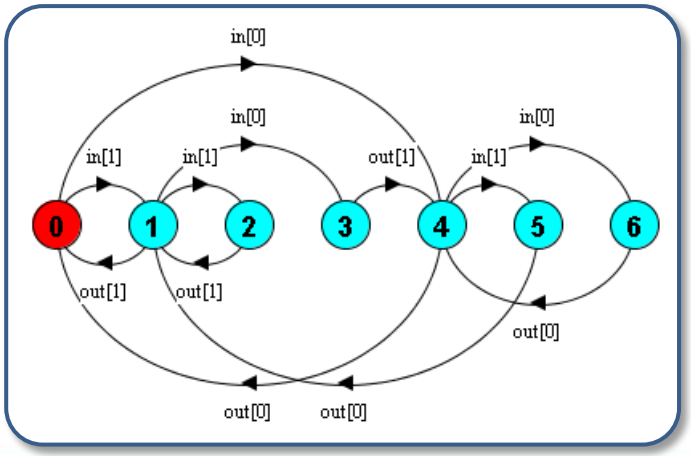
（入力値無制限）

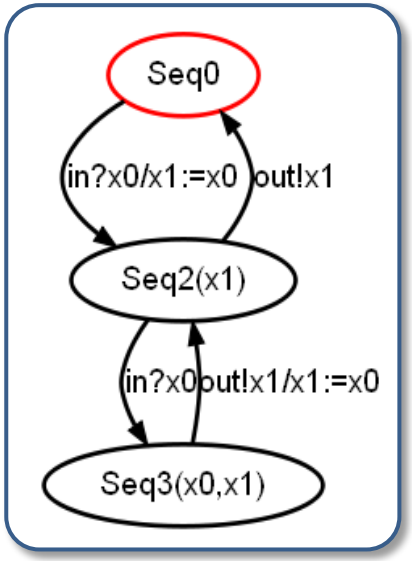


（入力値∈{0,1}）









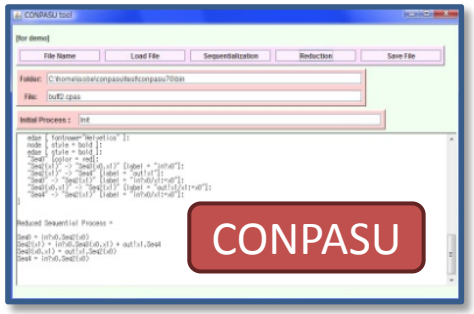
注：状態遷移図の表示にはGraphvizを使用

解析ツール CONPASU

- 解析ツール（CONPASU）の機能
- 逐次化機能
- 状態数削減機能

CONPASUの機能

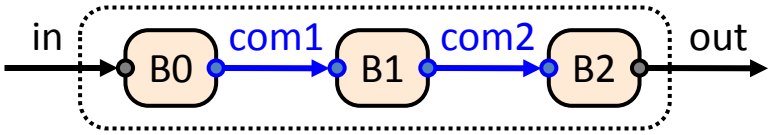
- CONPASU: 並行プロセスの動作を自動解析するためのツール（Javaで5,000行程度）
  - 逐次化機能: 並行動作を記号的意味論によって逐次的な動作に展開する機能
  - 状態数削減機能: 弱等価性を保存したまま状態数を減らす機能

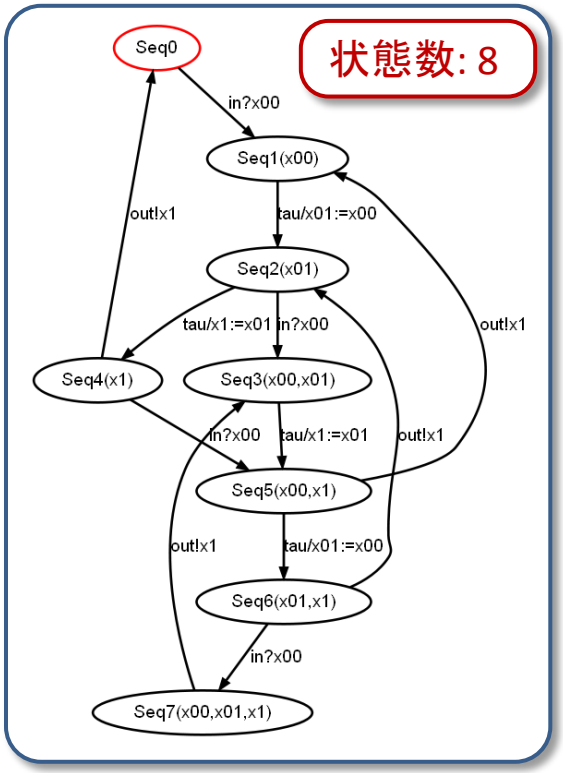


並行プロセス記述

```
B01 = (B0|B1|B2)＼{com1,com2}
 B0 = in?x. com1!x. B0
 B1 = com1?x. com2!x. B1
 B2 = com2? outx.!x. B3
```

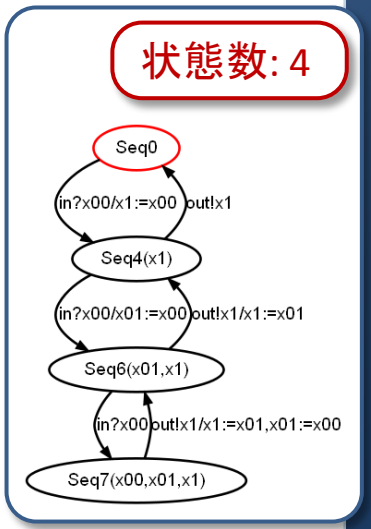
逐次化（CONPASU）





状態数削減（CONPASU）

≈



容量3のバッファ

注: 状態遷移図の表示にはGraphvizを使用

逐次化（記号的意味論）

- プロセス代数の記号的意味論は 1995年頃から様々な研究が行われている。

  本研究では下記の記号的意味論をベースにしている。

  [7] 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.

- 文献[7]との違い

  [7]では束縛変数を展開するときに変数名の衝突を避けるため新しい名前を割り当てる。

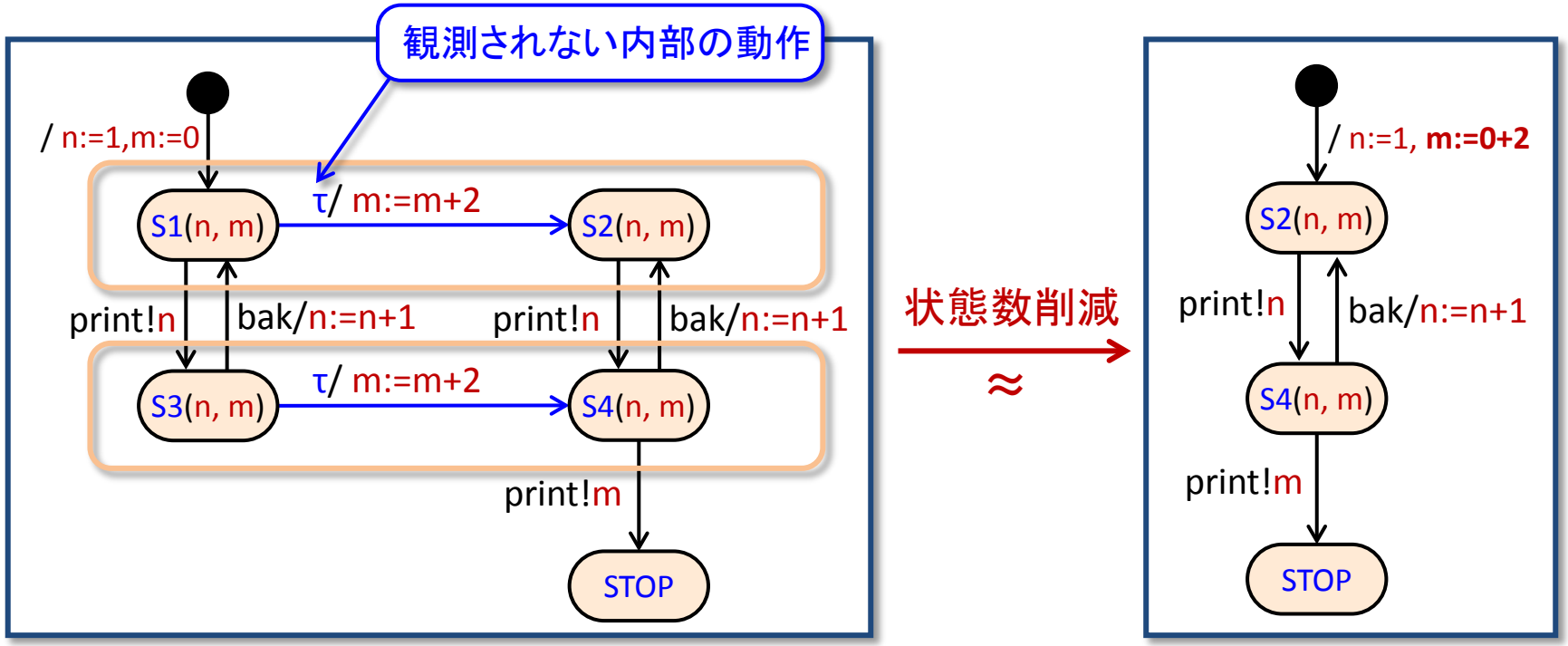
  ⇒ 新しい名前の割り当て方は書かれていない（ツール化に割り当て方が必要）。

  ⇒ ⇒本研究では変数に位置情報を付加して、変数名の衝突を避けている。

この記号的意味論の正しさは命題3.1に与えられている。

状態数削減

■ 状態削減の基本は弱等価な複数の状態を一つの状態に畳みこむことにある。
ただし、一般に記号的意味論では弱等価性を自動判定できない。
⇒ 自動判定できる範囲で弱等価な組を発見する方法を提案する。



S1(n,m) ≈ S2(n,m+2)

S3(n,m) ≈ S4(n,m+2)

この状態数削減法の正しさは命題4.1に与えられている。

CONAPSU適用例

- 並列数値計算の解析例
- 解析時間

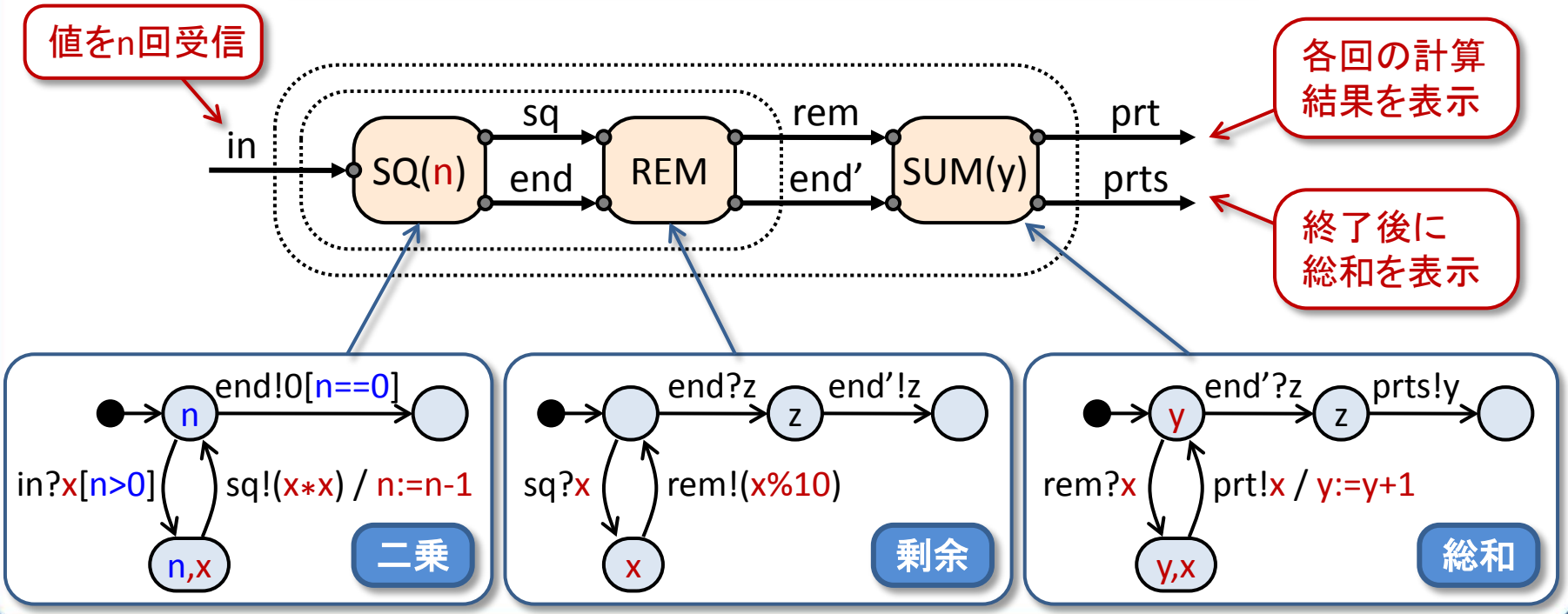
並列数値計算の解析例(1/4)

■ CAL(n) : 3つのプロセスから構成される並行プロセス

CAL(n) = (SQREM(n) | SUM(0))＼{rem,end'}
SQREM(n) = (SQ(n) | REM)＼{sq,end}

並行プロセス

SQ(n) = (n>0) => in?x.sq!(x∗x).SQ(n-1) + (n==0) => end!0.STOP
REM = sq?x.rem!(x%10).REM + end?z.end'!z.STOP
SUM(y) = rem?x.prt!x.SUM(y+x) + end'?z.prts!y.STOP



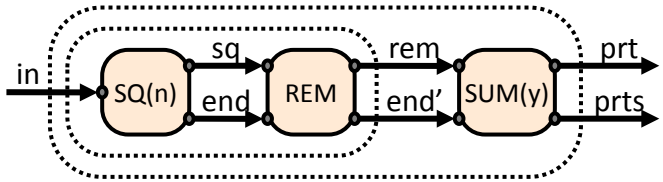
注: α[b] は条件 b が真ならばアクション α を実行できる条件付きのアクションを表す。

並列数値計算の解析例(2/4)

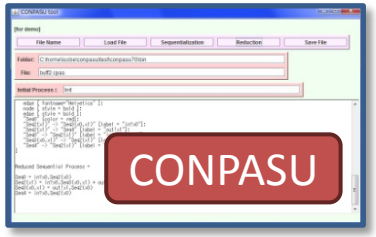
■ 並行プロセスCAL(c)の解析: 逐次化(状態数削減前)

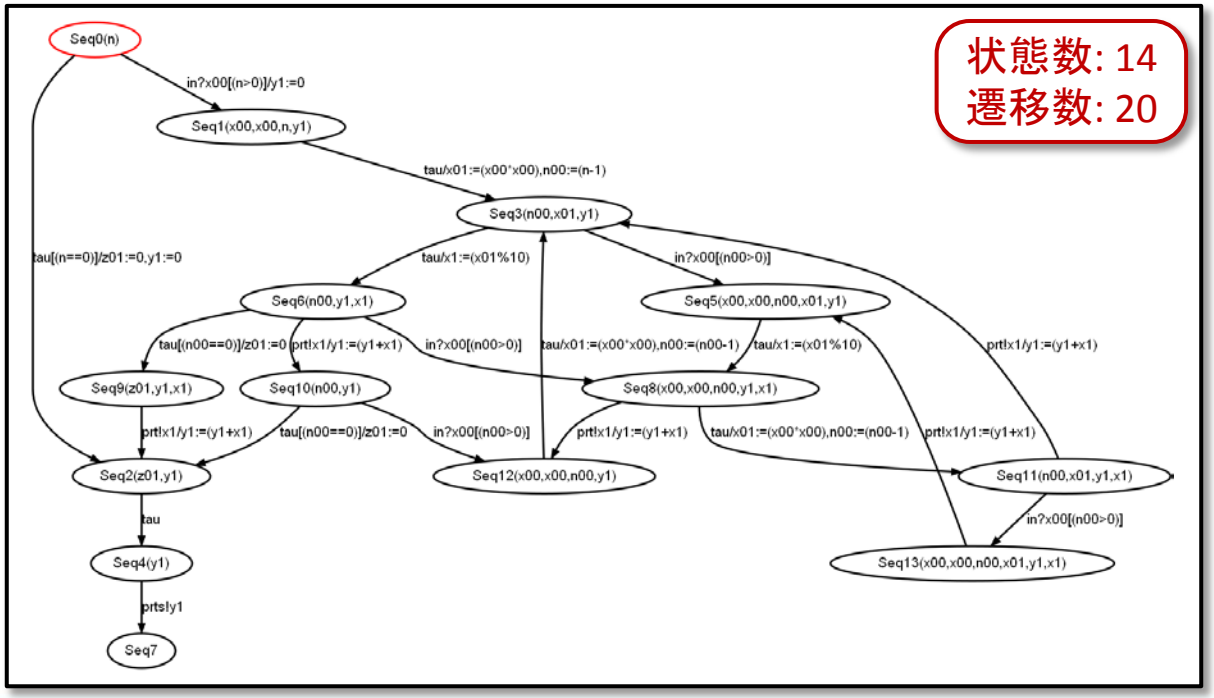
```
   CAL(n) = (SQREM(n) | SUM(0))＼{rem,end'}
SQREM(n) = (SQ(n) | REM)＼{sq,end}

   SQ(n) = (n>0) => in?x.sq!(x*x).SQ(n-1) + (n==0) => end!0.STOP
   REM = sq?x.rem!(x%10).REM + end?z.end'!z.STOP
   SUM(y) = rem?x.prt!x.SUM(y+x) + end'?z.prts!y.STOP
```

並行プロセス



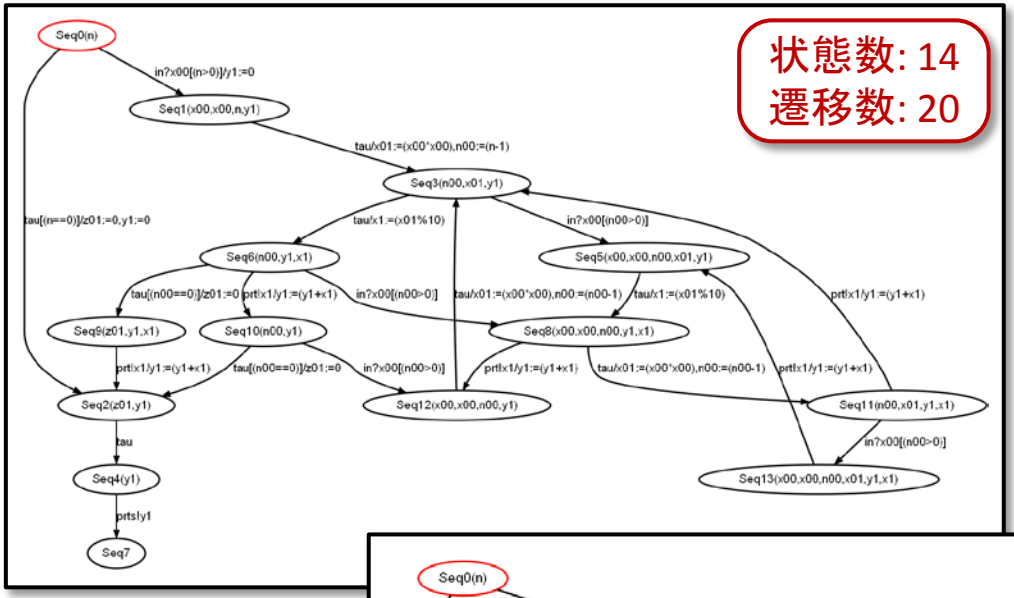
逐次化

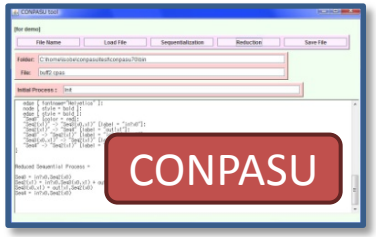




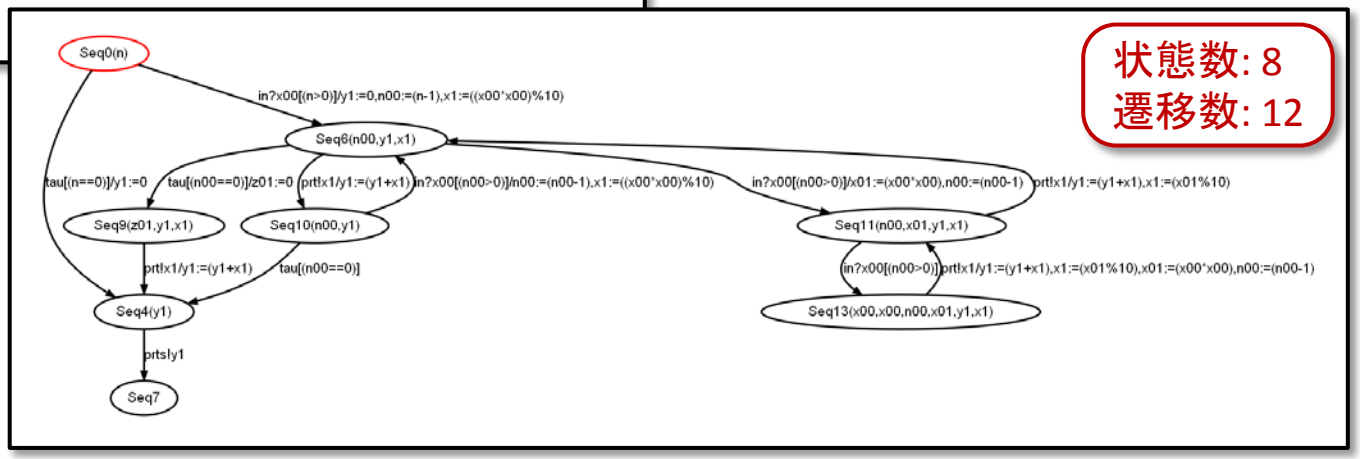
並列数値計算の解析例(3/4)

- 並行プロセスCAL(c)の解析: 状態数削減



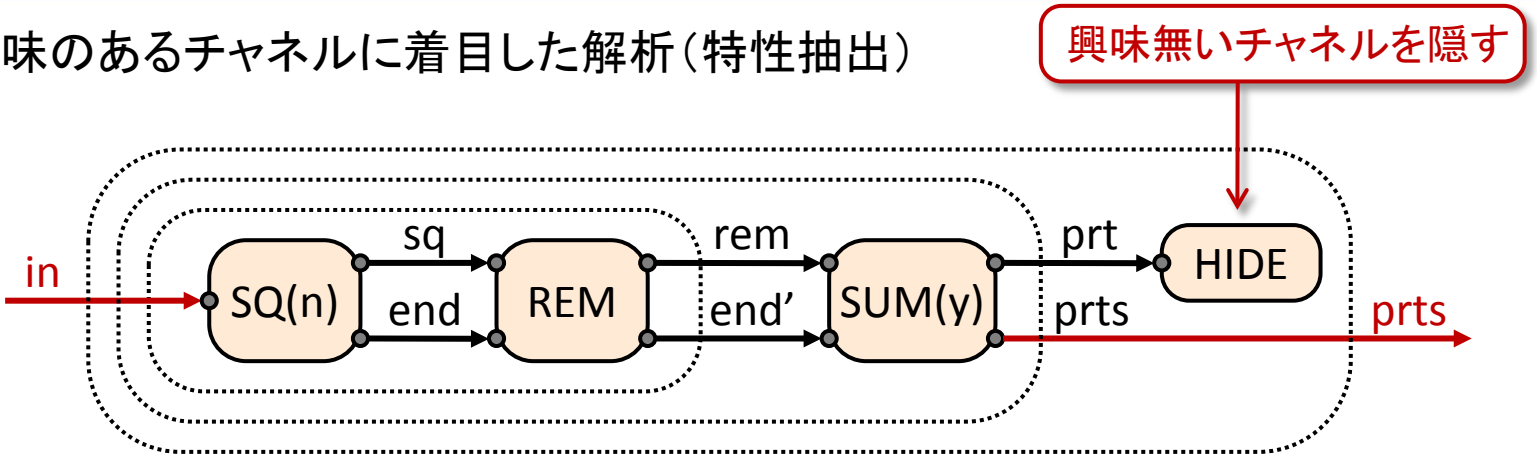


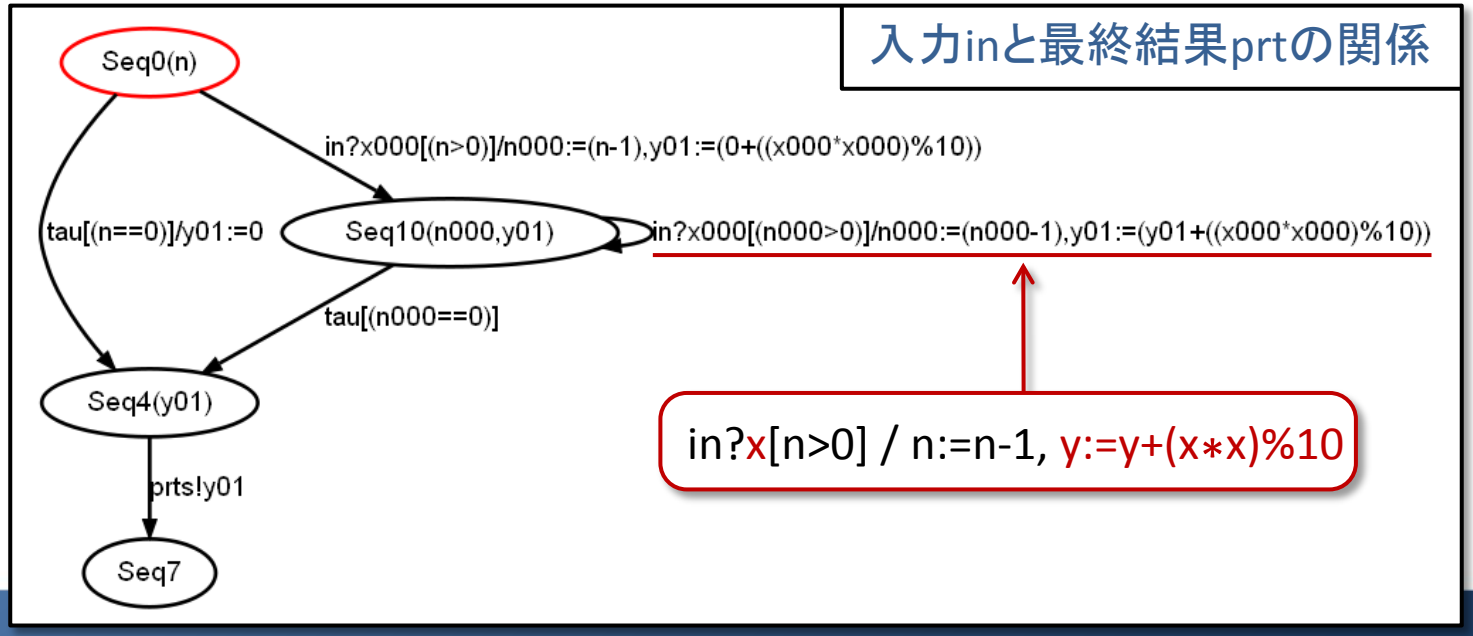
状態数削減



並列数値計算の解析例(4/4)

■ 興味のあるチャネルに着目した解析(特性抽出)

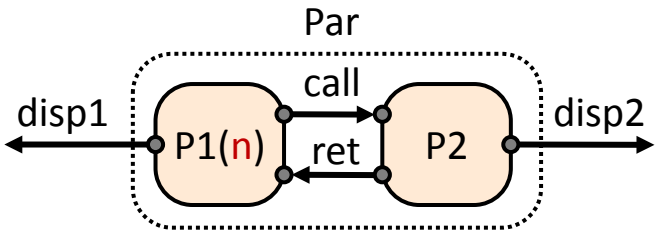




解析時間（参考値）

■ CONPASUによる解析時間

現在のCONPASUは試作機であるが、参考までに並行プロセスの解析（逐次化＋状態削除）にかかった時間の一例を示す。



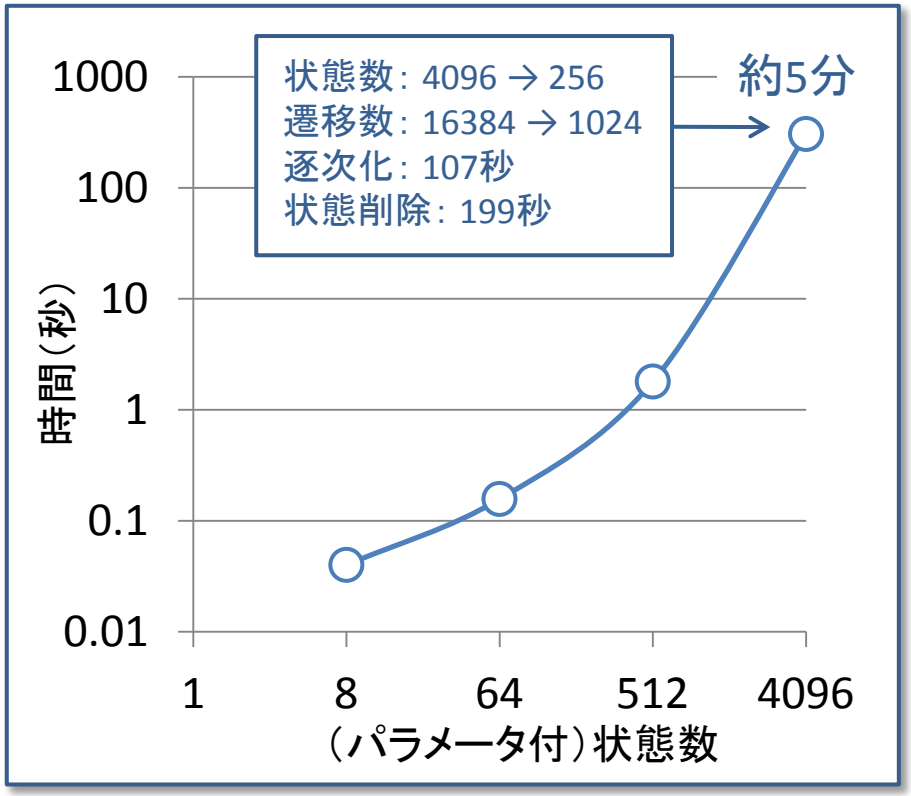

```
  Par  =  (P1(0)|P2)＼{call,ret}
P1(n)  =  disp1!n. call!n. ret?n. P1(n)
   P2  =  call?n. disp2!n. ret!(n+1). P2
```

COPY(m) = Par | … | Par
（Par | … | Par は m個）

COPY(m)の状態数は8^m個

m=1,2,3,4の場合のCOPY(m)の解析時間 ⇒

（Intel Core 2 Duo CPU P9600, 2.66GHz, 4GB RAM）

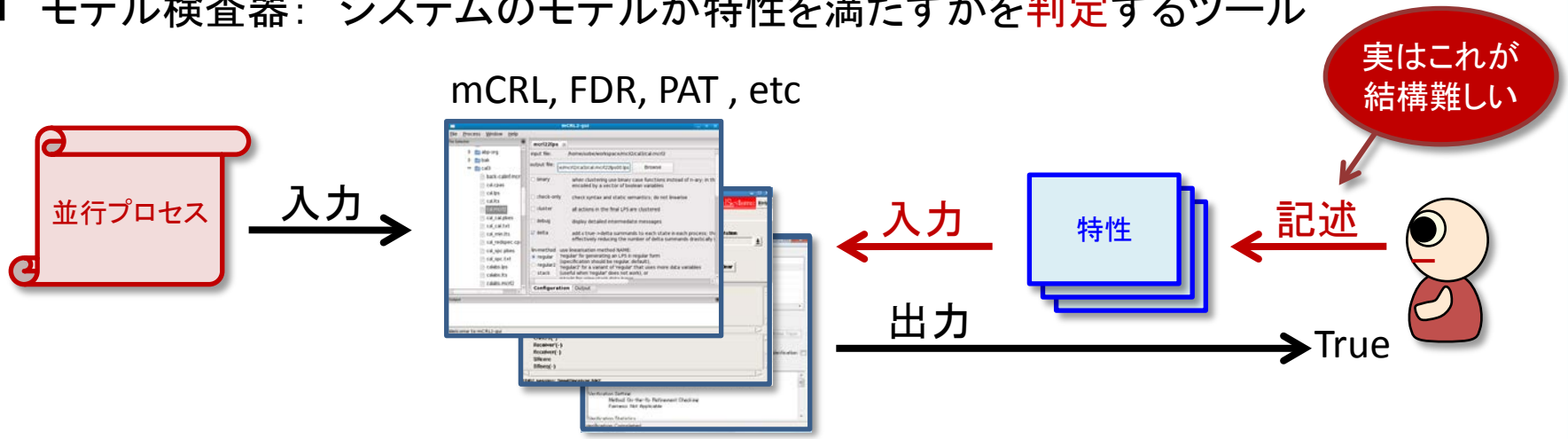


関連研究とまとめ

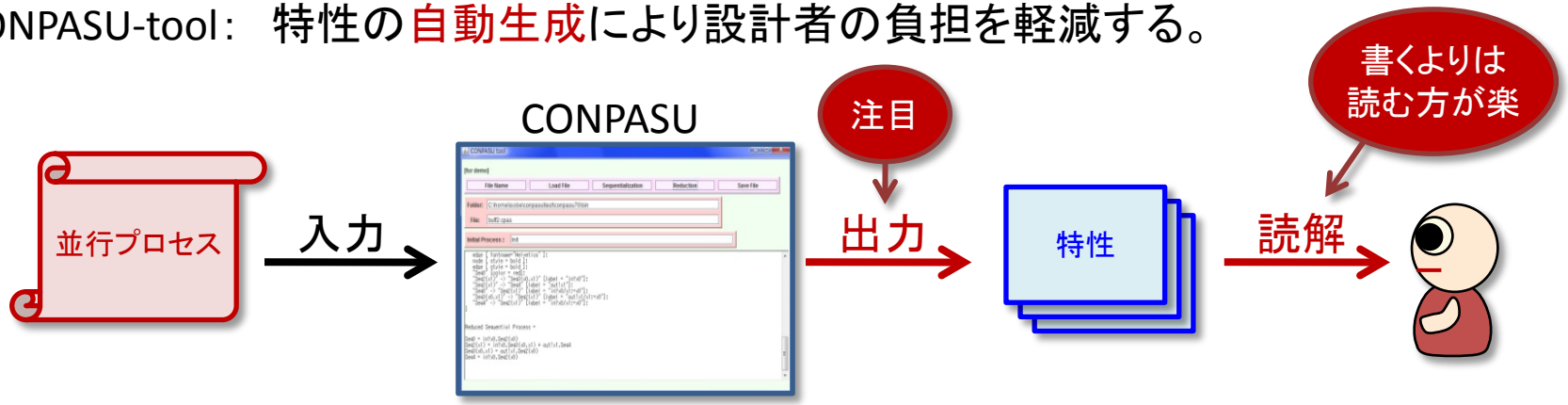
- モデル検査器との比較
- 状態遷移図の表示機能
- モデル検査器との相互補完
- まとめと今後の課題

比較:モデル検査

- モデル検査器: システムのモデルが特性を満たすかを判定するツール

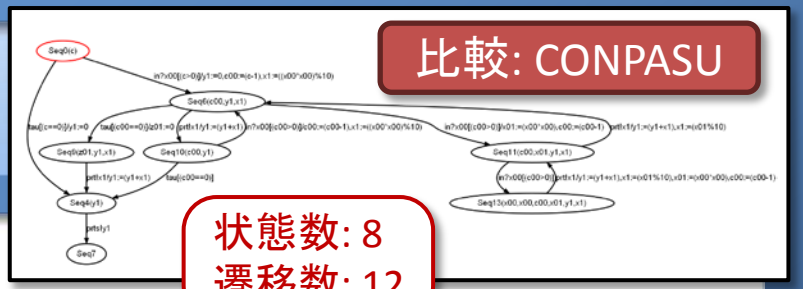


- CONPASU-tool: 特性の自動生成により設計者の負担を軽減する。

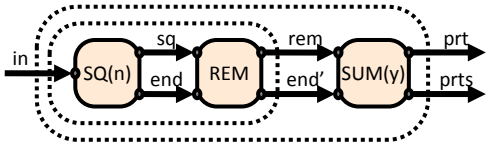


mCRL: http://www.mcrl2.org/mcrl2/wiki/index.php/Home , FDR: http://www.fsel.com/ , PAT: http://www.comp.nus.edu.sg/~pat/

比較：状態遷移図表示（mCRL2）



- モデル検査器mCRL2は状態遷移図を立体的に表示する機能がある。



並列数値計算CAL(3), 入力値∈{0..23}

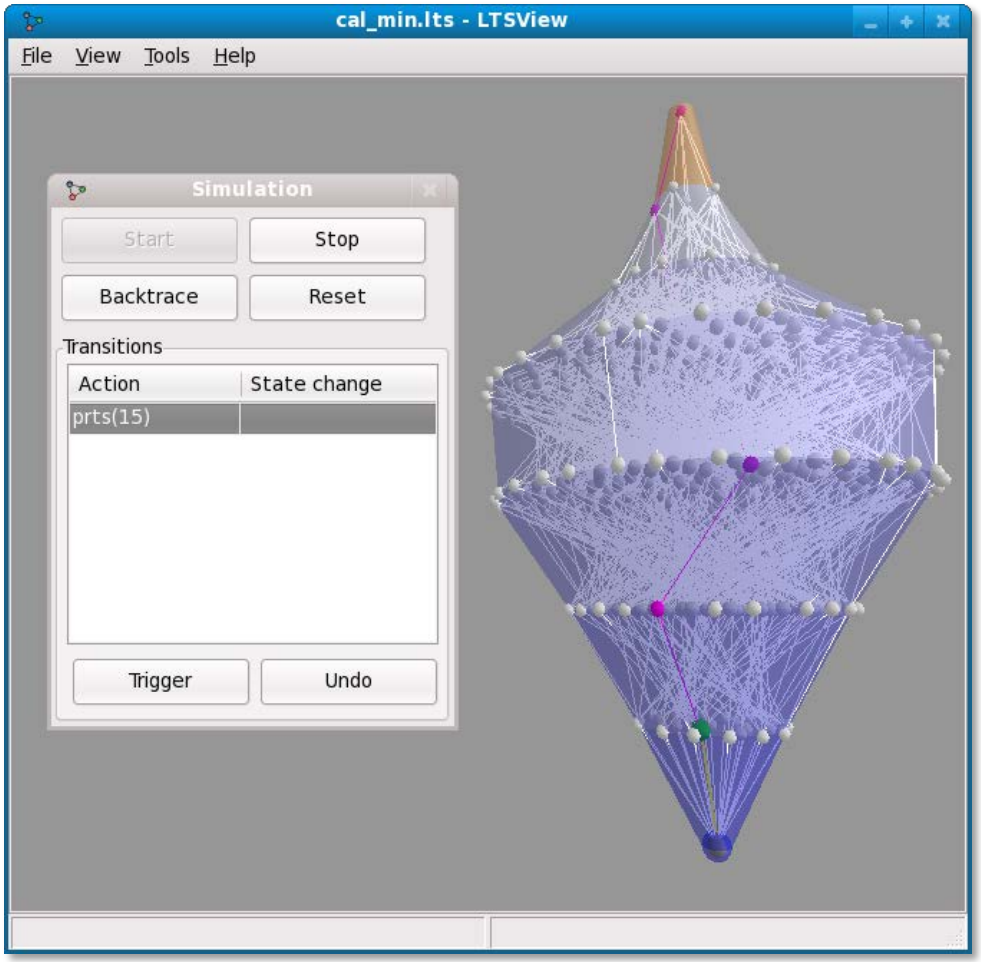
LTSへ変換 ↓ mcrl22lps, lps2lts

状態数：10,944
遷移数：22,176

状態数最小化 ↓ ltsconvert

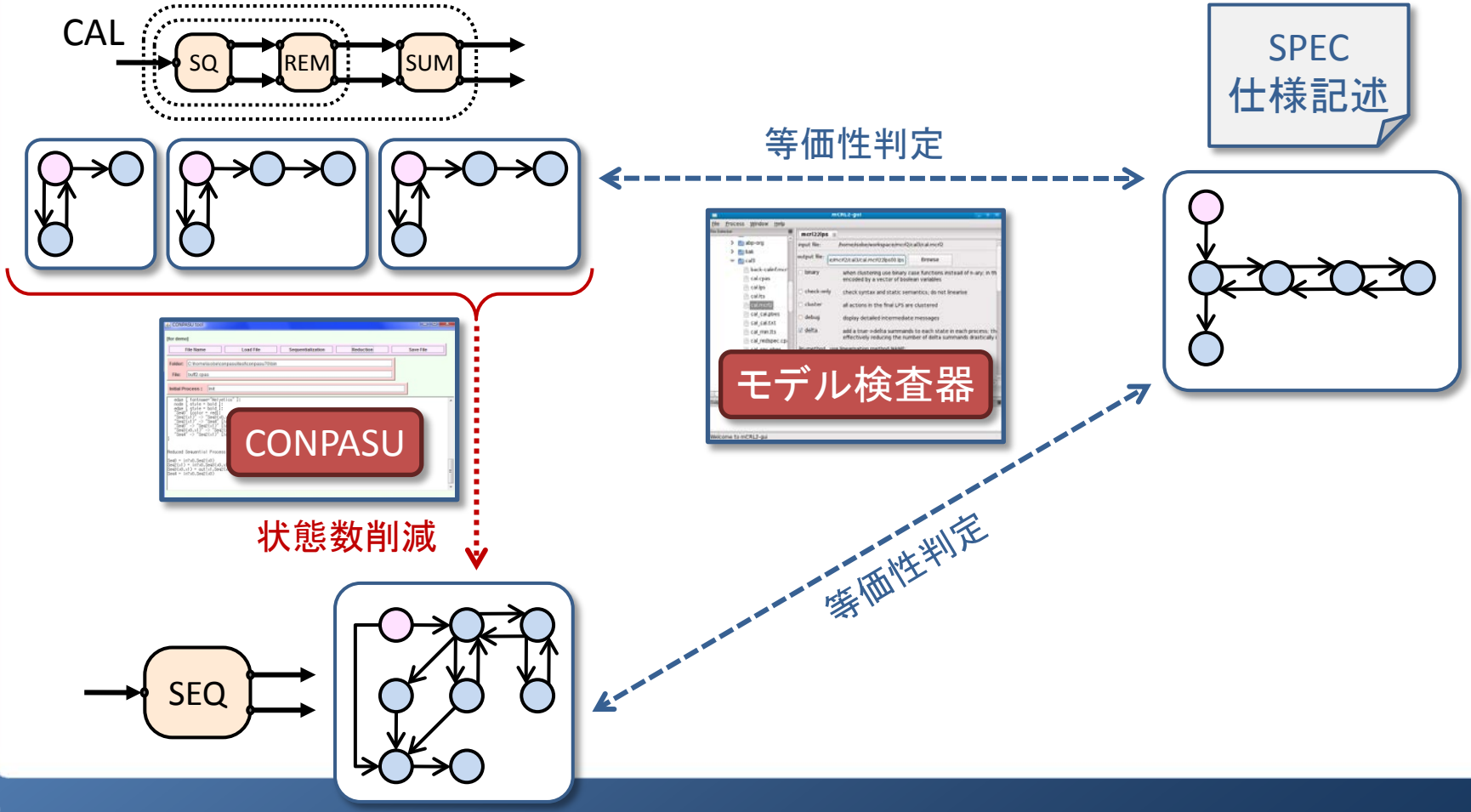
状態数：656
遷移数：3,056

ltsview



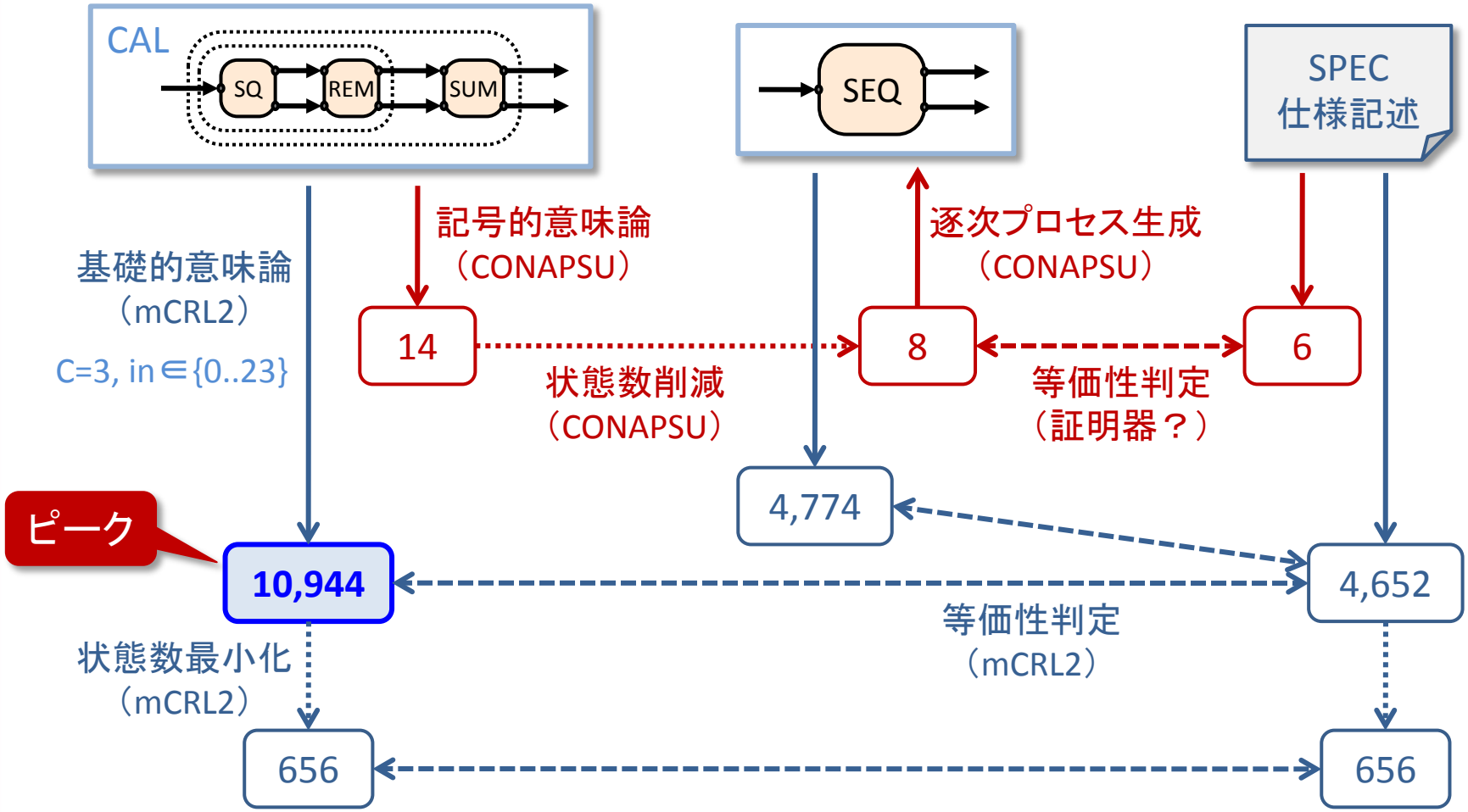
モデル検査の前処理器として

- モデル検査器は状態数有限な並行プロセスと仕様記述の等しさを自動判定できる。
  ⇒ モデル検査器で判定する前にCONPASUを状態数削減器として利用できる。



等価性判定に必要な状態数の例

■ 記号処理(CONPASU)によって等価性判定に必要な状態数のピーク値を減らせる。



まとめ：現状と今後の課題

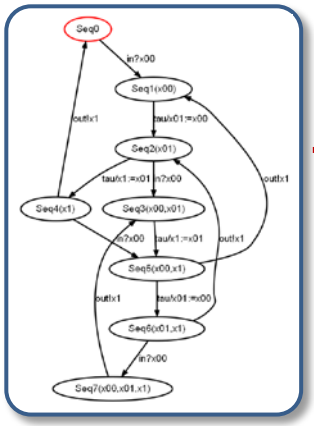
■ CONPASUの機能

- 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：弱等価性を保存したまま状態数を削減

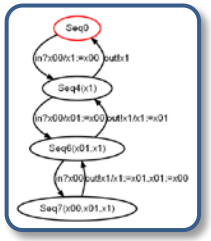
並行プロセス（CCS記述）

```
B01 = (B0|B1|B2)＼{com1,com2}
 B0 = in?x. com1!x. B0
 B1 = com1?x. com2!x. B1
 B2 = com2? outx.!x. B3
```

逐次化 →



状態数削減 ≈ →



プロセス生成 →

逐次プロセス（CCS記述）

```
Seq0 = in?x00.Seq4(x00)
Seq4(x1) = out!x1.Seq0 + in?x00.Seq6(x00,x1)
Seq6(x01,x1) = in?x00.Seq7(x00,x01,x1) + out!x1.Seq4(x01)
Seq7(x00,x01,x1) = out!x1.Seq6(x00,x01)
```

■ CONPASUの特徴

- 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）

■ 今後の課題

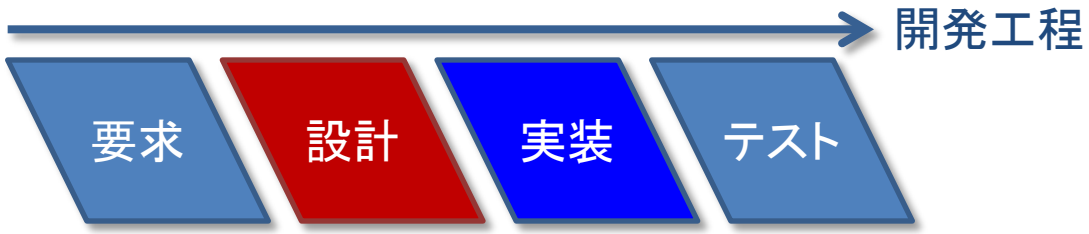
- 状態数削減機能の強化（条件付き内部アクションの削減など）。
- データ式の記号処理の強化（試作版：1+2 ≠ 2+1）。
- 解析結果（特性）の表現方法の改善。

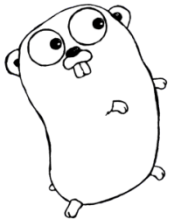
付録

- ライブラリ
- 連立方程式
- 状態遷移図表示例

補足:プロセス代数ベース並列プログラミング

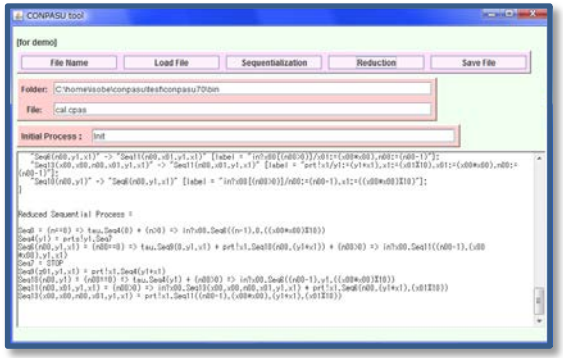
- プロセス代数ベースの様々なプログラミング言語/ライブラリが開発されている。
  ⇒ CONPASUで解析したモデルを少ないギャップで実装できる。





Go言語の公式マスコットGordon
http://golang.org/

プロセス代数ベース解析器



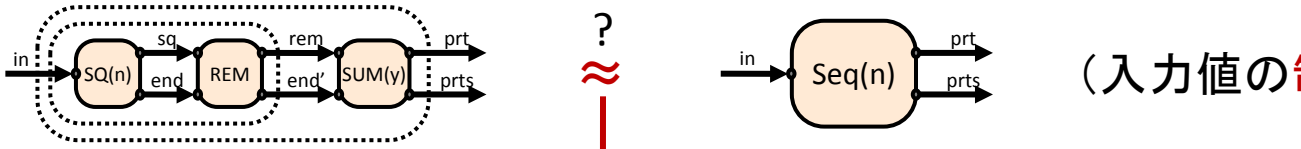
CONPASU-tool

プロセス代数ベースライブラリ/言語（同期型メッセージパッシング通信）

言語	ライブラリ	研究開発元
Java	JCSP	ケント大学（QuickStone）
C++	C++CSP	ケント大学
Haskell	CHP	ケント大学
Python	PyCSP	トロムソ大学＆コペンハーゲン大学
Python	Python-CSP	ウォルバーハンプトン大学
Go		Google
XC		XMOS（並列プロセッサ記述言語）

比較：記号的意味論による等価性判定方法

- 記号的意味論によって、2つの無限状態プロセスの弱等価性判定問題を連立方程式PBES（Parameterized Boolean Equation System）の解の存在問題に変換する方法が提案されており、mCRL2には試験的に実装されている。



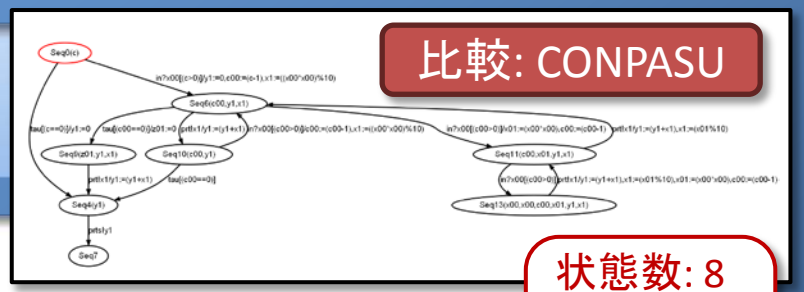
PBESへ変換　lpsbisim2pbesコマンド（mCRL2）

1/32

```
pbes nu Xms(s3_SQ: Pos, x_SQ: Nat, n_SQ: Int, s30_REM: Pos, x_REM: Nat, s31_SUM: Pos, x_SUM,y_SUM: Nat, s3_Seq0: Pos, x00_Seq0,x_Seq0,x01_Seq0: Nat, z01_Seq0,n00_Seq0: Int,
x1_Seq0,y1_Seq0: Nat, n_Seq0: Int) =
    ((((((((((val(!(s30_REM == 2 && s31_SUM == 1 && x_REM mod 10 < 10)) || Y1ms_tau(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0,
z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) && val(!(s30_REM == 3 && s31_SUM == 1)) || Y1ms_tau0(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0,
x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) && val(!(((s3_SQ == 1 && !(0 < n_SQ)) && s30_REM == 1) && s31_SUM == 3)) || Y1ms_prts(s3_SQ, x_SQ, n_SQ, s30_REM,
x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) && val(!(((s3_SQ == 1 && !(0 < n_SQ)) && s30_REM == 1) &&
s31_SUM == 2)) || Y1ms_prt(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) &&
val(!((s3_SQ == 2 && s30_REM == 1) && s31_SUM == 3)) || Y1ms_prts0(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0,
n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) && val(!((s3_SQ == 2 && s30_REM == 1) && s31_SUM == 2)) || Y1ms_prt0(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0,
x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) && val(!(s31_SUM == 3)) || Y1ms_prts1(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM,
s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) && val(!(s31_SUM == 2)) || Y1ms_prt1(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM,
y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) && (forall x0_SQ: Nat. val(!(s3_SQ == 1 && 0 < n_SQ)) || Y1ms_in1(s3_SQ, x_SQ, n_SQ,
s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0, x0_SQ))) && val(!((s3_SQ == 1 && !(0 < n_SQ)) &&
s30_REM == 1)) || Y1ms_tau1(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0)) &&
val(!(s3_SQ == 2 && s30_REM == 1)) || Y1ms_tau2(s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM, s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0,
y1_Seq0, n_Seq0)) && (((val(!(s3_Seq0 == 2)) || Y1sm_prts(s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0, s3_SQ, x_SQ, n_SQ, s30_REM, x_REM,
s31_SUM, x_SUM, y_SUM)) && (forall e10_Seq0: Enum3. val(!(C3_2(e10_Seq0, true, true, true) && C3_2(e10_Seq0, s3_Seq0 == 6 && n00_Seq0 == 0, s3_Seq0 == 4 && n00_Seq0 == 0, s3_Seq0 ==
1 && n_Seq0 == 0))) || Y1sm_tau(s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0, s3_SQ, x_SQ, n_SQ, s30_REM, x_REM, s31_SUM, x_SUM, y_SUM,
e10_Seq0))) && (forall e9_Seq0: Enum4, x000_Seq0: Nat. val(!(C4_2(e9_Seq0, true, true, true, true) && C4_2(e9_Seq0, s3_Seq0 == 7 && 0 < n00_Seq0, s3_Seq0 == 6 && 0 < n00_Seq0, s3_Seq0
== 4 && 0 < n00_Seq0, s3_Seq0 == 1 && 0 < n_Seq0))) || Y1sm_in1(s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0, s3_SQ, x_SQ, n_SQ, s30_REM, x_REM,
s31_SUM, x_SUM, y_SUM, e9_Seq0, x000_Seq0))) && (forall e4_Seq0: Enum4. val(!(C4_2(e4_Seq0, true, true, true, true) && C4_2(e4_Seq0, s3_Seq0 == 4, s3_Seq0 == 5, s3_Seq0 == 7, s3_Seq0
== 8 && 0 < n00_Seq0))) || Y1sm_prt(s3_Seq0, x00_Seq0, x_Seq0, x01_Seq0, z01_Seq0, n00_Seq0, x1_Seq0, y1_Seq0, n_Seq0,
e4_Seq0));
```

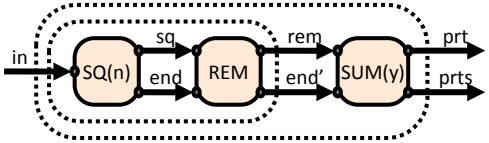
自動的には解けない…

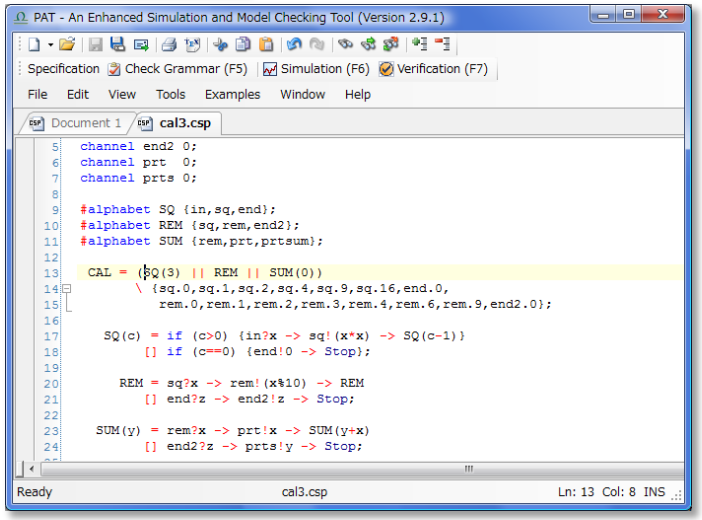
比較：状態遷移図表示（PAT）



- 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。

PATのGUI

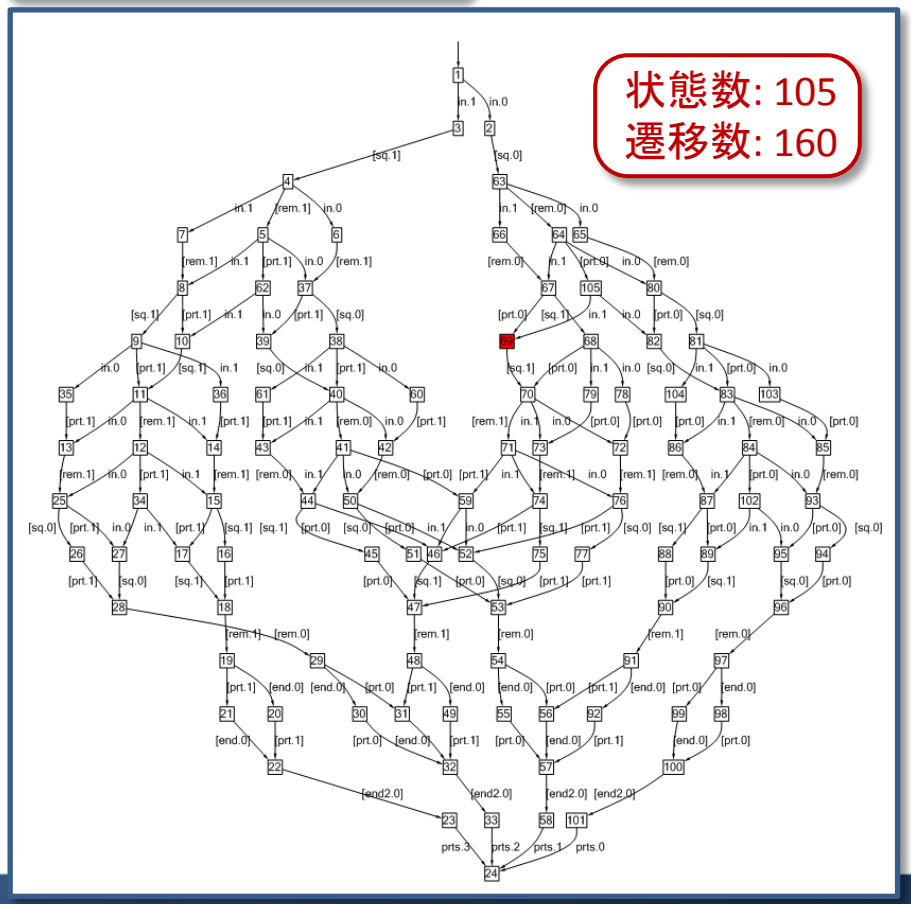




シミュレータの起動

C=3に固定, 入力値は{0,1}に有限化

CAL(3)の状態遷移図

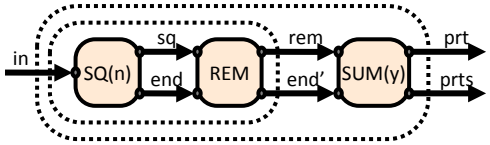


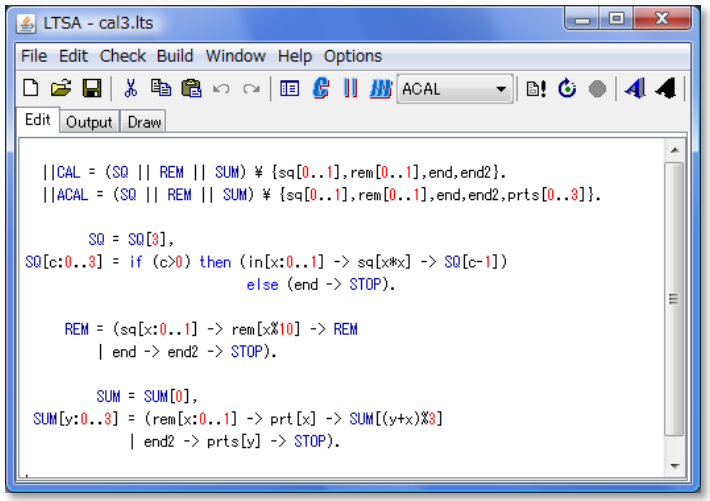
比較：状態遷移図表示（LTSA）

■ 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

C=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）