CSPプロセスの解析支援ツールの試作

～仕様の自動生成を目指して～

CONPASU-tool

(CONcurrent Process Analysis SUpport tool)

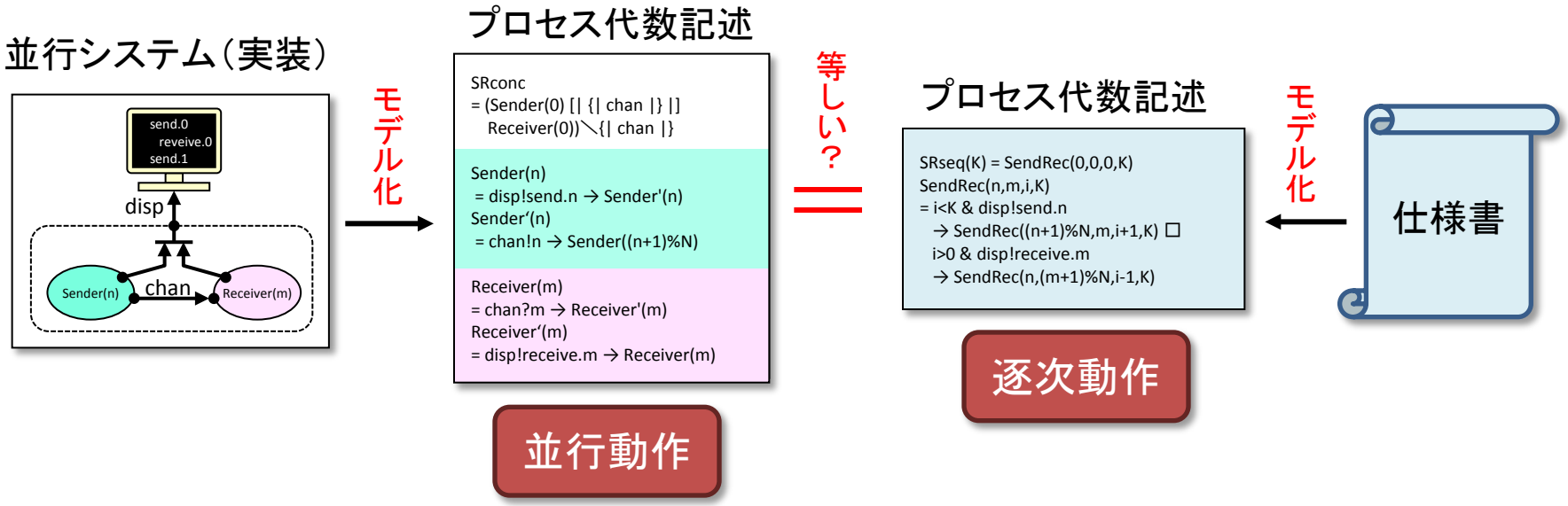
を開発中

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

第4回CSP研究会資料　2010.3.10

プロセス代数

■ **プロセス代数:**
並行システムを記述して解析するための理論



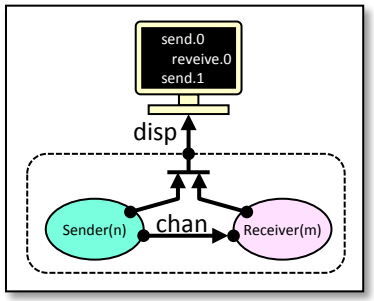
■ **代表的な基本的プロセス代数:**

- CCS (Calculus of communicating systems), R. Milner
- CSP (Communicating Sequential Processes), C.A.R Hoare
- ACP (Algebra of Communicating Processes), J.Bergstra and J.W.Klop

モデル検査器（プロセス代数系）

■ モデル検査器：等価関係や詳細化関係を判定するツール

並行システム（実装）



モデル化

プロセス代数記述

SRconc
= (Sender(0) [| {| chan |} |]
Receiver(0))＼{| chan |}

Sender(n)
= disp!send.n → Sender'(n)
Sender'(n)
= chan!n → Sender((n+1)%N)

Receiver(m)
= chan?m → Receiver'(m)
Receiver'(m)
= disp!receive.m → Receiver(m)

等しい？

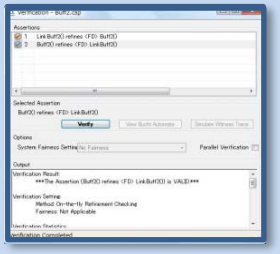
プロセス代数記述

SRseq(K) = SendRec(0,0,0,K)
SendRec(n,m,i,K)
= i<K & disp!send.n
→ SendRec((n+1)%N,m,i+1,K) □
i>0 & disp!receive.m
→ SendRec(n,(m+1)%N,i-1,K)

モデル化

仕様書

モデル検査器の例（CSP系）



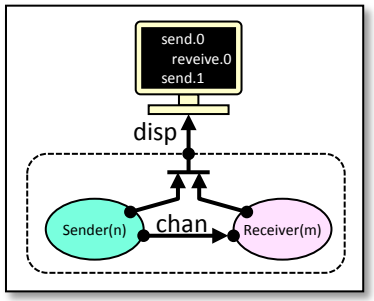
モデル検査器PAT
（シンガポール大学）

モデル検査器FDR
（オクスフォード大学）

意外に難しいのが…

■ 本来は仕様書が用意されていて、それをモデル化すればよいのだが…

並行システム（実装）



モデル化

プロセス代数記述

SRconc
= (Sender(0) [| {| chan |} |]
Receiver(0))＼{| chan |}

Sender(n)
= disp!send.n → Sender'(n)
Sender'(n)
= chan!n → Sender((n+1)%N)

Receiver(m)
= chan?m → Receiver'(m)
Receiver'(m)
= disp!receive.m → Receiver(m)

等しい？

プロセス代数記述

SRseq(K) = SendRec(0,0,0,K)
SendRec(n,m,i,K)
= i<K & disp!send.n
→ SendRec((n+1)%N,m,i+1,K) □
i>0 & disp!receive.m
→ SendRec(n,(m+1)%N,i-1,K)

モデル化

仕様書

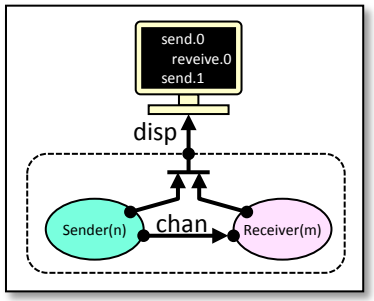
実装のモデルは比較的機械的に作成できる。

意外に仕様のモデル化は難しい。
- 必ずしも仕様が用意されていない。
- 並行動作を考慮した仕様は難しい。

並行プロセス解析支援ツール（CONPASU-tool）

■ **目標：仕様の自動生成ツールの開発**（現在は並行動作の逐次化機能まで）

並行システム（実装）



モデル化

プロセス代数記述

SRconc
= (Sender(0) [| {| chan |} |]
Receiver(0))＼{| chan |}

Sender(n)
= disp!send.n → Sender'(n)
Sender'(n)
= chan!n → Sender((n+1)%N)

Receiver(m)
= chan?m → Receiver'(m)
Receiver'(m)
= disp!receive.m → Receiver(m)

等しい

プロセス代数記述

SRseq(K) = SendRec(0,0,0,K)
SendRec(n,m,i,K)
= i<K & disp!send.n
→ SendRec((n+1)%N,m,i+1,K) □
i>0 & disp!receive.m
→ SendRec(n,(m+1)%N,i-1,K)

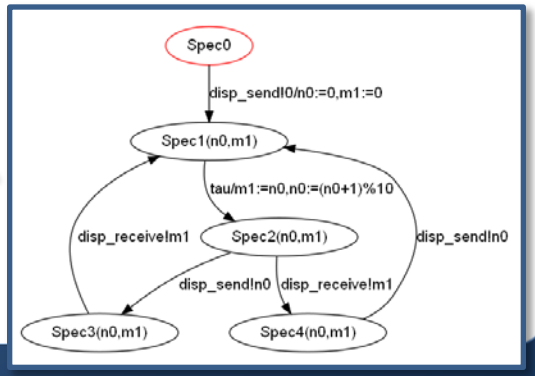
出力：逐次プロセスのFDR記述

入力：並行プロセスのFDR記述

FDR: CSPの標準的なモデル検査器
（現在はその構文の一部に対応）

開発中のツール
（CONPASU-tool）

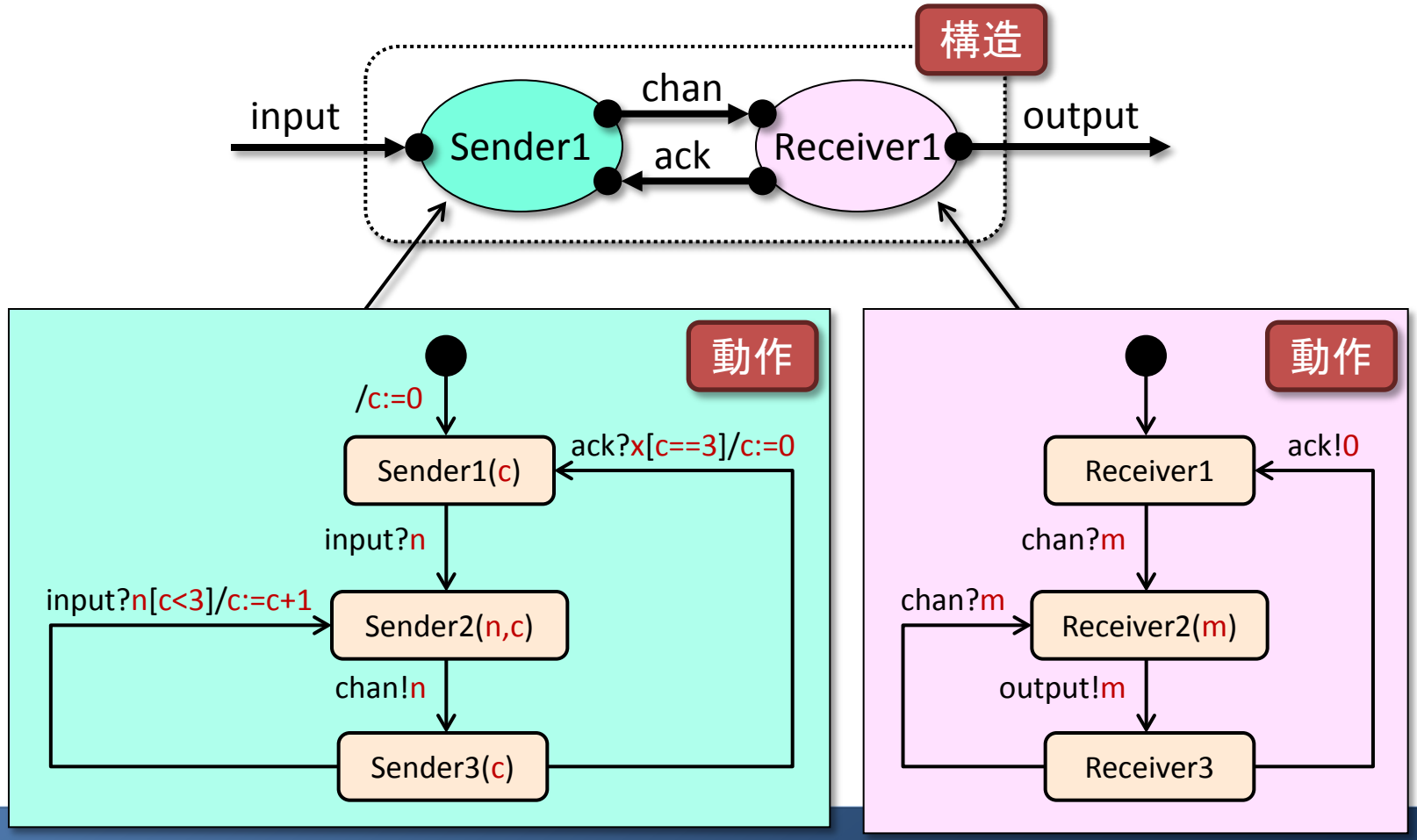
状態遷移図（Graphviz）



CONPASU-tool 適用例

■ データ転送の例

- データをチャネルinputからoutputへ転送する。
- Senderは転送3回に1回ackを待ってから次の受信（input）を行う。



並行プロセスFWのCSPモデル（FDR記述）

■ 並行システムFWの構造と動作のCSPによる記述

構造

```
FW    = (Sender1(0) [|{|chan,ack|}|] Receiver1)＼{|chan,ack|}
```

送信動作

```
Sender1(c)  = input?n → Sender2(n,c)
Sender2(n,c)  = chan!n → Sender3(c)
Sender3(c)  = (c<3) & Sender1(c+1) □ (c==3) & ack?x → Sender1(0)
```

受信動作

```
Receiver1  = chan?m → Receiver2(m)
Receiver2(m)  = output!m → Receiver3
Receiver3  = Receiver1 □ ack!0 → Receiver1
```

並行動作の逐次化

■ FWのCSPモデルをCONPASU-toolで自動的に逐次化可能。

FW = (Sender1(0) [|{|chan,ack|}|] Receiver1)＼{|chan,ack|}

Sender1(c) = input?n → Sender2(n,c)

Sender2(n,c) = chan!n → Sender3(c)

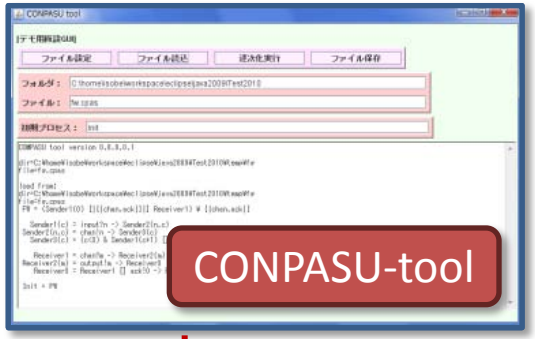
Sender3(c) = (c<3) & Sender1(c+1) □ (c==3) & ack?x → Sender1(0)

Receiver1 = chan?m → Receiver2(m)

Receiver2(m) = output!m → Receiver3

Receiver3 = Receiver1 □ ack!0 → Receiver1

入力



出力

等価な逐次プロセス

Spec0 = input?n0 → Spec1(n0,0)

Spec1(n0,c0) = tau → Spec2(c0,n0)

Spec2(c0,m1) = (c0<3) & input?n0 → Spec3(n0,(c0+1),m1) □ output!m1 → Spec4(c0)

Spec3(n0,c0,m1) = output!m1 → Spec5(n0,c0)

Spec4(c0) = (c0<3) & input?n0 → Spec5(n0,(c0+1)) □ (c0==3) & tau → Spec6(0)

Spec5(n0,c0) = tau → Spec2(c0,n0)

Spec6(c0) = input?n0 → Spec1(n0,c0)

Spec = Spec0＼{tau}

変数は変数のまま逐次化
（値で具体化されない！）

状態遷移図の表示（Graphvizを使用）

■ FWの動作を状態遷移図によっても確認できる。

FW = (Sender1(0) [|{|chan,ack|}|] Receiver1)＼{|chan,ack|}

Sender1(c) = input?n → Sender2(n,c)

Sender2(n,c) = chan!n → Sender3(c)

Sender3(c) = (c<3) & Sender1(c+1) □ (c==3) & ack?x → Sender1(0)

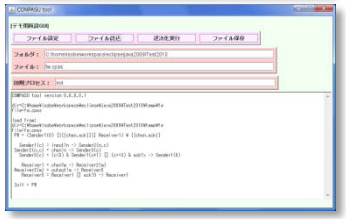
Receiver1 = chan?m → Receiver2(m)

Receiver2(m) = output!m → Receiver3

Receiver3 = Receiver1 □ ack!0 → Receiver1

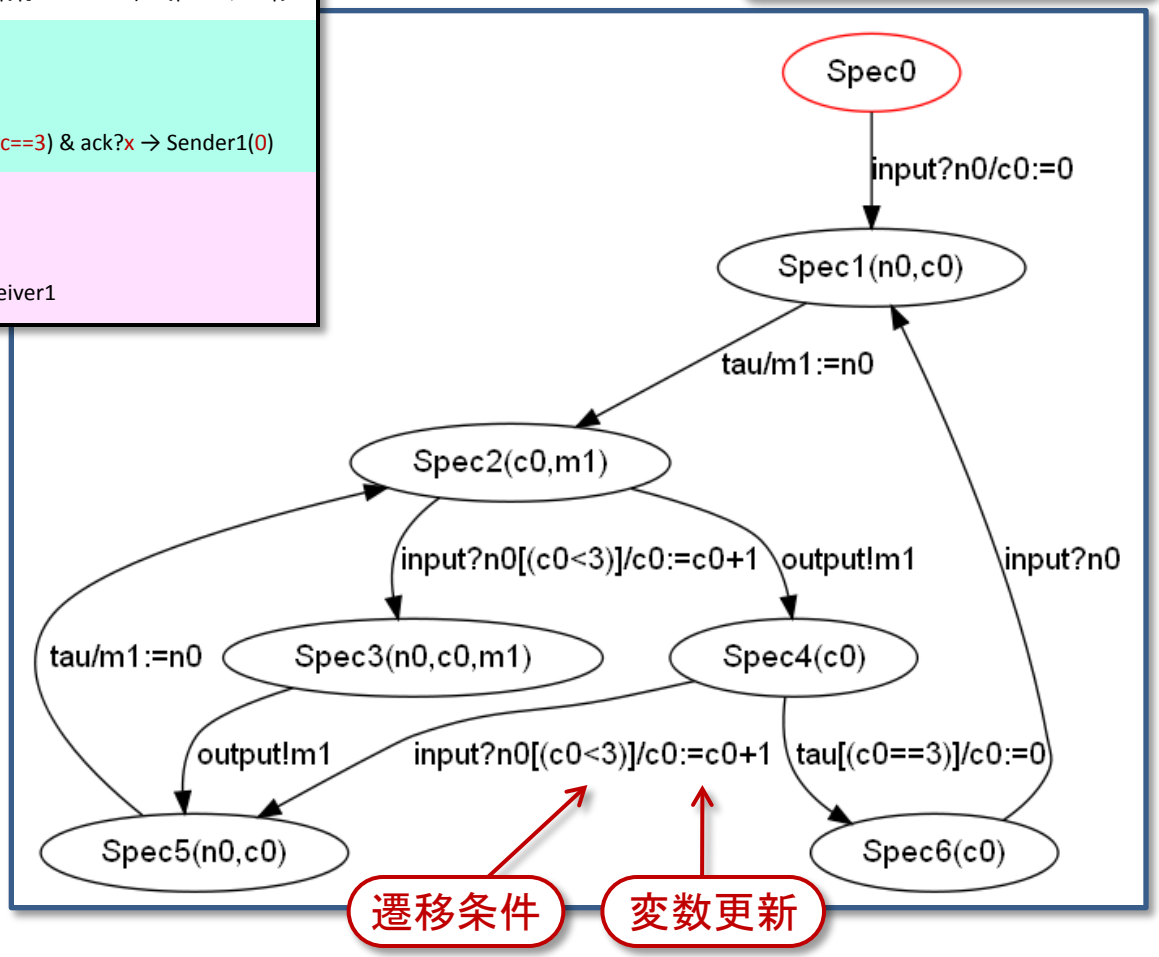
入力

CONPASU-tool



出力

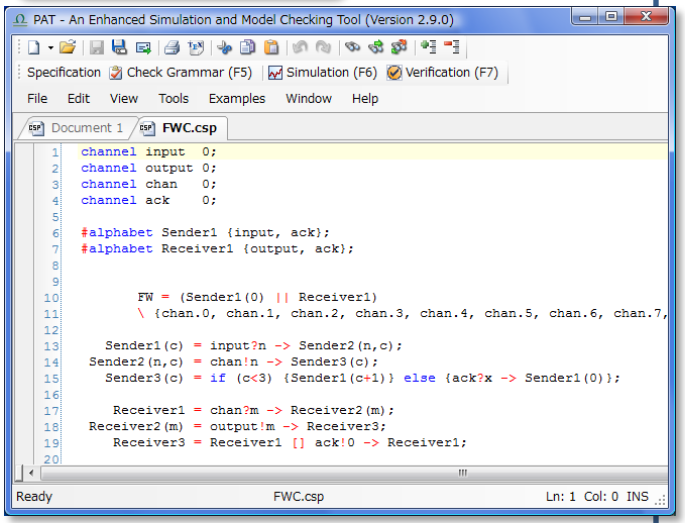
FWの状態遷移図



比較：モデル検査器PAT

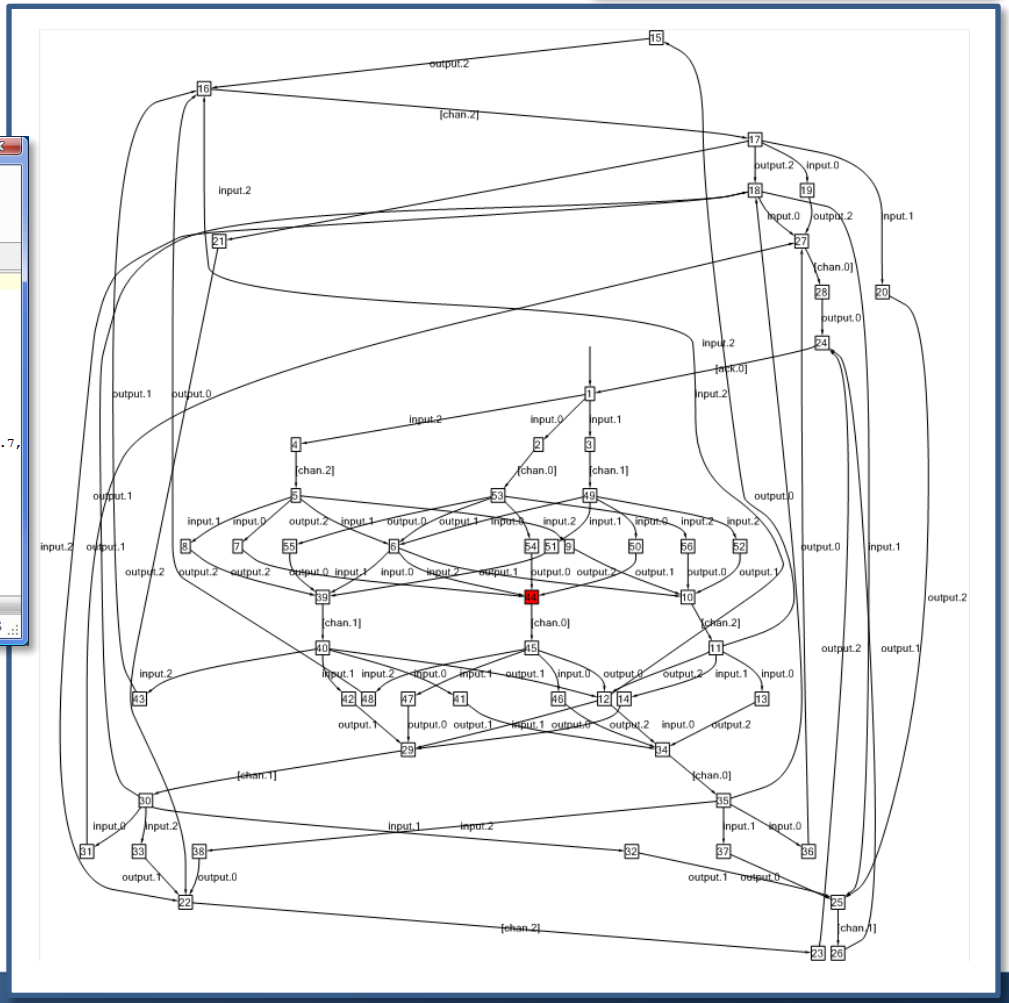
- **PATでも状態遷移図を表示できる。**
  変数は全て具体化されるため状態数が多くなる。

PATのエディタ



シミュレータの起動

FWの状態遷移図

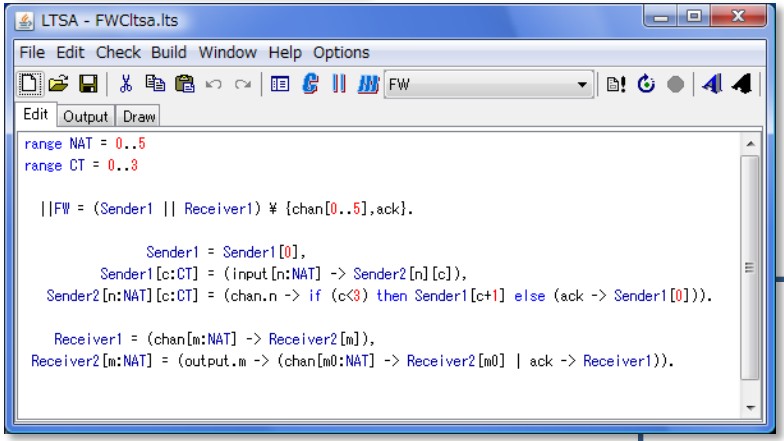


データは[0..2]に有限化

比較：モデル検査器LTSA

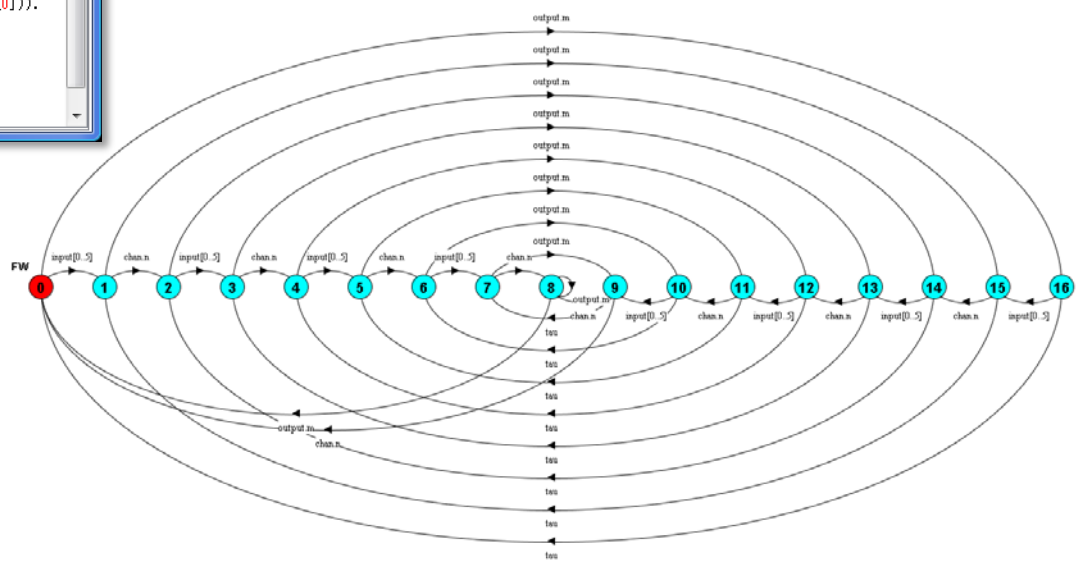
- **LTSAでも状態遷移図を表示できる。**
  変数は具体化されるが、状態数を最小化する機能がある。

LSTAのエディタ



Drawで表示

FWの状態遷移図
（状態数最小化済）

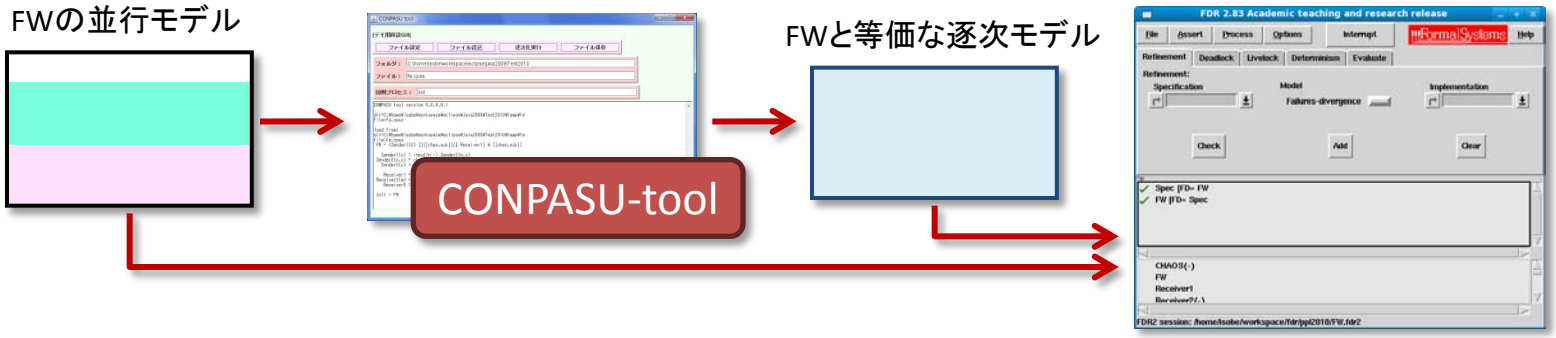


データは[0..5]に有限化

まとめと今後の課題

■ 現状

- FDR記述（一部）を読み込み、逐次化する機能を実装済（Javaで4,000行程度）。



実際に等価であること
をFDRで確認できる。

- 特徴：変数を値で具体化（計算）せずに展開している。

  長所：状態数を減らせる（可読性を高められる）。

  短所：条件付遷移の実行可能性を知るには動作をトレースする必要がある。

■ 今後の課題

- 内部遷移を考慮した状態数の削減（公理系を用いて変数を具体化せずに…）。
- 着目したイベントの関係を表す部分的な仕様の自動生成。
- 詳細化関係など、通常のモデル検査機能。