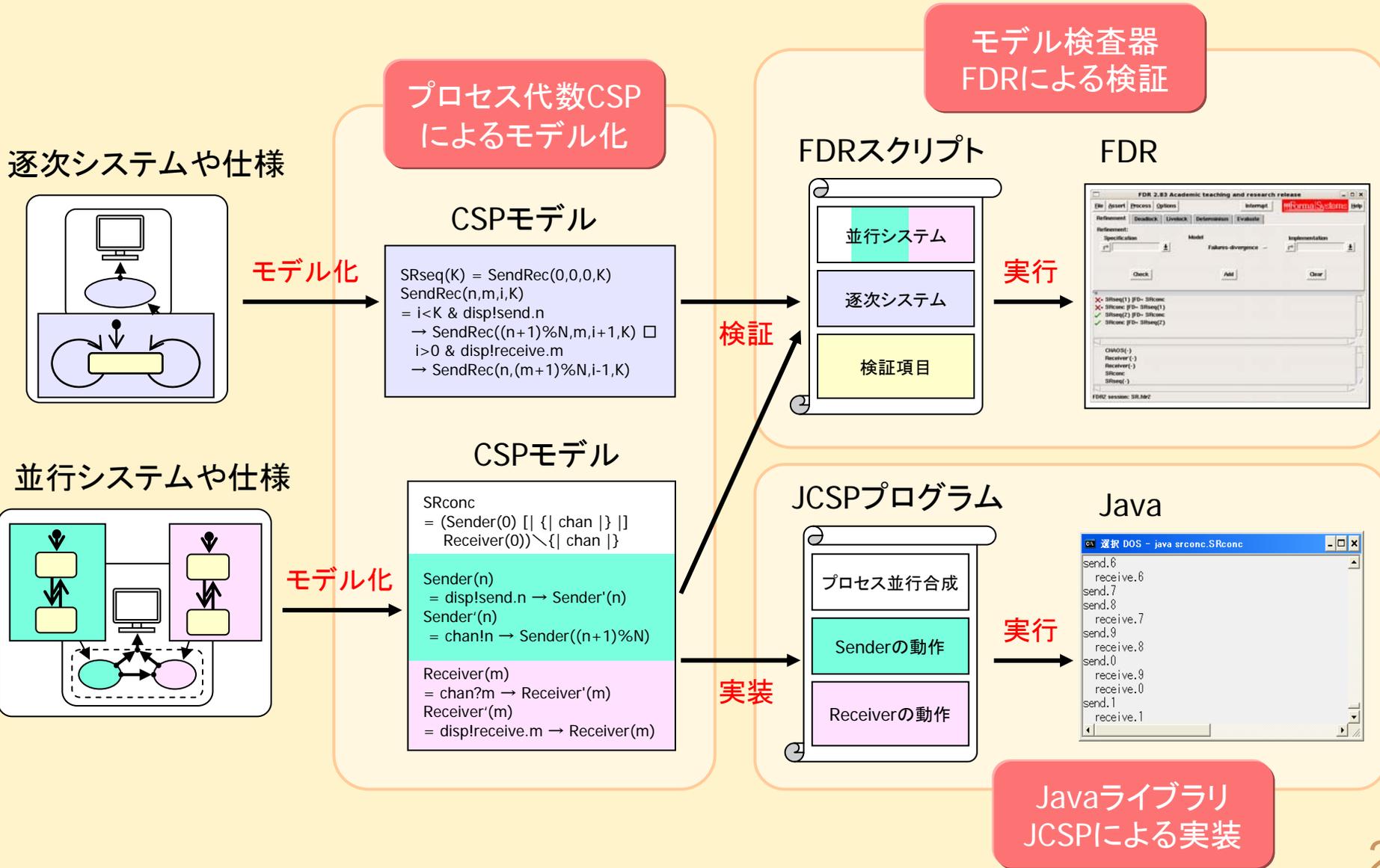


# プロセス代数CSPのモデル検査器FDRの紹介

磯部 祥尚

産業技術総合研究所AIST(ベンチャー開発センターに出向中) &  
国立情報学研究所NII(トップエスイーでCSP関係の講座を担当)

# 講義：並行システムの検証と実装



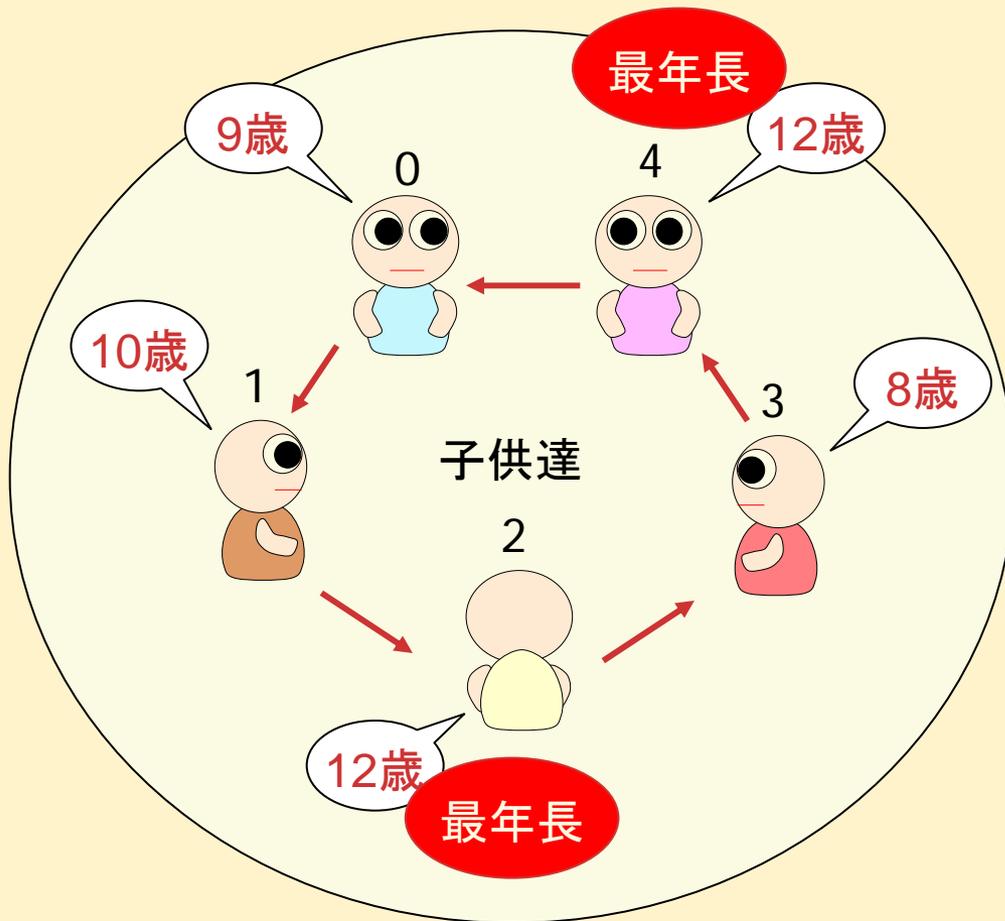
- (1) この講演で用いる例題(リーダー決定問題)の紹介
- (2) CSP(FDR)によるモデル化の例
- (3) FDRによる検証例
- (4) FDRによるデバッグの例
- (5) JCSPによる実装例



## リーダー決定問題

# リーダー決定問題(最年長者は誰?)

問題: 輪になって座ったM人の子供たちから最年長者(複数可)を探す。

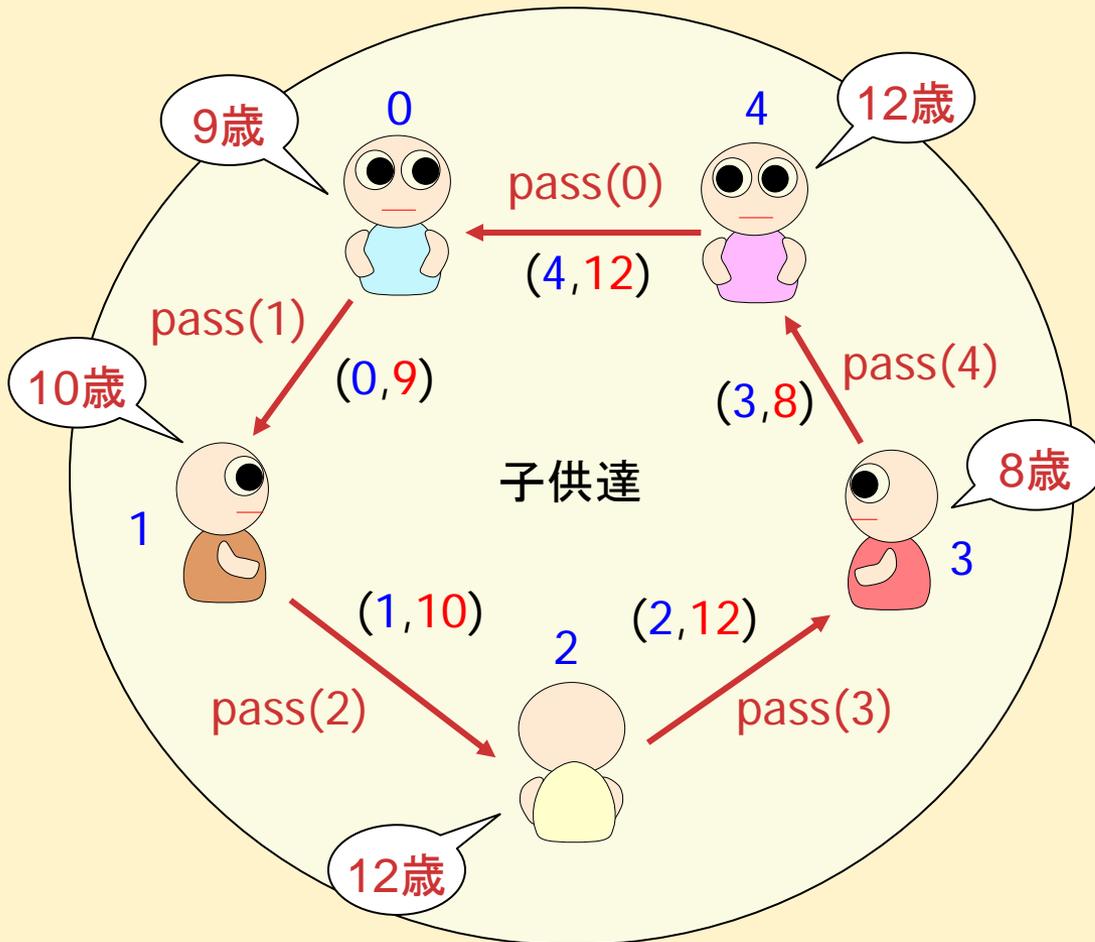


## 条件

- (1) 各子供は**右隣の子供**にのみ話しかけることができる。
- (2) 各子供は自分の名前(ID)と年齢の他に、**同時に2人まで**他人の名前(ID)と年齢を覚えることができる。
- (3) 大小の判断はできる。

# チャンネルの設定

子供たちは一方向のpassチャンネルでつながれている。

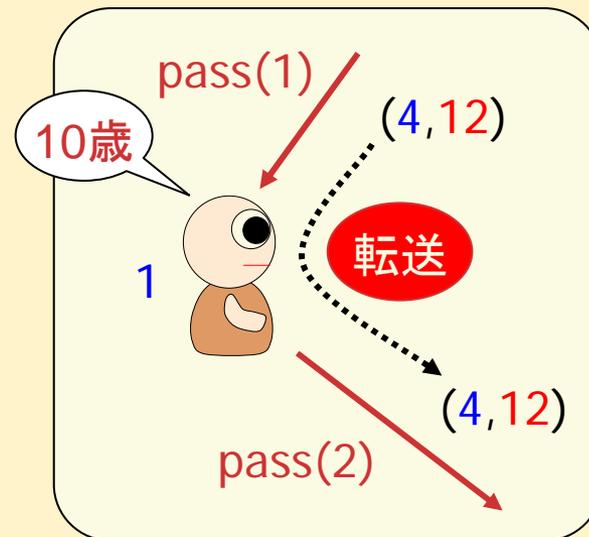
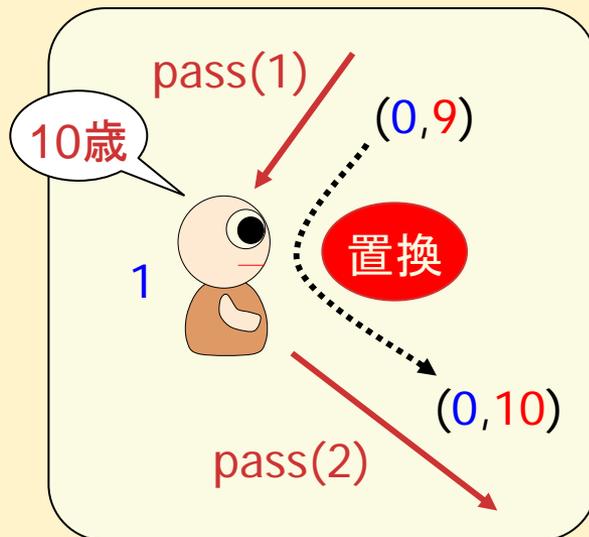


- IDがidの左側のチャンネル:  
pass(id)
- IDがidの右側のチャンネル:  
pass((id+1)%N)
- passで送受信するデータ:  
(ID, 年齢)

# データの転送方法

子供たちはある規則に従ってデータ転送を繰り返す。

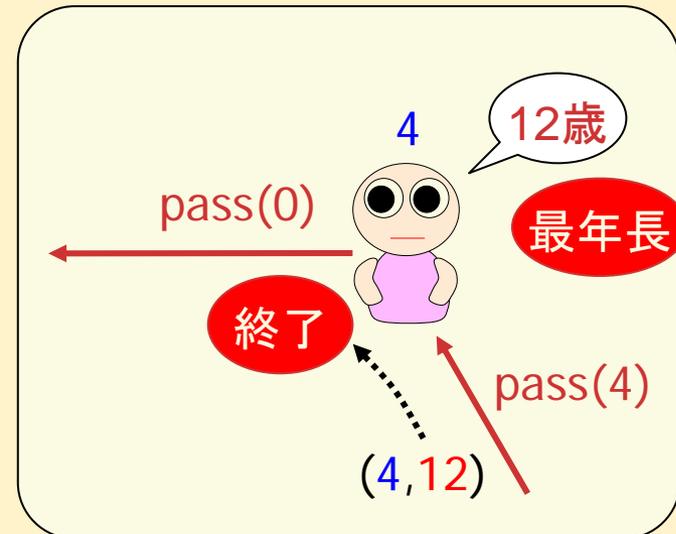
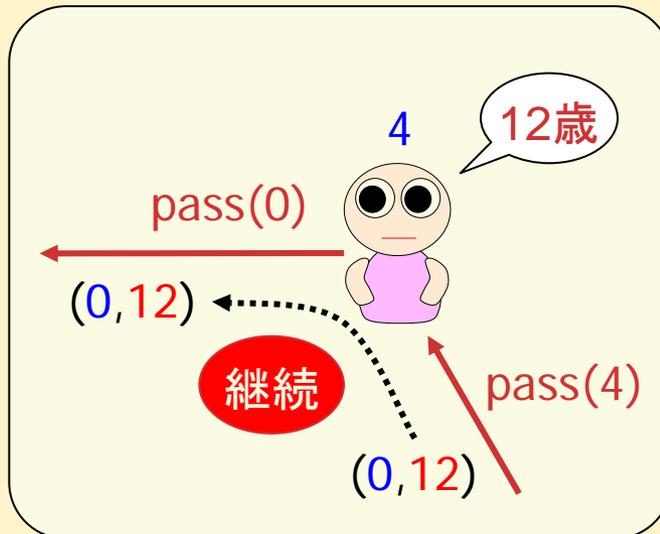
- 受取った年齢が自分より上ならばそのまま転送。
- 受取った年齢が自分より下ならば自分の年齢に置き換えて転送。



# データの転送方法

子供たちはある規則に従ってデータ転送を繰り返す。

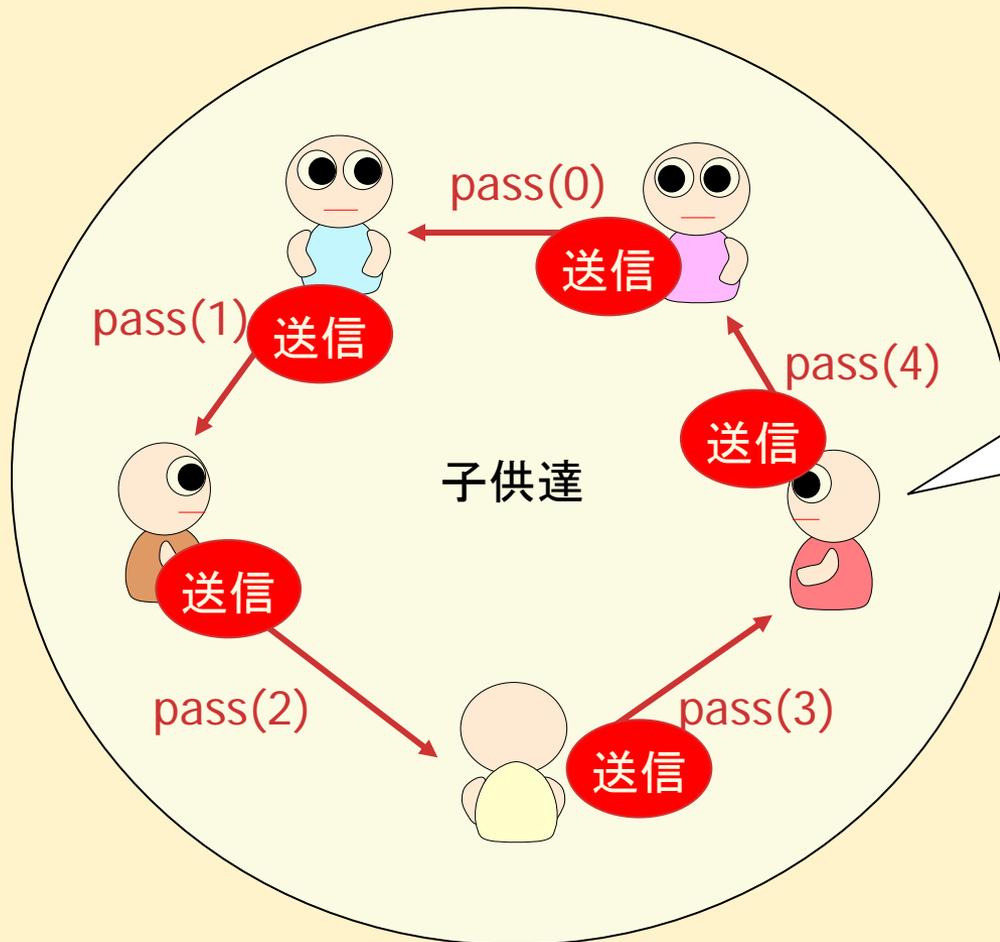
- 受取ったIDが自分のIDでないならば転送を**継続**。
- 受取ったIDが自分のIDならば転送を**終了**。  
⇒このとき受け取った年齢が自分と同じならばこれが**最大値**。



モデル化

# 各子供の動作

全員が送信してから受信をしようとすると何もできない(デッドロック)

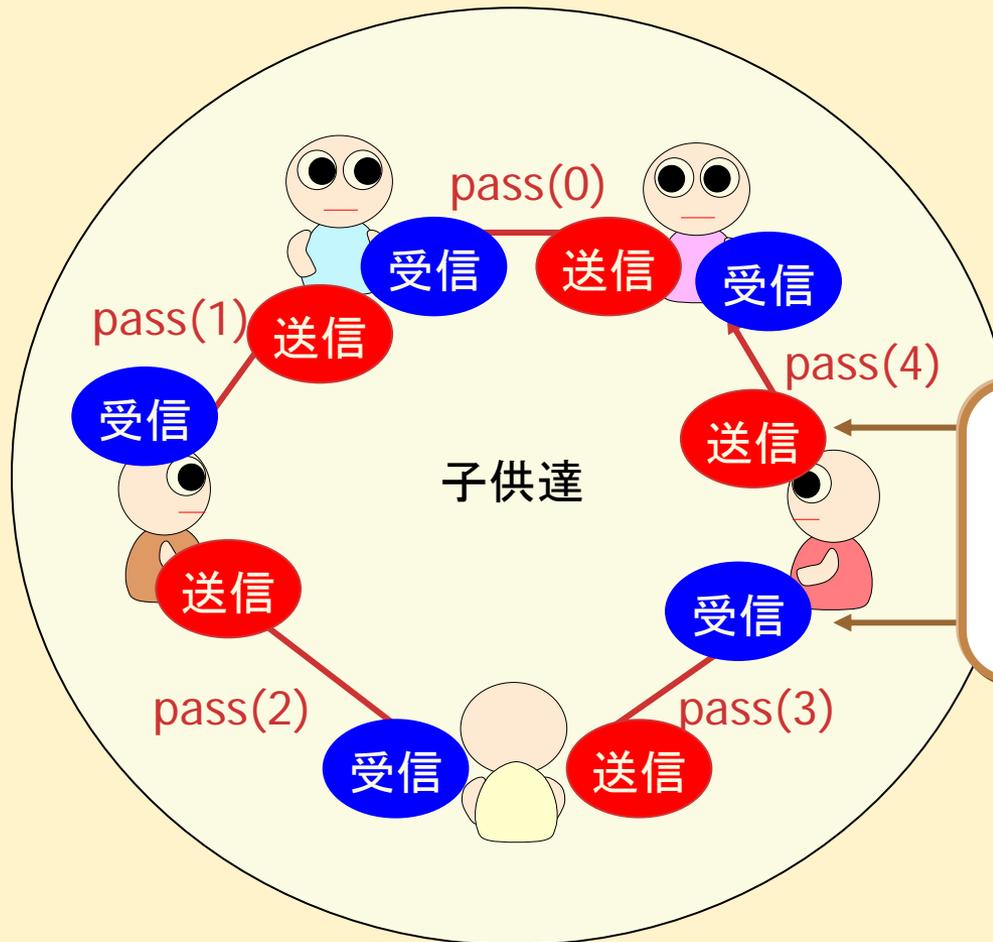


誰か最初に受信して～

passは送信と受信が同時に実行される同期型チャネルとする。

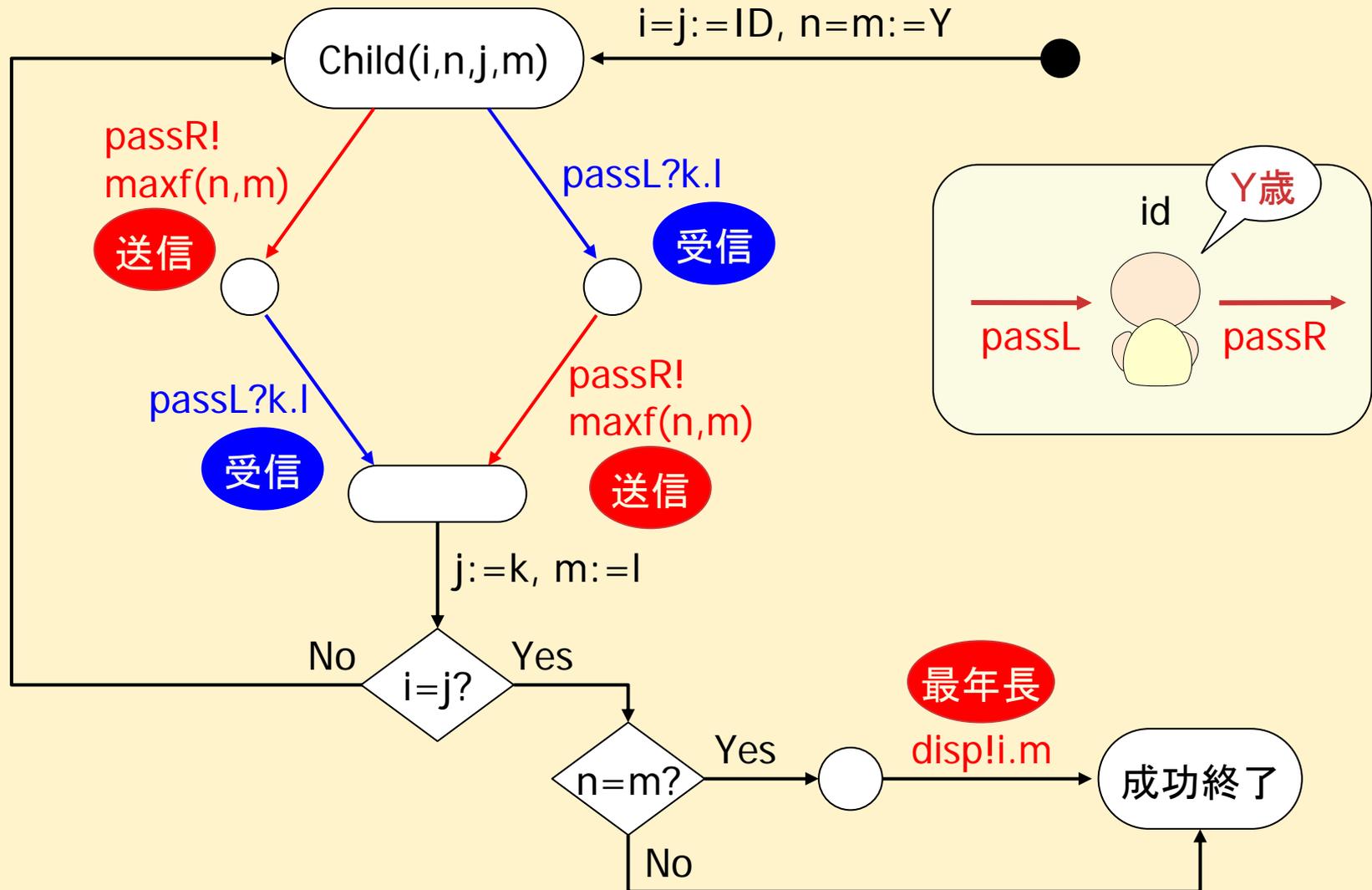
# 各子供の動作

送信か受信を選択的に実行できるようにする。

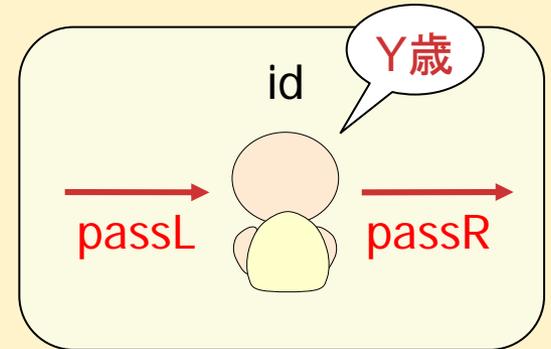
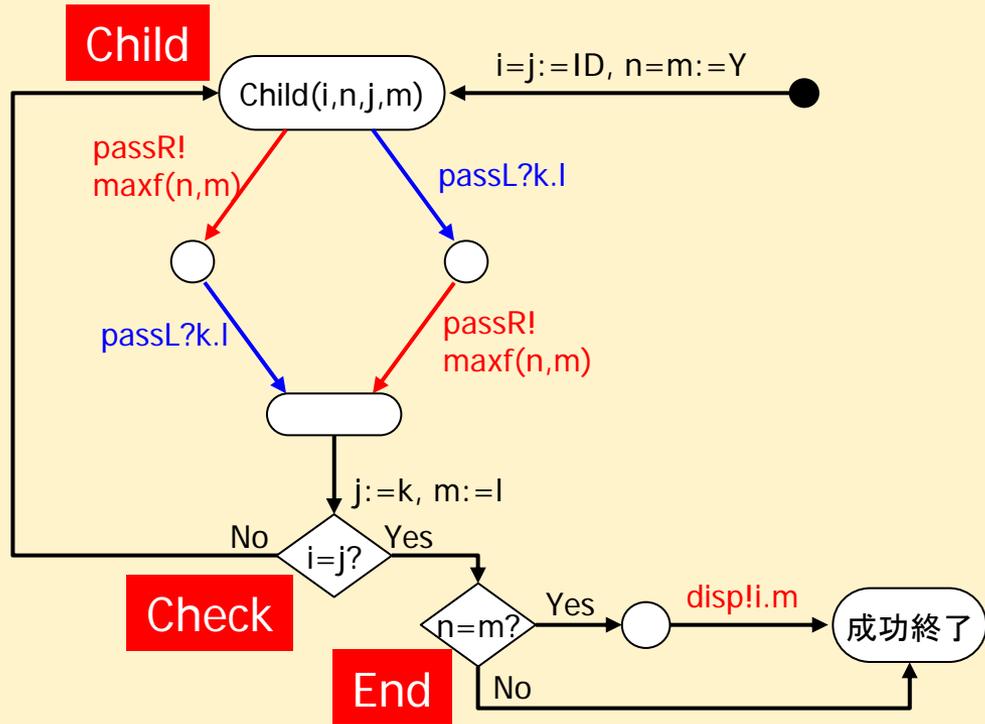


この送信と受信は外部の状況に応じて**選択的に**実行できる。  
(JCSP的にいえばAlternativeにする)

# 各子供のモデル化



# 各子供のFDRスクリプト

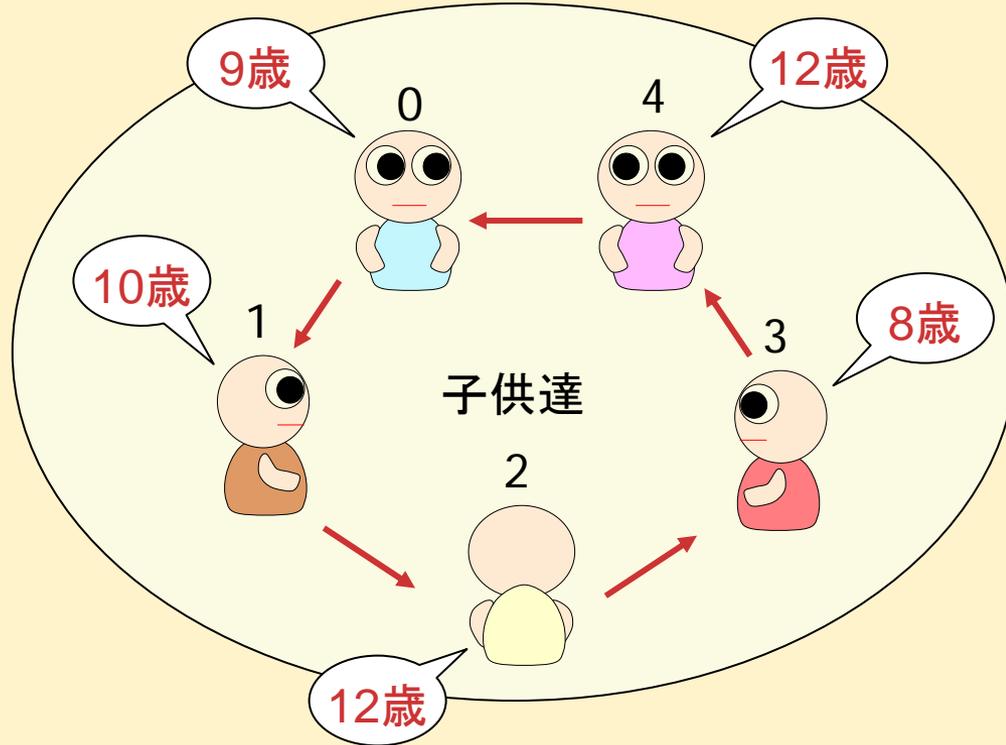


Child( $i,n,j,m$ ) =  $\text{pass.right}(i)!j.\text{maxf}(n,m) \rightarrow \text{pass.i?k.l} \rightarrow \text{Check}(i,n,k,l) []$   
 $\text{pass.i?k.l} \rightarrow \text{pass.right}(i)!j.\text{maxf}(n,m) \rightarrow \text{Check}(i,n,k,l)$

Check( $i,n,j,m$ ) = if ( $i==j$ ) then End( $i,n,m$ ) else Child( $i,n,j,m$ )

End( $i,n,m$ ) = if ( $n==m$ ) then  $\text{disp.i.m} \rightarrow \text{SKIP}$  else SKIP

# 子供達の並行合成



$$\text{Sys} = (|| i:\text{Id} @ [\text{Ev}(i)] \text{Child}(i, \text{read}(\text{list}, i), i, \text{read}(\text{list}, i))) \text{ } \text{¥} \{|\text{pass}| \}$$

FDR(CSP)の並行合記号  
(共有するチャンネル名を接続)

passの隠蔽  
(見えるのは結果dispのみ)

# 仕様のFDRスクリプト

仕様： 最年長の子供のIDを順不同で表示する。

IDが*i*の子供の年齢が最年長かを判定  
(*list*は子供全員の年齢のリスト)

```
Spec = || i:Id@[Ev2(i)] if (read(list,i) == maxl(list))  
      then disp.i.read(list,i) -> SKIP else SKIP
```

最年長の子供のIDを表示

並行合成なので複数いるときは順不同

検証

# FDRスクリプトの全体

```
-- 準備

N = 5    -- 整数の最大値
M = 5    -- 人数

Nat = {0..N} -- 整数の集合
Id = {0..(M-1)} -- 個人ID

right(i) = (i+1)%M -- 右隣のID

list = <0, 1, 4, 2, 1> -- 初期値

-- read(s,n) 列sの番めを読み出す

read(s,0) = head(s)
read(s,n) = read(tail(s),n-1)

-- maxf(n,m) nかmの大きい方を返す

maxf(n,m) = if (n > m) then n else m

-- maxl(s) 列s中の最大値を返す

maxl(<>) = 0
maxl(<n>) = n
maxl(<n>^s) = if n >= maxl(s) then n else maxl(s)

-- channel 宣言

-- pass.i.k.n 左隣からiへId付値k.nを送受信
-- disp.k.n 最大値n(kは所有者のID)を表示

channel pass : Id.Id.Nat
channel disp : Id.Nat
```

```
-- プロセス

Child(i,n,j,m) = pass.right(i)j.maxf(n,m) -> pass.i?k.l -> Check(i,n,k,l) [] -- 先に左に送信
                pass.i?k.l -> pass.right(i)j.maxf(n,m) -> Check(i,n,k,l) -- 先に右から受信

Check(i,n,j,m) = if (i==j) then End(i,n,m) else Child(i,n,j,m) -- 自分のIdが戻ったら終了

End(i,n,m) = if (n==m) then disp.i.m -> SKIP else SKIP

-- プロセスの合成

Ev(i) = { | pass.i, pass.right(i), disp.i | }
Sys = (| | i.Id @ [Ev(i)] Child(i,read(list,i),i,read(list,i)) | } ¥ { | pass | }

-- 仕様:最大値がdispから順不同に出力される

Ev2(i) = { | disp.i | }

Spec = || i.Id@[Ev2(i)]
        if (read(list,i) == maxl(list))
        then disp.i.read(list,i) -> SKIP else SKIP
```

```
-- 検証

assert Spec [FD= Sys
assert Sys [FD= Spec
```

データ部(型、関数、チャネルなど)

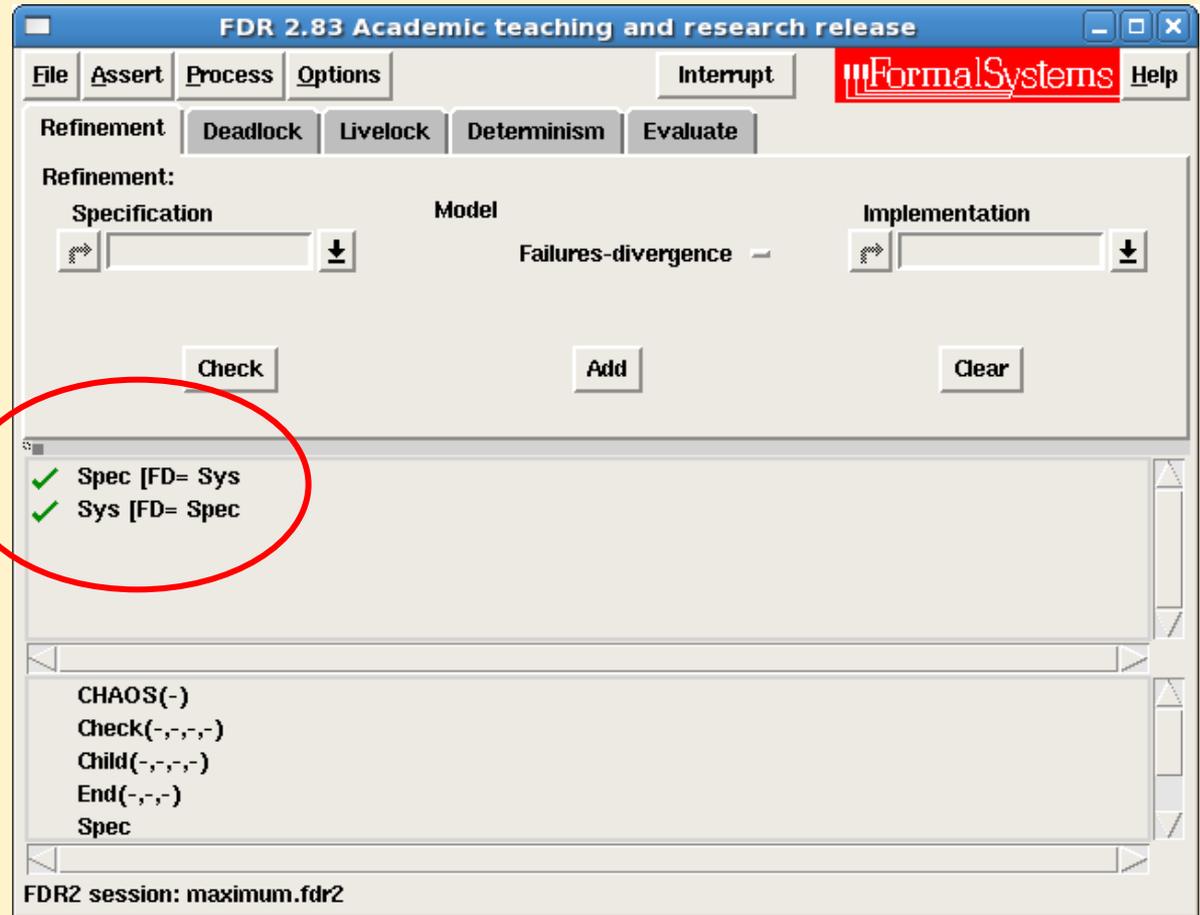
プロセス部(並行システム、仕様など)

検証部(詳細化関係、デッドロックなど)

# FDRによる検証例



入力



検証結果  
並行動作と仕様は等価

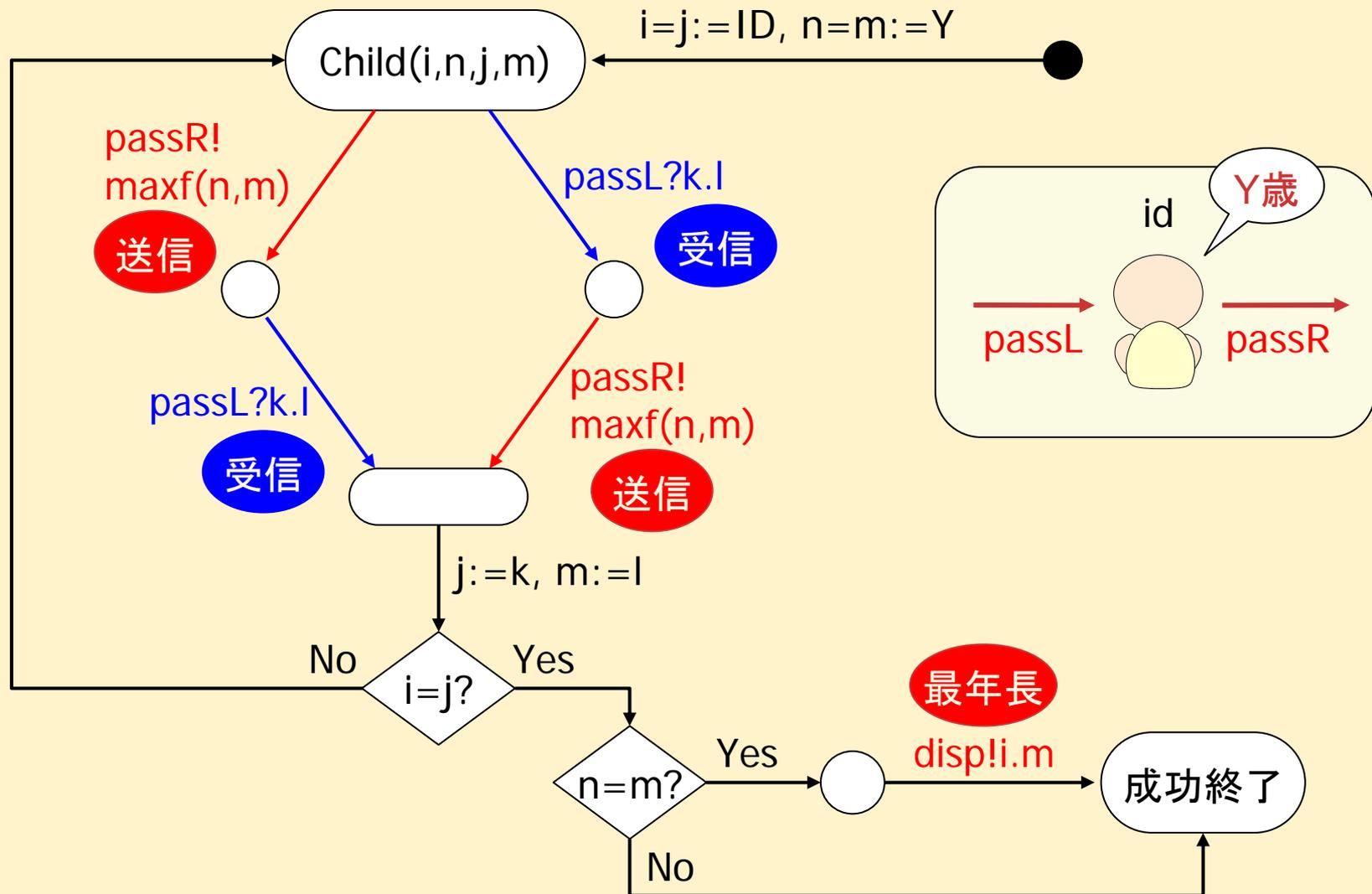
# FDRで検証可能な詳細化関係の種類と特徴

- CSPには**いくつかの種類**の詳細化関係がある(検証目的で使い分ける)。
- FDRではCSPの次の**3種類**の詳細化関係を検証できる。

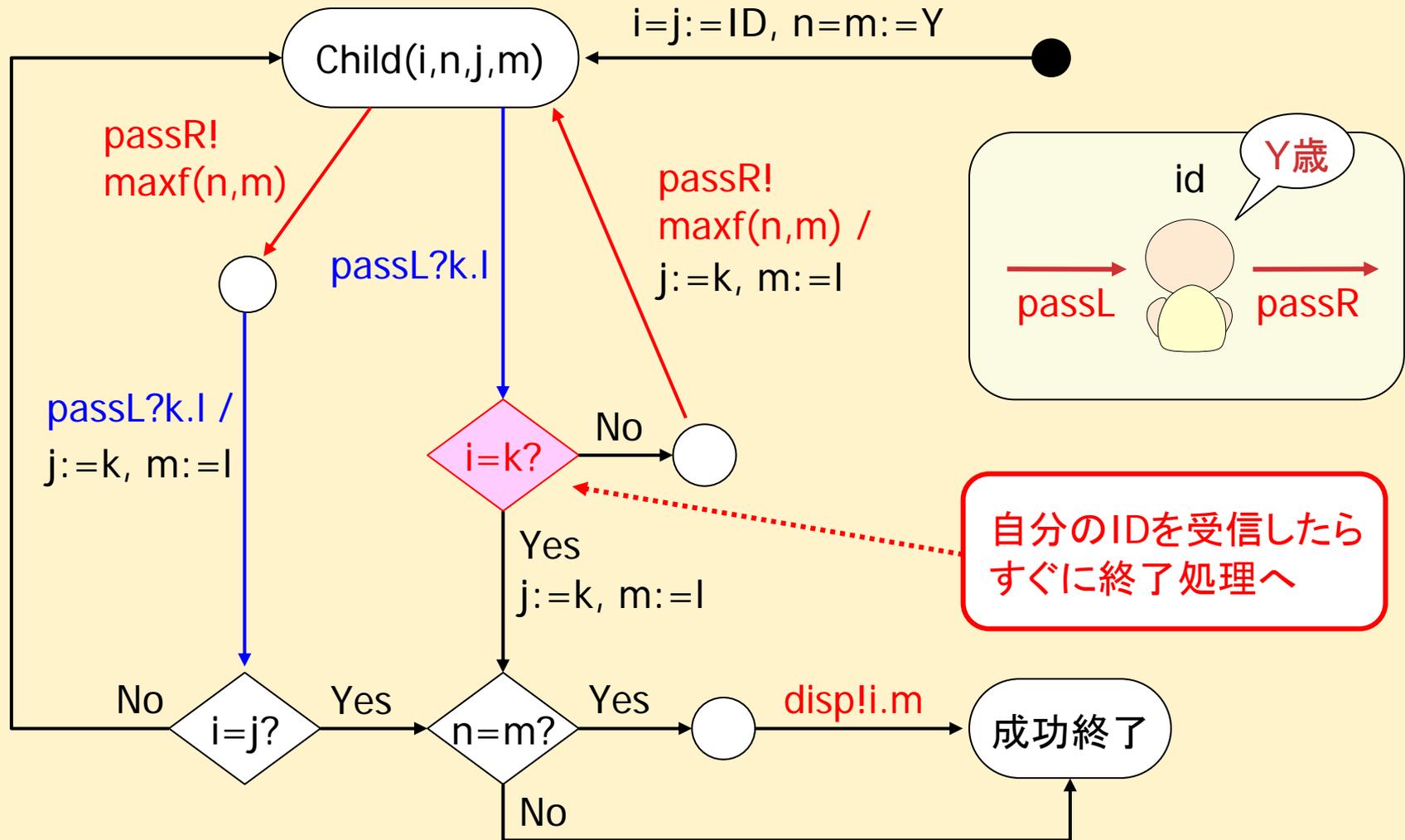
$P \sqsubseteq_T Q$	トレースに基づく最も簡単な詳細化関係: PからQへの詳細化の過程で <b>危険なトレース</b> が入り込まない。 (安全性検証)
$P \sqsubseteq_F Q$	トレースと <b>拒否</b> に基づく詳細化関係: PからQへの詳細化の過程で <b>拒否されるイベント</b> が増えない。 (デッドロックフリー性検証、活性検証)
$P \sqsubseteq_{FD} Q$	トレースと拒否と <b>内部ループ</b> に基づく詳細化関係: PからQへの詳細化の過程で <b>内部ループ</b> が増えない。 (ライブロックフリー性検証)

デバッグ

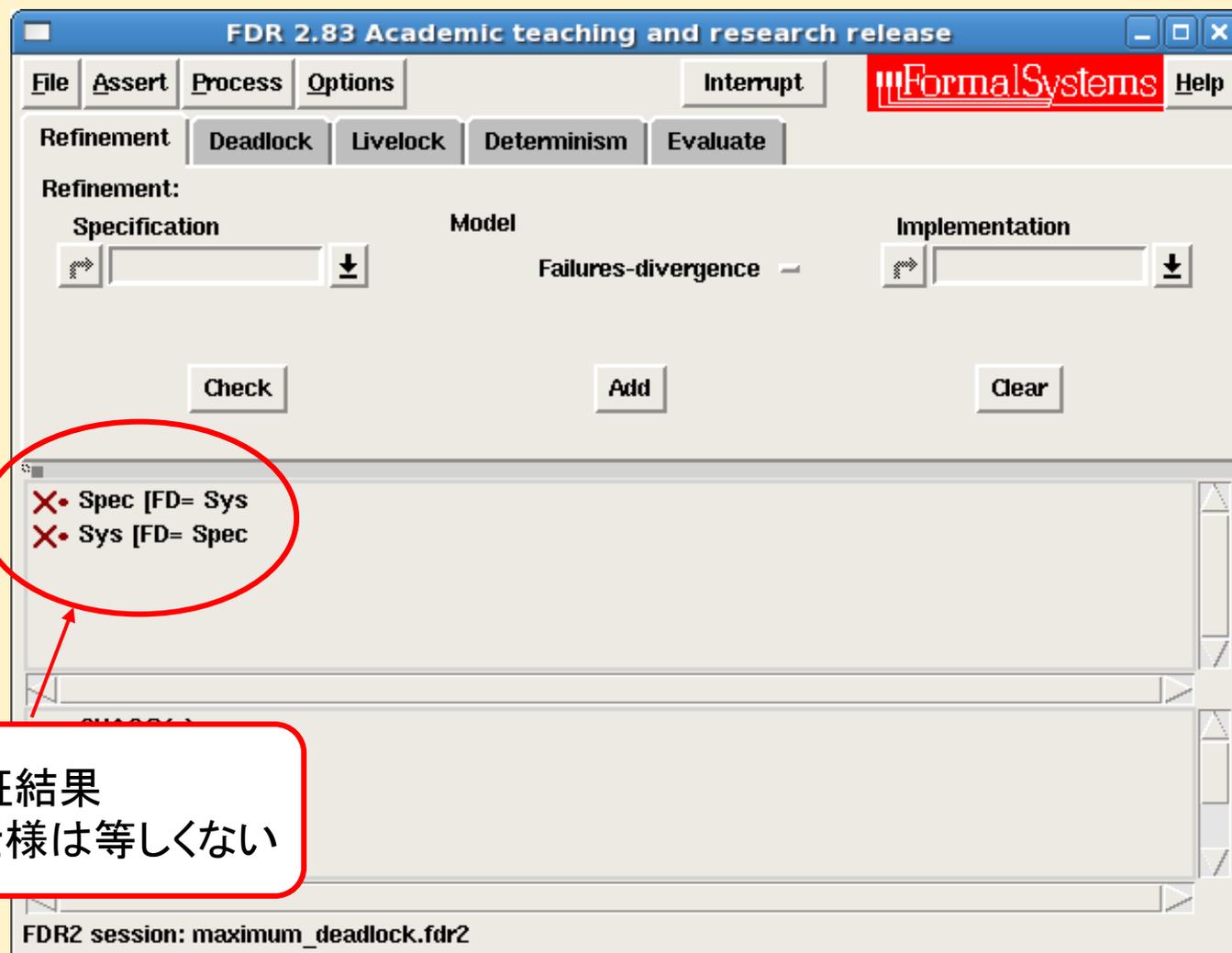
# 各子供のモデル化



# 子供のモデル化(デッドロック版)



# FDRによる検証(デッドロック版)



# デバッガの起動(デッドロック版)

pass.0から受信を待機中

The screenshot shows the FDR Debug 1 interface. At the top, there is a menu bar with 'File' and 'Help'. Below it, a dropdown menu shows 'Example 1 of 1'. The main area is divided into two panes. The left pane displays a process tree starting with 'Sys' at the root. It branches into several child processes, including 'Child(0,0,0,0)' which is highlighted with a red circle. The right pane shows a trace log for 'Child(0,0,0,0)' with the following entries: 'Performs pass.0.4.1', 'pass.1.0.0', 'pass.0.3.2', 'pass.1.4.1', 'pass.0.2.4', 'pass.1.3.2', 'pass.1.2.4', 'pass.0.1.4', and 'pass.1.1.4'. Below the trace log, there is an 'Accepts' section containing '{pass. 0}'. A red circle highlights the 'Accepts' section. At the bottom of the interface, there is a status bar with 'FDR2 det' on the left and 'Show' on the right.

File Help

Example 1 of 1

0 1

Sys

[..|..]

[..|..]

Child(2,4,2,4) Child(3,2,3,2) Child(1,1,1,1) [..|..]

Child(0,0,0,0) Child(4,1,4,1)

Child(0,0,0,0)

Performs

- pass.0.4.1
- pass.1.0.0
- pass.0.3.2
- pass.1.4.1
- pass.0.2.4
- pass.1.3.2
- pass.1.2.4
- pass.0.1.4
- pass.1.1.4

Accepts

{pass. 0}

Allowed...

Show

FDR2 det

ID=1の子供に着目

エラーに至るトレース

# デバッガの起動(デッドロック版)

The image shows the FDR Debugger interface with two windows. The main window displays a state transition tree for a system 'Sys'. The tree has a root node 'Sys' with children 'Child(2,4,2,4)', 'Child(3,2,3,2)', 'Child(1,1,1,1)', and 'Child(0,0,0,0)'. The 'Child(1,1,1,1)' node is highlighted with a red circle. A callout box points to it with the text 'ID=0の子供に着目'. To the right of the tree is a list of 'Performs' actions for 'Child(1,1,1,1)'. The 'Accepts' section is highlighted with a red circle and labeled '既に成功終了'. Below the tree, a callout box points to the 'Child(0,0,0,0)' node with the text 'ID=1の子供に着目'. A second window shows the execution trace for 'Child(0,0,0,0)'. The 'Performs' list includes 'pass.0.4.1', 'pass.1.0.0', 'pass.0.3.2', 'pass.1.4.1', 'pass.0.2.4', 'pass.1.3.2', 'pass.1.2.4', 'pass.0.1.4', and 'pass.1.1.4'. The 'Accepts' section shows '{pass. 0}'. A callout box points to the 'pass.0.1.4' action with the text 'エラーに至るトレース'. Another callout box at the top right points to the 'pass.0' accept action with the text 'pass.0から受信を待機中'.

既に成功終了

pass.0から受信を待機中

ID=0の子供に着目

ID=1の子供に着目

エラーに至るトレース

実装

# JCSPプログラムへの変換

```
-- 準備
N = 5    -- 整数の最大値
M = 5    -- 人数

Nat = {0..N} -- 整数の集合
Id = {0..(M-1)} -- 個人ID

right(i) = (i+1)%M -- 右隣のID

list = <0, 1, 4, 2, 1> -- 初期値

-- read(s,n) 列sの番めを読み出す

read(s,0) = head(s)
read(s,n) = read(tail(s),n-1)

-- maxf(n,m) nかmの大きい方を返す

maxf(n,m) = if (n > m) then n else m

-- maxl(s) 列s中の最大値を返す

maxl(<>) = 0
maxl(<n>) = n
maxl(<n>^s) = if n >= maxl(s) then n else maxl(s)

-- channel 宣言

-- pass.l.k.n 左隣からiへId付値k.nを送受信
-- disp.k.n 最大値n(kは所有者のID)を表示

channel pass : Id.Id.Nat
channel disp : Id.Nat
```

```
-- プロセス

Child(i,n,j,m) = pass.right(i)j.maxf(n,m) -> pass.l?k.l -> Check(i,n,k,l) [] -- 先に左に送信
                pass.l?k.l -> pass.right(i)j.maxf(n,m) -> Check(i,n,k,l) -- 先に右から受信

Child(i,n,j,m) = if (i==j) then End(i,n,m) else Child(i,n,j,m) -- 自分のIdが戻ったら終了

End(i,n,m) = if (n==m) then disp.i.m -> SKIP else SKIP

-- プロセスの合成

Ev(i) = { | pass.l, pass.right(i), disp.i | }
Sys = (| | i.Id @ [Ev(i)] Child(i,read(list,i),i,read(list,i)) | } { | pass | }

-- 仕様:最大値がdispから順不同に出力される

Ev2(i) = { | disp.i | }

Spec = || i.Id@[Ev2(i)]
        if (read(list,i) == maxl(list))
        then disp.i.read(list,i) -> SKIP else SKIP
```

```
-- 検証

assert Spec [FD= Sys
assert Sys [FD= Spec
```

変換

Javaプログラム  
(JCSPライブラリ使用)

この例でプログラミングは2時間程度

最初の実行でチャンネル生成を忘れて  
エラーが1つでたが、それを修正後は  
他のエラーはなし。

# JCSPプログラムの実行例(リーダー決定問題)

```
C:\ DOS
整数 ∈ [0,200], 人数 = 50

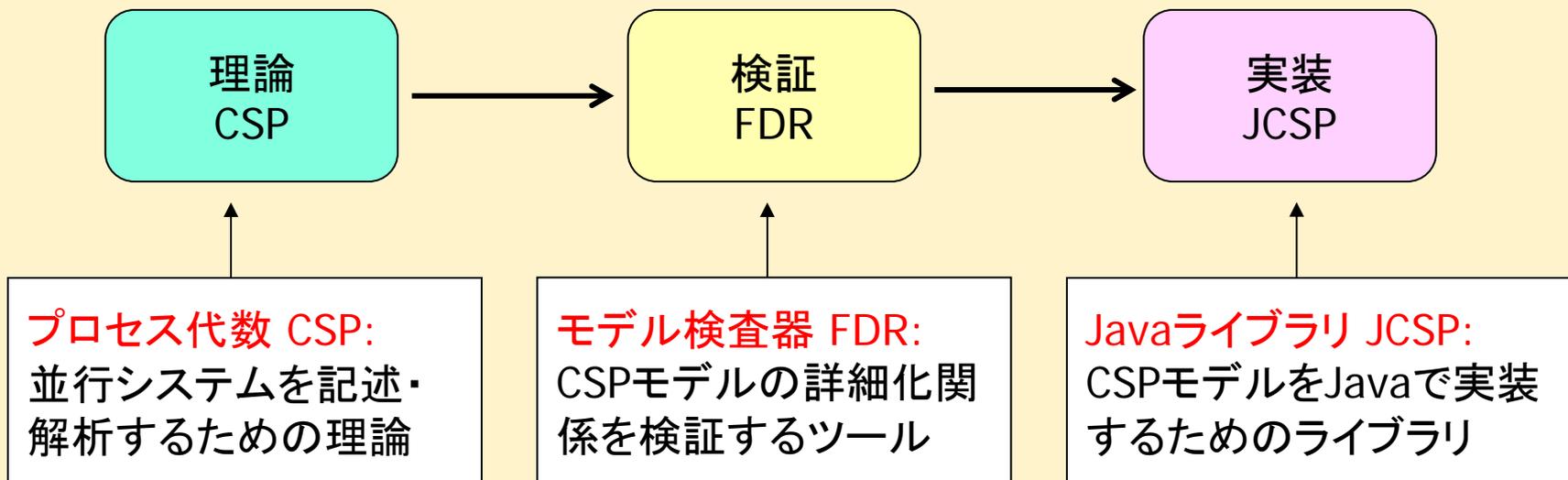
ID=0:196,      ID=1:140,      ID=2:58,      ID=3:57,      ID=4:192,
ID=5:28,       ID=6:184,      ID=7:33,      ID=8:196,     ID=9:144,
ID=10:85,      ID=11:127,     ID=12:109,    ID=13:55,     ID=14:123,
ID=15:157,     ID=16:93,      ID=17:155,    ID=18:50,     ID=19:25,
ID=20:174,     ID=21:84,      ID=22:130,    ID=23:5,      ID=24:187,
ID=25:106,     ID=26:22,      ID=27:190,    ID=28:82,     ID=29:27,
ID=30:3,       ID=31:135,     ID=32:44,     ID=33:176,    ID=34:129,
ID=35:7,       ID=36:70,      ID=37:133,    ID=38:196,    ID=39:88,
ID=40:133,     ID=41:193,     ID=42:115,    ID=43:138,    ID=44:134,
ID=45:72,      ID=46:150,     ID=47:62,     ID=48:193,    ID=49:36,

最大値
ID=0:196
ID=38:196
ID=8:196
```

まとめ

# まとめ

- (1) FDRでは**並行システムも仕様もCSPモデル**で表現する。
- (2) CSPモデル間の**詳細化関係**をFDRで検証できる。  
(デッドロック、ライブロック、決定性検証を含む)
- (3) デバッガを使ってエラーに至る**トレース**とその**エラー状態**を確認できる。
- (4) FDRで検証してからJCSP等で実装すると**致命的エラーを減らせる**。



# 付録 (FDRスクリプト)

-- 準備

P.1

N = 5 -- 整数の最大値  
M = 5 -- 人数

Nat = {0..N} -- 整数の集合  
Id = {0..(M-1)} -- 個人ID  
right(i) = (i+1)%M -- 右隣のID  
list = <0, 1, 4, 2, 1> -- 初期値

-- read(s,n) 列sの番めを読み出す

read(s,0) = head(s)  
read(s,n) = read(tail(s),n-1)

-- maxf(n,m) nかmの大きい方を返す

maxf(n,m) = if (n > m) then n else m

-- maxl(s) 列s中の最大値を返す

maxl(<>) = 0  
maxl(<n>) = n  
maxl(<n>^s) = if n >= maxl(s) then n else maxl(s)

-- channel 宣言

-- pass.i.k.n 左隣からiへId付値k.nを送受信  
-- disp.k.n 最大値n(kは所有者のID)を表示

channel pass : Id.Id.Nat  
channel disp : Id.Nat

-- プロセス

P.2

Child(i,n,j,m) =  
pass.right(i)!j.maxf(n,m) -> pass.i?k.l -> Check(i,n,k,l) []  
pass.i?k.l -> pass.right(i)!j.maxf(n,m) -> Check(i,n,k,l)

Check(i,n,j,m) = if (i==j) then End(i,n,m) else Child(i,n,j,m)

End(i,n,m) = if (n==m) then disp.i.m -> SKIP else SKIP

-- プロセスの合成

Ev(i) = { | pass.i, pass.right(i), disp.i | }  
Sys = ( | i:Id @ [Ev(i)] Child(i,read(list,i),i,read(list,i))  
¥ { | pass | } )

-- 仕様: 最大値がdispから順不同に出力される

Ev2(i) = { | disp.i | }

Spec = || i:Id@[Ev2(i)]  
if (read(list,i) == maxl(list))  
then disp.i.read(list,i) -> SKIP else SKIP

-- 検証

assert Spec [FD= Sys  
assert Sys [FD= Spec