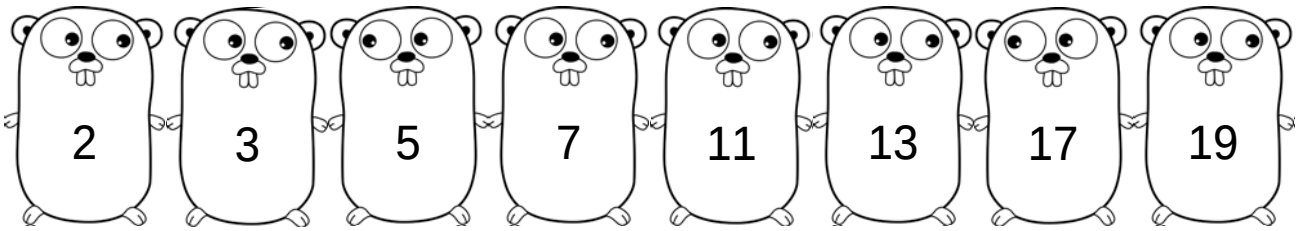
第19回CSP研究会

Gopher：Go言語のマスコット



The Gopher character is designed by Renée French.

並行処理のモデル化、検証、実装の概要

磯部 祥尚

国立研究開発法人 産業技術総合研究所 情報技術研究部門
ソフトウェアアナリティクス研究グループ

2017年5月27日

背景：Go言語の人気上昇

- 2016年後半から、Go言語（Google）の TIOBE Indexが上昇中
  - TIOBE Index：TIOBE Software社による各プログラミング言語の人気指数
  - Go言語は2016年のプログラミング言語大賞（TIOBE SW社）を受賞

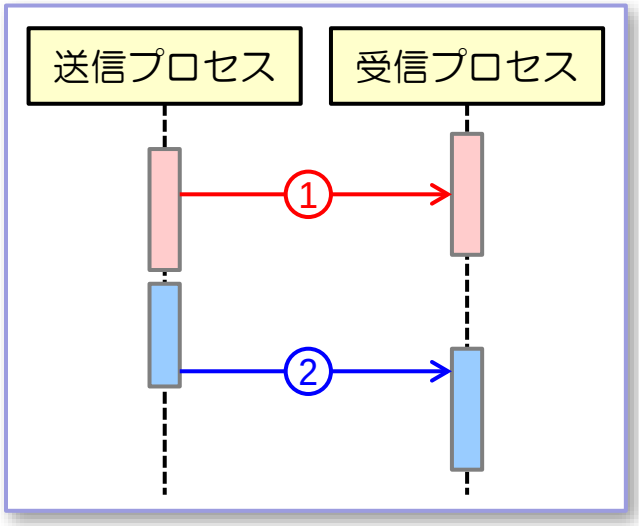
人気上昇の理由は、Go言語の学習の容易さと実用性の高さや、
最近の並列・分散環境の普及も考えられる。

背景：Go言語の並行処理モデル

- Go言語は並行処理モデルとして CSP *1 を採用：
  - チャネルを用いた同期型メッセージパッシング通信*2
  - 通信可能なチャネルの自動選択機能（select-case）

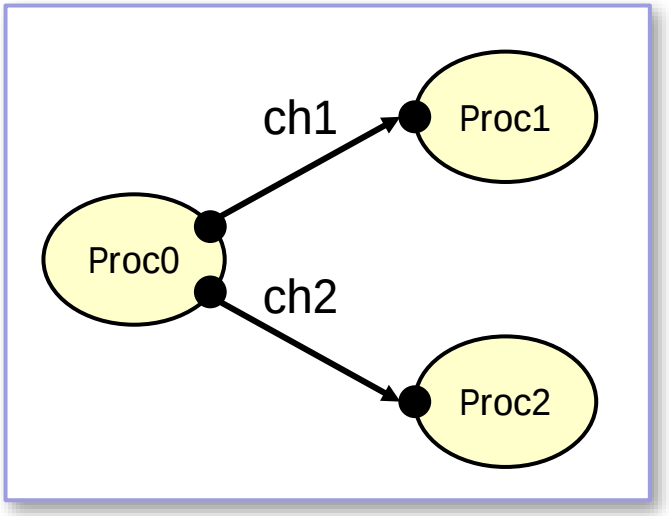
*1 CSP：並行動作の形式仕様記述言語とその解析手法（C. A. R. Hoare, Oxford大学, 1978）
*2 バッファ付チャネルも利用できるが、バッファフルの場合に送信がブロックされることが重要

送信と受信は同時



メッセージが確実に相手に伝わる
（取りこぼしがない）

通信可能なチャネルを自動選択



Proc0はProc1とProc2の状態を確認し、
受信可能な方に送信する

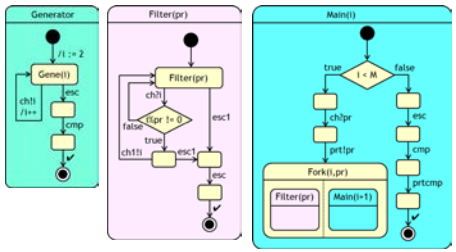
発表内容

- 並行処理の注意と対策：
  - 並行処理にはプロセス間の干渉による不具合（デッドロックなど）がある
  - そのような不具合は再現性が低いためテストによる発見が難しい

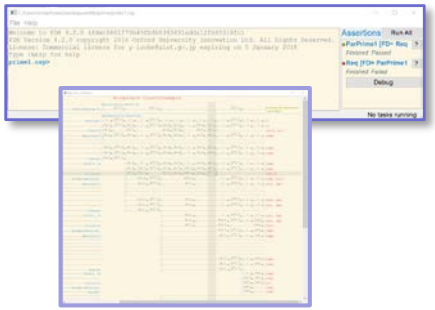
対策：プログラミングの前に並行処理の正しさを検証する

- 発表内容：
  - 並行処理をCSPによってモデル化し、モデル検査器FDRによって検証し、Go言語によって実装する方法について、素数生成器の例を用いて説明する

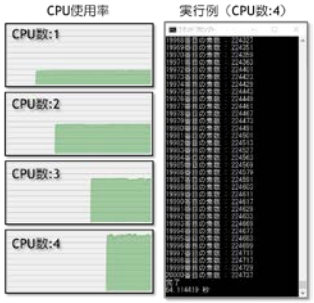
CSPによるモデル化



FDRによる検証・デバッグ

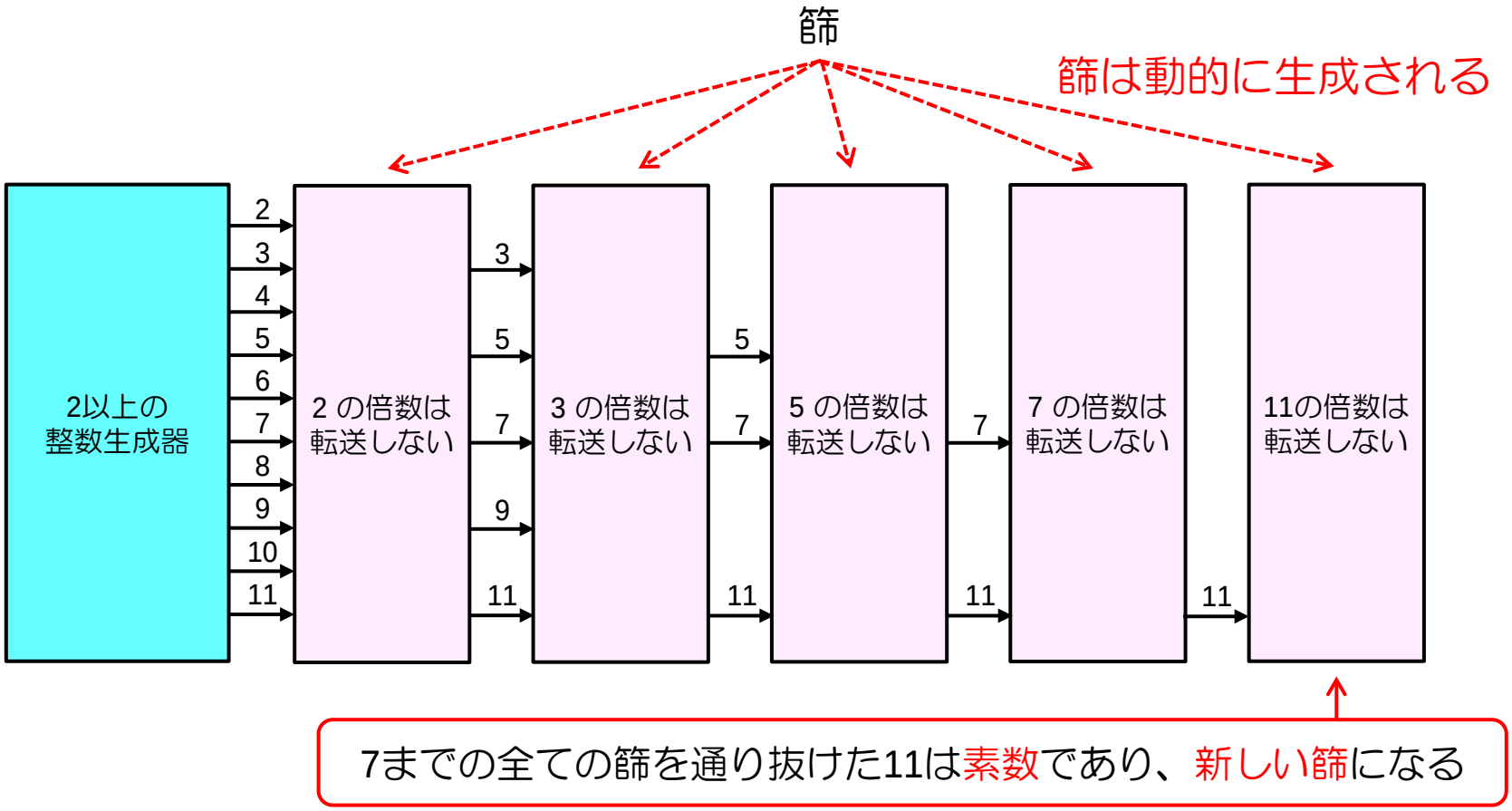


Goによる実装・実行



例：素数生成器（素数の篩）

- 素数の篩（ふるい）を例に並行処理のモデル化、検証、実装を説明する
  - 参考：Goの情報サイトのGo言語チュートリアル（part10）
    http://golang.jp/2009/11/198：素数の篩の説明
    https://play.golang.org/p/9U22NfrXeq：素数の篩のGoソースコード（今回、このコードをベースにした）

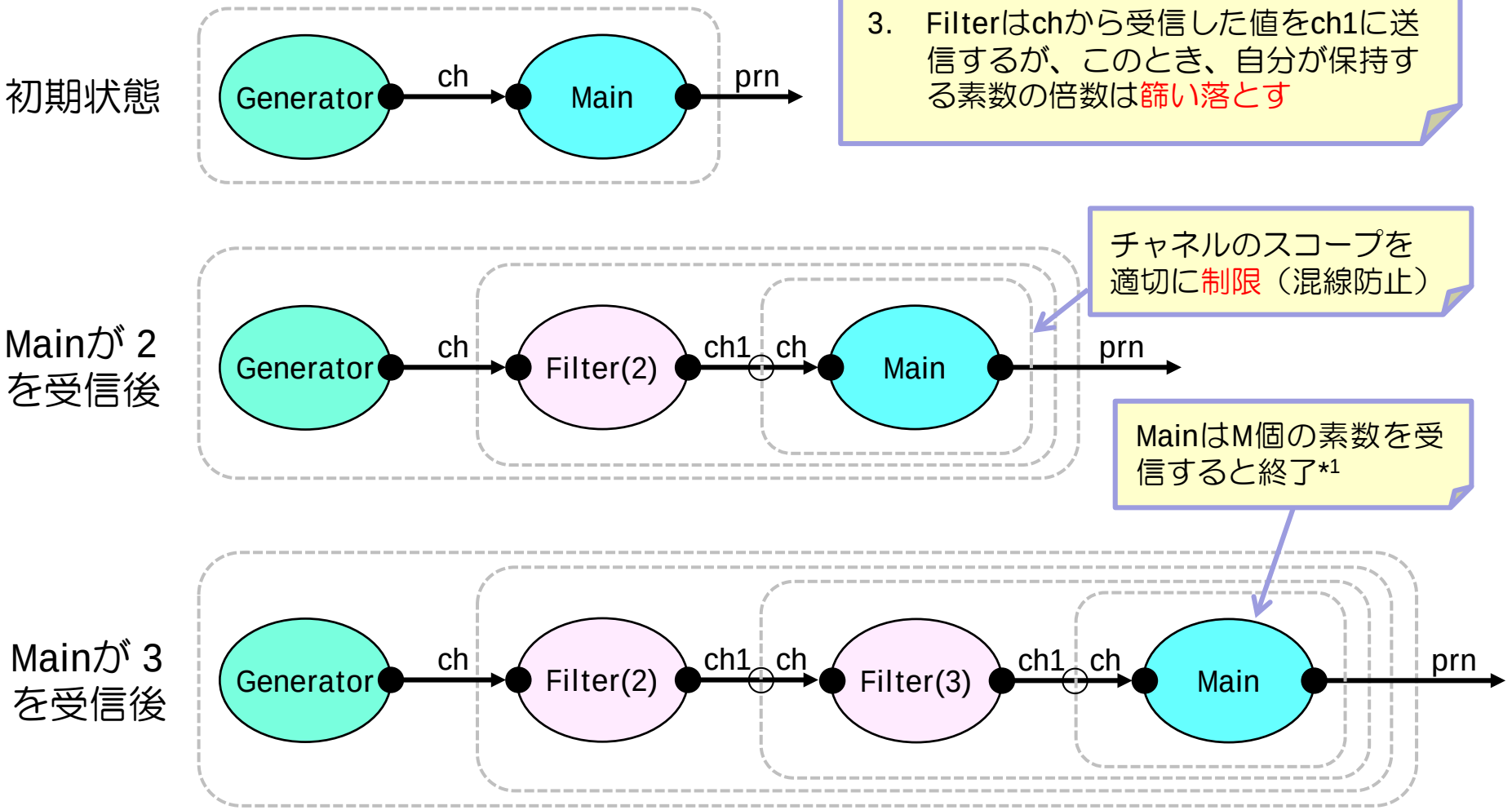


CSPによるモデル化

素数生成器の構造

1. Generatorは2以上の整数を順次生成しチャネルchに送信
2. Mainはchから受信した値を素数として表示（prn）し、その倍数を篩落とすプロセスFilterをGeneratorとの間に動的に生成する。
3. Filterはchから受信した値をch1に送信するが、このとき、自分が保持する素数の倍数は篩い落とす

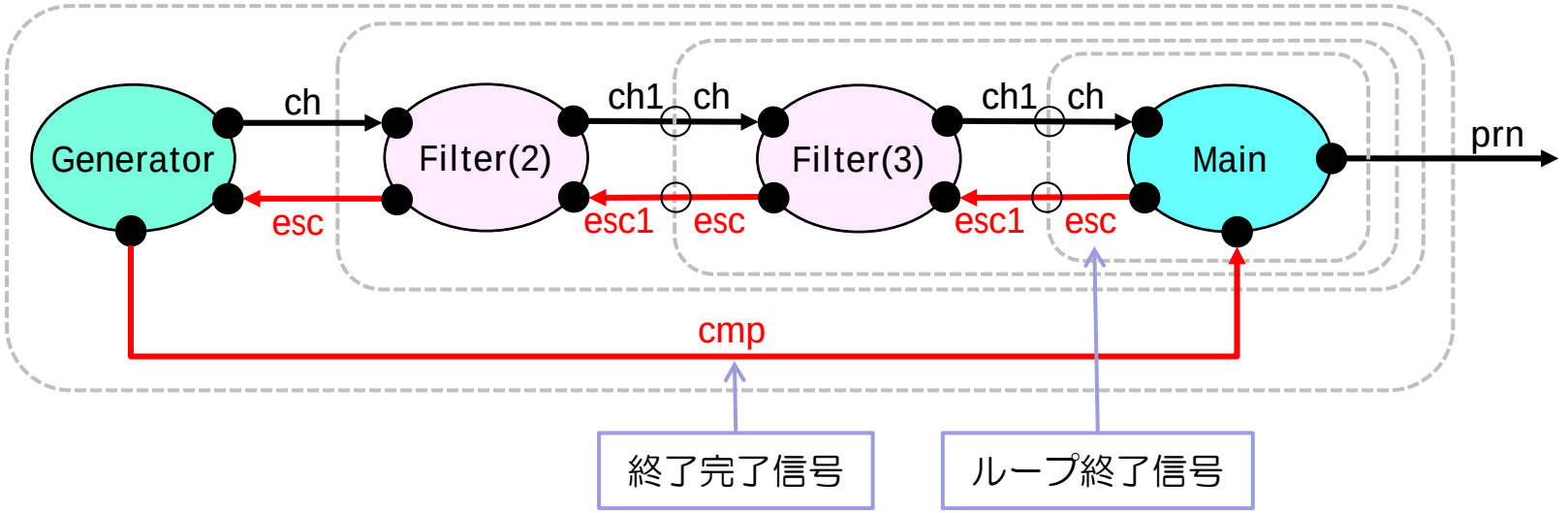
■ 素数生成器の構造



*1 Go言語ではMainが終了すると他のプロセスは強制終了される

素数生成器の修正（終了信号の追加）

- M個の素数を生成後、全てのプロセスが成功終了するように修正
  1. Mainは整数生成と篩い落としのループを終了させる信号（esc）を送信
  2. Filterは終了信号を受信するとループを終了し、終了信号を逆向きに転送
  3. Generatorは終了信号を受信するとループを終了し、完了信号（cmp）を送信
  4. Mainはcmpを受信後に終了*1

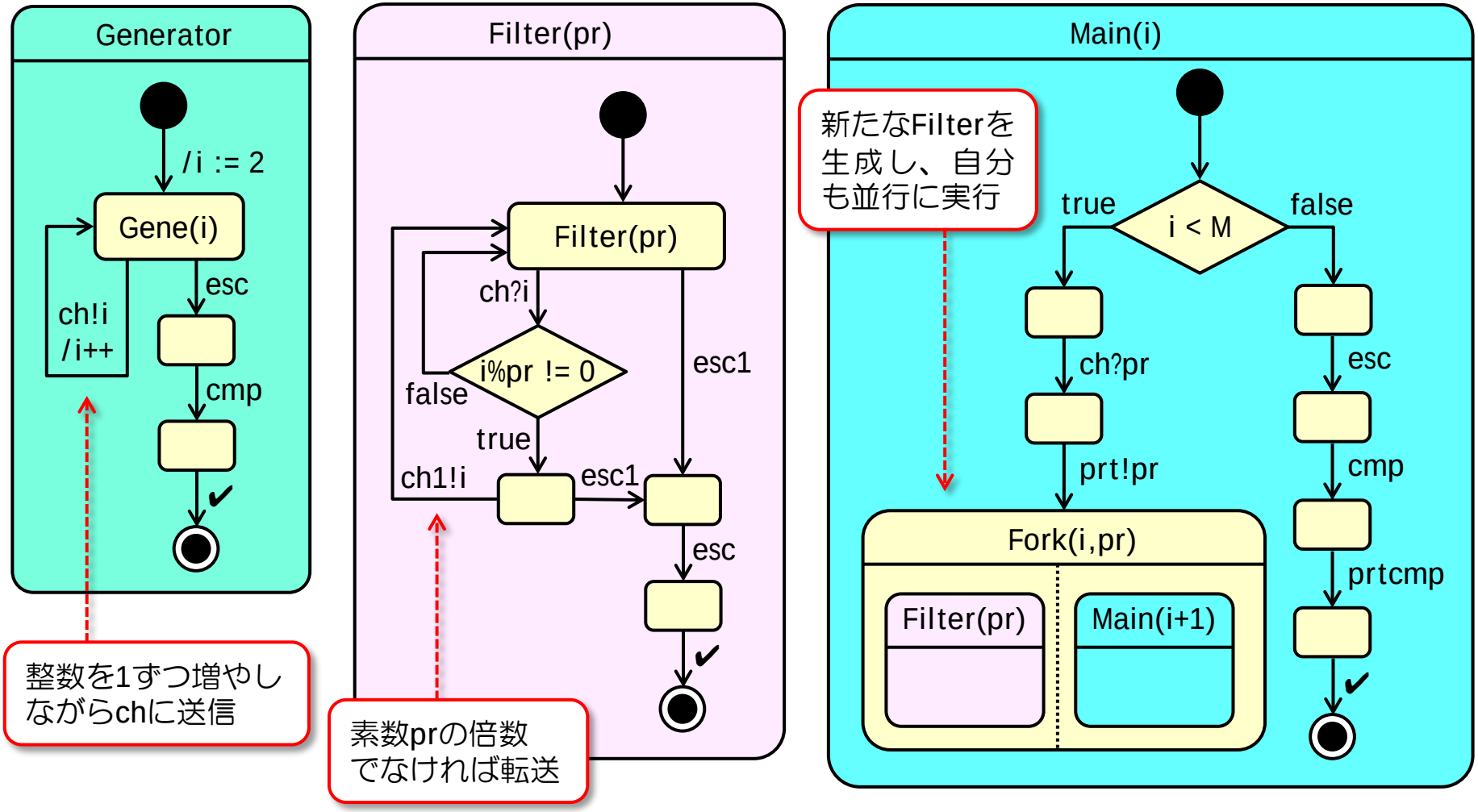


*1 Mainがcmpを待たない場合、esc信号がすべてのプロセスに届く前に、Mainが終了する可能性がある。

素数生成器の各プロセスの振舞い

ch!v：値vをchに送信
ch?x：chから受信した値を変数xに代入

■ 各プロセスの振舞いを表す状態遷移図

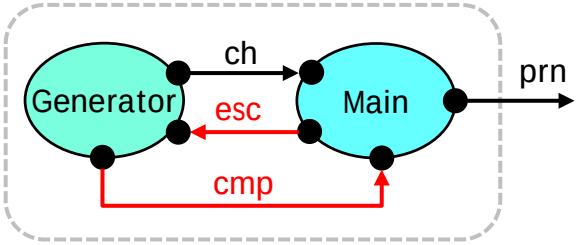


CSP：並行処理の形式仕様記述言語

- CSP *1：並行処理を論理的に記述する言語とその解析方法一式
  - CSPでは逐次処理を同期通信チャネルで接続して並行処理を組み立てる
  - CSPは機能安全の国際規格IEC61508で推奨されている形式手法のひとつ
  - CSPに基づく検証ツール（FDRなど）やプログラミング言語（Goなど）がある

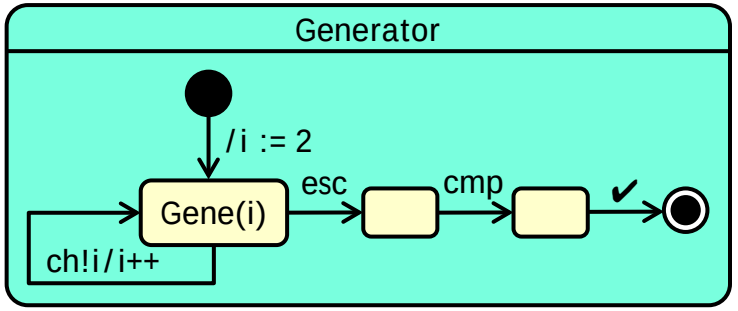
構造のCSP記述の例

```
     Chs = {|ch, esc, cmp|}
ParPrime = (Generator [|Chs|] Main) \ Chs
```



振舞いのCSP記述の例

```
Generator = Gene(2)
  Gene(i) = ch!i -> Gene(i+1)
            [] esc -> cmp -> SKIP
```

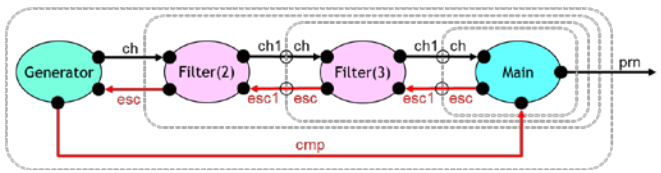


*1 CSP: Communicating Sequential Processes, C. A. R. Hoare, Oxford大学, 1978

素数生成器のCSP記述

- 構造図と状態遷移図からCSP記述に変換できる

素数生成器の構造図と状態遷移図



変換

```
M = 3
N = 14

-- --------------------
--       channel
-- --------------------

nametype Nat = {0..N}

channel ch, ch1 : Nat
channel prt : Nat
channel esc, esc1, cmp, prtcmp

 Chs = {|ch, esc, cmp|}
Chs1 = {|ch1, esc1|}
Rename1(P) = P[[ch <- ch1, esc <- esc1]]

-- -----------------
--      process
-- -----------------
-- Generator

Generator = Gene(2)
  Gene(i) = ((i < N) & ch!i -> Gene(i+1)) [] esc -> cmp -> SKIP

-- Filter
         Filter(0) = STOP
     Filter(prime) = ch?i -> FilterChk(prime,i) [] esc1 -> FilterEsc
FilterChk(prime,i) = if (i%prime != 0) then FilterSnd(prime,i)
                     else Filter(prime)
FilterSnd(prime,i) = ch1!i -> Filter(prime) [] esc1 -> FilterEsc
         FilterEsc = esc -> SKIP

-- Main
            Main = MainLoop(0)
     MainLoop(i) = if (i < M)then ch?prime -> MainPnt(i,prime)
                   else MainEsc
MainPnt(i,prime) = prt!prime -> MainNew(i,prime)
MainNew(i,prime) = (i < M) & Fork(i,prime)
         MainEsc = esc -> cmp -> prtcmp -> SKIP

-- composition

     ParPrime = (Generator [|Chs|] Main) ¥ Chs
Fork(i,prime) = (Filter(prime) [|Chs1|] Rename1(MainLoop(i+1))) ¥ Chs1
```

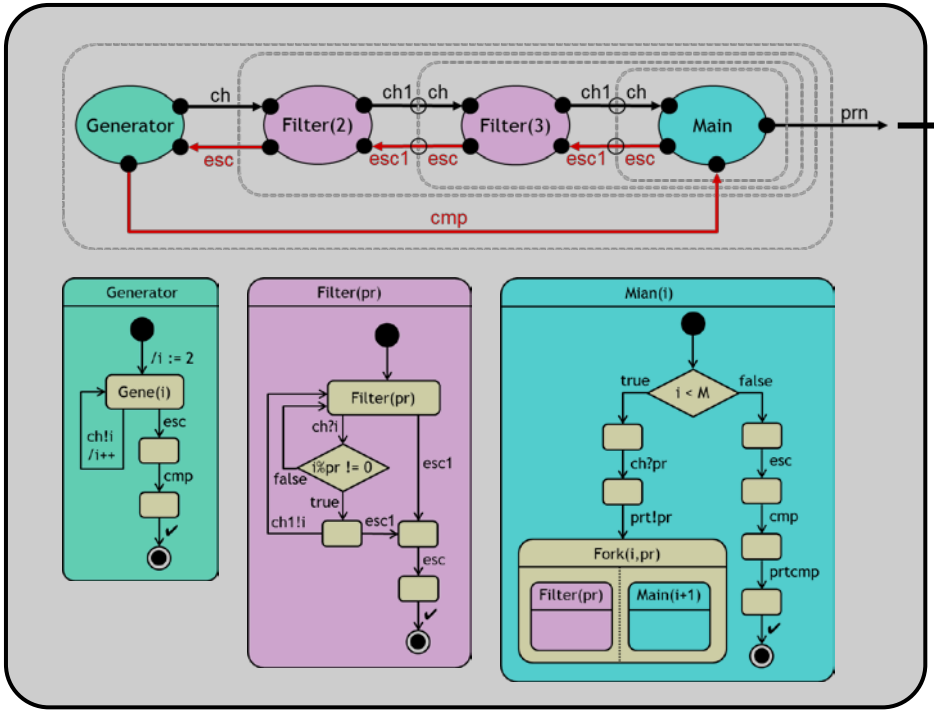
素数生成器のCSP記述

状態数が無限にならないように、生成する素数の個数は3、探索する整数の範囲は{0,…,14}に制限している

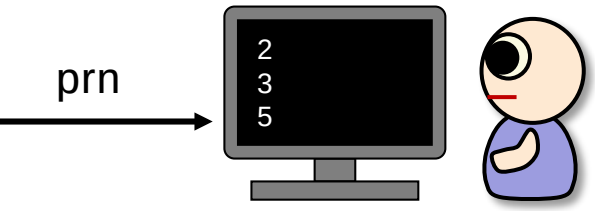
FDRによる検証

素数生成器の検証

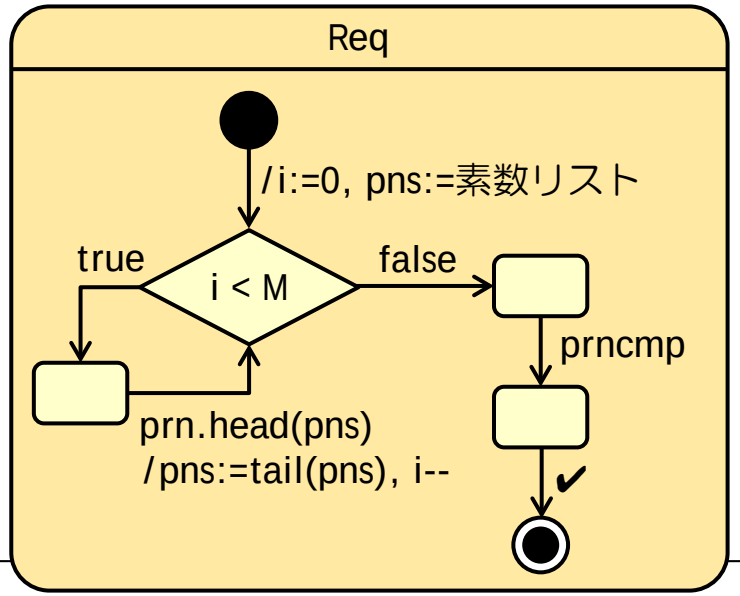
- 素数生成器が正しく動作することの検証
  - M個の素数が生成されるか？
  - 途中でデッドロックせずに成功終了するか？



素数生成器 ParPrime



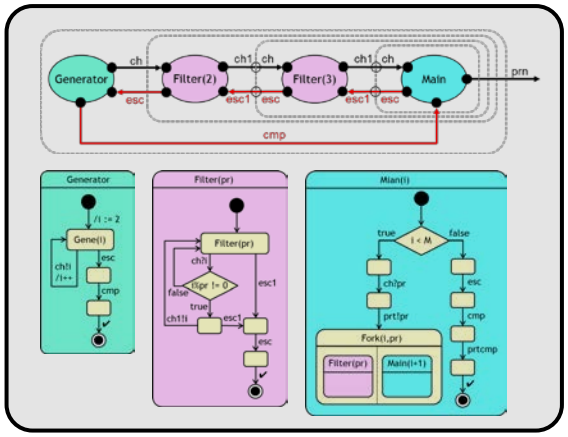
表示される内容に着目した要求 Req



素数生成器ParPrimeと要求Reqの等しさ判定

- ２つのCSP記述の振舞いの等しさを判定できる
  - ParPrime と Req のCSP記述を作成して振舞いの等しさを判定する

素数生成器 ParPrime

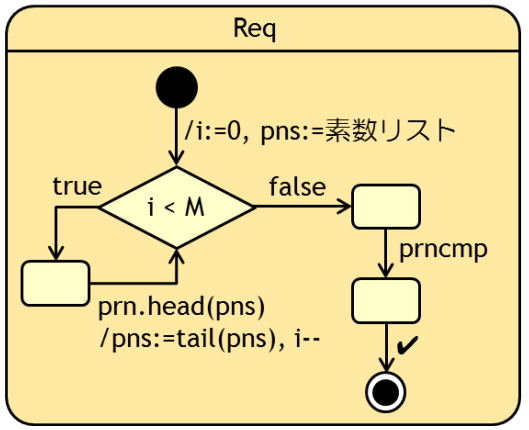


変換

ParPrimeの
CSP記述

?

要求 Req



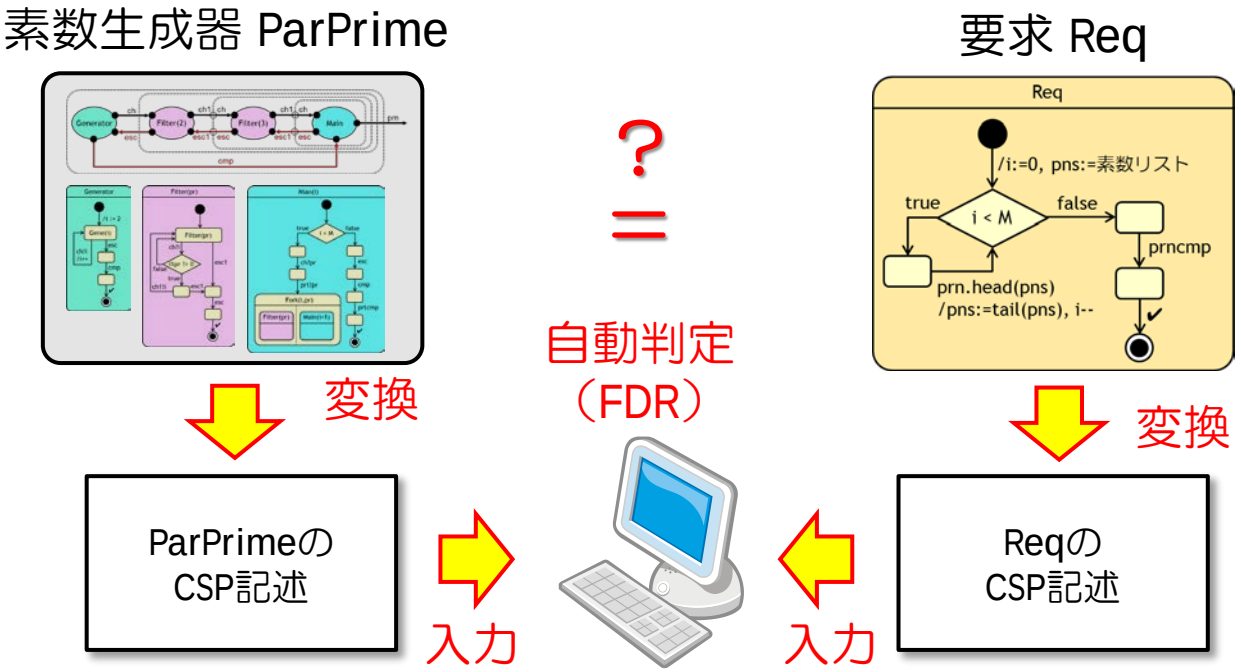
変換

Reqの
CSP記述

手作業でも判定できるが、限界がある...

FDR：モデル検査器（ツール）

- FDR：CSPのモデル検査器（Oxford大学）：
  - ２つのCSP記述の振舞いの等しさを自動判定可能
    - 正確には両方向の詳細化関係を判定する
  - 学術/研究目的でのみ無料で利用可能：
    - 最新版 FDR 4.2.0（2017/05/24現在：Windows 64bit, Linux x86 64bit, Mac OS X 64bit）
    - https://www.cs.ox.ac.uk/projects/fdr/



モデル検査器FDRによる判定結果

- ParPrime と Req の振舞いの等しさをFDRで判定
  - 検査では生成する素数の個数は3個（大きな数では状態数が急増する）
  - 検査結果は「等しい」

```
-- parameters --
M = 3
N = 8

-- --------------------
--      channel
-- --------------------
nametype Nat = {0..N}

channel ch, ch1 : Nat
channel prt : Nat
channel esc, esc1, cmp, prtcmp

Chs1 = {|ch1, esc1|}
 Chs = {|ch, esc, cmp|}
Rename1(P) = P[[ch <- ch1, esc <- esc1]]

-- -----------------
--      process
-- -----------------
-- Generator
Generator = Gene(2)
Gene(i) = ((i < N) & ch!i -> Gene(i+1)) [] esc -> cmp -> SKIP

-- Filter
        Filter(0) = STOP
     Filter(prime) = ch?i -> FilterChk(prime,i) [] esc1 -> FilterEsc
FilterChk(prime,i) = if (i%prime != 0) then FilterSnd(prime,i) else Filter(prime)
FilterSnd(prime,i) = ch1!i -> Filter(prime) [] esc1 -> FilterEsc
         FilterEsc = esc -> SKIP

-- Main
            Main = MainLoop(0)
     MainLoop(i) = if  (i < M)then ch?prime -> MainPnt(i,prime) else MainEsc
MainPnt(i,prime) = prt!prime -> MainNew(i,prime)
MainNew(i,prime) = (i < M) & Fork(i,prime)
         MainEsc = esc -> cmp -> prtcmp -> SKIP

-- composition
     ParPrime = (Generator [|Chs|] Main) ¥ Chs
Fork(i,prime) = (Filter(prime) [|Chs1|] Rename1(MainLoop(i+1))) ¥ Chs1

-- --------------------
--     requirement
-- --------------------
primeNums(2) = <2>
primeNums(n) = let pns = primeNums(n-1) within pns ^ checkPN(n,pns)

   checkPN(p,<>) = <p>
checkPN(p,<n>^s) = if (p%n == 0) then <> else checkPN(p,s)

Req = ReqLoop(M, primeNums(N))

    ReqLoop(0, pns) = prtcmp -> SKIP
     ReqLoop(i, <>) = STOP
ReqLoop(i, <p>^pns) = prt.p -> ReqLoop(i-1, pns)

-- --------------------
--   verification
-- --------------------
assert ParPrime [FD= Req
assert Req [FD= ParPrime
```

ParPrimeのCSP記述

ReqのCSP記述

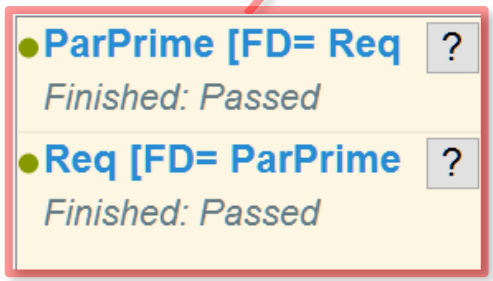
検査項目

FDR記述（基本的にはCSP記述）

入力

モデル検査器FDR4のスクリーンショット

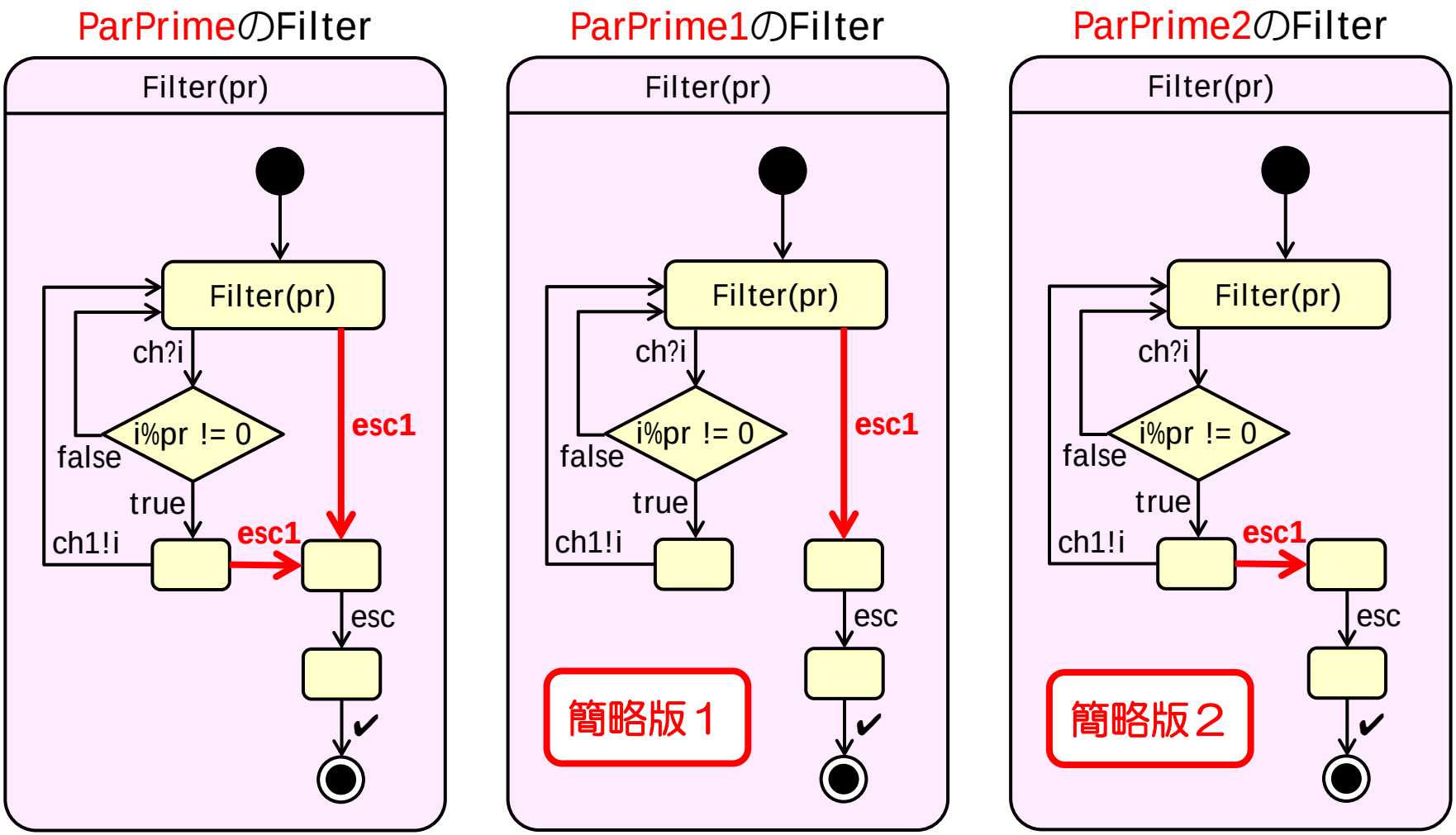
検査時間は約40秒
（CPU: Core i7-6567U）



両方向の詳細化関係成立

素数生成器の簡略版の検討

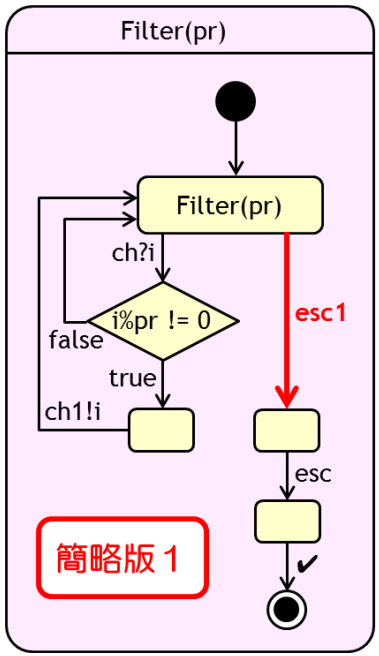
- ParPrime1,2: Filter(pr)の2つの esc1 の遷移の一方を削除した簡略版

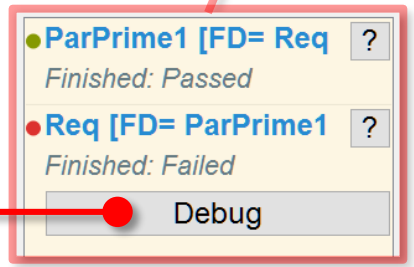


モデル検査器FDRによる簡略版１の判定結果

- FDRによる判定結果：
  - 簡略版１（ParPrime1）は要求Reqを満たさない

ParPrime1のFilter

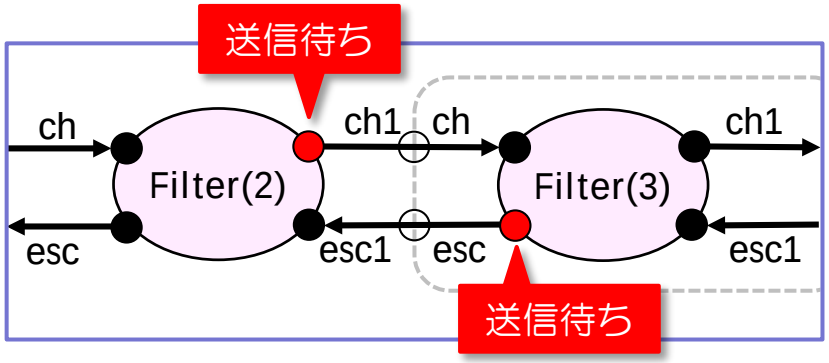




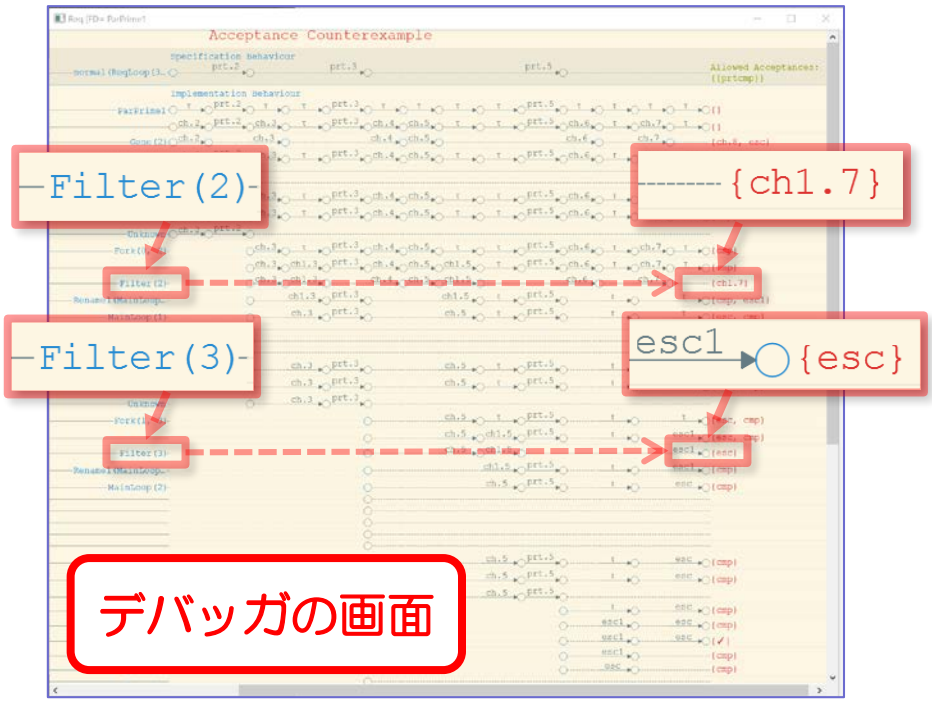
Reqを満たさない原因を分析するためデバッガを開く

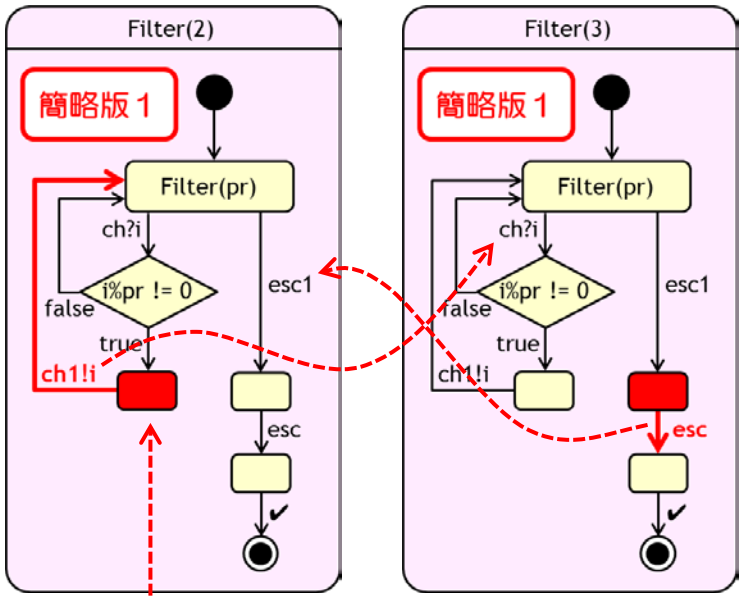
モデル検査器FDRによる簡略版１のデバッグ

- 簡略版１のデバッグ画面
  - 成功終了せずにデッドロックしている
    - Filter(2)はch1に7を送信待ち
    - Filter(3)はesc信号を送信待ち



デッドロックまでのトレースが表示される



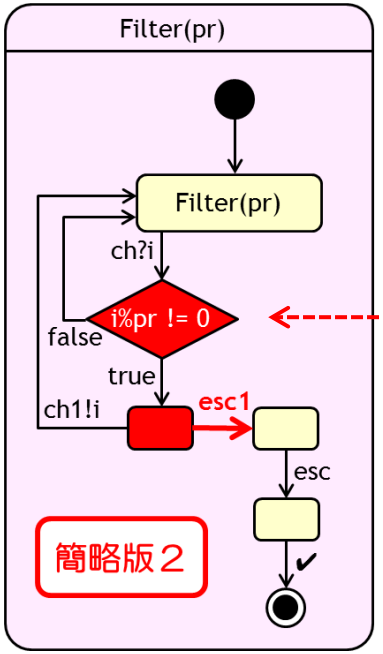


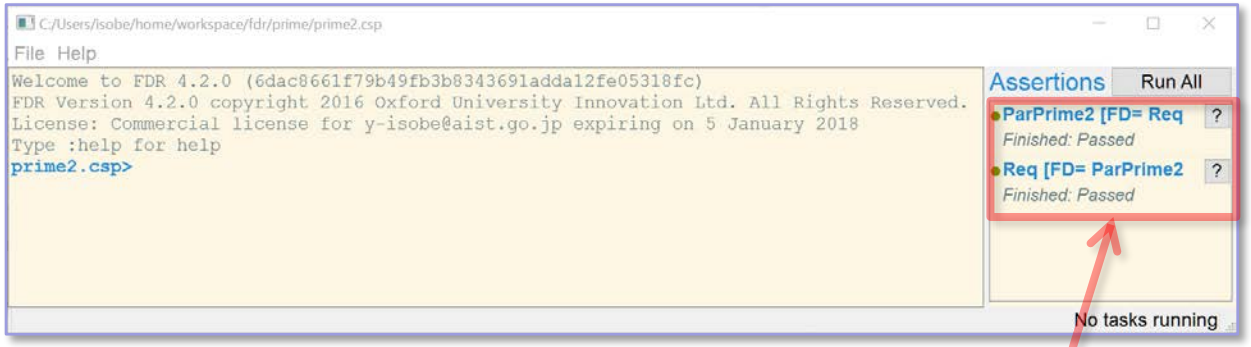
この状態がesc1遷移をもてば、
このデッドロックは回避できる

モデル検査器FDRによる簡略版２の判定結果

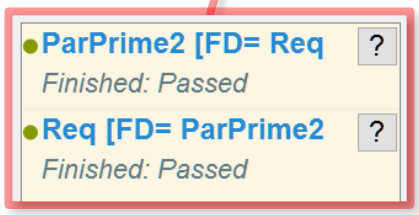
- FDRによる判定結果：
  - 簡略版２（ParPrime2）は要求Reqを満たす

ParPrime2のFilter





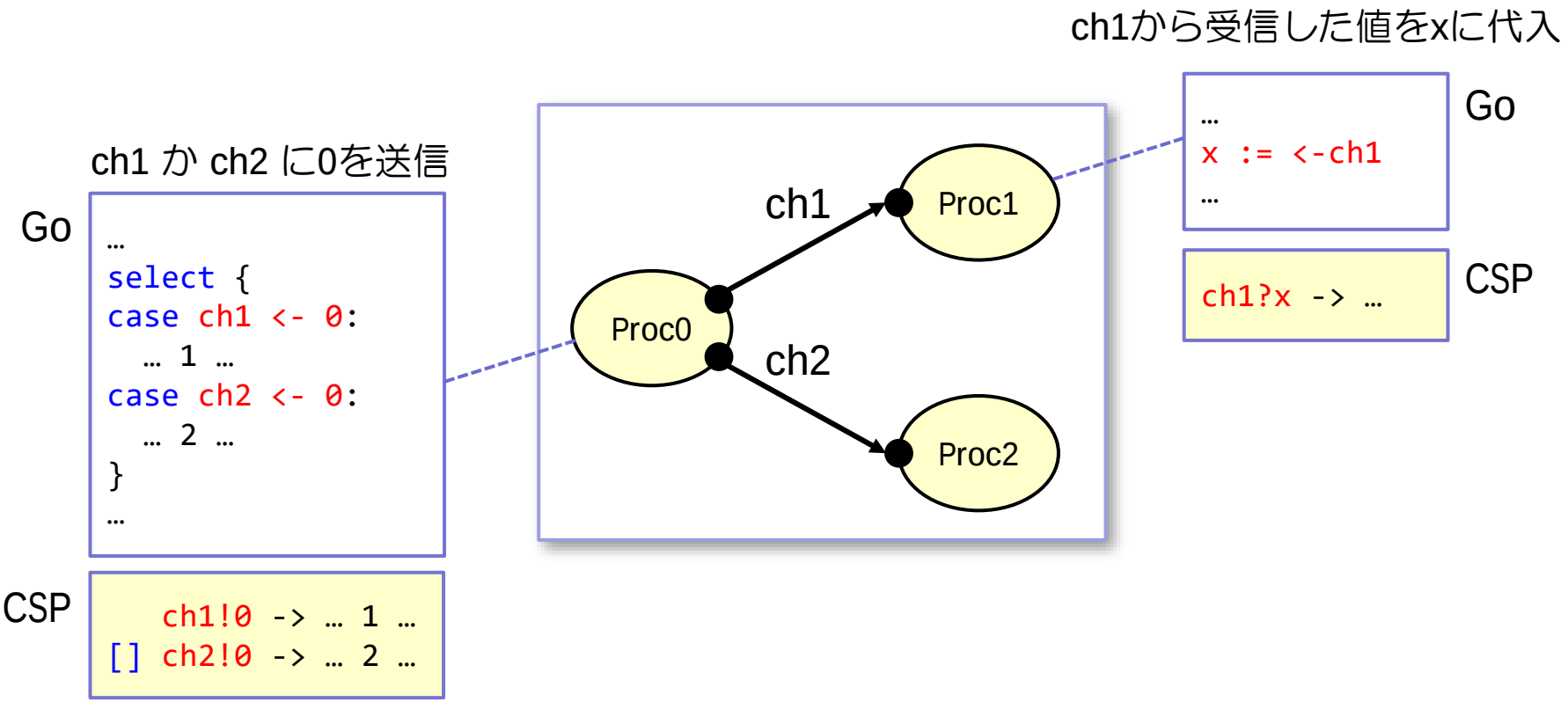
Reqは満たすが、この条件を満たさないと esc1で同期できないため、M個の素数を発見後も、内部で多くの計算を必要とする問題がある



GOによる実装

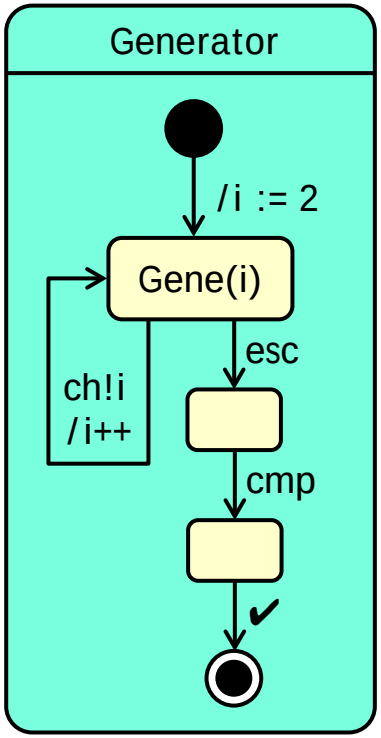
Go：プログラミング言語

- Go言語：
  - Googleによって開発されているプログラミング言語（2009年発表）
  - 最新版 Go 1.8.2（2017/05/24現在：Windows 64, Linux 64, Mac OS X 64）
  - Go言語は並行処理モデルとして CSPを採用



素数生成器のGoプログラム（Generator）

■ 素数生成器のGeneratorプロセスのGoプログラム

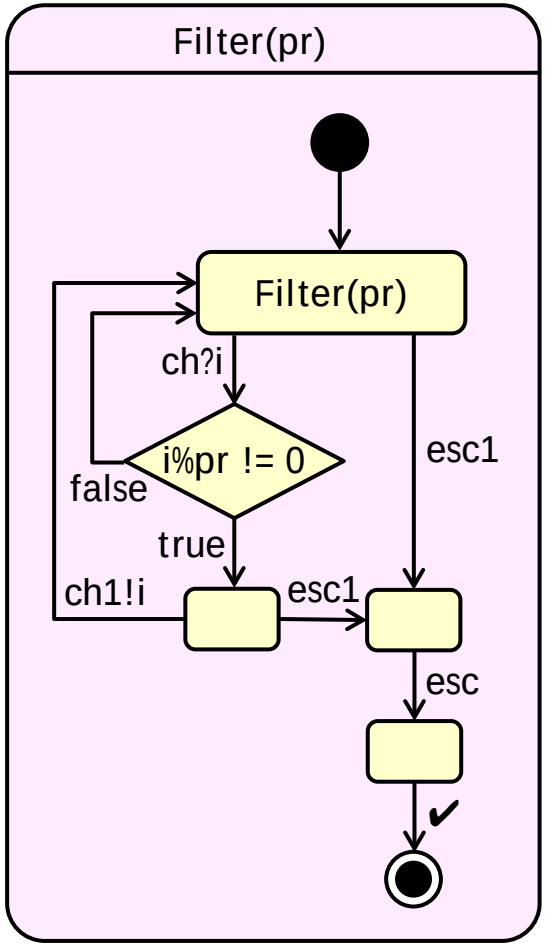


GeneratorのGoプログラム

```
// 自然数をchに繰返し送信、またはescから終了信号を受信
func Generator(ch chan<- int, esc <-chan bool,
               cmp chan<- bool) {
LOOP:
  for i := 2; ; i++ {
    select {
    case ch <- i: // iをチャネルchに送信
    case <-esc:   // 終了信号を受信
      break LOOP
    }
  }
  cmp <- true // 全てのgoroutineの終了完了送信
}
```

素数生成器のGoプログラム（Filter）

■ 素数生成器のFilterプロセスのGoプログラム

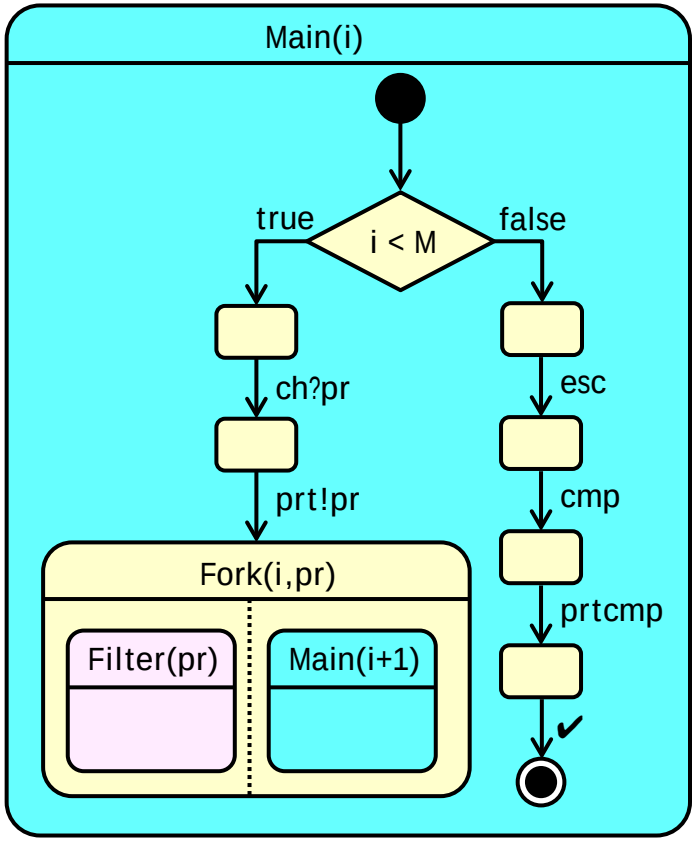


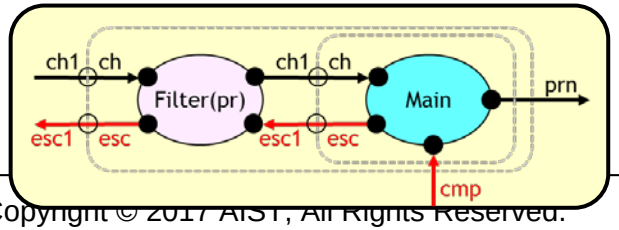
FilterのGoプログラム

```
// 受信した値が割り切れない場合は次に送信
func Filter(ch <-chan int, ch1 chan<- int,
            esc chan<- bool, esc1 <-chan bool,
            prime int) {
LOOP:
  for {
    select {
    case i := <-ch:          // 素数候補値の受信
      if i%prime != 0 {
        select {
        case ch1 <- i:       // 素数候補値の送信
        case <-esc1:         // 終了信号の受信
          break LOOP
        }
      }
    case <-esc1:             // 終了信号の受信
      break LOOP
    }
  }
  esc <- true                // 終了信号の送信
}
```

素数生成器のGoプログラム（Filter）

■ 素数生成器のMainプロセスのGoプログラム





MainのGoプログラム

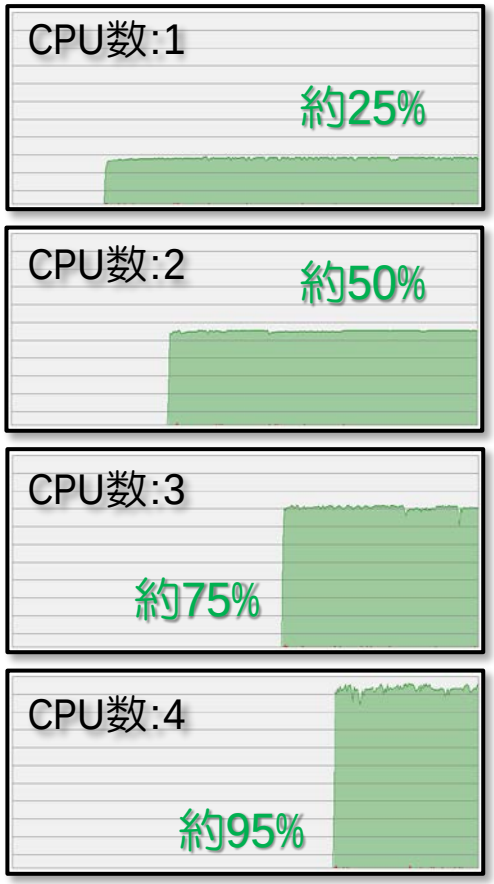
```
func main() {
  ch := make(chan int)   // チャネルch生成
  esc := make(chan bool) // チャネルesc生成
  cmp := make(chan bool) // チャネルcmp生成
  go Generator(ch, esc, cmp) // 生成

  for i := 0; i < 100; i++ {
    prime := <-ch
    fmt.Printf("%d番目の素数 : %d¥n", i+1, prime)
    ch1 := make(chan int)
    esc1 := make(chan bool)

    go Filter(ch, ch1, esc, esc1, prime) // 生成
    ch = ch1    // チャネル名変更
    esc = esc1  // チャネル名変更
  }
  esc <- true   // 終了信号の送信
  <-cmp         // 完了信号の受信
  fmt.Printf("完了¥n")
}
```

素数生成器のGoプログラムの実行例

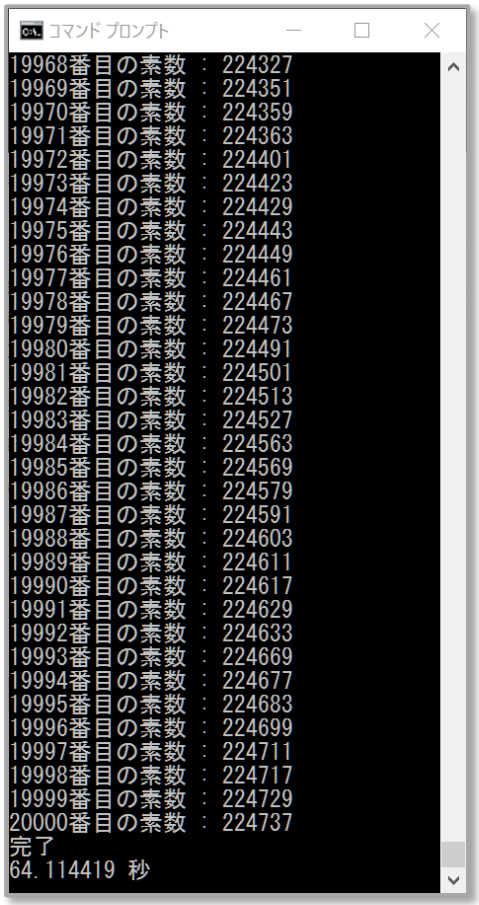
- 20,000個の素数を生成
  - 所要時間

CPU数	時間（秒）
1	160
2	136
3	86
4	64

CPU使用率



実行例（CPU数:4）



使用したパソコン

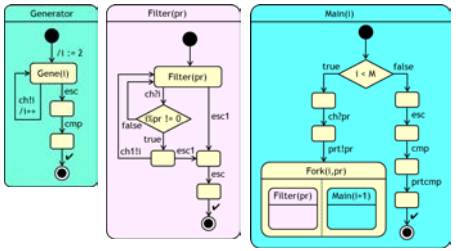
- SONY VAIO Z （VJZ13B1）
- CPU：インテル Core i7-6567U
  - 2コア
  - 4スレッド
  - 3.30～3.60GHz
- メモリ：16GB
- OS：Windows 10

おわりに

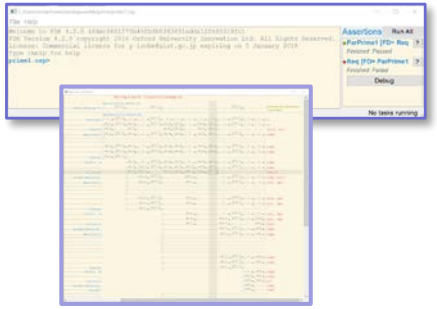
まとめ

- 並行処理をプログラミングの前に検証して信頼性・安全性の向上
  - 並行処理の不具合は再現性が低いためテストでの発見は難しい
  - Go言語の並行処理はCSP（シンプル）なためモデル化しやすい
    ⇒ CSPモデルを作成するだけでも不具合の発見につながる
  - CSPモデルの検証ツール（FDRなど）でタイミング網羅的な検査ができる
    ⇒ 発生確率の低い不具合も発見できる

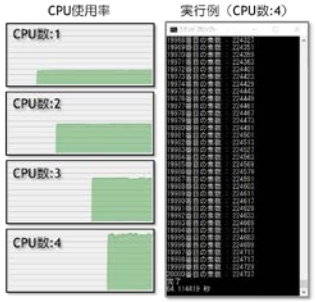
CSPによるモデル化



FDRによる検証・デバッグ



Goによる実装・実行



付録：モデル検査器の例

■ 付録：モデル検査器の例

- FDR：CSPの代表的なモデル検査器（オクスフォード大学、教育・研究目的ならば無料）
  https://www.cs.ox.ac.uk/projects/fdr/

  FDRの商用ライセンスも上記のサイトの Contact から購入できる

- PAT：CSPの高機能なモデル検査器（シンガポール大学、非営利研究目的ならば無料）
  http://sav.sutd.edu.sg/PAT/

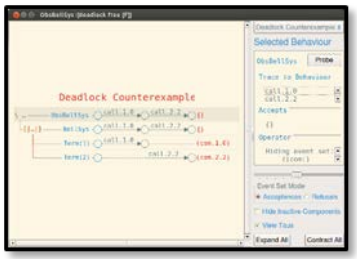
  PATの商用ライセンスはキャッツ株式会社から購入できる
  http://www.zipc.com/product/pat_pro/

- ProB：FDR互換性のモデル検査器（デュッセルドルフ大学とサウサンプトン大学、EPL v1.0）
  http://www.stups.uni-duesseldorf.de/ProB/

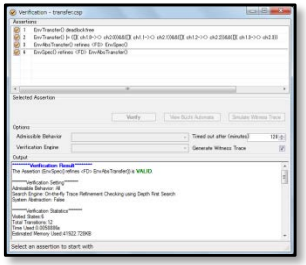
  ProBの有償サポートはformalmind社から受けることができる
  http://www.formalmind.com/

- SyncStitch：GUIが使いやすいモデル検査器（株式会社 PRINCIPIA、評価版あり）
  http://www.principia-m.com/jp/syncstitch.html

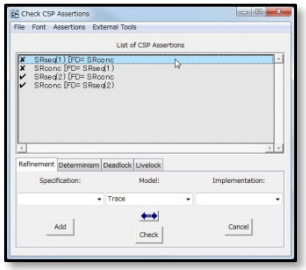
FDR



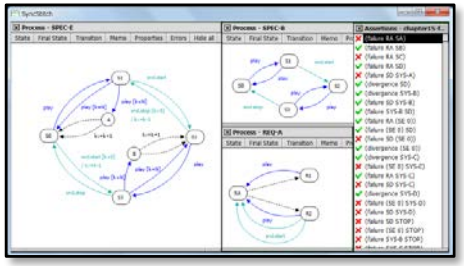
PAT



ProB



SyncStitch



付録：モデル検査器の例

■ CSPの教科書：

- C.A.R. Hoare, Communicating Sequential Processes, 2015.
  （http://www.usingcsp.com/cspbook.pdf から無料ダウンロード可能）

  CSPの提案者Hoare先生のCSPの最初の教科書です。

- A. W. Roscoe, The theory and practice of concurrency, Prentice Hall, 1998.
  （http://www.comlab.ox.ac.uk/people/bill.roscoe/publications/68b.pdf からダウンロード可能）

  Roscoe先生のCSPの教科書です。FDRの説明もあります。この本の改訂版が2010年に出版されています。
  A. W. Roscoe, Understanding Concurrent Systems, Springer, 2010.

- S. Schneider, Concurrent and Real-time Systems: the CSP Approach, John Wiley & Sons, 1999.
  （http://www.computing.surrey.ac.uk/personal/st/S.Schneider/books/CRTS.pdf からダウンロード可能）

  時間付CSP（Timed-CSP）の教科書です。時間制約をCSPのイベントに付けて検証することができます。

- 磯部祥尚, 並行システムの検証と実装
  – 形式手法CSPに基づく高信頼並行シスム開発入門, 近代科学者, 2012.

  CSP, FDR, JCSPを活用するための入門書です