A　CRIL の基本動作意味定義

以下に文献 [1] の CRIL における基本動作意味を示す.

Expression のルールは値表現の評価，Instructions は CRIL に含まれる命令の 1 ステップごとの可逆実行意味 $\rightleftharpoons$ を定義する.

Expressions:

$$\frac{k \text{ is a constant}}{(\rho,\sigma) \rhd k \rightsquigarrow k}\ \textbf{Con} \qquad \frac{}{(\rho[x \mapsto m],\sigma) \rhd x \rightsquigarrow m}\ \textbf{Var} \qquad \frac{}{(\rho[x \mapsto m_1],\sigma[m_1 \mapsto m_2]) \rhd M[x] \rightsquigarrow m_2}$$

$$\frac{(\rho,\sigma) \rhd right_1 \rightsquigarrow m_1 \qquad (\rho,\sigma) \rhd right_2 \rightsquigarrow m_2 \qquad m_3 = m_1 \odot m_2}{(\rho,\sigma) \rhd right_1 \odot right_2}\ \textbf{Exp}$$

Instructions:

$$\frac{(\rho,\sigma) \rhd e \rightsquigarrow m_3 \qquad m_2 = m_1 \oplus m_3}{x \oplus\texttt{=}\ e \rhd (\rho[x \mapsto m_1],\sigma) \rightleftharpoons (\rho[x \mapsto m_2],\sigma)}\ \textbf{AssVar}$$

$$\frac{(\rho,\sigma) \rhd e \rightsquigarrow m_3 \qquad m_2 = m_1 \oplus m_3}{\texttt{M[}x\texttt{]}\ \oplus\texttt{=}\ e \rhd (\rho[x \mapsto m_4],\sigma[m_4 \mapsto m_1]) \rightleftharpoons (\rho[x \mapsto m_4],\sigma[m_4 \mapsto m_2])}\ \textbf{AssArr}$$

$$\frac{}{x\ \texttt{<->}\ y \rhd (\rho[x,y \mapsto m_1,m_2],\sigma) \rightleftharpoons (\rho[x,y \mapsto m_2,m_1],\sigma)}\ \textbf{SwapVarVar}$$

$$\frac{}{x\ \texttt{<->}\ \texttt{M[}y\texttt{]} \rhd (\rho[x,y \mapsto m_1,m_3],\sigma[m_3 \mapsto m_2]) \rightleftharpoons (\rho[x,y \mapsto m_2,m_3],\sigma[m_3 \mapsto m_1])}\ \textbf{SwapVarA}$$

$$\frac{}{\texttt{M[}x\texttt{]}\ \texttt{<->}\ \texttt{M[}y\texttt{]} \rhd (\rho[x,y \mapsto m_3,m_4],\sigma[m_3,m_4 \mapsto m_1,m_2]) \rightleftharpoons (\rho[x,y \mapsto m_3,m_4],\sigma[m_3,m_4 \mapsto m_2,m_1])}\ \textbf{Sw}$$

$$\frac{}{\texttt{V}\ x \rhd (\rho[x \mapsto 0],\sigma) \rightleftharpoons (\rho[x \mapsto 1],\sigma)}\ \textbf{V-op} \qquad \frac{}{\texttt{P}\ x \rhd (\rho[x \mapsto 1],\sigma) \rightleftharpoons (\rho[x \mapsto 0],\sigma)}\ \textbf{P-op}$$

$$\frac{}{\texttt{skip} \rhd (\rho,\sigma) \rightleftharpoons (\rho,\sigma)}\ \textbf{Skip} \qquad \frac{(\rho,\sigma) \rhd e \not\rightsquigarrow 0}{\texttt{assert}\ e \rhd (\rho,\sigma) \rightleftharpoons (\rho,\sigma)}\ \textbf{Assert}$$

図 1: CRIL の基本動作意味定義 (1)

基本ブロック b に対して，入力ポイントを $\mathsf{entry}(b)$, 出力ポイントを $\mathsf{exit}(b)$, 命令を $\mathsf{inst}(b)$ とする．それぞれの表現 E に出現する変数，ヒープの集合を $\mathsf{Var}(E)$ で表すとすると，参照されるメモリの集合 $\mathsf{read}(b)$, 更新・参照されるメモリの集合 $\mathsf{write}(b)$ を以下のように定義する.

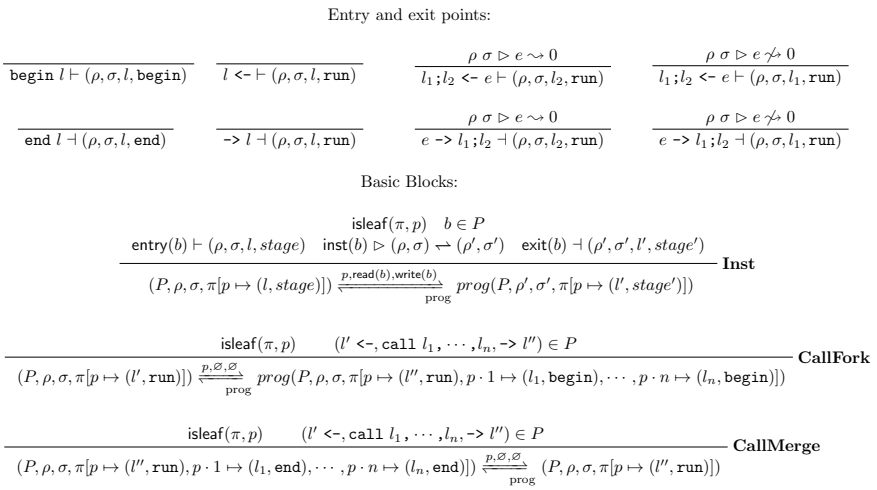


図 2: CRIL の基本動作意味定義 (2)

$$\mathsf{read}(b) = \mathsf{Var}(\mathsf{entry}(b)) \cup \mathsf{Var}(\mathsf{inst}(b)) \cup \mathsf{Var}(\mathsf{exit(b)})$$

$$\mathsf{write}(b) = \begin{cases} \{x\} & \mathsf{inst}(b) = x \oplus e \text{ のとき} \\ \{\texttt{M}\} & \mathsf{inst}(b) = \texttt{M[}x\texttt{]} \oplus e \text{ のとき} \\ \{x, y\} & \mathsf{inst}(b) = x\texttt{<->}y \text{ のとき} \\ \{x, \texttt{M}\} & \mathsf{inst}(b) \in \{\texttt{M[}y\texttt{]<->}x, x\texttt{<->M[}y\texttt{]}\} \text{ のとき} \\ \{\texttt{M}\} & \mathsf{inst}(b) = \texttt{M[}x\texttt{]<->M[}y\texttt{]} \text{ のとき} \\ \{s\} & \mathsf{inst}(b) \in \{\texttt{P}\ s, \texttt{V}\ s\} \text{ のとき} \\ \varnothing & \text{その他の場合} \end{cases}$$

プロセス IDpid を持つプロセスの実行状態は $(\ell, stage)$ で表す．$stage \in \{\texttt{begin}, \texttt{end}, \texttt{run}\}$ であり，$stage = \texttt{run}$ のとき，プロセスが ℓ のラベルをもつ基本ブロックの命令を実行していることを示す．$stage = \texttt{begin}$, $stage = \texttt{end}$ は順方向実行では，ℓ の基本ブロックでプロセスブロックが起動された状態，終了している状態を示す．逆方向実行では，起動，終了が逆である状態を示す．

(P, ρ, σ, π) はプログラム P の状態を表す．ここで，ρ は変数に対する割当て，σ はヒープの値を示す．π は，プロセス ID とそのプロセスの状態の対応を示す．$\mathsf{isleaf}(\pi, p)$ は，プロセス IDp が子プロセスを持たないとき真となる．

図 2 において **Inst** ルールは π で実行中のプロセスの中で待機状態になく実行可能な一つのプロセスが 1 ステップの命令を実行する遷移を示し，**CallFork** は順方向実行の `call` 命令による子プロセスの生成，**CallMerge** は，生成した子プロセスがすべて終了したとき，`call` 命令を終了して次の基本ブロックに制御を渡すことを表す．対称的に，逆方向実行では，**CallMerge** によって子プロセスを生成し，**CallFork** によって子プロセスの終了を待機する．

B　例: エラトステネスの篩

図 3 にエラトステネスの篩の手法に沿って，100 までの素数を計算する Concurrent Janus のプログラムを示す．配列 p はインデックスの数がどのふるいにかけられたかを記録する大域変数であり，p[i] が 0 であるとき，i は素数である．

max は自然数列の上限，maxrt は max の平方根であり，生成される篩の上限として与える．nextlock，plock は同期変数である．

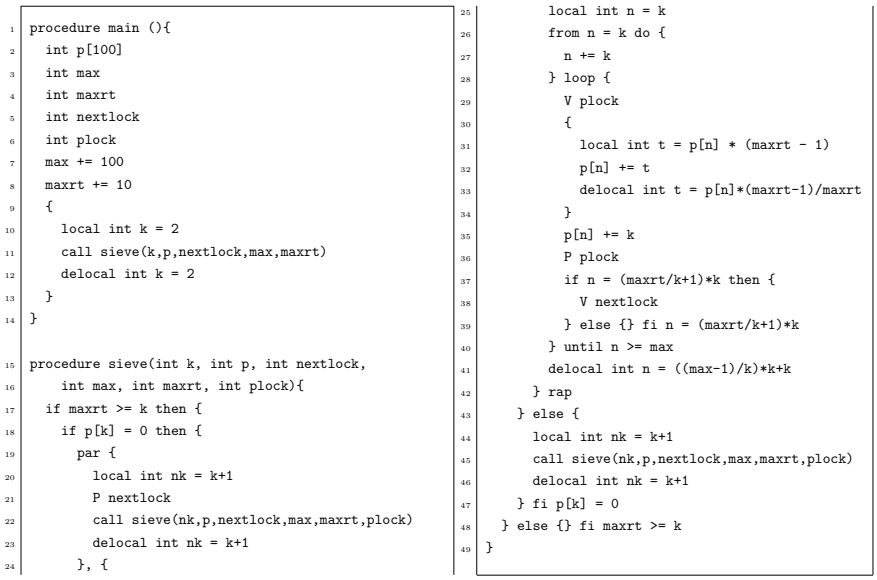
篩の作成が行われない状況での篩の実行は順方向，逆方向とも実行時に決定される．ここでは，実行順所を示すために p の要素として，2,3,5,7 の篩の実行順序を新たな篩にかかった場合，前の値を maxrt 倍して自分の篩の値を加えることで maxrt 進数で表している．例えば，maxrt =10 の場合，3,7,2 の順番で篩にかかった場合は，372 が格納される．

プログラムの構成　手続き sieve は，引数 k が maxrt 以下の場合に動作し，k が素数と判定された場合は，par..rap によって 2 つのプロセスが生成される．前半 (20 行目 ∼23 行目) は次の篩を生成するプロセスであり，後半 (25 行目 ∼41 行目) は篩の本体の動作のプロセスである．26 行目 ∼40 行目のループが篩の振舞いを記述しており，p の内容を k ごとに参照して内容を更新する．k が素数と判定されない場合は，20 行目 ∼24 行目と同様に次の篩を生成するが，素数でないので篩の本体は記述しない．

順方向実行における振舞い　k<=maxrt が成立する間 (18 行目 ∼47 行目) は順に素数に対して篩を生成する．k が素数である場合は，k の倍数の p のインデックスに 2 の篩の適用を示す数を書き込む (31 行目 ∼33 行目)．この動作を max まで繰り返す (26 行目 ∼40 行目)．nextlock は k の書込みが maxrt を上回る nextlock によって 21 行目の待機を開放し，k+1 の篩を生成する．k+1 が素数の場合は，上と同様の動作を行い，素数でない場合は何も行わず，k をインクリメントして再帰的に sieve を呼出す (45 行目)．k が maxrt は何もしない (48 行目)．

maxrt=10 の場合，k が 2,3,5,7 に対して 26 行目から 40 行目のループが並行に実行されるプロセスが実行される．(その他のプロセスも生成されるが直ちに終了する.）k=10 までは順番を保って篩が生成される必要がある．例えば，k=4 での sieve の呼出しが k=2 を追い越すと 18 行目の素数チェックが正しくなくなる．このために，nextlock によって順番を保証している．

逆方向実行における振舞い　逆方向実行でも k=2 から sieve を実行する．p[k] に素数が格納されている場合は 0 なので，47 行目の逆方向の分岐で k が素数である場合は 26 行目 ∼40 行目のループを逆方向に実行して，p の k の倍数マークを消去する．それと並行に 22 行目で次の逆方向の篩を生成する．



```
procedure main (){
  int p[100]
  int max
  int maxrt
  int nextlock
  int plock
  max += 100
  maxrt += 10
  {
    local int k = 2
    call sieve(k,p,nextlock,max,maxrt)
    delocal int k = 2
  }
}

procedure sieve(int k, int p, int nextlock,
    int max, int maxrt, int plock){
  if maxrt >= k then {
    if p[k] = 0 then {
      par {
        local int nk = k+1
        P nextlock
        call sieve(nk,p,nextlock,max,maxrt,plock)
        delocal int nk = k+1
      }, {
        local int n = k
        from n = k do {
          n += k
        } loop {
          V plock
          {
            local int t = p[n] * (maxrt - 1)
            p[n] += t
            delocal int t = p[n]*(maxrt-1)/maxrt
          }
          p[n] += k
          P plock
          if n = (maxrt/k+1)*k then {
            V nextlock
          } else {} fi n = (maxrt/k+1)*k
        } until n >= max
        delocal int n = ((max-1)/k)*k+k
      } rap
    } else {
      local int nk = k+1
      call sieve(nk,p,nextlock,max,maxrt,plock)
      delocal int nk = k+1
    } fi p[k] = 0
  } else {} fi maxrt >= k
}
```

図 3: Concurrent Janus によるエラトステネスの篩

k が素数でない場合は，何もせず次の篩を生成する．26 行目 ∼40 行目のループが maxrt 以下になる直前で 28 行目の V nextlock で待機する．nextlock の更新は順方向で 2,3,5,7 のプロセスの順番で行った依存関係がアノテーション DAG に保存されているため，この依存関係に従って，7,5,3,2 の順番に p の倍数のマークを消去する．

図 4 と図 5 に CRIL への変換結果を示す．図 6, 図 7 に実行結果を示す．ここでは実行ごとに p の素数でないインデックスの値は異なり，篩が並行に実行されていることを示す．いずれの場合も依存関係に基づいて逆実行によって初期状態に逆実行されることを示している．

参考文献

[1] Shunya Oguchi and Shoji Yuen. CRIL: A concurrent reversible intermediate language. In Claudio Antares Mezzina and Georgiana Caltais, editors, *Proceedings Combined 30th International Workshop on Expressiveness in Concurrency and 20th Workshop on Structural Operational Semantics, EXPRESS/SOS 2023, and 20th Workshop on Structural Operational SemanticsAntwerp, Belgium, 18th September 2023*, Vol. 387 of *EPTCS*, pp. 149–167, 2023.

```
begin main
indent 0
#p[100] += 0
#max += 0
#maxrt += 0
#nextlock += 0
#plock += 0
max += 100
maxrt += 10
indent 0
$k += 2
set $k:1 k
set $p:1 p
set $nextlock:1 nextlock
set $max:1 max
set $maxrt:1 maxrt
call sieve
unset $maxrt:1 maxrt
unset $max:1 max
unset $nextlock:1 nextlock
unset $p:1 p
unset $k:1 k
$k -= 2
unindent 0
unindent 0
end main
```

```
begin sieve
indent 1
$tmp0 += maxrt >= k
$tmp0 -> l0;l1
l0 <-
$tmp0 -= maxrt >= k
indent 0
$tmp1 += p[k] == 0
$tmp1 -> l4;l5
l4 <-
$tmp1 -= p[k] == 0
indent 0
call $1.0.0.0, $1.0.0.1
unindent 0
$tmp25 += p[k] == 0
-> l6
l5 <-
$tmp1 -= p[k] == 0
indent 1
$tmp23 += k + 1
$nk += $tmp23
$tmp23 -= k + 1
set $k:1 nk
set $p:1 p
set $nextlock:1 nextlock
set $max:1 max
set $maxrt:1 maxrt
set $plock:1 plock
call sieve
unset $plock:1 plock
unset $maxrt:1 maxrt
unset $max:1 max
unset $nextlock:1 nextlock
unset $p:1 p
unset $k:1 nk
$tmp24 += k + 1
$nk -= $tmp24
$tmp24 -= k + 1
unindent 1
$tmp25 += p[k] == 0
-> l7
l6;l7 <- $tmp25
$tmp25 -= p[k] == 0
unindent 0
$tmp26 += maxrt >= k
-> l2
```

```
l1 <-
$tmp0 -= maxrt >= k
indent 1
unindent 1
$tmp26 += maxrt >= k
-> l3
l2;l3 <- $tmp26
$tmp26 -= maxrt >= k
unindent 1
end sieve

begin $1.0.0.0
indent 0
$tmp2 += k + 1
$nk += $tmp2
$tmp2 -= k + 1
P nextlock
set $k:1 nk
set $p:1 p
set $nextlock:1 nextlo
set $max:1 max
set $maxrt:1 maxrt
set $plock:1 plock
call sieve
unset $plock:1 plock
unset $maxrt:1 maxrt
unset $max:1 max
unset $nextlock:1 next
unset $p:1 p
unset $k:1 nk
$tmp3 += k + 1
$nk -= $tmp3
$tmp3 -= k + 1
unindent 0
end $1.0.0.0
```

図 4: 図 3 のプログラムの変換結果 (1)

```
begin $1.0.0.1
indent 1
$n += k
$tmp4 += n == k
-> l8
l8;l12 <- $tmp4
$tmp4 -= n == k
indent 0
n += k
unindent 0
-> l9
l11 <-
$tmp18 -= n >= max
indent 1
V plock
indent 0
$tmp5 += maxrt - 1
$tmp6 += p[n] * $tmp5
$t += $tmp6
$tmp6 -= p[n] * $tmp5
$tmp5 -= maxrt - 1
p[n] += t
$tmp7 += maxrt - 1
$tmp8 += p[n] * $tmp7
$tmp9 += $tmp8 / maxrt
$t -= $tmp9
$tmp9 -= $tmp8 / maxrt
$tmp8 -= p[n] * $tmp7
$tmp7 -= maxrt - 1
unindent 0
p[n] += k
P plock
$tmp10 += maxrt / k
$tmp11 += $tmp10 + 1
$tmp12 += $tmp11 * k
$tmp13 += n == $tmp12
$tmp13 -> l13;l14
l13 <-
$tmp13 -= n == $tmp12
$tmp12 -= $tmp11 * k
$tmp11 -= $tmp10 + 1
$tmp10 -= maxrt / k
indent 1
V nextlock
unindent 1
$tmp14 += maxrt / k
$tmp15 += $tmp14 + 1
$tmp16 += $tmp15 * k
$tmp17 += n == $tmp16
-> l15
l14 <-
$tmp13 -= n == $tmp12
$tmp12 -= $tmp11 * k
$tmp11 -= $tmp10 + 1
$tmp10 -= maxrt / k
indent 2
unindent 2
$tmp14 += maxrt / k
$tmp15 += $tmp14 + 1
$tmp16 += $tmp15 * k
$tmp17 += n == $tmp16
-> l16
l15;l16 <- $tmp17
$tmp17 -= n == $tmp16
$tmp16 -= $tmp15 * k
$tmp15 -= $tmp14 + 1
$tmp14 -= maxrt / k
unindent 1
$tmp4 += n == k
-> l12
l9 <-
$tmp18 += n >= max
$tmp18 -> l10;l11
l10 <-
$tmp18 -= n >= max
$tmp19 += max - 1
$tmp20 += $tmp19 / k
$tmp21 += $tmp20 * k
$tmp22 += $tmp21 + k
$n -= $tmp22
$tmp22 -= $tmp21 + k
$tmp21 -= $tmp20 * k
$tmp20 -= $tmp19 / k
$tmp19 -= max - 1
unindent 1
end $1.0.0.1
```

図 5: 図 3 のプログラムの変換結果 (2)

```
$ go run ./runtime sieve_f.crl
P0,0>
begin main
indent 0
-> _0

P0,1>
_0 <-
#p[100] += 0
-> _1

<略>

P0,23>
_22 <-
unindent 0
end main

Debug >> Execution finished
var
---Symbol Status---
p -> [1,100] ->
[0,0,0,0,2,0,23,0,2,3,25,0,23,0,27,35,2,0,23,0,25,
37,2,0,32,5,2,3,27,0,325,0,2,3,2,57,32,0,2,3,25,
0,327,0,2,35,2,0,32,7,25,3,2,0,32,5,27,3,2,0,325,
0,2,37,2,5,32,0,2,3,275,0,32,0,2,35,2,7,32,0,25,
3,2,0,372,5,2,3,2,0,352,7,2,3,2,5,32,0,72,3]
max -> [101] -> 100
k:0.0 -> [105] -> 0
maxrt -> [102] -> 10
plock -> [104] -> 0
nextlock -> [103] -> 0
<略>
```

```
bwd
Debug >> Backward mode set
run
Debug >> Running...
P0,23>
end main
unindent 0
_22 <-

<略>
P0,0>
-> _0
indent 0
begin main

Debug >> Execution finished
var
---Symbol Status---
p -> [1,100] ->
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
plock -> [104] -> 0
maxrt -> [102] -> 0
k:0.0 -> [105] -> 0
nextlock -> [103] -> 0
<略>
```

図 6: 実行結果 (1)

```
$ go run ./runtime sieve_f.crl
P0,0>
begin main
indent 0
-> _0

P0,1>
_0 <-
#p[100] += 0
-> _1

<略>

P0,23>
_22 <-
unindent 0
end main

Debug >> Execution finished
var
---Symbol Status---
p -> [1,100] ->
[0,0,0,0,2,0,23,0,2,3,25,0,23,0,27,35,2,0,23,0,25,
37,2,0,23,5,2,3,27,0,253,0,2,3,2,57,23,0,2,3,52,
0,723,0,2,53,2,0,23,7,52,3,2,0,23,5,72,3,2,0,532,
0,2,73,2,5,32,0,2,3,572,0,32,0,2,53,2,7,32,0,52,
3,2,0,732,5,2,3,2,0,532,7,2,3,2,5,32,0,72,3]
max -> [101] -> 100
maxrt -> [102] -> 10
k:0.0 -> [105] -> 0
plock -> [104] -> 0
nextlock -> [103] -> 0
<略>
```

```
bwd
Debug >> Backward mode set
run
Debug >> Running...
P0,23>
end main
unindent 0
_22 <-

<略>

P0,0>
-> _0
indent 0
begin main

Debug >> Execution finished
var
---Symbol Status---
p -> [1,100] ->
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
plock -> [104] -> 0
maxrt -> [102] -> 0
k:0.0 -> [105] -> 0
nextlock -> [103] -> 0
<略>
```

図 7: 実行結果 (2)