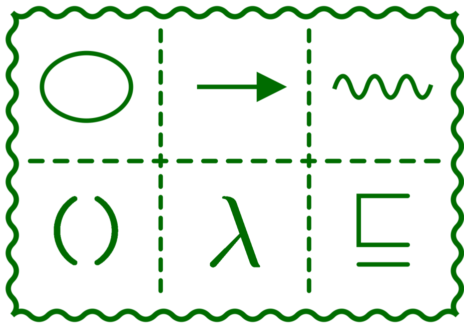


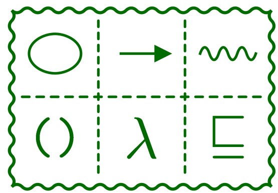
並行システムの検証と実装

第11章 並行システムの実装 0
CSP 実装支援ライブラリ MCCSP による実装

PRINCIPIA Limited

初谷 久史

©2015 PRINCIPIA Limited



システムの設計（振る舞い側面）

上流へ

要求

振る舞い
仕様化

振る舞い
仕様

比較

比較結果

コンポーネント
分割と
振る舞いモデル化

コンポーネント
モデル 0

コンポーネント
モデル 1

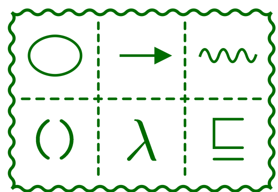
コンポーネント
モデル N–1

合成

システムモデル

詳細化または
実装へ

©2015 PRINCIPIA Limited



モデルから実装へ

上流から

コンポーネント
モデル k

実装

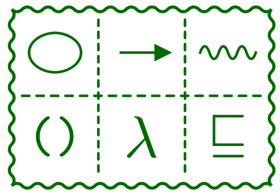
コンポーネント
コード k

Compile
& Link

実装支援ライブラリ

実行可能コード

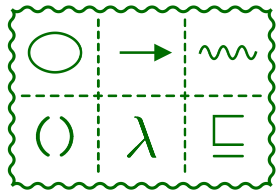
©2015 PRINCIPIA Limited



並行システムの実装

- CSP 実装支援ライブラリ MCCSP による実装
- 同期機構による実装
- 不可分操作によるロックフリーアルゴリズムの実装

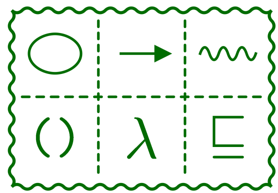
©2015 PRINCIPIA Limited



モデルから実装へ: 2つのケース

- システムを CSP の考え方（イベントによる同期型相互作用）で構築する
  - システムの振る舞いを CSP でモデル化し検査する
  - CSP の考え方をサポートするプログラミング言語やライブラリを使って実装する
- システムをオペレーティングシステムが提供する同期機構を使って実装する
  - CSP でモデル化した同期機構を部品としてシステムのモデルを作成し検査する
  - 同期機構を使って実装する

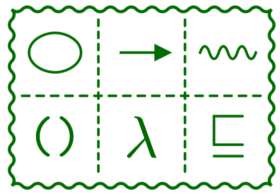
©2015 PRINCIPIA Limited



2つのケースの比較

	利点	欠点
CSP	考え方がシンプルで，モデル化しやすく，検査において理論とツールの支援を受けやすい	オペレーティングシステム，プログラミング言語，ライブラリ等の支援が少ない
同期機構	現在の計算機アーキテクチャに適合しており性能が出しやすい	モデル化が難しく，検査すべき性質も表現しにくい．モデルの規模が大きくなりがちでツールの能力を超えやすい

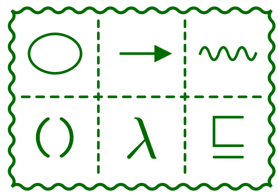
©2015 PRINCIPIA Limited



CSP 実装支援

- プログラミング言語
  - OCCAM-π
  - Go
  - Rust
  - XC (XMOS)
- ライブラリ
  - C++CSP2 (C++)
  - JCSP (Java)
  - PyCSP (Python)
  - CHP (Haskell)
  - Jibu (C#)
  - MCCSP (C)

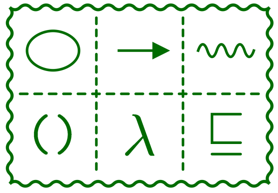
©2015 PRINCIPIA Limited



MCCSP

- CSP の実装を支援するC言語ライブラリ
- 特徴
  - CSP を基礎とするイベント同期型相互作用
  - pthread との組み合わせを想定
  - チャネル送受信のサポート
  - マルチ同期，インターリーブ結合，受信ガードは非サポート

©2015 PRINCIPIA Limited



MCCSP

Thread

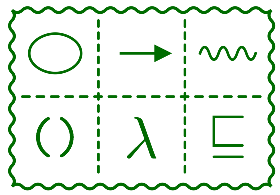
Channel

send

Process

receive

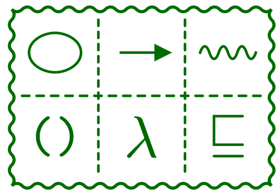
©2015 PRINCIPIA Limited



MCCSP: 型

- `mccsp_process_t` プロセスオブジェクト
  - `intptr_t v[]` ユーザ使用領域の先頭アドレス
- `mccsp_channel_t` チャネル
- `mccsp_sync_t` 同期記述子
  - `mccsp_channel_t ch` チャネル
  - `intptr_t slot` 送受信データ
  - `int rs` 送受信指定 `MCCSP_SEND, MCCSP_RECEIVE`
  - `int active` 記述子の有効・無効

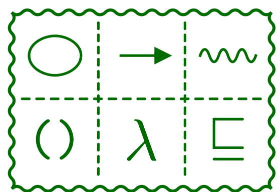
©2015 PRINCIPIA Limited



MCCSP: 関数

- プロセスオブジェクトの作成と削除
  - `mccsp_create_process`
  - `mccsp_delete_process`
- チャネルの作成と削除
  - `mccsp_create_channel`
  - `mccsp_delete_channel`
- 同期
  - `mccsp_sync`

©2015 PRINCIPIA Limited



mccsp_create_process

プロセスオブジェクトを作成する

```
mccsp_process_t mccsp_create_process(size_t size)
```

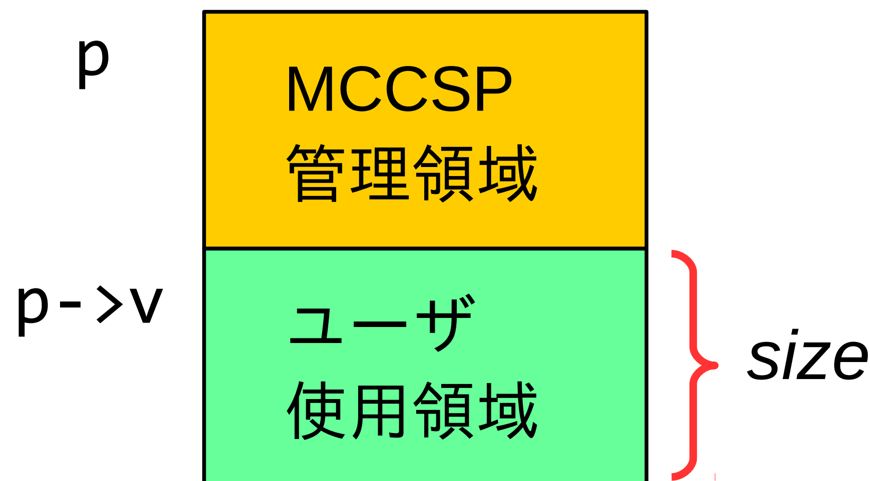
Parameters

size ユーザ使用領域の大きさ

Return Value

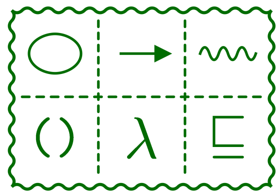
プロセスオブジェクト

作成に失敗した場合は NULL



プロセスオブジェクトの構造

©2015 PRINCIPIA Limited



mccsp_delete_process

プロセスオブジェクトを削除する

```
int mccsp_delete_process(mccsp_process_t p)
```

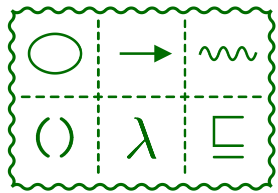
Parameters

p　　プロセスオブジェクト

Return Value

成功した場合は 0，失敗した場合は非0

©2015 PRINCIPIA Limited



mccsp_create_channel

チャネルを作成する

```
mccsp_channel_t mccsp_create_channel(
    const char *name, int id)
```

Parameters

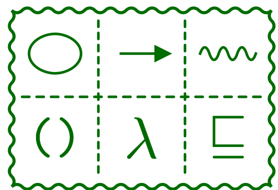
name	チャネル名文字列へのポインタ
id	チャネル ID

Return Value

チャネル

作成に失敗した場合は NULL

©2015 PRINCIPIA Limited



mccsp_delete_channel

チャネルを削除する

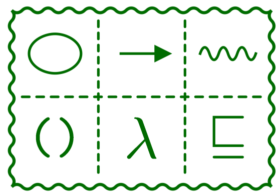
```
int mccsp_delete_channel(mccsp_channel_t ch)
```

Parameters

ch チャネル

Return Value

成功した場合は 0，失敗した場合は非0



mccsp_sync

同期する

```
int mccsp_sync(mccsp_process_t *process,
     mccsp_sync_t *syncv, int num_entries)
```

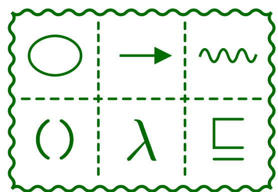
Parameters

process	プロセスオブジェクト
syncv	同期記述子配列へのポインタ
num_entries	同期記述子配列の長さ

Return Value

同期した同期記述子のインデックス
エラーが発生した場合は負数

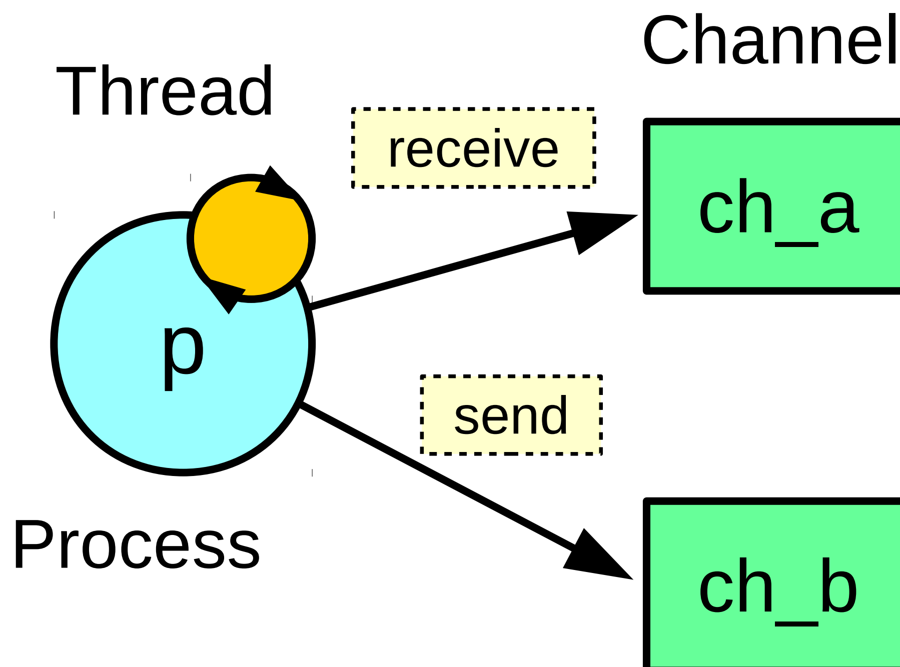
©2015 PRINCIPIA Limited



MCCSP

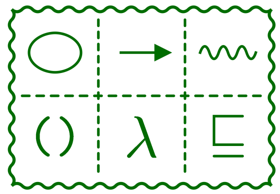
同期記述子の配列 syncv

syncv[0]	ch = ch_a rs = MCCSP_RECEIVE active = 1
syncv[1]	ch = ch_b rs = MCCSP_SEND active = 1 slot = 送信値




```
k = mccsp_sync(p, syncv, 2);
```

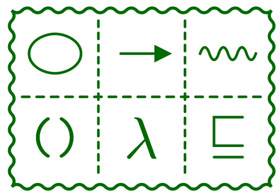
ch_a から受信した場合は 0, ch_b に送信した場合は 1 が返る
受信した値は syncv[0].slot に入る



スレッドの作成と終了

- 作成
  - pthread_create
- 終了
  - return or pthread_exit
- 終了待ち
  - pthread_join

©2015 PRINCIPIA Limited



pthread_create

スレッドを作成する

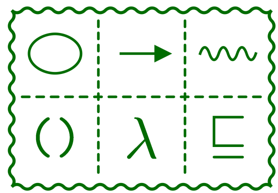
```
int pthread_create(pthread_t *thread,
     const pthread_attr_t *attr,
     void *(*start_routine)(void*), void *arg)
```

Parameters

thread	作成したスレッドの ID を格納する変数へのポインタ
attr	属性（既定値の場合は NULL を指定）
start_routine	スレッドの処理を記述した関数へのポインタ
arg	*start_routine* に渡す引数

Return Value

成功の場合は 0，失敗の場合はエラー番号



pthread_join

スレッドの終了を待つ

```
int pthread_join(pthread_t thread, void **value_ptr)
```

Parameters

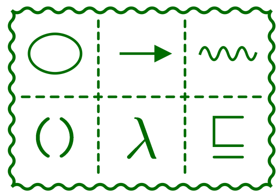
thread スレッド ID

value_ptr スレッドの終了値を格納する変数へのポインタ

Return Value

成功の場合は 0，失敗の場合はエラー番号

©2015 PRINCIPIA Limited



pthread_exit

スレッドを終了する

```
void pthread_exit(void *value_ptr)
```

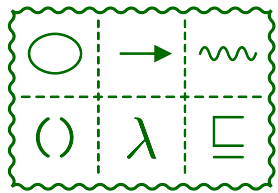
Parameters

value_ptr　　スレッドの終了値

Return Value

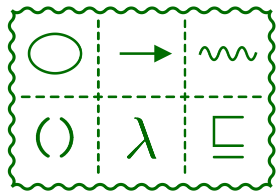
なし

©2015 PRINCIPIA Limited



例: Producers-Consumer

©2015 PRINCIPIA Limited



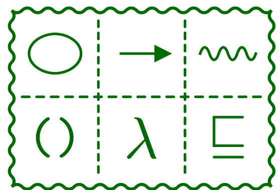
Producer

example1.c

```
void *producer(void *arg) {
    mccsp_process_t self = (mccsp_process_t)arg;
    struct pdesc_producer *p =
             (struct pdesc_producer *)self->v;
    mccsp_sync_t syncv[1];
    int i, k;
    syncv[0].ch = p->ch;
    syncv[0].rs = MCCSP_SEND;
    syncv[0].active = 1;
    for (i = 0; i < M; ++i) {
        syncv[0].slot = p->id * M + i;
        k = mccsp_sync(self, syncv, 1);
    }
    return NULL;
}
```

```
struct pdesc_producer {
  mccsp_channel_t ch;
  int id;
};
```

M: 送信メッセージ数



Consumer (1/2)

example1.c

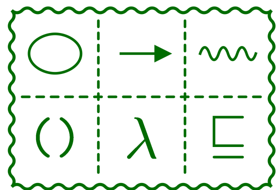
NP: Producer 数

```
void *consumer(void *arg) {
    mccsp_process_t self = (mccsp_process_t)arg;
    struct pdesc_consumer *p =
                (struct pdesc_consumer *)self->v;
    mccsp_sync_t syncv[NP];
    int i, j, k;

    for (i = 0; i < NP; ++i) {
        syncv[i].ch = p->ch[i];
        syncv[i].rs = MCCSP_RECEIVE;
        syncv[i].active = 1;
    }
    ...
```

```
struct pdesc_consumer {
  mccsp_channel_t ch[NP];
};
```

syncv 初期化

©2015 PRINCIPIA Limited



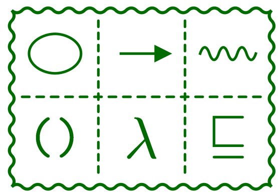
Consumer (2/2)

example1.c

受信ループ

```
void *consumer(void *arg) {
    ...
    for (i = 0; i < M * NP; ++i) {
        k = mccsp_sync(self, syncv, NP);
    }
    return NULL;
}
```

©2015 PRINCIPIA Limited

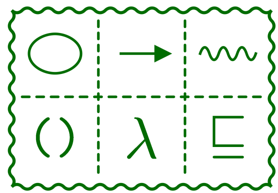


main (1/5)

example1.c

```
int main(void)
{
    mccsp_process_t c, p[NP];
    struct pdesc_producer *pdesc_p;
    struct pdesc_consumer *pdesc_c;
    pthread_t cth, pth[NP];
    int i, r;
    void *retcode;
   ...
```

©2015 PRINCIPIA Limited

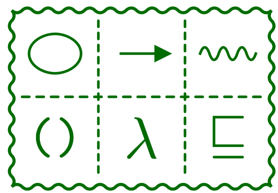


main (2/5)

example1.c

プロセスオブジェクトとチャネルの作成

```
c = mccsp_create_process(
            sizeof(struct pdesc_consumer));
pdesc_c = (struct pdesc_consumer *)c->v;
for (i = 0; i < NP; ++i) {
    p[i] = mccsp_create_process(
                sizeof(struct pdesc_producer));
    pdesc_p = (struct pdesc_producer *)p[i]->v;
    pdesc_p->id = i;
    pdesc_c->ch[i] = pdesc_p->ch =
            mccsp_create_channel("ch", i);
}
```



main (3/5)

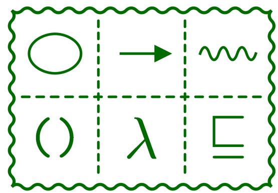
example1.c

スレッド作成

```
for (i = 0; i < NP; ++i) {
    pthread_create(&pth[i], NULL, &producer, p[i]);
}
pthread_create(&cth, NULL, &consumer, c);
```

```
int pthread_create(pthread_t *thread,
     const pthread_attr_t *attr,
     void *(*start_routine)(void*), void *arg)
```

©2015 PRINCIPIA Limited



main (4/5)

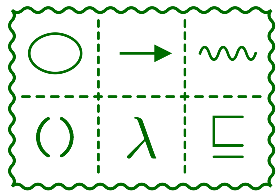
example1.c

スレッド終了待ち

```
for (i = 0; i < NP; ++i) {
    pthread_join(pth[i], &retcode);
}
pthread_join(cth, &retcode);
```

```
int pthread_join(pthread_t thread, void **value_ptr)
```

©2015 PRINCIPIA Limited



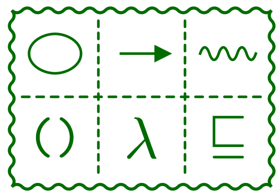
main (5/5)

example1.c

クリーンアップ: プロセスオブジェクトとチャネルの削除

```
for (i = 0; i < NP; ++i) {
    mccsp_delete_channel(pdesc_c->ch[i]);
    mccsp_delete_process(p[i]);
}
mccsp_delete_process(c);
```

©2015 PRINCIPIA Limited



実行例

Producer 数 NP = 6
送信数 M = 10

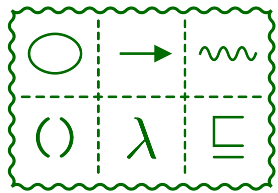
送受信でメッセージ表示

```
consumer sync 3 received 35
producer 3 sent 35
...
producer 2 sent 28
consumer sync 3 received 38
producer 3 sent 38
consumer sync 2 received 29
producer 2 sent 29
consumer sync 3 received 39
producer 3 sent 39
consumer sync 4 received 40
producer 4 sent 40
consumer sync 5 received 50
producer 5 sent 50
consumer sync 4 received 41
producer 4 sent 41
consumer sync 5 received 51
producer 5 sent 51
consumer sync 4 received 42
producer 4 sent 42
consumer sync 5 received 52
producer 5 sent 52
consumer sync 4 received 43
producer 4 sent 43
consumer sync 5 received 53
producer 5 sent 53
consumer sync 4 received 44
producer 4 sent 44
consumer sync 5 received 54
producer 5 sent 54
consumer sync 4 received 45
producer 4 sent 45
consumer sync 5 received 55
producer 5 sent 55
consumer sync 4 received 46
producer 4 sent 46
consumer sync 5 received 56
producer 5 sent 56
consumer sync 4 received 47
producer 4 sent 47
consumer sync 5 received 57
producer 5 sent 57
consumer sync 4 received 48
producer 4 sent 48
consumer sync 5 received 58
producer 5 sent 58
consumer sync 4 received 49
producer 4 sent 49
consumer sync 5 received 59
producer 5 sent 59

hatsugai@ephesus ~/projects/mccsp/trunk/rc0
$
```

各 Producer からの受信数カウント

```
 3  0  0  0  0  0
 4  0  0  0  0  0
...
10  3  1  0  0  0
10  3  2  0  0  0
10  4  2  0  0  0
10  4  3  0  0  0
10  5  3  0  0  0
10  5  4  0  0  0
10  6  4  0  0  0
10  6  5  0  0  0
10  7  5  0  0  0
10  7  5  1  0  0
10  8  5  1  0  0
10  8  6  1  0  0
10  9  6  1  0  0
10  9  7  1  0  0
10 10  7  1  0  0
10 10  8  1  0  0
10 10  9  1  0  0
10 10 10  1  0  0
10 10 10  2  0  0
10 10 10  3  0  0
10 10 10  4  0  0
10 10 10  5  0  0
10 10 10  6  0  0
10 10 10  7  0  0
10 10 10  8  0  0
10 10 10  9  0  0
10 10 10 10  0  0
10 10 10 10  1  0
10 10 10 10  1  1
10 10 10 10  1  2
10 10 10 10  2  2
10 10 10 10  3  2
10 10 10 10  4  2
10 10 10 10  5  2
10 10 10 10  6  2
10 10 10 10  7  2
10 10 10 10  8  2
10 10 10 10  9  2
10 10 10 10 10  2
10 10 10 10 10  3
10 10 10 10 10  4
10 10 10 10 10  5
10 10 10 10 10  6
10 10 10 10 10  7
10 10 10 10 10  8
10 10 10 10 10  9
10 10 10 10 10 10

hatsugai@ephesus ~/projects/mccsp/trunk/rc0
$
```

©2015 PRINCIPIA Limited



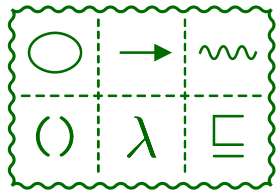
実行例

送信後，ランダムに待ち時間を入れた場合

~/projects/mccsp/trunk/rc0

```
  1  1  1  0  0  0
  1  1  1  1  0  0
  1  1  1  1  1  0
  1  1  1  1  1  1
  1  1  2  1  1  1
  2  1  2  1  1  1
  2  2  2  1  1  1
  2  2  2  2  1  1
  2  2  2  2  2  1
  2  2  2  2  2  2
  2  2  2  3  2  2
  3  2  2  3  2  2
  3  3  2  3  2  2
  3  3  3  3  2  2
  3  3  3  3  3  2
  3  3  3  3  3  3
  3  3  3  4  3  3
  4  3  3  4  3  3
  4  4  3  4  3  3
  4  4  4  4  3  3
  4  4  4  4  4  3
  4  4  4  4  4  4
  4  4  4  5  4  4
  4  4  4  6  4  4
  4  5  4  6  4  4
  5  5  4  6  4  4
  5  6  4  6  4  4
  5  6  5  6  4  4
  6  6  5  6  4  4
  6  6  6  6  4  4
  6  6  6  7  4  4
  6  6  6  7  5  4
  6  6  6  7  5  5
  6  6  6  7  6  5
  6  6  6  7  6  6
  6  6  6  7  7  6
  7  6  6  7  7  6
  7  7  6  7  7  6
  7  7  7  7  7  6
  7  7  7  7  7  7
  7  7  7  8  7  7
  8  7  7  8  7  7
  8  8  7  8  7  7
  8  8  8  8  7  7
  8  8  8  8  8  7
  8  8  8  8  8  8
  8  8  8  9  8  8
  8  9  8  9  8  8
  9  9  8  9  8  8
  9  9  9  9  8  8
  9  9  9  9  9  8
  9  9  9  9  9  9
  9  9  9 10  9  9
  9  9  9 10 10  9
 10  9  9 10 10  9
 10 10  9 10 10  9
 10 10 10 10 10  9
 10 10 10 10 10 10

hatsugai@ephesus ~/projects/mccsp/trunk/rc0
$
```

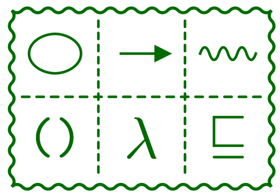
©2015 PRINCIPIA Limited



演習: スケジューラの実装

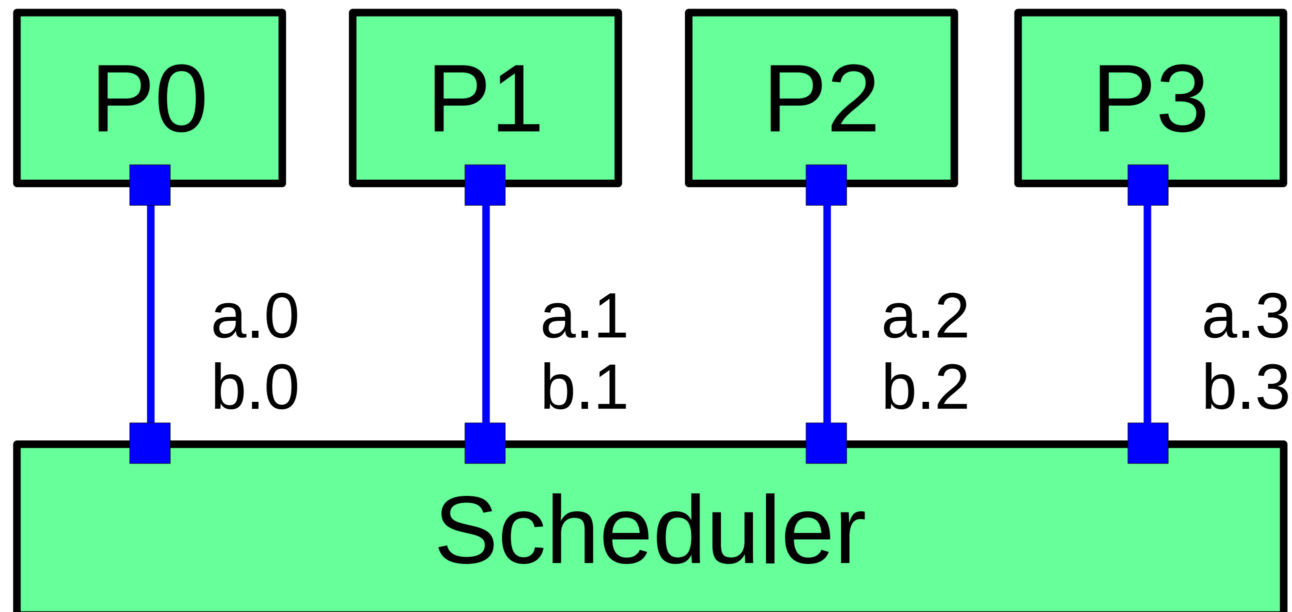
- スケジューラを実装してください
  - 仕様を1つのプロセスとして実装してください
    - 中央集権型のスケジューラになります
  - 分散型のスケジューラを実装してください
    - バグがある版と修正版の2つを実装してください

©2015 PRINCIPIA Limited

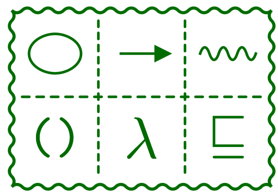


スケジューリング問題

- 各プロセス Pk は仕事の開始許可 a.k を受けて仕事を開始し，終了したら b.k で通知する
- 仕事を開始する順序は a.0, a.1, a.2, a.3 の順序で循環しなければならない

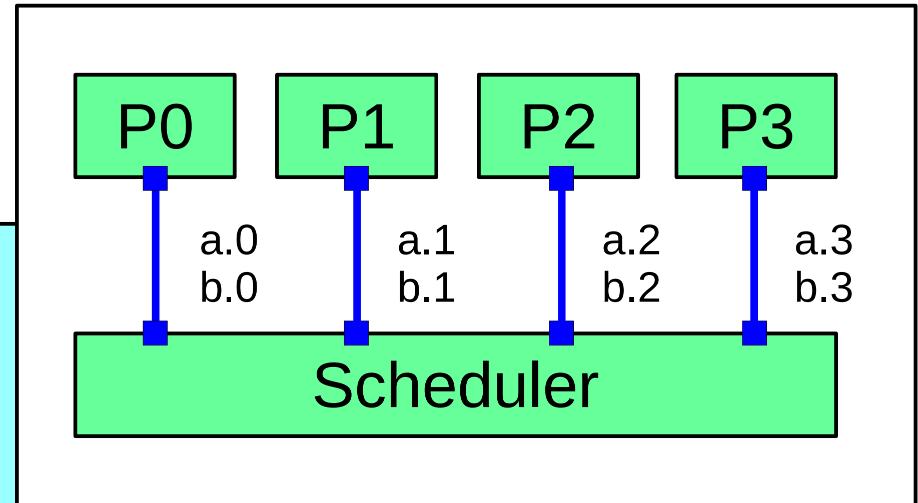


出典: Communicating and Mobile Systems: the π-calculus, Robin Milner, 1999



スケジューラの仕様

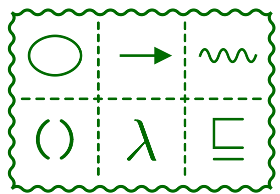
中央集権型スケジューラ



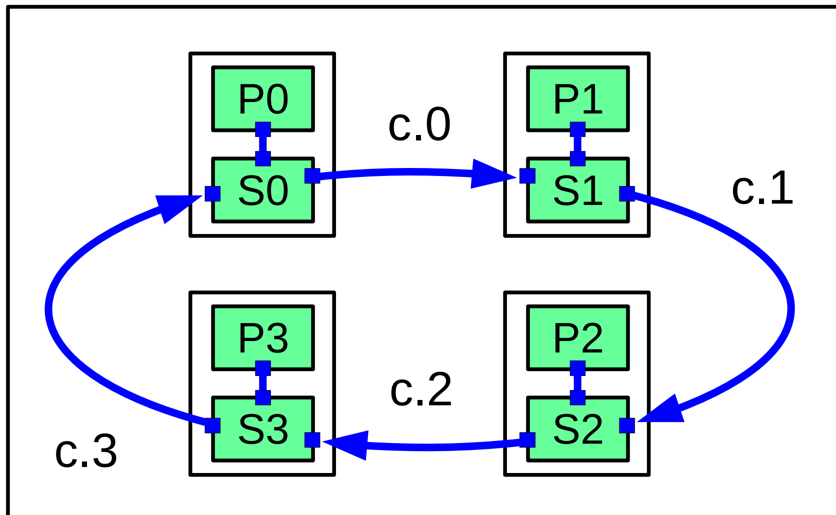

```
(define-process SPEC (S 0 '()))

(define-process (S k ws)
  (alt
    (if (member k ws)
        STOP
        (! a (k)
           (S (mod (+ k 1) N) (cons k ws))))
    (xalt j ws
      (! b (j) (S k (remove j ws))))))
```

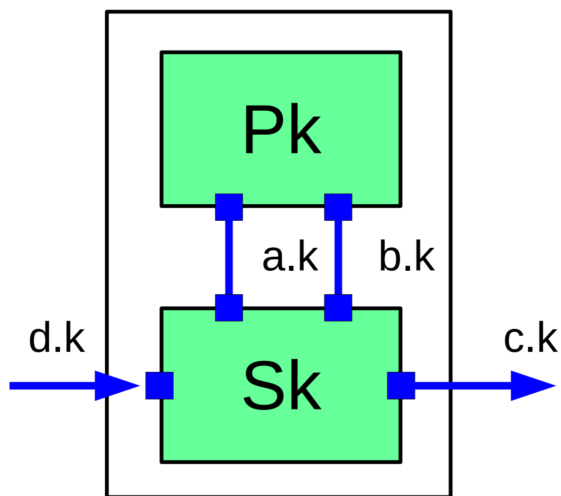
k:　次に開始するプロセス
ws: 仕事中のプロセスリスト

©2015 PRINCIPIA Limited

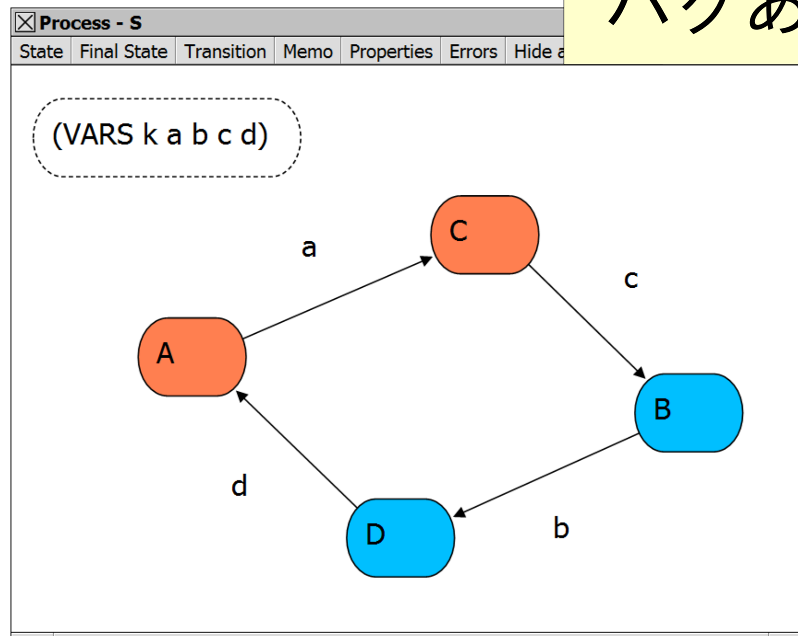


分散型スケジューラ

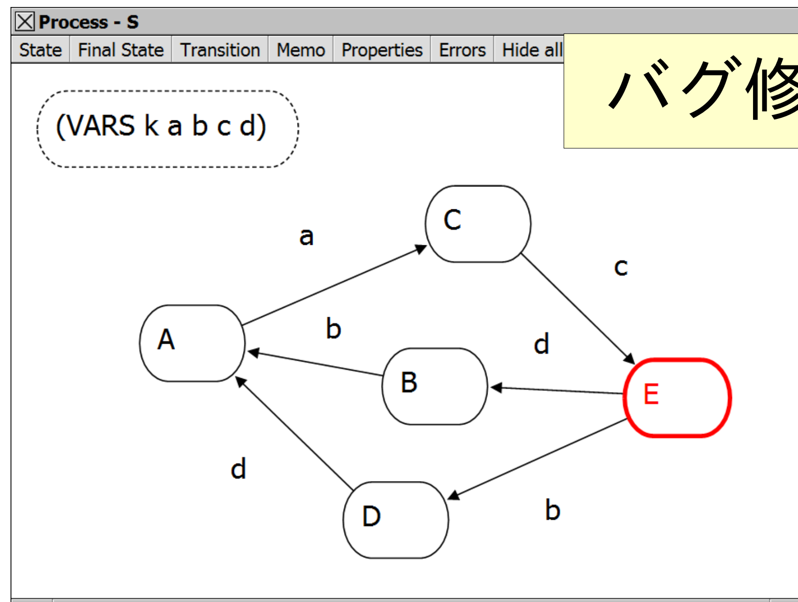




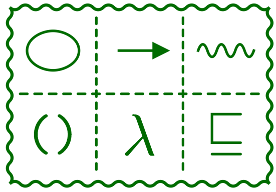
バグあり版



バグ修正版



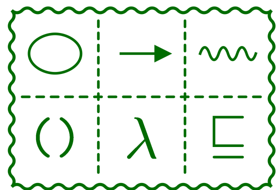
©2015 PRINCIPIA Limited



Pk: worker thread

```
void *worker_thread(void *arg) {
    mccsp_process_t self = (mccsp_process_t)arg;
    struct worker *p = (struct worker *)self->v;
    mccsp_sync_t syncv_a[1], syncv_b[1];

    syncv_a[0].ch = p->a;
    syncv_a[0].rs = MCCSP_RECEIVE;
    syncv_b[0].ch = p->b;
    syncv_b[0].rs = MCCSP_SEND;
    syncv_a[0].active = syncv_b[0].active = 1;

    while (1) {
        mccsp_sync(self, syncv_a, 1);
        mccsp_sync(self, syncv_b, 1);
    }
    return NULL;
}
```

```
struct worker {
  int id;
  mccsp_channel_t a;
  mccsp_channel_t b;
};
```



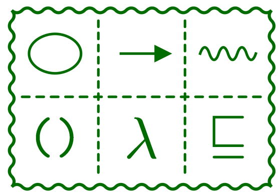
Pk: worker thread

sched_worker.c

スケジューラが誤ったイベントを提示していないかチェックする

```
mccsp_sync_t syncv[2];
int k;

syncv[0].ch = p->a;
syncv[0].rs = MCCSP_RECEIVE;
syncv[1].ch = p->b;
syncv[1].rs = MCCSP_SEND;
syncv[0].active = syncv[1].active = 1;

while (1) {
    k = mccsp_sync(self, syncv, 2);
    assert(k == 0);
    k = mccsp_sync(self, syncv, 2);
    assert(k == 1);
}
```

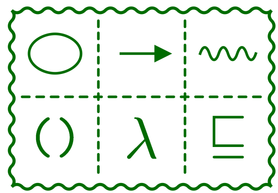
a, b 両方待つ



チャネル作成

```
mccsp_channel_t a[N], b[N];
int i;

for (i = 0; i < N; ++i) {
    a[i] = mccsp_create_channel("a", i);
    b[i] = mccsp_create_channel("b", i);
}
```

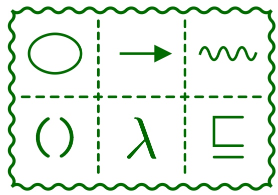
©2015 PRINCIPIA Limited



Worker 起動

```
pthread_t th;
int i;

for (i = 0; i < N; ++i) {
    struct worker *p;
    worker[i] = mccsp_create_process(
                     sizeof(struct worker));
    p = (struct worker *)worker[i]->v;
    p->id = i;
    p->a = a[i];
    p->b = b[i];
    pthread_create(&th, NULL, &worker_thread, worker[i]);
}
```

©2015 PRINCIPIA Limited



仕様（中央集権型）実装のヒント

- 次に仕事を開始する worker ID を k, 仕事中の worker ID 集合を ws とするとき

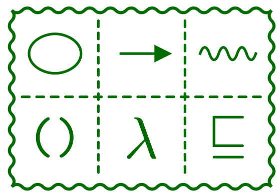
$k \in ws$ ならばスケジューラが提示する

イベントの集合は $\{ b_j \mid j \in ws \}$

$k \notin ws$ ならばスケジューラが提示する

イベントの集合は $\{ a_k \} \cup \{ b_j \mid j \in ws \}$

- a, b すべてを含む N * 2 の大きさの syncv を用意する
- syncv のメンバ active を使って集合 ws を表す



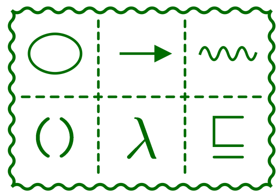
クイズ

- バグがある場合とない場合で差が出るような Worker の状況を考えてください

Worker のメインループ

```
while (1) {
    k = mccsp_sync(self, sync, 2);
    assert(k == 0);
    if (...) sleep(...);
    k = mccsp_sync(self, sync, 2);
    assert(k == 1);
    if (...) sleep(...);
}
```

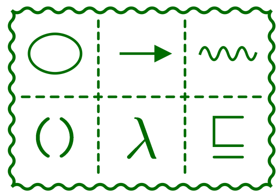
©2015 PRINCIPIA Limited



中央集権型スケジューラ (1/2)

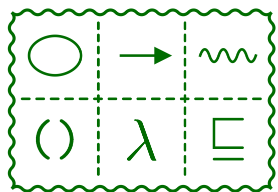
```
mccsp_sync_t syncv[N * 2];

sched = mccsp_create_process(0);

for (i = 0; i < N; ++i) {
    syncv[i].ch = a[i];
    syncv[i].rs = MCCSP_SEND;
    syncv[i].active = (i == 0);
    syncv[i + N].ch = b[i];
    syncv[i + N].rs = MCCSP_RECEIVE;
    syncv[i + N].active = 0;
}
```

©2015 PRINCIPIA Limited



中央集権型スケジューラ (2/2)

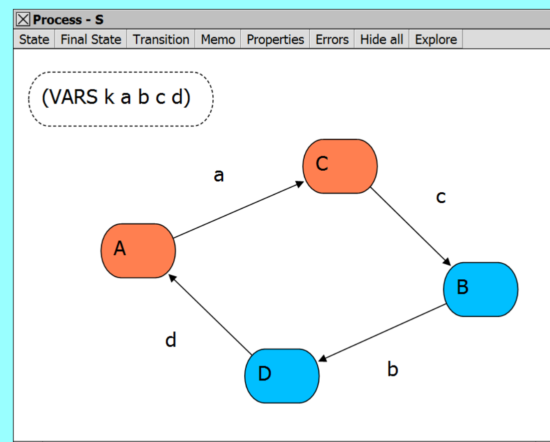
```
while (true) {
    k = mccsp_sync(sched, syncv, N * 2);
    if (k == next) {
        syncv[next].active = 0;
        syncv[next + N].active = 1;
        next = (next + 1) % N;
        if (!syncv[next + N].active)
            syncv[next].active = 1;
    } else {
        syncv[k].active = 0;
        if (k - N == next)
            syncv[next].active = 1;
    }
}
```

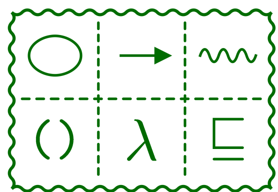


分散型スケジューラ: バグあり

```
mccsp_sync_t sync_A[1], sync_C[1], sync_B[1], sync_D[1];

sync_A[0].ch = p->a;
sync_A[0].rs = MCCSP_SEND;
sync_A[0].active = 1;

sync_C[0].ch = p->c;
sync_C[0].rs = MCCSP_SEND;
sync_C[0].active = 1;

sync_B[0].ch = p->b;
sync_B[0].rs = MCCSP_RECEIVE;
sync_B[0].active = 1;

sync_D[0].ch = p->d;
sync_D[0].rs = MCCSP_RECEIVE;
sync_D[0].active = 1;
```

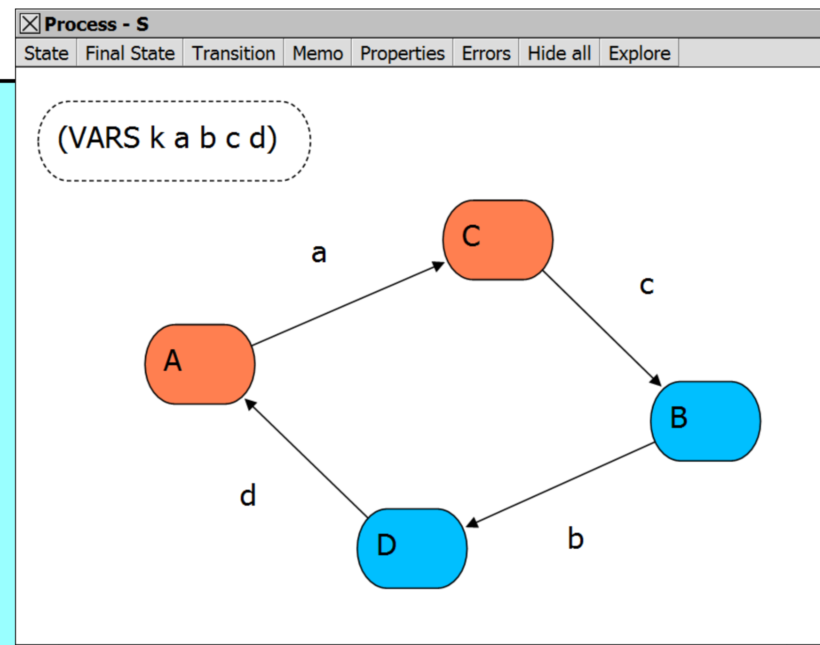
```
struct controller {
  int id;
  mccsp_channel_t a;
  mccsp_channel_t b;
  mccsp_channel_t c;
  mccsp_channel_t d;
};
```



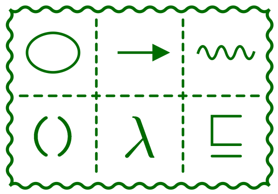


分散型スケジューラ: バグあり

```
if (p->id > 0) {
   mccsp_sync(self, sync_D, 1);
}

while (true) {
   mccsp_sync(self, sync_A, 1);
   mccsp_sync(self, sync_C, 1);
   mccsp_sync(self, sync_B, 1);
   mccsp_sync(self, sync_D, 1);
}
```

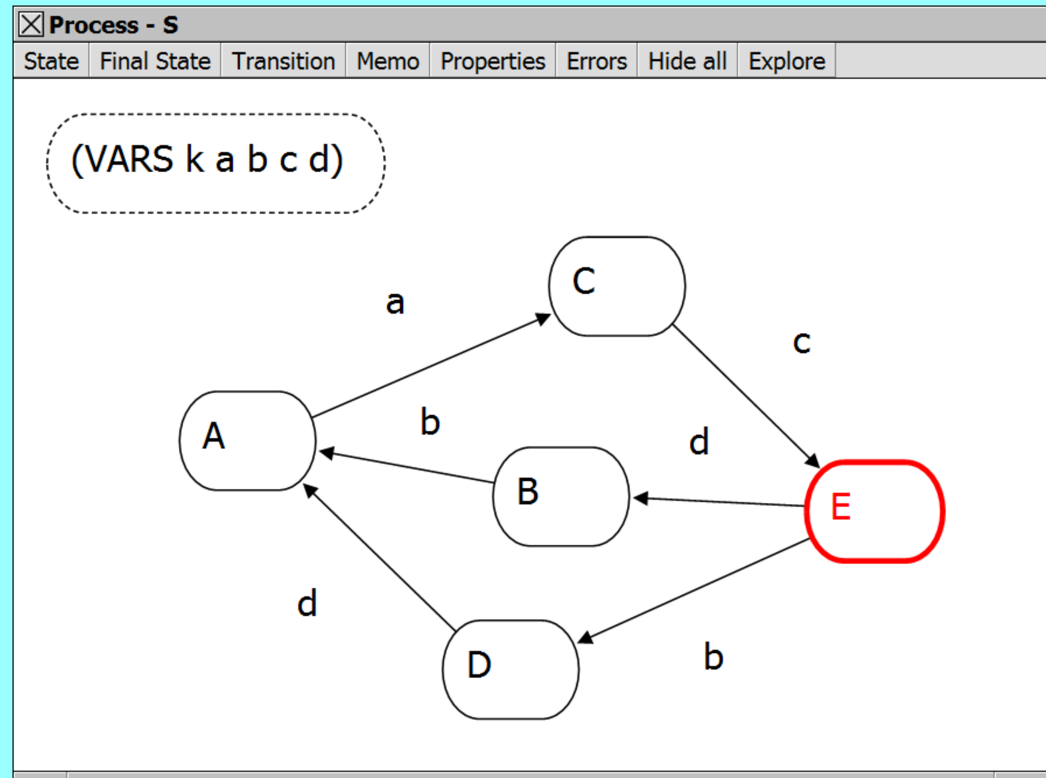


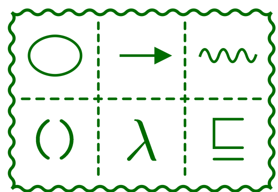
©2015 PRINCIPIA Limited



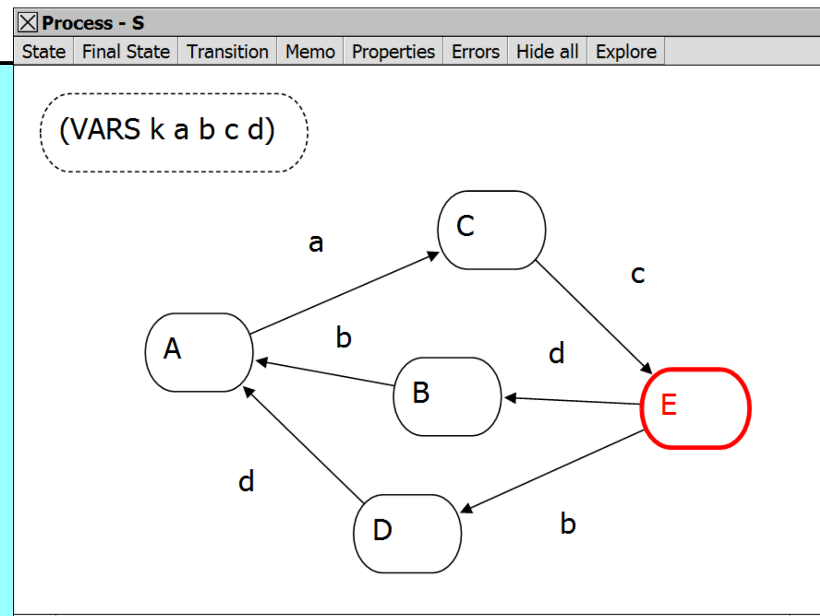
分散型スケジューラ: バグなし

```
mccsp_sync_t sync_A[1], sync_C[1], sync_B[1], sync_D[1], sync_E[2];

sync_A[0].ch = p->a;
sync_A[0].rs = MCCSP_SEND;

sync_C[0].ch = p->c;
sync_C[0].rs = MCCSP_SEND;

sync_B[0].ch = p->b;
sync_B[0].rs = MCCSP_RECEIVE;

sync_D[0].ch = p->d;
sync_D[0].rs = MCCSP_RECEIVE;

sync_E[0].ch = p->d;
sync_E[0].rs = MCCSP_RECEIVE;
sync_E[1].ch = p->b;
sync_E[1].rs = MCCSP_RECEIVE;

sync_A[0].active = sync_C[0].active = sync_B[0].active =
  sync_D[0].active = sync_E[0].active = sync_E[1].active = 1;
```



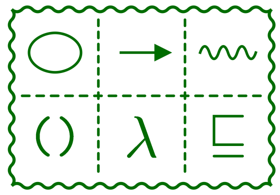


分散型スケジューラ: バグなし




```
if (p->id > 0) {
   mccsp_sync(self, sync_D, 1);
}

while (true) {
   mccsp_sync(self, sync_A, 1);
   mccsp_sync(self, sync_C, 1);
   k = mccsp_sync(self, sync_E, 2);
   if (k == 0)
      mccsp_sync(self, sync_B, 1);
   else
      mccsp_sync(self, sync_D, 1);
}
```

©2015 PRINCIPIA Limited



まとめ

- CSP 実装支援ライブラリを使うと，CSP モデルの構造そのままに直接実装することができる
  - 実装の段階で問題を混入する可能性を減らせる
  - モデル修正時の対応も容易