

STOC事例WG デイスクッションメモ
SpecCの改良へ向けての検討事項について

作成者：高田 広章，本田 晋也（豊橋技術科学大学）
最終更新：2001年2月26日

【並列実行のセマンティクスについて】

par のセマンティクスの問題

前回のWGで提示した通り，par のセマンティクスが疑問がある．具体的には，par による並列実行の単位が，ノンプリエンテイクタに実行されるのか（その場合，どのオペレーションでスケジューリングが起ころか？），プリエンテイクタに実行されるのである．

前回のWGでは，ノンプリエンテイクタという話であったが，その後の他の関係者に確認したところ，プリエンテイクタという回答だった．

例えば，

```
int x = 0;
int y = 0;

behavior a {
  main() {
    x = y + 1;
  }
}

behavior b {
  main() {
    y = x + 1;
  }
}
```

と定義されている時に，

```
par {
  a.main();
  b.main();
}
```

を実行した場合に，x と y がどうなるかが定義されない．ノンプリエンテイクタに実行されるなら，x = 1, y = 2 か，x = 2, y = 1 になる．それに対して，プリエンテイクタに実行されるなら，x = 1, y = 1 になる可能性もある．

ノンプリエンテイクタ側からは，プリエンテイクタの方が良いと思われる．ただし，その場合には，排他制御のためのプリミティブと，一時的に例外処理を禁止するためのプリミティブが必要である．一方ハードウェア側からは，変数参照は一つ前の状態を参照し，必ず x = 1, y = 1 になるのが都合が良いと思われる（VHDL で process の外に定義した変数だとこうなる）．

排他制御のプリミティブ

排他制御プリミティブとしては，wait/notify と相性のよいモニタの導入が有力候補であろう．モニタからハードウェア生成できるかについては，模射が必要である．モニタのシンタックスとしては，Java の synchronized が参考に

なる．ただし，Java の synchronized は制限が強いという意見を聞いたことがある（詳しくは知らないが）ので，その点も調査が必要であろう．

channel は自動的にモニタであると考えれば，これまでのサンプルプログラムの多くはそのまま使える．

一時的に例外処理を禁止するプリミティブ

一時的に例外処理を禁止するためのプリミティブがないと，非同期な例外処理時に一貫性を保つことが難しくなる．

一時的に例外処理を禁止するためのプリミティブにより，一時的にノンプリエンテイクタにもなるものとするれば，排他制御のプリミティブと共用できる可能性もある（要検討）．

ちなみに，Java には非同期な例外処理がない（言い換えると，他のスレッドに例外処理を要求することはできない）ため，一時的に例外処理を禁止するプリミティブは必要ない．

バッファ付き変数

VHDL で process の外に定義した変数と同等の振舞いをする変数を，ここでは，バッファ付き変数と呼ぶ．バッファ付き変数を導入する場合は，通常の変数と明確に区別して扱うべきである．例えば，

```
buffered int var @ event;
```

により，var を notify(event) される度に更新される/バッファ付き変数と定義する方法が考えられる．最初の例では，x と y を buffered にすることにより，必ず x = 1, y = 1 になる．

【pipe について】

まだ記述に使っていないため感性的な議論ではあるが，他のプリミティブが基本的な機能であるのと比較して，やや複雑なプリミティブが混ざっているという印象がある．

```
pipe {
  a.main();
  b.main();
  c.main();
}
```

は，pipelined を使わない限り，

```
par {
  a.main();
}
par {
  a.main();
  b.main();
}
while (1) {
  par {
    a.main();
    b.main();
    c.main();
  }
}
```

と等価であると考えられ、par で記述してもそれほど複雑さではない。逆に、piped を使った場合、このようには展開できない。むしろ、piped を展開できるようなプリミティブ（上で述べたバッファ付き変数が有力候補）を導入すべきではないかと思う。また、暗黙に無限ループになるのも、プリミティブとしてはどうかと思う。

無限ループについては、一つのアイデアに過ぎないが、pipe に繰り返し条件を導入する方法が考えられる。繰り返し条件は、「入力がある」という条件が使いやすいと思われる。つまり、

```
pipe (条件式) {
  a.main();
  b.main();
  c.main();
}
```

は、

```
if (条件式) {
  a.main();
  if (条件式) {
    par {
      a.main();
      b.main();
    }
    while (条件式) {
      par {
        a.main();
        b.main();
        c.main();
      }
    }
  }
  par {
    b.main();
    c.main();
  }
}
else {
  b.main();
  c.main();
}
/* 入力が一つのみ */
}
```

【その他の検討項目】

詳しくは検討していないが、以下の点も検討を要すると思われる。

- ・並列実行単位の動的スケジューリング
- ・behavior の main 以外のメソッドの意義
- ・fsm のシンタックスとセマンティクス
- ・イベントのOR待ちで、どのイベントで起こされたかの判別方法
- ・イベントのAND待ち

以上