JAIST Repository
https://dspace.jaist.ac.jp/

Title	モデル検査のためのアスペクト指向でのモデル記述支援環境に関する研究
Author(s)	大野，真一朗
Citation	
Issue Date	2008-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/4297
Rights	
Description	Supervisor:岸知二，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

モデル検査のためのアスペクト指向での
モデル記述支援環境に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

大野　真一朗

2008 年 3 月

修 士 論 文

モデル検査のためのアスペクト指向での モデル記述支援環境に関する研究

指導教官　岸　知二　特任教授

審査委員主査　岸　知二　特任教授
審査委員　片山　卓也　教授
審査委員　落水　浩一郎　教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

610017 大野　真一朗

提出年月: 2008 年 2 月

Copyright © 2008 by Shinichiro Ohno

概 要

本研究では、モデル検査器 SPIN を用いた検証における仕様記述言語でのモデル記述をアスペクト指向技術を用いて支援するための、文法、言語処理系、検証方法を提案する。ある検証項目を検証するために「性質を確認するための記述」が横断的に現れるモデル記述に対して、他の検証項目を検証したい際に、この部分を人手により書き換えるのは煩雑である。

よって、この部分をアスペクトとしてカプセル化し検証項目ごとに、アスペクトを変更するだけで、一つのモデルに対して様々な検証行える方式を実現する。

目 次

第 1 章　はじめに **1**
1.1　研究背景 1
1.1.1　社会的背景 1
1.1.2　SPIN での検証 1
1.1.3　検証モデルの変化 1
1.1.4　「確認したい性質の変化」に伴う「モデルの変化」とその問題点 2
1.1.5　モデルに横断的な変更 3
1.2　論文の構成 3

第 2 章　目的 **5**
2.1　解決したい問題点 5
2.2　研究目的とアプローチ 6
2.3　本研究の付帯的効果 6

第 3 章　関連技術 **9**
3.1　SPIN/promela 9
3.1.1　プロセス 9
3.1.2　データオブジェクト 10
3.1.3　メッセージチャネル 11
3.1.4　制御フロー 14
3.1.5　ラベル 16
3.2　アスペクト指向技術 17
3.2.1　アスペクトの例 17
3.2.2　ジョインポイントモデル 17

第 4 章　言語の設計思想 **20**
4.1　アスペクト指向言語でのジョインポイントの指定 20
4.1.1　ジョインポイント 20
4.1.2　ポイントカット 20
4.2　promela に対する考察 21
4.2.1　promela でのチャネルの重要性 21
4.2.2　promela でのジョインポイントの指定 21
4.3　文法の要件 22
4.4　本文法でのジョインポイント 22
4.5　要件の実現 22
4.5.1　要件 1 の実現 22

4.5.2 要件２の実現 22
4.6 本文法での範囲指定の考え方 23
4.6.1 範囲を持つ言語要素 23
4.6.2 範囲に名前を付ける方法 23
4.6.3 任意の範囲を指定する方法 24
4.6.4 範囲の指定の意味 25
4.6.5 範囲を意味１で扱う利点 25
4.6.6 範囲を意味２で扱う利点 25
4.6.7 本文法での範囲の扱い 27

第 5 章 文法の提案 **28**
5.1 アドバイス 29
5.2 範囲を持たない言語要素のポイントカット 30
5.2.1 chan（チャネル操作） 30
5.2.2 else , break 33
5.2.3 xs, xr, goto, print, assert, expr, decl, assign 34
5.3 範囲を持つ言語要素のポイントカット 36
5.3.1 if, do, atomic, d_step, option, init, never, trace, notrace 36
5.3.2 unless 39
5.3.3 proctype 41
5.3.4 label 43
5.4 ポイントカットの演算子、操作 46
5.4.1 ALLSTMNT 46
5.4.2 AND,OR 49
5.4.3 CONTAINS 51
5.4.4 REMOVESCOPE 54
5.5 その他の規則 56
5.5.1 アスペクトの評価順序 56
5.5.2 演算子の結合規則 56
5.5.3 セパレータの扱い 56

第 6 章 実装 **57**
6.1 言語処理系の概要 57
6.2 言語処理系設計思想 58
6.2.1 内部モデルを用いる理由 58
6.2.2 内部モデルに XML を用いる理由 60
6.3 promela から XML への変換 60
6.4 アスペクトから Xpath 式への変換 60
6.5 ジョインポイントの絞込み 61
6.6 アドバイスの作用 61
6.7 XML から promela への変換 62

第 7 章　適用例 **63**
7.1　適用例１：同一の検証対象に対して、異なった検証を行う際への適用 63
7.1.1　検証１：チャネル受信値のアサーションによるチェック 64
7.1.2　検証２：else ガードの付け替えによる、考慮外のメッセージ受信の検出 65
7.1.3　検証３：STATE1 の具象化 65
7.1.4　検証４：STATE2,3 の抽象化 66
7.2　適用例２：類似しているが、異なる検証対象のモデルを効率的に記述する際への適用 66
7.2.1　OSEK/VDX 66
7.2.2　検証モデル 67
7.3　適用例に対する考察 69
7.3.1　適用例１ 69
7.3.2　適用例２ 70

第 8 章　まとめ **72**
8.1　まとめ 72
8.2　今後の課題 72
8.2.1　本研究の文法におけるアドバイス 72
8.2.2　扱えないモデル 72
8.3　今後の展望 73

第 9 章　最後に **74**
9.1　謝辞 74

付 録 A　promela の BNF **77**

付 録 B　OSEK/VDX **80**
B.1　状態遷移図 80
B.2　ウィーブ前の promela 83
B.3　ウィーブ後の promela 85

図 目 次

1.1 注目している範囲の変化 2
1.2 検証モデルの抽象度の変化 3

2.1 類似しているが異なる検証対象の例 7
2.2 モデルの書き換え 8

3.1 チャネルのイメージ 11
3.2 ログ出力の例 18
3.3 AspectJ でのジョインポイントモデル 19

4.1 promela の局所化 21
4.2 範囲の意味の違いによるウィーブの違い 26
4.3 提案する局所化手法 27

5.1 if,do ポイントカット用いたウィーブ 38
5.2 unless(before) ポイントカットの例 40
5.3 unless(after) ポイントカットの例 40
5.4 ラベルの範囲の図 44
5.5 allStmnt 47
5.6 AND によるジョインポイントの絞込み 50
5.7 contains の説明図 52
5.8 contains の動作 53

6.1 言語処理系の全体図 57
6.2 避けたい処理方式 59
6.3 処理の分割 59

7.1 OSEK/VDX における基本タスクと資源の状態遷移 67
7.2 モデルより取り除く遷移 68

B.1 タスク２、資源なしの複合状態 81
B.2 タスク２、資源１の複合状態 82

ソースコード目次

2.1 元々のモデル 5
2.2 検証を行うためにアサーションを埋め込んだモデル 6
3.1 proctype の例 9
3.2 active をつけた proctype の例 10
3.3 run 式で proctype をインスタンス化する例 10
3.4 mtype の例 11
3.5 atomic の例 14
3.6 d_step の例 14
3.7 if の例 15
3.8 do の例 15
3.9 unless の例 16
3.10 unless の例２ 16
3.11 ラベルの付いた中括弧で複文を囲う例 17
5.1 提案する文法の BNF 28
5.2 chan ポイントカットの例（ウィーブ前) 31
5.3 chan ポイントカットの例（ウィーブ後） 31
5.4 else break ポイントカットの例（ウィーブ前） 33
5.5 else break ポイントカットの例（ウィーブ後） 33
5.6 assert expr ポイントカットの例（ウィーブ前） 35
5.7 assert expr ポイントカットの例（ウィーブ後） 35
5.8 proctype ポイントカットの例（ウィーブ前） 41
5.9 proctype ポイントカットの例（ウィーブ前） 41
5.10 ラベルの範囲 43
5.11 label ポイントカット（ウィーブ後) 44
5.12 allStmnt の説明用モデル 46
5.13 allStmnt の説明用モデル (ウィーブ後) 48
5.14 and によるポイントカットの絞込み (ウィーブ前) 49
5.15 and によるポイントカットの絞込み (ウィーブ後) 49
5.16 contains の説明用モデル 1 51
5.17 contains の説明用モデル 2(ウィーブ後) 52
5.18 List5.12 と同様のモデル 54
5.19 List5.12 と同様のモデル（ウィーブ後１) 54
5.20 List5.12 と同様のモデル（ウィーブ後２) 55
7.1 検証モデル 63
B.1 タスク２つ資源１つのモデル 83

B.2 osek のシステムシステムコール (osekoslib.spin) 84
B.3 ウィーブ後の promela 85

表 目 次

3.1 基本データ型 10
3.2 定義済み関数一覧 13

4.1 範囲を持つ言語要素とそうでないものの分類 23

5.1 操作の一覧 31
5.2 各ポイントカットが選び出すジョインポイント 34
5.3 各言語要素でのフィールドの意味の違い 35
5.4 各ポイントカットが選び出すジョインポイント[1] 36
5.5 各ポイントカットがブロックとして扱う範囲 37
5.6 演算子の結合優先度、結合方向 56

6.1 各処理とソースコードの対応 58
6.2 アドバイスと処理メソッドの対応 62

7.1 取り除く遷移の遷移先 68
7.2 本研究の提案手法と不変表明による検証時の状態数比較 70

第1章　はじめに

1.1　研究背景

1.1.1　社会的背景

近年、ソフトウェアは金融、公共、産業、交通システムから自動車、家電など至る所で利用されている。そのため、そのソフトウェアの誤りが、社会与える影響は大きくソフトウェアの信頼性向上が大きな課題となっている。そうした背景の中、ソフトウェアの正しさを確かめる手法の一つであるモデル検査 [3] が、ソフトウェア検証の新たな選択肢として注目されている。モデル検査は、検証対象の有限状態モデルと、そのモデル上で満たしたい論理的性質を与えると、モデル上でその性質が成り立つかを自動的に検証できる技術であり、そうしたモデル検査を支援するツールの一つに SPIN[1][11] がある。

1.1.2　SPIN での検証

SPIN で検証を行うには、仕様記述言語 promela で検証対象をモデル化し、SPIN を用いて検証器を生成し、その検証器を実行することで、仕様の正しさを確かめる。モデルに対して論理的な性質を与えたいときは時相論理式 (LTL) という時間に関して言及することが出来る論理式を用いて性質を記述し検証を行う。

1.1.3　検証モデルの変化

検証時に記述される promela による検証モデルは、検証目的に基づいて記述される為、一般に検証目的に応じて異なったものとなる。例えば、検証目的が「ある機能Aに関わる性質の検証」の場合には、「機能Aに注目したモデル」を作成する必要がある。次に、検証目的が「ある機能Bに関わる性質の検証」に変化した場合には、先ほどのモデルを「機能Bに注目したモデル」へと書き換える必要がある。この様に、検証目的が変化するとそれに対応して、検証モデルも変化する。

検証目的の変化による検証モデルの変化の理由には、次のようなものが挙げられる。

1. 検証により確認したい性質の変化
   先ほどの例では、ある性質を検証するためにモジュールAの検証モデルを記述したが、それ以外の性質についても検証したい場合には、検証モデルの変更が必要である。

2. 注目している範囲の変化
   先ほどの例では、モジュールAについて注目して検証モデルを記述したが、範囲の変化の一例として、そのほかのモジュールも含めたモデルに関しても検証を行いたい場合がある。この場合にはA以外にもモジュールB、C、D…とモデル化の範囲を広げて必要があり、検証モデルの変更が必要である。（図 1.1）

3. 検証モデルの抽象度の変化
   検証モデルは検証目的に応じて、検証対象から必要な部分を取り出してきて、それ以外の部分に関しては、省略するなどの抽象化が必要であり。検証目的の変化により、この抽象度を変更必要があり、それに応じてモデルが変更が必要である。(図 1.2)

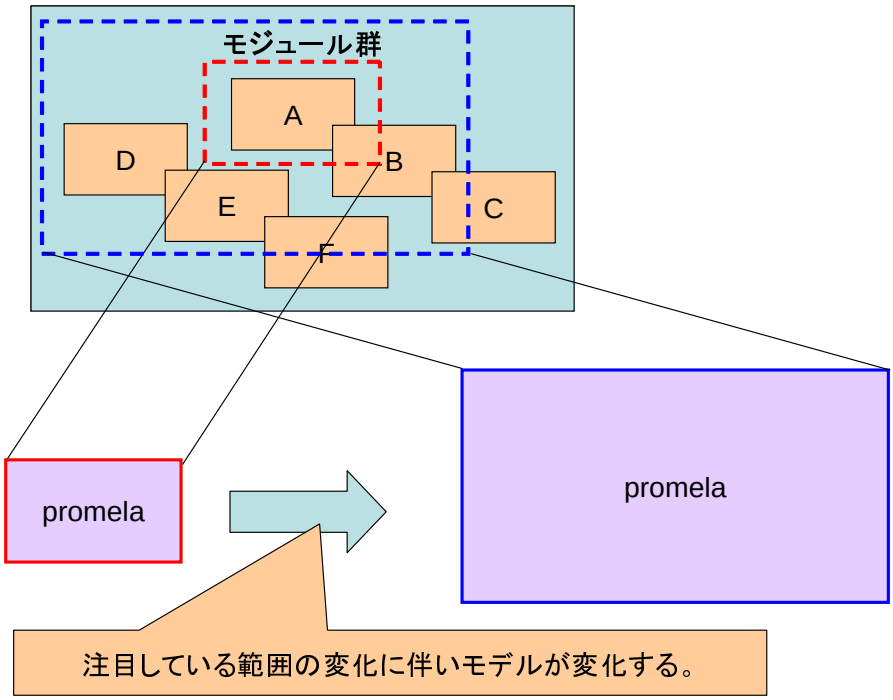

図 1.1: 注目している範囲の変化

この様に、検証対象は同一であっても検証モデルは検証目的に応じて変化する。

1.1.4 「確認したい性質の変化」に伴う「モデルの変化」とその問題点

検証モデルはどのように変更されるかを 1 の「確認したい性質の変化」を例に考察する。モデルに対して確認したい性質を与える方法としては、先に説明した LTL を用いる方法と、その他にアサーション(表明)を用いる方法がある。アサーションとは、モデル中にその時点で満たして欲しい性質をアサーションとして論理式とともに記述し、検証時にその論理式に反する状態があった場合、SPIN がその結果を共にアサーションの場所を出力するというものである。

LTL を用いた検証と、アサーションを用いた検証を比較すると、LTL を用いた検証では与えた LTL を SPIN が never claim という満たしてはいけない性質を記述したオートマトンを自動的に生成し、検証を行えるため検証モデル自体に変更を加える必要がなく、確認したい性質が変わったとしても、LTL を変更すればその関心事の検証を行うことができる。

しかし、アサーションを用いた検証では

- モデル中にアサーションを記述しなければならない。

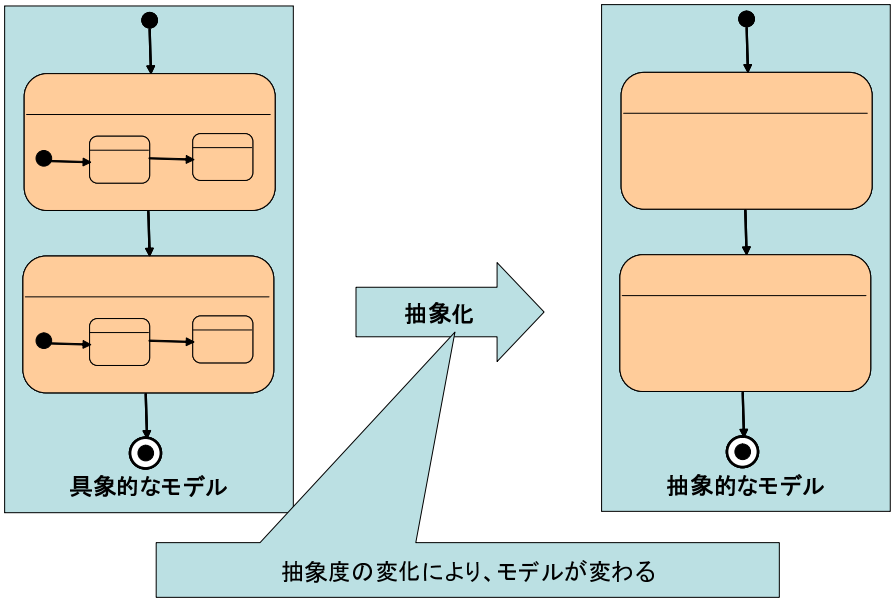

図 1.2: 検証モデルの抽象度の変化

- アサーションがモデル中に横断的に現れることがある。

などの理由から、ある性質を検証するために記述されたモデルを他の性質の検証に用いる場合、煩雑なモデルを書き換えを行わなければならない。

1.1.5 モデルに横断的な変更

「確認したい性質の変化」に伴う「モデルの変化」においては、その性質がモデル中に横断的に存在する仕組みや機能に興味がある場合に、変更が煩雑であることを述べた。この「横断的関心事」に関する問題は、「性質の変化」以外の「注目している範囲の変化」「抽象度の変化」によるモデルの変化においても問題となることがある。

よって、本研究では promela に対してアスペクト指向拡張を行い、この問題を解決するための、文法、言語処理系を提案し、「検証目的」の変化に伴う「モデルの変化」に対応する。

1.2 論文の構成

本稿では、２章にて本研究で解決したい問題点と共にどのようなアプローチでこの問題点を解決するかを述べ、３章にて本研究に関連のある技術について説明する。

４章では本研究で提案するアスペクト指向言語の文法の要件を考察し、実際にその要件をどのように実現するかについて述べる。

５章では４章で説明した設計思想に基づいて、定義されたアスペクト指向言語の文法を説明する。

６章では、文法の言語処理系の実装について必要な説明を行う。

７章では、promela モデルに対して本言語処理系を用いて、アスペクトを作用させる例を示す。また、本研究の提案する文法、言語処理系の有用性についてこの適用例を用いて考察を行う。

そして８章にて、本研究のまとめ、今後の課題について述べる。

第2章　目的

本研究の目的は以下の通りである。

「同一の検証対象の検証モデルについて、検証目的の変化によるモデルの横断的な変更」をアスペクト指向技術を用いて実現する。

本章では、「検証目的の変化による、モデルの横断的変更」の典型例として「同一の検証対象に対する、確認したい性質の変化に伴うモデルの変化」の例を用いる。そして、何が問題点でありそれをどのように解決するかを研究のアプローチとして述べる。また、本研究における付帯的効果としての「異なった検証対象であるが、類似しているモデル」の生成を実現についても説明する。

2.1　解決したい問題点

同一の検証対象に対する、確認したい性質の変化に伴うモデルの変化」の例として、チャネル通信を多用するモデルを考える (List2.1)。

このモデル上でチャネルの受信後の状態に関心があるような性質を新たに検証する時に起こるモデルの変化として「アサーションの埋め込み」が考えられる。

新たな性質に関する検証をアサーションを用いて行うにはチャネルの受信文（９、１４、２０行目）のあとにその性質を確認するためのアサーションを埋め込む必要がある。(List2.2)

List 2.1: 元々のモデル

```
1  mtype = { msg1 , msg2 …}
2  chan ch = [0] of {mtype}
3  active proctype P()
4  {
5    mtype rcvMsg;
6
7  INIT:printf("init\n");
8    …
9    ch?rcvMsg;
10   …
11
12 WAIT:printf("wait\n");
13   …
14   ch?rcvMsg;
15   …
16
17 TERMINATE:
18   printf("terminate\n");
19   …
20   ch?rcvMsg;
21   …
22   …
```

List 2.2: 検証を行うためにアサーションを埋め込んだモデル

```
mtype = { msg1 , msg2 …}
chan ch = [0] of {mtype}
active proctype P()
{
  mtype rcvMsg;

INIT:printf("init\n");
  …
  ch?rcvMsg;
  assert(rcvMsg != fin);
  …

WAIT:printf("wait\n");
  …
  ch?rcvMsg;
  assert(rcvMsg != fin);
  …

TERMINATE:
  printf("terminate\n");
  …
  ch?rcvMsg;
  assert(rcvMsg != fin);
  …
  …
```

このようなモデルの変化を人手によるモデルの書き換えによって対応するのは煩雑であり、書き換えミスも起こる可能性がある。

このように、モデル中に横断的に現れる仕組みや機能に注目した検証を行う際には、検証支援系による支援が必要である。

2.2 研究目的とアプローチ

先の例の問題点は、一つの原因に起因する。

それは、spin の仕様記述言語である promela が、モデル中に横断する仕組みや機能（横断的関心事）をうまく扱うための仕組みを提供していない点である。

一般に多くの言語は横断的関心事を扱うことが出来ないため、C++や Java 等の言語で、これを扱うために、aspectC++や aspect/J が開発された。promela でも、これと同様に横断的関心事を扱うためには、promela におけるアスペクト指向言語と言語処理系が必要である。よって、本研究では、先の例のようにモデル中に横断的にあわられる仕組みや機能をアスペクトとみなして、spin の仕様記述言語である promela にアスペクト指向を適用した、アスペクト指向言語、言語処理系を開発し、それらを用いた検証手法を提案する。

2.3 本研究の付帯的効果

これまで、「同一の検証対象のモデルの変化」を扱ってきたが、本研究の付帯的効果として「異なった検証対象のモデルの生成」が考えられる。

図 2.1 の検証対象Ａ、検証対象Ｂは類似しているが、「状態５がない」という点で異なる。このようなモデルを promela でモデル化する際、従来は両方のモデルを記述してきたが、これは非効率である。そこで、この検証対象Ａだけを promela でモデル化し、状態１、状態３、状態４からの状態５への状態遷移を削除

することにより、検証対象Ｂのモデルを自動的に生成する方法を考える。(図 2.2）今回の例では、状態５への遷移はモデル中に３箇所だけであった為、人手による遷移の書き換えにより、これを実現できるかもしれない。しかし

- 状態５への遷移が非常に多い場合
- 状態５以外にも、状態３、状態２などが存在しない類似したモデルも生成したい場合

などに、先の例と同様に人手によるモデルの書き換えは煩雑である。

よって、検証対象は異なるが、モデルが類似している場合に、そのモデル中に横断的に存在する仕組みや機能を変更することにより、新たな検証モデルをが生成可能な場合には、本検証支援系による支援が有用である。

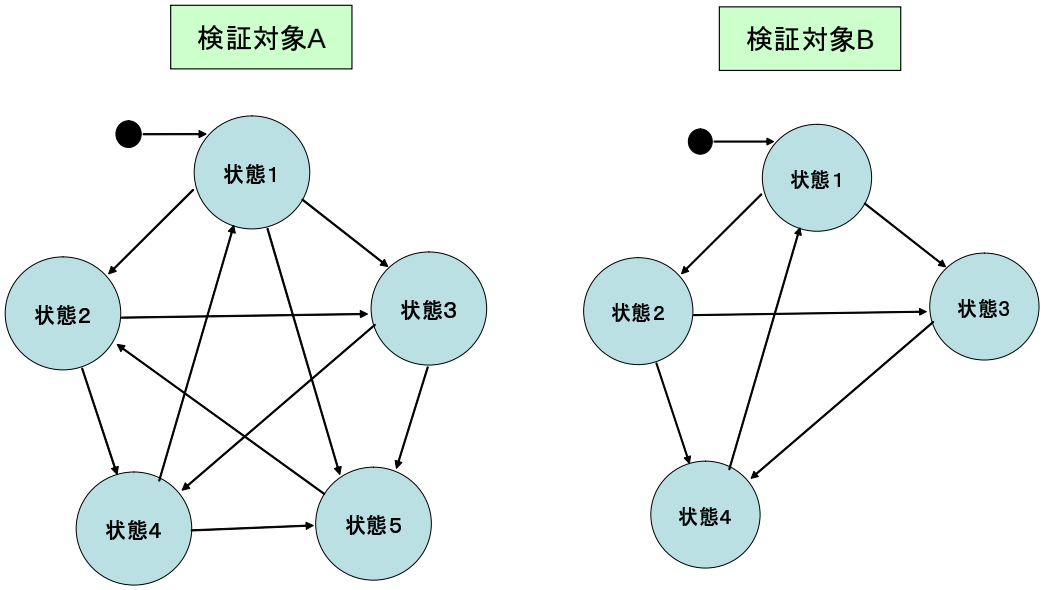

図 2.1: 類似しているが異なる検証対象の例

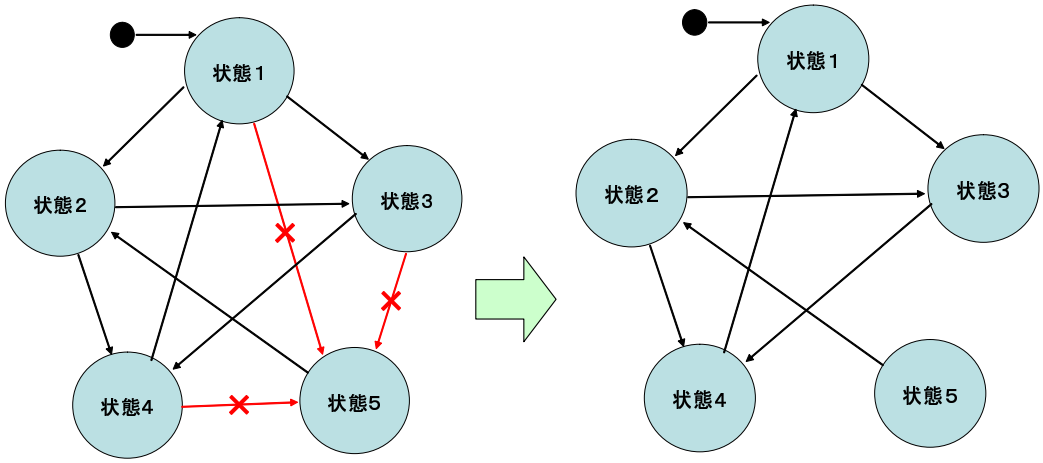

図 2.2: モデルの書き換え

第3章　関連技術

3.1　SPIN/promela

本研究は、promela の言語要素、文法に着目したものであるため、本稿中の説明においても言語要素名を用いる場合が多い。よって、本節にて必要な説明をおこなう。なお、本節に記載がないものに関しては、参考文献 [1] を参照。

SPIN(Simple Promela INterpreter) は、AT&T ベル研究所、コンピュータサイエンス研究センターにて Gerard.J.Holzmann が中心になって開発した、分散ソフトウェアシステムの形式的検証に用いられる、オープンソースのモデル検査器である。[11]

SPIN で検証を行う際は promela(PROcess MEta LAnguage) という仕様記述言語で仕様を記述する。promela は基本的には K&R の C 言語と同じような記法であるが、一部異なる点があり、それらはガードやメッセージパッシングの記法である。これらの記法は、E.W.Dijkstra の Guarded Command[4] や C.S.R.Hoare の CSP(Communicating sequential processes)[5] に影響をうけた記法となっている。[1]

promela のモデルは次の３種類の基本型のオブジェクトから構築される。

- プロセス
- データオブジェクト
- メッセージチャネル

本節ではまずは、この基本型のオブジェクトを説明し、次に制御フローとラベルについて説明をする。

3.1.1　プロセス

promela は最低１つ以上の proctype（プロセス）が宣言され、インスタンスとして存在していなくてはならない。プロセスを宣言するための proctype は List 3.1 のように記述される。

List 3.1: proctype の例

```
1 proctype SimpleProcess(){
2         printf("hello World\n")
3 }
```

3.1 は SimpleProcess という名前のプロセスを宣言したが、これだけではコンパイルエラーになる。なぜならば、インスタンス化されている proctype が一つもない為であり、この proctype をインスタンス化する必要がある。proctype をインスタンス化するのには２つの方法がある。

- proctype の宣言に active をつける
- run proctype 名 (引数リスト) という形で、run 式を用いる。

表 3.1: 基本データ型

タイプ名	範囲
bit	0,1
bool	*false,true*
byte	0..255
chan	0..255
mtype	0..255
pid	0..255
short	$-2^{15}..2^{15}-1$
int	$-2^{31}..2^{31}-1$
unsigned	$0..2^{15}-1$

List 3.2: active をつけた proctype の例

```
1 active proctype SimpleProcess(int i){
2         printf("hello World\n")
3 }
```

List 3.3: run 式で proctype をインスタンス化する例

```
1 proctype SimpleProcess(int i){
2         printf("hello World\n");
3 }e
4
5 init{ run SimpleProcess(4) }
```

List 3.2 は proctype に active が付けられているため、最初からインスタンス化される。List 3.3 の場合は、５行目の init 型プロセスで初めて run 式によりインスタンス化される。なお、init 型プロセスとは、大域変数の初期化、メッセージチャネルの初期化、プロセスの起動などが行われる、C 言語の main 関数のようなプロセスである。List 3.2 と List 3.3 で異なる点は、active でインスタンス化された proctype には、引数を渡すことができない点である。（もし、仮引数を持っていれば０に初期化される。）つまり、List 3.2 の仮引数は０で初期化され List 3.3 は 4 で初期化されることになる。

3.1.2 データオブジェクト

promela の基本データ型とその範囲は表 3.1 の通りである。変数のスコープは、proctype 内か大域の２種類であり、複数の proctype から参照されるような、プロセス間通信用のチャネルや変数は大域で宣言しておく必要がある。

配列は１次元配列のみであるが、構造体 (typedef) を使えば、多次元配列も利用可能である。mtype は、C 言語の列挙型 (enum) と同じであり、List 3.4 のサンプルを実行すると、mtype 中の一番最初の要素から、３，２，１と出力される。また、mtype を印字するために、mtype の値により if で出力を判断するようなロジックを書く必要はなく、printm を用いることにより mtype 型の要素の中の列挙した名前で出力を行うことができる。

List 3.4: mtype の例

```
mtype = {msg1, msg2, msg3}
active proctype mtype_test()
{
        printf("%d,%d,%d\n",msg1,msg2,msg3);
        printm(msg1);
        printf("\n");
}

/*----------------------出力--------------------------
      3,2,1
      msg1
1 process created
----------------------------------------------------*/
```

3.1.3 メッセージチャネル

メッセージチャネルは、メッセージの送受信を行うためのメッセージ通信路をモデル化するのに用いられる。

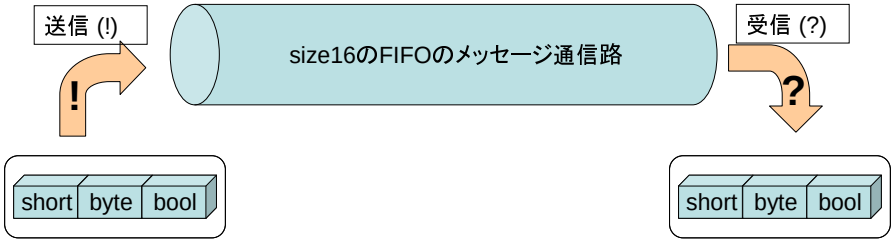

図 3.1: チャネルのイメージ

チャネルの宣言

チャネルの宣言は次のようにして記述する。

```
chan チャネル名 = [サイズ]  of  {データ型, データ型, …}
```

次の宣言は、バッファのサイズが１６のメッセージに short 型、byte 型、bool 型の変数を持つ FIFO のメッセージ通信路を宣言する。図 3.1 のように、真ん中のメッセージ通信路を宣言したことになる。

```
chan qname = [16] of {short, byte, bool}
```

送信

チャネルへメッセージを送信する場合には、次のように記述する。なお、指定されたチャネルが一杯のときは、デフォルトではブロックする。

```
チャネル名 ! 式,  式…
```

次の文は、qname というチャネルに、i,j,k をを含むメッセージを送信する。図 3.1 の、チャネルに対してメッセージを送信する左側の矢印の操作を行ったことになる。送信操作の記述は、下から 2 行はどちらも正しく、チャネルの第一パラメータが、メッセージタイプを表す場合にこのような記法を用いる。

```
short i = 0;
byte  j = 0;
bool  k = true;
qname!i, j, k;
qname!i (j, k)
```

受信

チャネルからメッセージを受信する場合には、次のように記述する。なお、指定されたチャネルが空のときはデフォルトではブロックする。

```
チャネル名 ? 変数, 変数  ‥
```

次の文は、qname というチャネルからメッセージを受信し、変数 i,j,k へ書き込む。図 3.1 の、チャネルからメッセージ受信する右側の矢印の操作を行ったことになる。送信操作のように、チャネルの第一パラメータがメッセージタイプを表す場合、一番下の記法を用いる。

```
qname?i, j, k;
qname?i (j, k)
```

また、受信操作において変数の部分に定数を書くと、その部分に書かれた定数とメッセージを取り出すようになる。

```
qname?100,j,k
```

先の 100 の部分に、ある変数を指定してその変数値とメッセージのその部分の値が等しかったときに受信したいときは次のように記述する。

```
int val = 100;
qname?eval(val), j, k
```

定義済み関数

promela には、定義済みのチャネルの中のメッセージ数を調べるための関数 len がある。次のように記述すると、qname の中にメッセージがいくつ入っているかがわかる。

```
len(qname)
```

また、表 3.2 のような良く使う関数が定義済み関数として用意されている。

表 3.2: 定義済み関数一覧

関数名	意味
empty	指定されたチャネルが空ならば true を返す。
nempty	指定されたチャネルが空でなければ true を返す。
full	指定されたチャネルが一杯ならば true を返す。
nfull	指定されたチャネルが一杯でなければ true を返す。

チャネルのテスト

次の式は qname?msg が実行可能かどうかを確認を行う。実行可能であれば、true を返すが、実際の受信は行われない。

```
qname?[msg]
```

promela の BNF を見ると分かるが、このチャネルのテストは recv_poll という名前の非終端記号である。他のチャネルの受信や送信は stmnt つまり文の一つであるが、このチャネルのテストは any_expr に属する。つまり、このチャネルのテストは文ではなく評価を返す式であり、副作用を与えることができないため、実際の受信は行われない。

ポーリング

通常の受信ではチャネルからメッセージを取り出すと、そのメッセージはチャネルから削除されるが、削除せずに取り出しだけを行いたいときに次のようにポーリングを用いる。

```
qname?<msg>
```

この文は、qname というチャネルからメッセージを取り出し msg という変数に格納するが、チャネルの中のメッセージは削除されない。

整列された送信とランダム受信[1]

整列された送信は、次のように普通の送信の！の代わりに！を二つ使って記述する。

[1] 参考文献 [1] には、*SORTED SEND AND RANDOM RECEIVE* と記述されている。*(p.48)* また、*Random receive*(ランダム受信) という言葉について *holzmann* は *p49* にて、*so the term "random receive" is a bit of a misnomer* と述べているためこの言葉は少々誤称なのかもしれない。

```
qname!!msg
```

この送信は、整数順にチャネル内をソートしてメッセージを挿入する。

例えばプロセスが１から１０までの数字をランダムに送信した場合、チャネル内が自動的のソートされて、整数順に格納する。

ランダム受信は、次のように？を 2 つ使って記述する。

```
qname??msg
```

普通の受信は、チャネル内の先頭のメッセージを取り出すが、この受信はチャネル内のメッセージのうち msg にマッチしたものを取り出す。

3.1.4 制御フロー

atomic

atomic は不可分な複文を記述するときに用いる。List 3.5 の例は、変数 val を宣言して、チャネルから val へ値を受信し、アサーションで val が 0 ではないことを表明する例である。atomic でこれらの処理が囲まれているため、一連の動作を割り込まれずに実行する。

List 3.5: atomic の例

```
1 active proctype atomic_example()
2 {
3         atomic
4         {
5                 int val;
6                 ch?val;
7                 assert(val != 0)
8         }
9 }
```

List 3.5 の例は、不可分な複文であるが、もしチャネル ch からメッセージを受信する際に ch の中のメッセージ数が０個であり、受信文でブロックした場合には、実行は他のプロセスに移る。これを許したくない場合は、d_step を用いる。

d_step

List 3.6 の例では、atomic の代わりに d_step が使われているため、受信でブロックした場合は、エラーになる。

List 3.6: d_step の例

```
1 active proctype d_step_example()
2 {
3         d_step
4         {
5                 int val;
6                 ch?val;
7                 assert(val != 0)
8         }
9 }
```

if

if 文は、複数のガード条件から実行可能なものを選択する文である。ガード条件のうち実行可能なものが一つだけある場合にはそれが選ばれるが、もし複数あれば実行可能なもののうち、ランダムに一つが選ばれる。

List 3.7: if の例

```
1 active proctype if_example()
2 {
3         if
4                 ::(a != b) -> option1
5                 ::(a == b) -> option2
6         fi
7 }
```

List 3.7 の例の場合、a と b が等しい場合には、2 行目のガード条件は 0 と評価され、3 行目のガード条件は 1 と評価されるため、option2 が実行される。

do

do 文は繰り返しの選択を記述するときに用いる文である。if 文と同じようにガード条件のうち実行可能なものを選び実行するが、これを反復して行う点が if 文と異なる。

List 3.8: do の例

```
1 active proctype do_example()
2 {
3         do
4                 ::counter++
5                 ::counter--
6                 ::(counter == 0) -> break
7         od
8 }
```

List 3.8 の例は、counter の値をインクリメント、またはデクリメントしているうちに counter の値が 0 になると do 文から脱出する例である。代入文はいつでも実行可能なため、break するまで 4,5,6 行目のガードからランダムに 1 つ選ばれ実行される。[2]

unless

unless は、以下のように用いる。

```
{ P } unless { E }
```

この文の実行は P から始まるが、その実行の前に E の実行可能性を確かめる。もし E が実行できない場合には P の実行が進む。

例えば、List 3.9 の例では、D を印字する 10 行目の print 文の実行可能性をまず確かめ、印字文はいつでも実行可能なため、D,E と印字され実行が終わる。

[2] Rules for executability（実行可能性のルール）から、print statement（印字文）と assignments（代入文）は常に実行可能である。詳しくは参考文献 [1] の Rules for executability(p51) を参照。

List 3.9: unless の例

```
1  active proctype unless_example()
2  {
3          {
4                  printf("A\n");
5                  printf("B\n");
6                  printf("C\n")
7
8          }unless{
9
10                 printf("D\n");
11                 printf("E\n")
12         }
13 }
```

List 3.9 の unless の直後に false を入れた場合には、常に unless 以降が実行出来ないため ABC と印字される。では、List 3.10 のようにこのプログラムを書き換えると結果はどうなるだろうか。この場合出力は、A D E となる。

理由は、A を実行後に flag に true が代入され、10 行目の flag が true(式文の評価が 1) となり実行可能になるため D と E の印字文が実行される。

List 3.10: unless の例 2

```
1  active proctype unless_example()
2  {
3          bool flag;
4          {
5                  printf("A\n");
6                  flag = true;
7                  printf("B\n");
8                  printf("C\n")
9
10         }unless{
11                 flag;
12                 printf("D\n");
13                 printf("E\n")
14         }
15 }
```

3.1.5 ラベル

ラベルは、文に名前をつける為の文であり、goto 文の参照先、end,progress ラベル[3]などに用いられる。本研究では、ラベルを範囲指定の方法として用いる。この概念はラベルの BNF 上でのシンタックスに注目して考案した概念である。よって本節では特にラベルのシンタックスについて説明を行う。ラベルは文、BNF では stmnt に":"を用いて次の形式で付加される。

```
ラベル名  :  stmnt
```

BNF では stmnt には、if 文、do 文、atomic 文、d_step 文、チャネル操作文等があるが、この stmnt に"{ sequence }"が含まれることに注目してほしい。つまり、List 3.11 のようにラベル中括弧に付加された場合は、ラベル付けされた中括弧が内部の複文を囲むということであり、これを用いると proctype 内の任意の範囲に対して名前を付けて指定できるということである。[4]

[3]参考文献 [1] の META LABELS(p76) を参照

[4]ただし、これはシンタックス上の話であり、中括弧にラベルを付けその proctype をプロセス名@ラベル名で参照するという実験を行った結果、中括弧のうち最初の一行を実行するときのみ true であり、それ以外の中括弧内は false であった。

List 3.11: ラベルの付いた中括弧で複文を囲う例

```
active proctype label_example()
{
        EXAMPLE:{
                                skip;
                                skip;
                                skip
                }
}
```

3.2 アスペクト指向技術

アスペクト指向技術とは、プログラム全体にわたって横断的に利用される仕組みや機能を、アスペクトという形にカプセル化すると共に、それらがプログラム中のどこで利用されるかをポイントカットとして、記述することにより、仕組みや機能をそれらが利用される場所（ジョインポイント）に対応付ける技術である。

プログラム全体にわたって横断的に利用される機能（横断的関心事）の例としては、以下がよく挙げられる。

1. セキュリティ
2. ログ出力
3. データの永続性
4. ビジネスロジック

3.2.1 アスペクトの例

例えば、このうちログ出力について、アスペクトがどのように有効に働くかを考えてみる。図 3.2 の左側の黒い帯は、ログ出力を行いたい部分を表す。この図のように、ログ出力がシステム全体で様々な場所から用いられる場合、ログ出力を行うメソッドやクラスの仕様が変更が、そのログ出力を使っていた部分（黒い帯の部分）すべてに影響し、多くの修正が必要である。

一方、アスペクト指向を用いて記述された、図の右側のようなシステムについて考える。このシステムでは、ログ出力をアスペクトとして実現している。このアスペクトの中には、このログ出力を行うためのコードと共に、「どこでログ出力が呼ばれる点を絞り込む条件（ポイントカット）」が記述されている。

このような状況であれば、ログ出力の変更は、アスペクト内にカプセルかされているため、システム全体に影響を与えずに、変更を行うことができる。

3.2.2 ジョインポイントモデル

「アスペクト」にはそのアスペクトが「どこで呼ばれるか（ポイントカット）」が記述してあるが、そのポイントカットによりアスペクトを適切に実行するための仕組みをジョインポイントモデルという。

ジョインポイントモデルの構成要素は、以下の通りである。

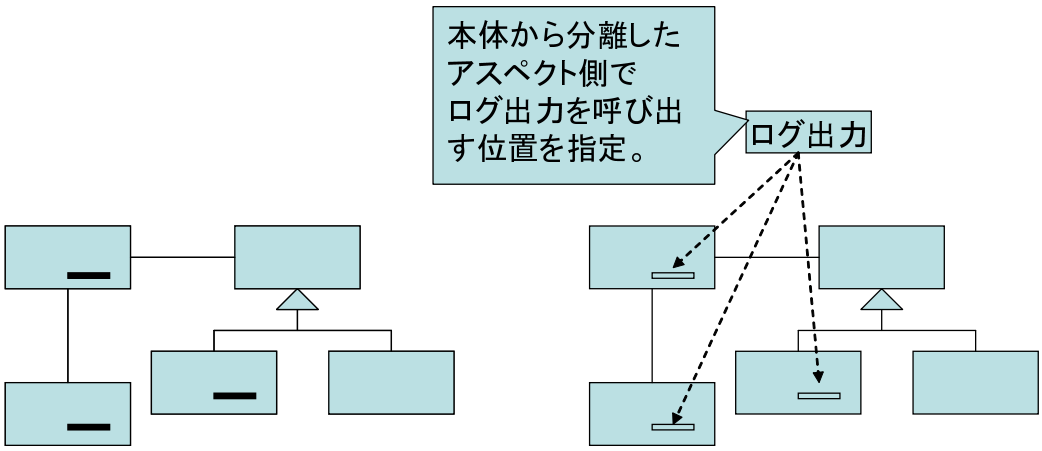

図 3.2: ログ出力の例

- ジョインポイント
  ジョインポイントとは、プログラムの実行時にアドバイスの実行を割り込ませることが、可能な点のことである。

- ポイントカット
  条件によって絞り込まれた、ジョインポイントの集合であり、この集合に対してアドバイスが作用する。

- アドバイス
  ポイントカットに含まれるジョインポイントに実行が差し掛かったときの実行されるコードのことである。先ほどのログ出力の例だと、ログ出力を行うコードがこのアドバイスにあたる。

AspectJ では、3.3 のように、ジョインポイントを絞り込む。

まず、java のプログラム中のあるスレッドの実行がジョインポイントに差し掛かる。この時、実行が差し掛かったジョインポイントにたいして、アスペクトでしていされたポイントカットによって、このジョインポイントが絞り込まれている場合、そのポイントカットに関連付けられたアドバイスが実行される。

AspectJ での、ジョインポイント、ポイントカットにはどのようなものがあるかは、４章にて説明する。

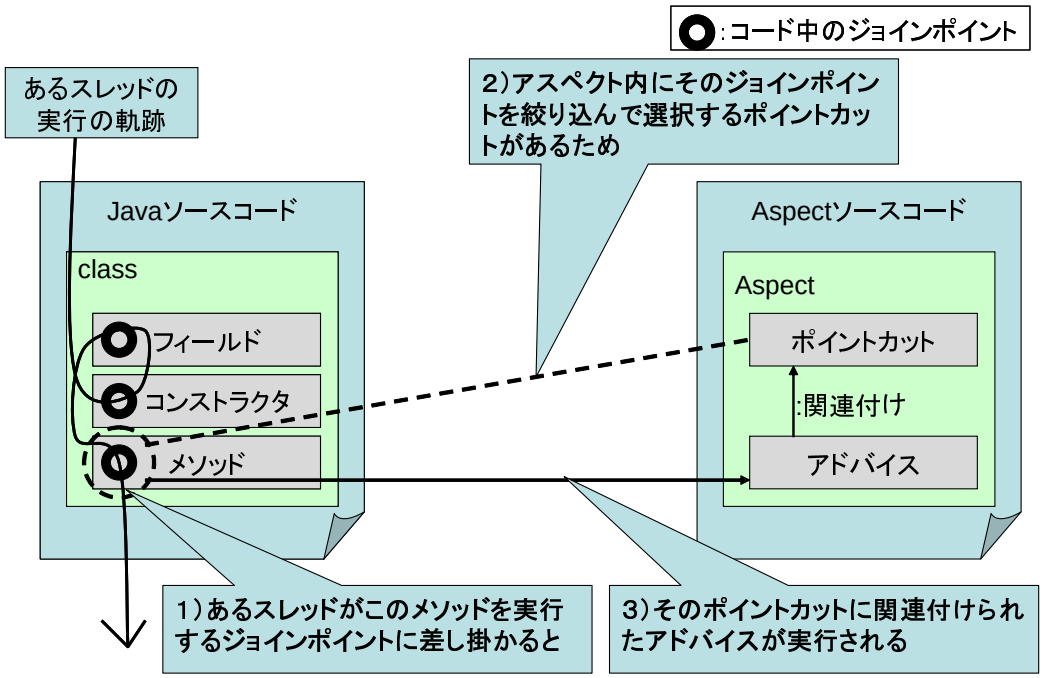

図 3.3: AspectJ でのジョインポイントモデル

第4章　言語の設計思想

本章では、promela と既存のアスペクト指向言語について考察を行い、それを踏まえたうえでアスペクト指向を promela に適用する際に、提案する言語に必要な要件を述べる。次に、各要件を実現する方法について述べ、最後に本言語の特徴的な点である、「範囲を持つ言語要素」の扱いについて述べる。

4.1　アスペクト指向言語でのジョインポイントの指定

アスペクト指向言語には、AspectJ[16] や JBossAOP[17] などがある。これらの対象言語は Java であるため、ポイントカットは Java の言語要素であるクラス、インターフェイス、メソッドなどを用いて記述することができる。例えば AspectJ での、ジョインポイント、ポイントカット、アスペクトでは次のような Java の概念を扱える。

4.1.1　ジョインポイント

メソッド、コンストラクタの呼び出し位置や実行位置、フィールドの参照位置や代入位置、クラス、インスタンスの初期化位置などが、ジョインポイントである。

4.1.2　ポイントカット

メソッド、コンストラクタに関するジョインポイントを指定するためのポイントカットにはメソッドパターン（クラス名やメソッドのシグニチャ）、コンストラクタパターン（修飾子、クラス名、引数の型）フィールドパターン（クラス名、修飾子、フィールドの型）などを指定するものがある。

一般にアスペクトを記述する際には、どのようなジョインポイントにアスペクトを作用させたいかを考え、適切なポイントカットを記述する。例えば、画像を描画するようなアプリケーションが、次の３つのようなメソッドを持っていたとする。

- init() 描画範囲の初期化を行うメソッド
- draw() 描画範囲に図形を描画するメソッド
- clear() 描画範囲の図形を消去するメソッド

Java の場合、図 4.1 のように各処理を、メソッドとして分割して、実装を行っておくことにより、「メソッドが実装しているコード上の範囲」と「プログラムの意味上の範囲」を一致させることができる。この例の場合、「描画範囲の初期化を行う」という意味に対応するコード上の範囲は「init メソッド内」である。

つまり、「初期化を行う場所」にアスペクトを作用させたいと考えたときは、それに対応する「init メソッド」をポイントカットとして指定することにより、適切なジョインポイント（＝コード上で初期化を行っている場所）を絞り込むことができる。

4.2 promela に対する考察

4.2.1 promela でのチャネルの重要性

promela は本来、プロトコル検証モデルを記述するための仕様記述言語であり、プロトコル検証におけるメッセージ伝送路のモデル化には、チャネルが用いられる [2]。

一方、ソフトウェアのモデル化においても、同期や排他制御といった一般に並行動作するプロセスが持つ相互の関係をモデル化するのに、チャネルが用いられる。

プロトコル検証におけるメッセージ伝送路、ソフトウェアのモデル化における、同期、排他制御の機構は、いずれも検証モデルの重要な要素である。よって、それらの要素に関する性質の確認においても、チャネルに関わる性質が重要となるため、本研究ではチャネルに関わるアスペクトの指定を細かく行える必要があると考えた。

4.2.2 promela でのジョインポイントの指定

promela は、関数を宣言することが出来ない。マクロや inline などは、関数のような機能を提供するが、コンパイル前にテキストの置換でこの機能を実現しているので、プリプロセス後は、関数としての意味を持たない。このため、Java のように「意味上の範囲」を言語要素を用いて指定することはできない。つまり図 4.1 のように、promela で実装されたモデルには意味上の範囲は存在しているのだが、それを指し示すための言語要素（関数、メソッド）などがないため意味上の範囲を指し示すことができない。AspectJ の場合は、アスペクトを実装するときに、まず「意味上の範囲」つまり先ほどの例だと、画像の描画や消去などを指定したいと考えそれに対応するコード上の範囲を指定することにより、意図したジョインポイントを指定することができる。しかし、このような言語要素がない promela では、意図した意味上の範囲にアスペクトを作用させることが難しいのである。

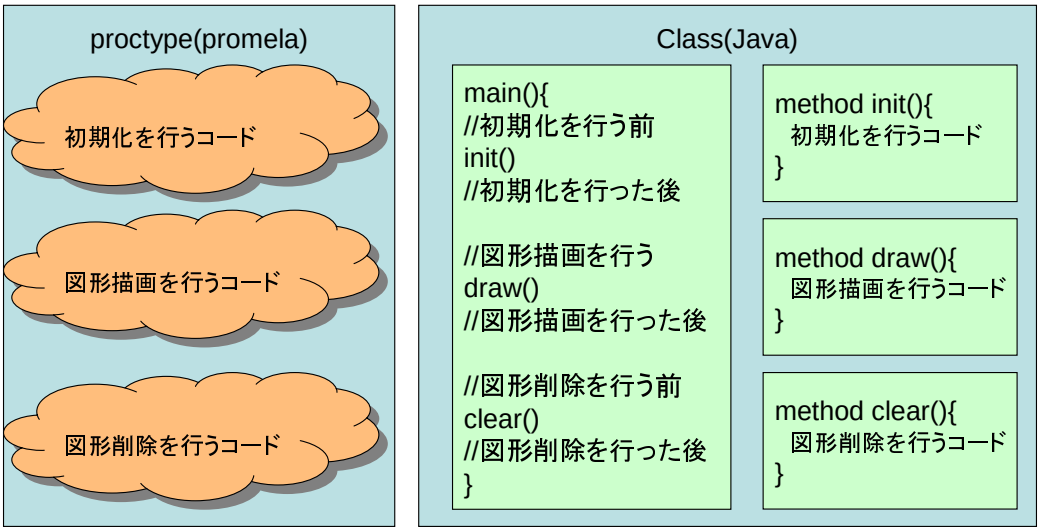

図 4.1: promela の局所化

4.3 文法の要件

先の考察を踏まえて、本研究で提案する文法の要件は次の通りである。

1. チャネル通信に関する言語要素を指定する文法は特に高い表現力を持つ必要がある。
2. proctype 中の任意の範囲を指定でき、その範囲に対してアスペクトを作用させることが出来るような仕組みを持つ。

4.4 本文法でのジョインポイント

本文法の要件をどのように実現するかを述べる前に、本文法でのジョインポイントについて説明する。ジョインポイントは promela の文法上正しくコードを割り込ませることが出来る点である必要がある。promela の実行は、任意の文[1]を「実行可能」か「ブロックされる」のどちらかに評価し、実行を進める。つまり、実行の単位は文 (BNF の stmnt) であり、ジョインポイントを「文 (stmnt) の実行」とすることにより promela の実行単位をジョインポイントとすることができる。

4.5 要件の実現

4.5.1 要件１の実現

要件１を実現するためには、チャネル操作を行う文を指定する際に様々な条件（チャネル名、操作、メッセージ）を指定して、適切なジョインポイントを絞り込めるような文法を設計する必要がある。promela でのチャネル操作[2] は、send, receive, sorted-send, random-receive, polling があるが、これらを指定でき、かつチャネル名、メッセージも柔軟に指定できれば、promela の文法上の全てチャネル操作を指定できることとなる。よって、本文法では、チャネル名、メッセージ名の正規表現での指定、チャネル操作の指定が行える文法を提案する。これに加えて、アスペクト内でチャネル操作のメッセージを参照する機能を持つ文法を、本研究では提案する。

4.5.2 要件２の実現

要件２を実現するためには、promela の言語要素を用いて任意の範囲を指定出来る必要がある。これが実現されれば、本文法のユーザーは次のような手順を踏んでアスペクトを記述すればよい。なお括弧の中は先ほどの例との対応である。

1. アスペクトを作用させたい意味上の範囲を考える。
   (描画範囲の初期化を行った後に処理をしたいと考える。)

[1] 任意の式 (expression) ではない。たとえば true && false && true を評価するとき、true を評価して true まで実行が進むということはない、(true && false && true) という各式全体を式文 (expression statement) として評価を行い、全体の結果が false となるのでこの式文でブロックする。

[2] チャネルのテスト（メッセージ部分に大括弧がつくもの）は、操作ではなく式である。チャネルに対して副作用を与えるものではなく BNF 上も式文 (expression statement) に分類されているため、本文法でも式文を指定するための expr というポイントカットに含めることとした。

2. その意味上の範囲に対応する、モデル記述上の範囲を promela の言語要素を用いてポイントカットとして指定する。
(描画範囲の初期化という意味を持つ、コード上の範囲は init メソッドの実行なので、java の言語要素であるメソッドを用いて、「init メソッドの実行」をポイントカットとして指定する。)

3. 2 で指定した範囲に対して、どのような処理を割り込ませるかを考える。
(描画範囲の初期化の後に行う処理を、アドバイスをして記述する。)

モデル記述の意味上の範囲に対応するコード上の範囲を promela の言語要素を用いて、指定するための方法について次節で説明を行う。

4.6 本文法での範囲指定の考え方

本節では、本文法でアスペクトが作用する範囲をどのように指定するかについて説明する。これは、promela の言語要素を「範囲を持つもの」と「そうではないもの」に分類し「範囲を持つもの」を指定した場合に、その指定に対して 2 つの意味を持たせることによりさまざまなコード上の範囲を指定するものである。そして、5 章にて、この考え方を実装した文法について説明を行う。

4.6.1 範囲を持つ言語要素

java では、メソッドという言語要素が、ある範囲のコードを囲っていることによりメソッドを指定するとコード上の範囲を指定するとこができる。

promela でも、ある範囲を囲う言語要素があるかを考えると、そのようなものは存在する。つまり、「範囲を持つ」という視点から言語要素を眺めると、範囲を持っているものとそうではないものに分類できる。その分類は表 4.1 の通りである。

表 4.1: 範囲を持つ言語要素とそうでないものの分類

分類	言語要素
範囲を持つ言語要素	if , do , atomic , d_step , unless ,option , init , never , trace , notrace , proctype , label
そうではないもの	send , receive , xr , xs , goto , print , assert , expr , one_decl , assign

例えば、if と goto を比較すると if は内部にガードを複数含み promela のモデル中の if を指し示すと、その if で囲まれた範囲を指し示すことができる。goto の場合は内部に言語要素は含まないので、goto 文を指し示すとの一文のみを指し示したことになる。

4.6.2 範囲に名前を付ける方法

範囲を持つ言語要素を指定することにより、その言語要素が囲う範囲を指定することが出来る。しかし、promela のモデル中には if 文は複数出現するため、if と書かれただけではどの if 文を指定したいのかが分からない。そこで p.16 で説明したラベルを用いる。

ラベルは次のように記述により、BNF での stmnt に名前を付けることができる。

```
ラベル名  :  stmnt
```

promela の BNF での stmnt のうち、「範囲を持つ言語要素」は「if,do,atomic,d_step, 中括弧」であり、これらにはラベルにより名前を付けることが出来る。よって、特定の範囲をもつ言語要素で囲まれた範囲は言語要素名＋ラベル名という形で指定することができる。たとえば、言語要素 if ＋ラベル名 FOO で指定される範囲は次の通りである。

```
if
    ::skip
fi;

/*FOO というラベルでラベル付けされた if の範囲は、ここから*/
FOO:if
    ::skip
    ::skip
fi;
/*ここまで*/
```

4.6.3 任意の範囲を指定する方法

先ほどは、範囲を持つ言語要素にラベルを付加することにより、範囲を指定していたが、次のようなモデルは範囲を持つ言語要素がないため、範囲を指定することが出来ない。

```
ch!init;
printf("init\n");
ch?ack;
ch!data(1,2);
printf("send data\n");
ch?ack;
```

このようなモデル中で、ある範囲に名前を付けたい場合には BNF の stmnt の中の中括弧を用いる。中括弧で BNF の sequence を囲うとその中括弧自体が BNF 上では stmnt となるためラベルを付けることが出来る。つまりこのようなモデルを次のようにすると、ある範囲に対して名前を付けることが出来る。

```
INIT:{
    ch!init;
    printf("init\n");
    ch?ack;
    }
SEND:{
    ch!data(1,2);
    printf("send data\n");
    ch?ack;
    }
```

4.6.4 範囲の指定の意味

ここで、「範囲の指定」に対して２つの意味を与える。その２つの意味とは以下である。

- 意味１：範囲を持つ言語要素を指定すると、その範囲自体をブロックとして指定する。
- 意味２：範囲を持つ言語要素を指定すると、その範囲中の全てのジョインポイントを指定する。

この２つの違いを説明するために、atomic という言語要素に対して、/*aspect*/というコメントを事後に割り込ませる例を図 4.2 示す。

意味１で範囲を考えると範囲自体をブロックとして扱うので、そのブロックの事後、つまり atomic の中括弧の後にコメントが追加される。(図 4.2 の右上)

意味２で範囲を考えると範囲中の全てのジョインポイントが指定されるので、各 skip の事後にコメントが追加される。(図 4.2 の右下)

4.6.5 範囲を意味１で扱う利点

範囲を持つ言語要素を意味１で扱うと、範囲自体に対して操作を行うことができる。つまり、ポイントカットで指定された if,do,option 等をブロックとして書き換え、事前・事後へコードの追加ができる。

4.6.6 範囲を意味２で扱う利点

「範囲を持つ言語要素を指定すると、その範囲中の全てのジョインポイントを指定できる。」と考えると、「範囲＝ジョインポイントの集合」として考えることが出来る。範囲を持つ言語要素をこのように考えると、ポイントカットに複合条件を用いることができる。複合条件とは「プロセス名 P の中のチャネル送信」というように、アスペクトの作用する範囲を限定するための条件である。

これは「範囲を持つ言語要素」が囲う範囲をジョインポイントの集合とみなし、その集合に対し和集合、積集合をとりその集合で、ポイントカットが作用する範囲を限定するというものである。

例えば図 4.3 の左側の promela では「範囲を表す言語要素」は次である。

1. proctype
2. if

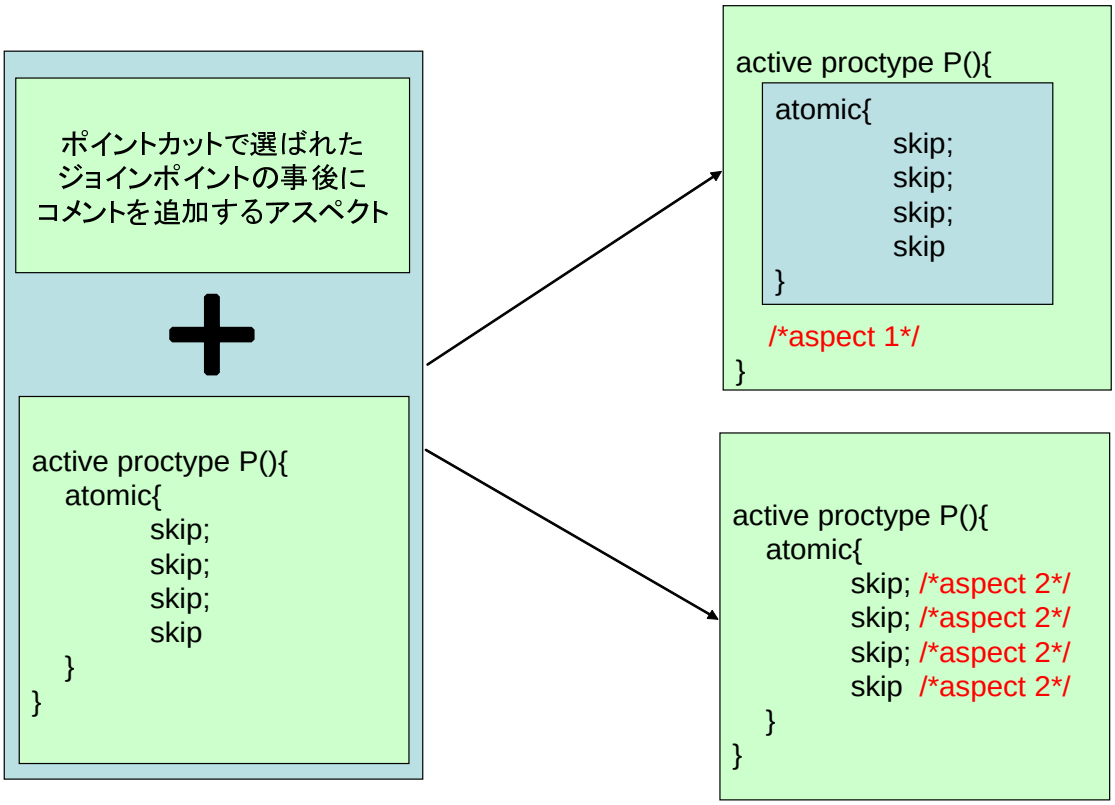

図 4.2: 範囲の意味の違いによるウィーブの違い

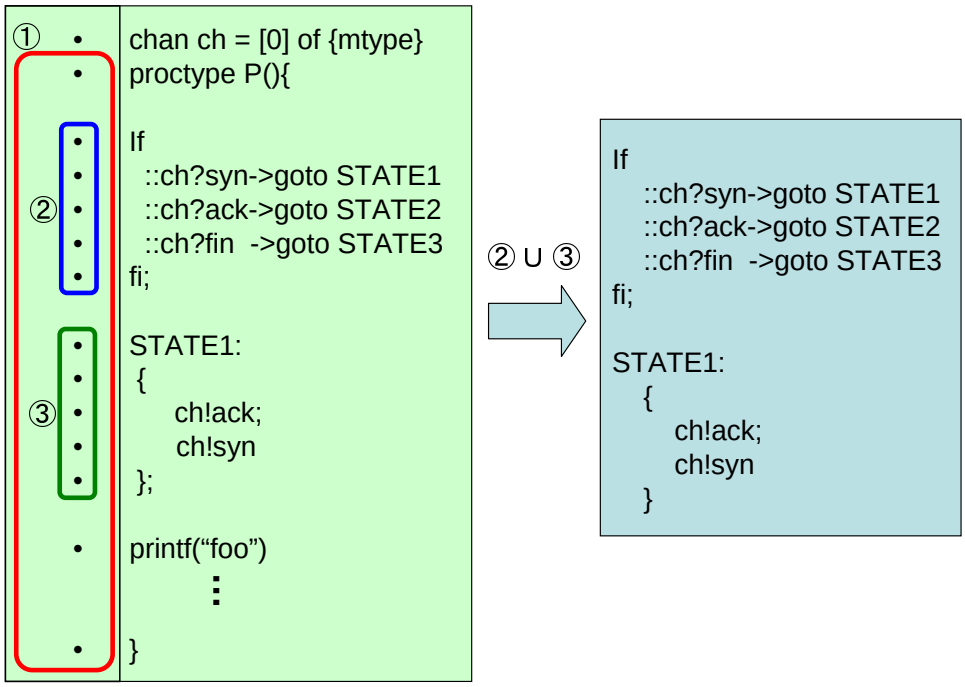

図 4.3: 提案する局所化手法

3. ラベル付けされた中括弧
4. if の中の option(付録 A.promela の BNF を参照)

このうち 2 と 3 の和集合をとると右側の promela のように範囲を絞りこむことが出来る。このようにして proctype 中の範囲を絞りこみ、適切なジョインポイントにアスペクトを作用させる。

4.6.7 本文法での範囲の扱い

これまで、「範囲を持つ言語要素」を指定した場合の「範囲」の扱いについて述べたが、本文法では「意味 1」、「意味 2」の両方を利用できるような文法を提案する。これは、「範囲を持つ言語要素」をポイントカットを用いて指定した場合、「意味 1」としてジョインポイントを絞り込むが、演算子 allStmnt を用いることにより、「意味 2」としてジョインポイントを絞り込むというものである。詳しくは、次章 5 にて説明する。

第5章　文法の提案

本研究で提案する文法の BNF を図 5.1 に示す。本章では、まず本文法でのアドバイスについて説明する。次に、範囲を持もたない goto,print 等の言語要素を指定するためのポイントカット、範囲を持つ if,do,atomic 等の言語要素を指定するためのポイントカットを説明し、ポイントカットの複合条件を指定するための演算子について説明する。そして、演算子の結合規則、アスペクトの評価順など全ポイントカット共通する規則について説明する。

List 5.1: 提案する文法の BNF

```
aspects          : /* empty */
                 | aspect

aspect           : advice ':' pointcut '{#' PROMELA '#}'

advice           : BEFORE
                 | AFTER
                 | AROUND

pointcut         : primitive
                 | '(' pointcut ')'
                 | pointcut      '&&'     pointcut
                 | pointcut      '||'     pointcut
                 | pointcut      CONTAINS pointcut
                 | ALLSTMNT      '(' pointcut ')'
                 | REMOVESCOPE   '(' pointcut ')'

primitive        : chan
                 | scope_op_no_arg        '()'
                 | scope_op_one_arg       '(' str_reg ')'
                 | unless
                 | point_op_no_arg        '()'
                 | point_op_one_arg       '(' str_reg ')'


chan             : CHAN '(' str_reg  ',' channel_op ',' str_reg ')'

str_regexp       : ARG
                 | REXEXP '(' ARG ')'

channel_op       : '!'  | '!!'
                 | '?'  | '??' | '?p' | '??p'
                 | '!*' | '?*'


scope_op_no_arg  : IF
                 | DO
                 | ATOMIC
                 | D_STEP
                 | OPTION
                 | INIT
                 | NEVER
                 | TRACE
                 | NOTRACE
```

```
scope_op_one_arg: PROCTYPE
                | LABEL

unless          : UNLESS '()'
                | UNLESS '(' BEFORE ')'
                | UNLESS '(' AFTER  ')'


point_op_no_arg : ELSE
                | BREAK


point_op_one_arg: XS
                | XR
                | GOTO
                | PRINT
                | ASSERT
                | EXPR
                | DECL
                | ASSIGN
```

5.1 アドバイス

本文法におけるアドバイスはbefore,after,around の三種類である。各アドバイスは次のような意味を持つ。

1. before
   ジョインポイントのコードの直前に、アスペクトのコードを追加する。

2. after
   ジョインポイントのコードが直後に、アスペクトのコードを追加する。

3. around
   ジョインポイントのコードと、アスペクトのコードを入れ替える。

5.2 範囲を持たない言語要素のポイントカット

5.2.1 chan（チャネル操作）

説明

モデル中のチャネル操作を行っているジョインポイントを選び出すためのポイントカット

記法

```
chan(［チャネル名］,［操作］,［メッセージ］)
```

このポイントカットでは絞り込みたいジョインポイントの、チャネル名、操作、メッセージを指定する。このポイントカットで絞り込まれるジョインポイントは、各フィールドの条件を and で評価したものである。各フィールドの記述方法は次の通りである。

[**チャネル名**] チャネル名のフィールドには次のどちらかを記述する。

- ”文字列”
  指定された文字列を名前に含むチャネルに対して操作を行っているジョインポイントを選び出す。
- regexp(”正規表現”)
  指定された正規表現にマッチする名前のチャネルに対して操作を行っているジョインポイントを選び出す。

[**メッセージ**] メッセージのフィールドには次のどちらかを記述する。

- ”文字列”
  指定された文字列を含むメッセージ、変数を用いてチャネル操作を行っているジョインポイントを選び出す。
- regexp(”正規表現”)
  指定された正規表現にマッチするメッセージ、変数を用いてチャネル操作を行っているジョインポイントを選び出す。

[**操作**] 絞り込みたいジョインポイントのチャネル操作が行っている操作を指定する。指定できる操作とその意味は表 5.1 の通りである。

使用例

List5.2 に対して以下のアスペクトを作用させると、List5.3 のようになる。

一つ目のアスペクトは、「チャネル名に”ch”を含むチャネルに対して送信を行っている文」のジョインポイントを選び出すので、プロセス P 内のチャネル送信の後に、アドバイスが挿入される。

二つ目のアスペクトは、「チャネル名に”ch”を含むチャネルからポーリングで受信を行っている文」のジョインポイントを選び出すので、プロセス Q の中の 2 行目の受信文の後にアドバイスが挿入される。

表 5.1: 操作の一覧

[操作]	意味
!	送信を行っているジョインポイントを選び出す。
!!	整列された送信 (SortedSend) を行っているジョインポイントを選び出す。
!*	「!」と「!!」で選ばれるすべてのジョインポイントを選び出す。
?	受信を行っているジョインポイントを選び出す。
??	ランダム受信を行っているジョインポイントを選び出す。
?p	ポーリングを行っているジョインポイントを選び出す。
??p	ランダム受信かつポーリングを行っているジョインポイントを選び出す。
?*	「?」,「??」,「?p」,「??p」で選ばれる全てのジョインポイントを選び出す。

```
after : chan("ch","!" ,""    ){#   printf("aspect1\n") #}
after : chan("ch","?p",""    ){#   printf("aspect2\n") #}
```

List 5.2: chan ポイントカットの例（ウィーブ前)

```
1  mtype = { msg1 , msg2 }
2  chan ch = [3] of {mtype};
3  active proctype P()
4  {
5    ch!msg1;
6    ch!msg2
7  }
8
9  active proctype Q()
10 {
11   ch?msg1;
12   ch?<msg2>
13 }
```

List 5.3: chan ポイントカットの例（ウィーブ後）

```
1  mtype = { msg1 , msg2 }
2  chan ch = [3] of {mtype};
3  active proctype P()
4  {
5    ch!msg1;printf("aspect1\n");
6    ch!msg2;printf("aspect1\n")
7  }
8
9  active proctype Q()
10 {
11   ch?msg1;
12   ch?<msg2>;printf("aspect2\n")
13 }
```

注意点

チャネルに関する式として、本稿 3.1.3 で説明したチャネルのテストがあるがこれは、文ではなく式であるので、チャネルのテストを含むような式文のジョインポイントを選び出す場合は、expr ポイントカット

を用いる。

5.2.2 else , break

説明

モデル中の else,break のジョインポイントを指定するためのポイントカット

記法

```
else()
break()
```

else,break の後に括弧が付いているが、括弧の中には何も指定しない。

使用例

次のアスペクトを、List5.4 に作用させると、List5.5 のようなモデルに変換される。

```
after : else()  {#  printf("else_Aspect\n"); #}

after : break() {#  printf("break_aspect\n"); #}
```

List 5.4: else break ポイントカットの例（ウィーブ前）

```
1 active proctype P()
2 {
3         do
4                 ::skip -> break
5                 ::else -> assert(false)
6         od;
7 }
```

List 5.5: else break ポイントカットの例（ウィーブ後）

```
1 active proctype P()
2 {
3         do
4                 ::skip ; break ; printf("else_Aspect\n") ;
5                 ::else ; printf("break_Aspect\n") ; assert(false) ;
6         od;
7 }
```

注意点

特になし。

5.2.3 xs, xr, goto, print, assert, expr, decl, assign

説明

モデル中の xs, xr, goto, print, assert , expr, decl, assign のジョインポイントを指定するためのポイントカット

記法

記法は以下の通りである。なお、各ポイントカットが選び出すジョインポイントは表 5.2 の通りである。

```
xs     ([文字列 or 正規表現])
xr     ([文字列 or 正規表現])
goto   ([文字列 or 正規表現])
print  ([文字列 or 正規表現])
assert ([文字列 or 正規表現])
expr   ([文字列 or 正規表現])
decl   ([文字列 or 正規表現])
assign ([文字列 or 正規表現])
```

[文字列 or 正規表現] の中には、次のどちらかが入る。

- ”文字列”
- regexp(”正規表現”)

また、各言語要素で [文字列 or 正規表現] のフィールドで指定するものの意味が異なる。その意味は表 5.3 の通りである。

表 5.2: 各ポイントカットが選び出すジョインポイント

ポイントカット名	そのポイントカットが選び出すジョインポイント
xs	チャネルアサーション (eXclusive Send)
xr	チャネルアサーション (eXclusive Receive)
goto	goto 文
print	printf または printm 文
assert	assert 文
expr	式文 (expression statement)
decl	変数定義全体
assign	代入文

使用例

次のようなアスペクトを、List5.6 に作用させると、List5.7 のようになる。

表 5.3: 各言語要素でのフィールドの意味の違い

ポイントカット名	[文字列 or 正規表現] のフィールドで指定されるものの意味
xs	チャネルアサーションに指定されているチャネル名
xr	チャネルアサーションに指定されているチャネル名
goto	goto に指定されている、ジャンプ先のラベル名
print	printf または printm 文の括弧の中身
assert	アサーションの評価式
expr	式文 (expression statement の評価式全体
decl	変数定義全体（型名、変数名も含む)
assign	代入文全体

一つ目のアスペクトは、「文字列"timeout"を含む式文」のジョインポイントを選び出すので、５行目の timeout 式を含む式文の後にアドバイスが挿入される。

二つ目のアスペクトは、「評価式に文字列"false"を含む assert 文」のジョインポイントを選び出すので、６行目の assert 文の後に、アドバイスが挿入される。

```
after:expr("timeout") {# printf("aspect1\n"); #}
after:assert("false") {# printf("aspect2\n"); #}
```

List 5.6: assert expr ポイントカットの例（ウィーブ前）

```
1 active proctype P()
2 {
3   if
4     ::skip;
5     ::timeout->goto Terminate
6     ::else->assert(false);
7   fi;
8 Terminate:skip
9 }
```

List 5.7: assert expr ポイントカットの例（ウィーブ後）

```
1 active proctype P()
2 {
3   if
4     ::skip;
5     ::timeout;printf("aspect1\n");goto Terminate
6     ::else;assert(false);printf("assert2\n");
7   fi;
8 Terminate:skip
9 }
```

注意点

特になし。

5.3 範囲を持つ言語要素のポイントカット

5.3.1 if, do, atomic, d_step, option, init, never, trace, notrace

説明

モデル中の if, do, atomic, d_step, option, init, never, trace, notrace の範囲を意味１[1]（ブロック）として指定するためのポイントカット

記法

記法は以下の通りである。なお、各ポイントカットが選び出すジョインポイントは表 5.4 の通りである。

```
if      ()
do      ()
atomic  ()
d_step  ()
option  ()
init    ()
never   ()
trace   ()
notrace ()
```

表 5.4: 各ポイントカットが選び出すジョインポイント[1]

ポイントカット名	そのジョインポイントが選び出すジョインポイント
if	if 文が囲う範囲をブロック (範囲の意味１）として選び出す。
do	do 文が囲う範囲をブロック (範囲の意味１）として選び出す。
atomic	atomic 文が囲う範囲をブロック (範囲の意味１）として選び出す。
d_step	d_step 文が囲う範囲をブロック (範囲の意味１）として選び出す。
option	option が囲う範囲をブロック (範囲の意味１）として選び出す。
init	init が囲う範囲をブロック (範囲の意味１）として選び出す。
never	never が囲う範囲をブロック (範囲の意味１）として選び出す。
trace	trace が囲う範囲をブロック (範囲の意味１）として選び出す。
notrace	notrace が囲う範囲をブロック (範囲の意味１）として選び出す。

なお、表 5.4 中の「if 文が囲う範囲」などの表現は、表 5.5 の範囲の通りである。

たとえば、if ポイントカットであれば、このポイントカットがブロックとして選び出す範囲は、if から fi の範囲であり、BNF 上では IF options FI がこの範囲に対応する。

[1]「範囲の意味１、２」は本稿 4.6.4(p.25) を参照。

表 5.5: 各ポイントカットがブロックとして扱う範囲

ポイントカット名	範囲の始点	範囲の終点	BNF 上の対応
if	if	fi	IF options FI
do	do	od	DO options OD
atomic	atomic{	}	ATOMIC{ sequence }
d_step	d_step{	}	D_STEP{ sequence }
option	::	:: または、fi,od	options の中の::から次の::まで
init	init{	}	INIT[priority]{ sequence }
never	never{	}	never{ sequence }
trace	trace{	}	trace{ sequence }
notrace	notrace{	}	notrace{ sequence }

使用例

図 5.1 の一番左のモデルに対して、次のようなアスペクトを作用させる。

```
after : if() {# printf("aspect1\n"); #}
before: do() {# printf("aspect2\n"); #}
```

このアスペクトを作用させるとモデルは、図の一番右のようになる。各アスペクトがモデルにどのように作用するかを説明する。

一つ目のアスペクトは、if ポイントカットを用いているため if で囲まれた範囲をブロックとして扱う。つまりポイントカットに対して before アドバイスが書かれた場合にはモデルの 1 行目に、after アドバイスの場合”fi;”のあとにアドバイスのコードが追加される。この例では、after アドバイスなので図の位置 (aspect1 の印字文の位置）にコードが追加されている。

二つ目のアスペクトは、一つ目のアスペクトと同様に do ポイントカットにより、do で囲まれた範囲

注意点

このポイントカットは、範囲を意味 1 [1]（ブロック）として扱う。範囲を、意味 2 [1]として扱いたい場合には、allStmnt 操作を用いる。

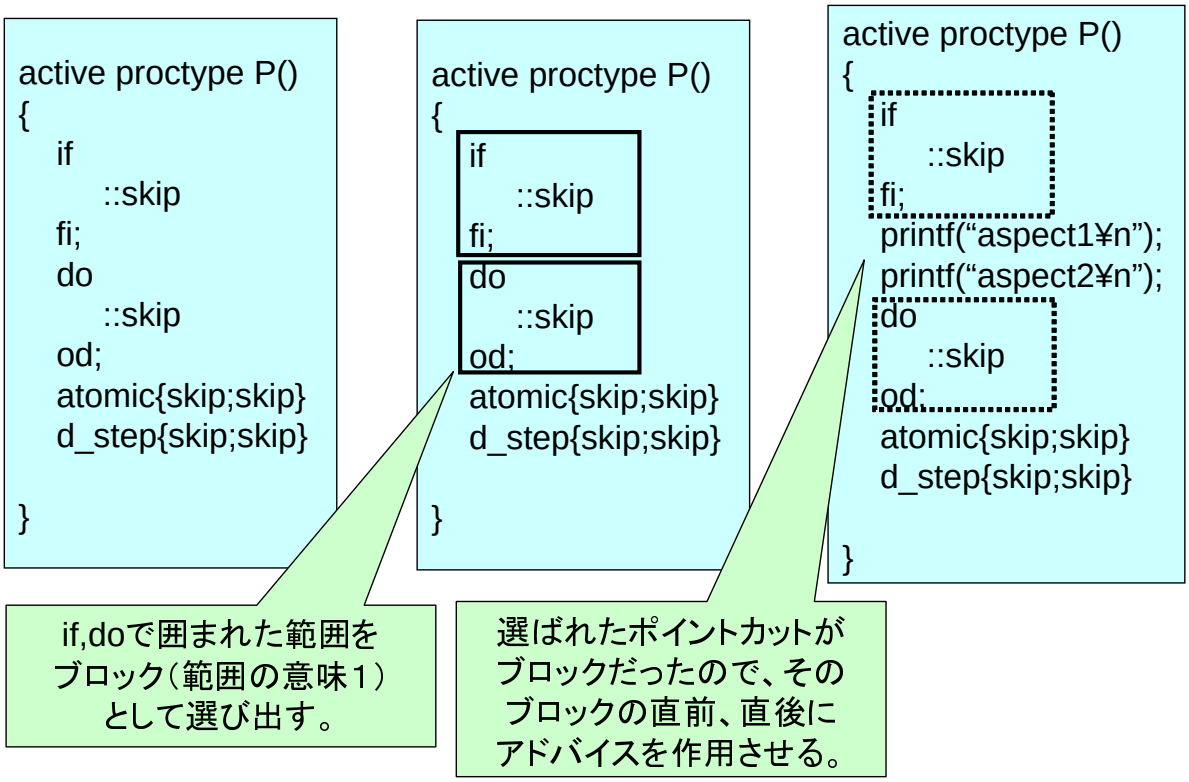

図 5.1: if,do ポイントカット用いたウィーブ

5.3.2 unless

説明

unless の前、後の stmnt またはその両方の stmnt のジョインポイントを指定するためのポイントカットである。

BNF では unless の定義は次のとおりである。

```
stmnt unless stmnt
```

unless は前後に中括弧を付けてもちられることが多いがそれは stmnt の一つに"{ sequence }"があるからである。このポイントカットは、unless の前後の stmnt によって次のような意味を持つ。

- stmnt が「範囲を持つ言語要素」
  その範囲を意味１（ブロック）として扱いそのジョインポイントを選び出す。
- stmnt が「範囲を持たない言語要素」
  その stmnt のジョインポイントを選び出す。

記法

```
unless( before | after | <空白> )
```

括弧の中のパラメータには、before か after か空白（"unless()"と書いた場合）が入る。それぞれの意味は次の通りである。

- **before**
  unless の前の stmnt のジョインポイントを選び出す。
- **after**
  unless の後の stmnt のジョインポイントを選び出す。
- <空白>
  unless の前後両方の stmnt のジョインポイントを選び出す。

使用例

unless ポイントカットの使用例を図 5.2, 図 5.3 に示す。

まず、図 5.2 について説明する。この例は、unless の前の stmnt をアドバイス around により書き換える例である。

プロセス P の最初の unless は stmnt の部分が文であるので、その文のジョインポイントに対してアドバイスを作用させる。残りの unless は stmnt の部分が「範囲を持つ言語要素」のため、その範囲をブロックとして扱い around アドバイスを作用させる。よって、ウィーブを行うと図の右のように、各 unless の左側が全て印字文に置きかれられる。

まず、図 5.3 について説明する。この例は、unless の後の stmnt をアドバイス around により書き換える例である。

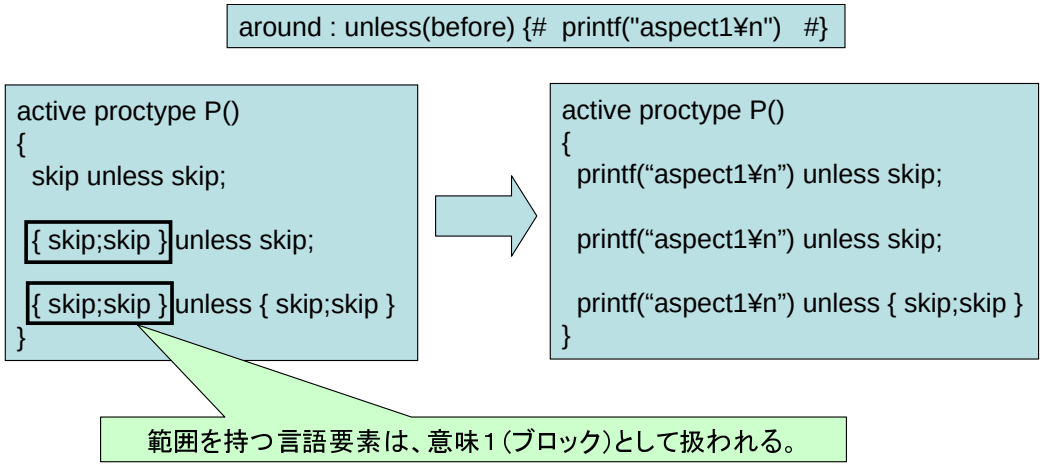

図 5.2: unless(before) ポイントカットの例

先ほどの例と同様に stmnt の部分が「範囲を持つ言語要素」の場合その範囲をブロックとして扱うため、この例では最後の unless の右側の stmnt の部分がブロックとして扱い、アドバイスを作用させる。

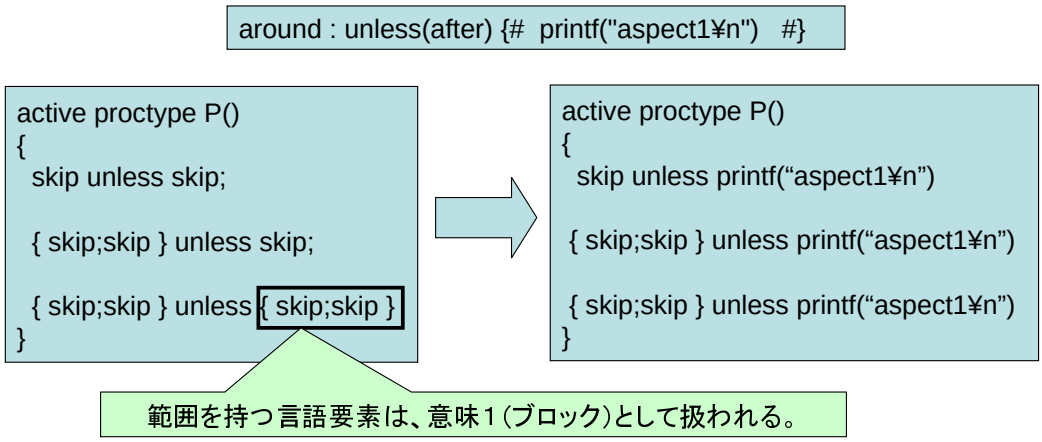

図 5.3: unless(after) ポイントカットの例

注意点

特になし。

5.3.3 proctype

説明

モデル中の proctype を意味１（ブロック）として指定するためのポイントカット

記法

```
proctype([プロセス名])
```

プロセス名のフィールドには次のどちらかを記述する。

- ”文字列”
  指定された文字列をプロセス名に含む proctype の囲う範囲を意味１（ブロック）として扱う。
- regexp(”正規表現”)
  指定された正規表現にマッチする名前の proctype の囲う範囲を意味１（ブロック）として扱う。

使用例

List5.8 のモデルに対して、次のアスペクトを作用させると、List5.9 のようになる。この例は、プロセス名”Q”の proctype を指定するポイントカットにより、この proctype が囲う範囲を意味１（ブロック）として扱い、around アドバイスを作用させることにより、proctype Q をコメントに置き換えている。

```
around : proctype("Q") {# /* aspect */ #}
```

List 5.8: proctype ポイントカットの例（ウィーブ前）

```
1  active proctype P()
2  {
3    skip;
4  }
5
6  active proctype Q()
7  {
8    skip;
9  }
10
11 active proctype R()
12 {
13   skip
14 }
```

List 5.9: proctype ポイントカットの例（ウィーブ前）

```
1  active proctype P()
2  {
3    skip;
4  }
5
6  /* aspect */
7
8  active proctype R()
```

```
{
  skip
}
```

注意点

proctype ポイントカット単独で用いることも出来るが、設計時点では allStmnt と一緒に使うことを想定している。

5.3.4 label

説明

モデル中のラベル付けされた文 (stmnt) のジョインポイントを指定するためのポイントカット

記法

```
label([ラベル名])
```

ラベル名のフィールドには次のどちらかを記述する。

- ”文字列”
  指定された文字列をラベル名に含む「ラベルの範囲」を意味１（ブロック）として扱う。
- regexp(”正規表現”)
  指定された正規表現にマッチする名前の「ラベルの範囲」を意味１（ブロック）として扱う。

ラベルの範囲とは　ラベルは BNF では次のように定義されている。

```
name ':' stmnt      /* labeled statement */
```

ラベルは BNF での stmnt に付けることができ、stmnt には「範囲を持つ言語要素」と範囲を持たない言語要素」の両方を含むため、ラベルはどちらの言語要素にも付加することができる。

本文法では、ラベルが印字文のような複文ではないような文に付加されていて、label ポイントカットによってこの印字分のジョインポイントが選び出された場合には、「ラベル＋印字文」の範囲を意味１として扱う。

例えば、List5.10 のようなモデルに対して次のポイントカットを用いてジョインポイントを選び出す場合を考える。

ポイントカット１により選び出されるのは、「名前が”LABEL1”のラベルがつけられた文 (stmnt) を含むラベルの範囲」なので、printf の印字文が選ばれるのではなく、図 5.4 のように、「ラベル＋ラベルが付加されていた文」を範囲をみなし、その範囲をブロックとして扱う。

ポイントカット２により選び出されるのは、「名前が”LABEL2”のラベルがつけられた文 (stmnt) を含むラベルの範囲」なので、「ラベル＋中括弧で囲まれた複文」を範囲とみなしてその範囲をブロックとして扱う。

```
label("LABEL1")      /* ポイントカット１*/
label("LABEL2")      /* ポイントカット２*/
```

List 5.10: ラベルの範囲

```
1  active proctype P()
2  {
3    LABEL1:printf("label 1\n");
4
5    LABEL2:{
6      printf("label 2\n");
7      skip
```

```
8     }
9   }
```

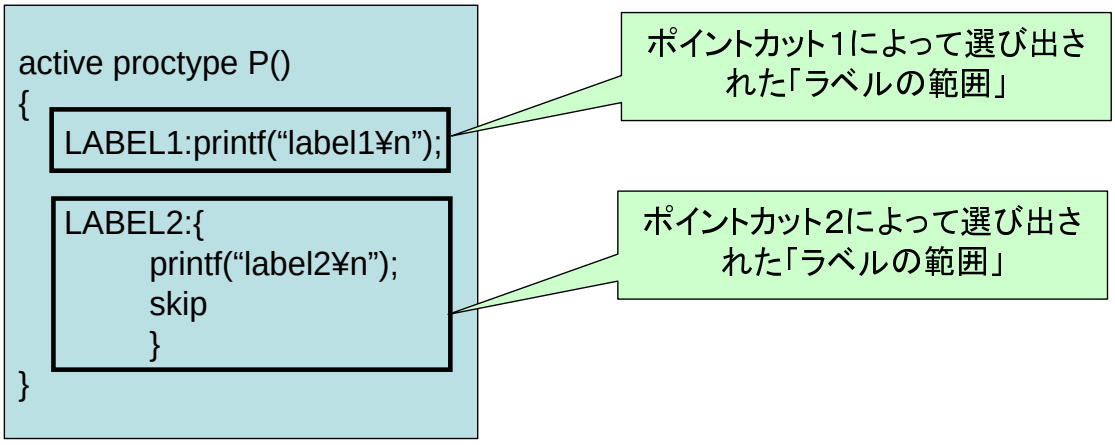

図 5.4: ラベルの範囲の図

使用例

List5.10 のモデルに対して、次のアスペクトを作用させると List5.11 のようになる。

一つ目のアスペクトは、名前が"LABEL1"のラベルを含むラベルの範囲に対して、around アドバイスを作用させるので、そのラベルの範囲自体がコメントに置き換わる。

二つ目のアドバイスは、名前が"LABEL2"のラベルを含むラベルの範囲に対して、before アドバイスを作用させるので、ラベルの範囲の直前にコメントが挿入される。注意していただきたいのは、コメントが挿入されるのはラベルと中括弧の間ではなく、ラベルの範囲の前である点である。これは、このポイントカットによって絞り込まれるジョインポイントが、中括弧の範囲ではなくラベルの範囲であることを表している。

三つ目のアドバイスは、二つ目と同じポイントカットで絞り込まれた、ラベルの範囲にたいして after アドバイスによりラベルの範囲の直後にコメントが挿入される。

```
around : label("LABEL1") {#  /* aspect1 */  #}
before : label("LABEL2") {#  /* aspect2 */  #}
after  : label("LABEL2") {#  /* aspect3 */  #}
```

List 5.11: label ポイントカット（ウィーブ後)

```
1  active proctype P()
2  {
3    /* aspect1 */
4
5    /* aspect2 */
6    LABEL2:{
7      printf("label 2\n");
8      skip
9    }
10   /* aspect3 */
```

```
}
```

注意点

ラベルが付加された stmnt の中の文に対して、アドバイスを作用させたいときは、allStmnt を用いる。

5.4 ポイントカットの演算子、操作

5.4.1 ALLSTMNT

説明

「範囲を持つ言語要素」を絞り込むポイントカットが絞り込んだ範囲中に含む全てのジョインポイントを取り出す。

この allStmnt は、本文法において重要な意味を持つ。前章にて、「範囲を持つ言語要素」に次の二つの意味をもたせることにより、文法の要件２を実現すると述べた。

- 意味１：
  範囲を持つ言語要素を指定すると、その範囲自体をブロックとして指定する。
- 意味２：
  範囲を持つ言語要素を指定すると、その範囲中の全てのジョインポイントを指定する。

本文法では「範囲を持つ言語要素」を指定するためのポイントカットを用いてジョインポイントを絞り込むと、そのジョインポイントは全て意味１での範囲としてアドバイスを作用させる仕様になっている。この allStmnt はその意味１での範囲を、意味２での範囲に変換するのに用いる。

意味１を意味２へどのように変換するかを、例を用いて説明する。List5.12 のモデルに次の proctype ポイントカットを用いてジョインポイントを絞り込むと、図 5.5 の左側のように、プロセス名 P の proctype が囲う範囲をジョインポイントとして絞り込む。

```
proctype("P")
```

図 5.5 の左側の円のように、このポイントカットによって、絞り込まれたジョインポイントの数は１つである。

次のように、このポイントカットに allStmnt を用いて、範囲が持つ意味を変換する。

```
allStmnt( proctype("P") )
```

allStmnt 次の手順で、意味１を意味２へと変換する。

1. まず、与えられたポイントカットが絞り込んだ「ジョインポイントの集合」の中から範囲を指すジョインポイントを絞り込む。
   (図 5.5 の左側の円から、範囲を指すジョインポイントである点を取り出す)
2. 絞り込んだ、範囲を指すジョインポイントが内包する「全てのジョインポイント」を取り出す。
   (図 5.5 では、１で取り出したジョインポイントが囲う範囲の全てのジョインポイント（skip,skip,printf) を取り出す。)
3. 2 で取り出したジョインポイントの集合を、元々のジョインポイントの集合に追加する。
   (図 5.5 の右側の円に２で取り出したジョインポイントを加える。)

List 5.12: allStmnt の説明用モデル

```
1 active proctype P()
2 {
```

```
3     skip;
4     skip;
5     printf("end");
6 }
```

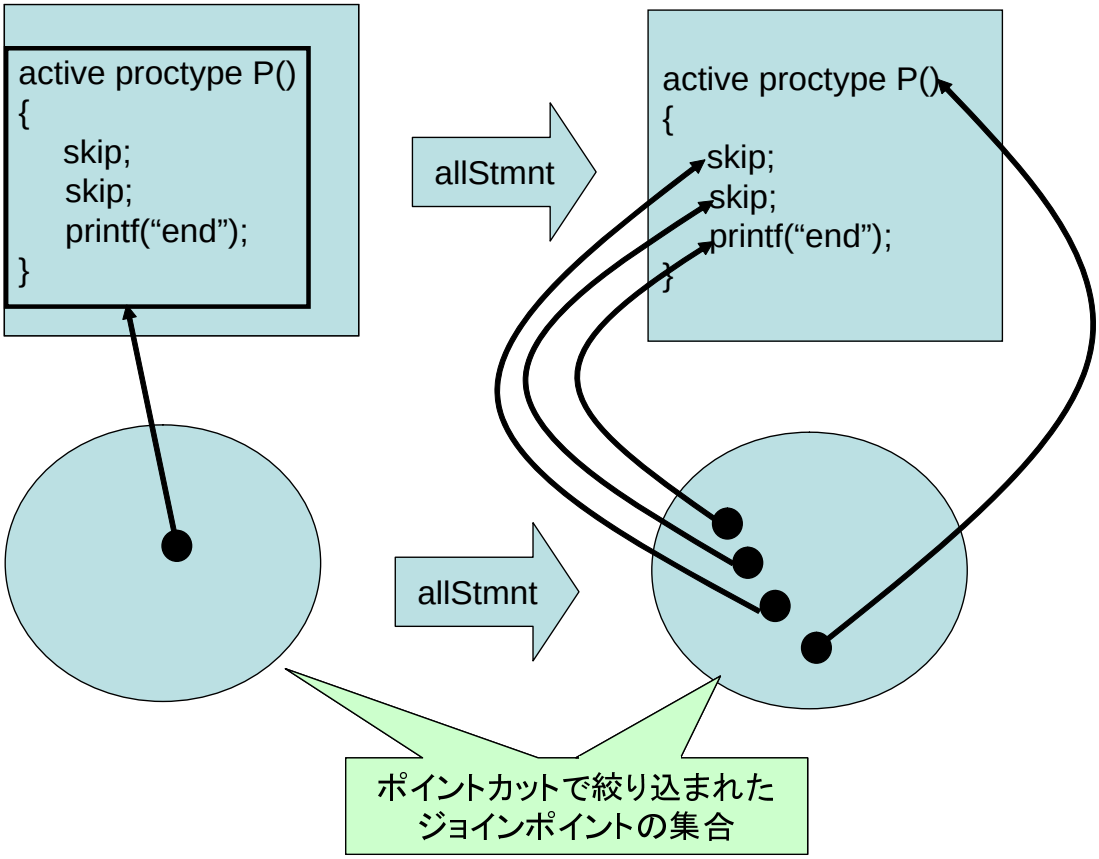

図 5.5: allStmnt

記法

```
allStmnt([pointcut])
```

[pointcut] のフィールドには、任意のポイントカットを指定する。

使用例

List5.12 のモデルに対して次のアスペクトを作用させると 5.13 のようになる。

```
after : allStmnt( proctype( "P" ) ) {#  /* aspect */ #}
```

List 5.13: allStmnt の説明用モデル (ウィーブ後)

```
1 active proctype P()
2 {
3   skip; /* aspect */
4   skip; /* aspect */
5   printf("end"); /* aspect */
6 }
7 /* aspect */
```

注意点

allStmnt は、アドバイスを作用させるジョインポイントの集合内に「範囲を表す言語要素」のジョインポインも含むために、List5.13 のように範囲に対してもアドバイスが作用する。

「範囲を持たない言語要素」のみにアドバイスを作用させたい場合には removeScope を用いて「範囲を持つ言語要素」をジョインポイントの集合から取り除く必要がある。

5.4.2 AND,OR

説明

ポイントカットで絞り込まれたジョンポイントの集合に対して、和集合 (or)、積集合 (and) のジョインポイントの集合を返す。

記法

```
[pointcut] && [pointcut]     /* and */
[pointcut] || [pointcut]     /* or  */
```

使用例

List5.14 のモデルに対して次のアスペクトを作用させると、List5.15 のようになる。

このアスペクトは、図 5.6 のように print ポイントカットによって、１つのジョインポイントが allStmnt の付いた proctype ポイントカットによって４つのジョインポイントが絞り込まれる。

この２つのジョインポイントの集合に対して、積集合をとると、各集合に共通に含まれる要素は、printf 文のポイントカットだけなので、この printf 文のポイントカットに対して after アドバイスが作用される。

よって、List5.15 のように printf 文の直後にコメントが追加される。

```
after : print() && allStmnt( proctype( "P" )) {#  /* aspect */  #}
```

List 5.14: and によるポイントカットの絞込み (ウィーブ前)

```
1 active proctype P()
2 {
3   skip;
4   skip;
5   printf("end");
6 }
```

List 5.15: and によるポイントカットの絞込み (ウィーブ後)

```
1 active proctype P()
2 {
3   skip;
4   skip;
5   printf("end"); /* aspect */
6 }
```

注意点

特になし。

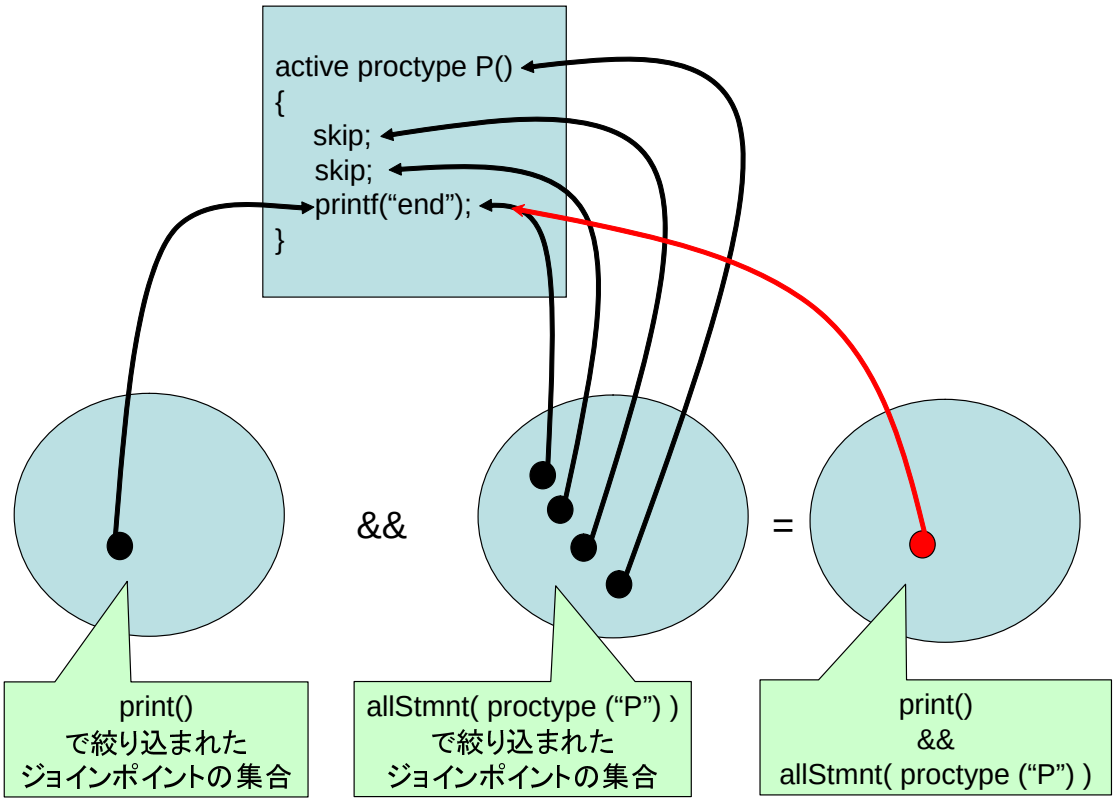

図 5.6: AND によるジョインポイントの絞込み

5.4.3 CONTAINS

説明

左辺のポイントカットによって絞り込まれたジョインポイントのうち、右辺のポイントカットで絞り込まれたジョインポイントを内部に「範囲を持つ言語要素」の範囲を意味１（ブロック）として取り出す。

記法

```
[pointcut] contains [pointcut]
```

[pointcut] には任意のポイントカットを記述する。

説明にて用いた「内部に持つ」という言葉はどのような意味を持つかを説明する。例として List5.16 のようなモデルを考える。

List 5.16: contains の説明用モデル 1

```
1  active proctype P()
2  {
3    atomic{
4      assert(true);
5      skip;
6      d\_step{
7        printf("in d_step");
8        skip
9      }
10   }
11 }
```

このモデルには、「範囲を持つ言語要素」が３個含まれている。それは、次の通りである。

- プロセス名 P の proctype
- atomic
- d_step

この３個の言語要素が、各範囲内にもつジョインポイントは図 5.7 のように、proctype は atomic、atomic は assert と skip と d_step、d_step は printf と skip である。

ここで注意していただきたいのは、proctype が直接 d_step を持っていない点である。つまり、「範囲を持つ言語要素」が「内部にもつジョインポイント」とはその言語要素が直接囲っている範囲にあるジョインポイントことを言う。

この List5.16 のモデルに対して、次のポイントカットでジョインポイントを絞り込む。

使用例

List5.16 のモデルに対して、以下のアスペクトを作用させると、List 5.17 のようになる。

```
after : allStmnt(proctype("P")) contains print() {# /* aspect */ #}
```

このアスペクトは、次の順序でポイントカットを絞り込む。

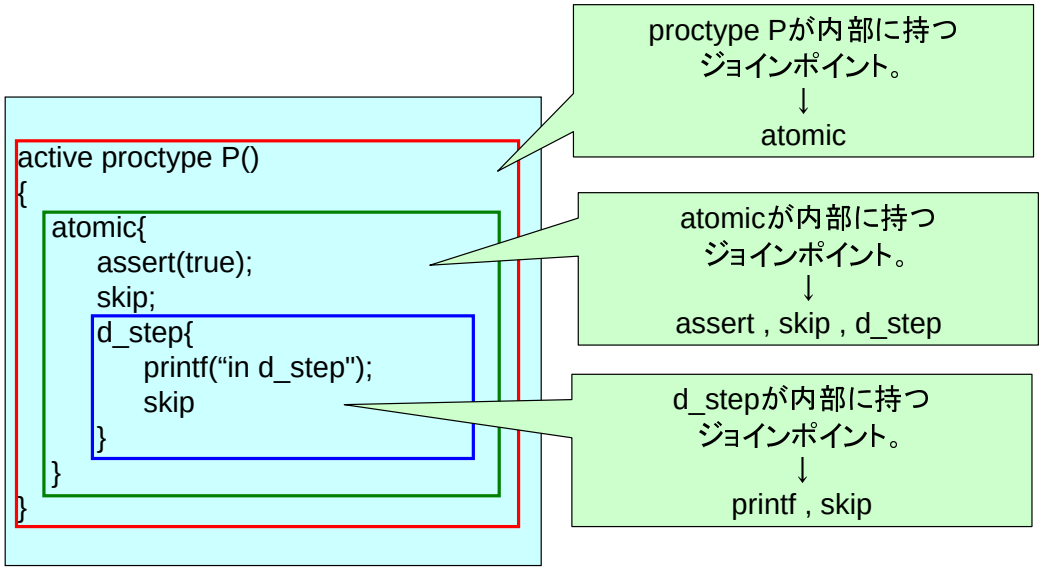

図 5.7: contains の説明図

1. まず proctype ポイントカットと print ポイントカットの評価を行う。
2. 次に、演算子の結合優先度が高い allStmnt が評価されて、proctype 中の全てのジョインポイントを選び出す。
   (図 5.8 の左側の円に対応)
3. proctype 中の全てのジョインポイント中で、内部に print ポイントカットによって選ばれたジョインポイント（= printf("in d_step"); という印字文）を含む「範囲を持つ言語要素」を選び出す。
   (図 5.8 の右側の破線の円に対応)

このようにして、d_step の囲う範囲がジョインポイントとして選び出される。その範囲に対して、after アドバイスが作用するので、d_step の範囲の後にコメントが挿入される。(List5.17)

List 5.17: contains の説明用モデル 2(ウィーブ後)

```
active proctype P()
{
  atomic{
    assert(true);
    skip;
    d\_step{
      printf("in d_step");
      skip
    }
    /* aspect */
  }
}
```

注意点

特になし。

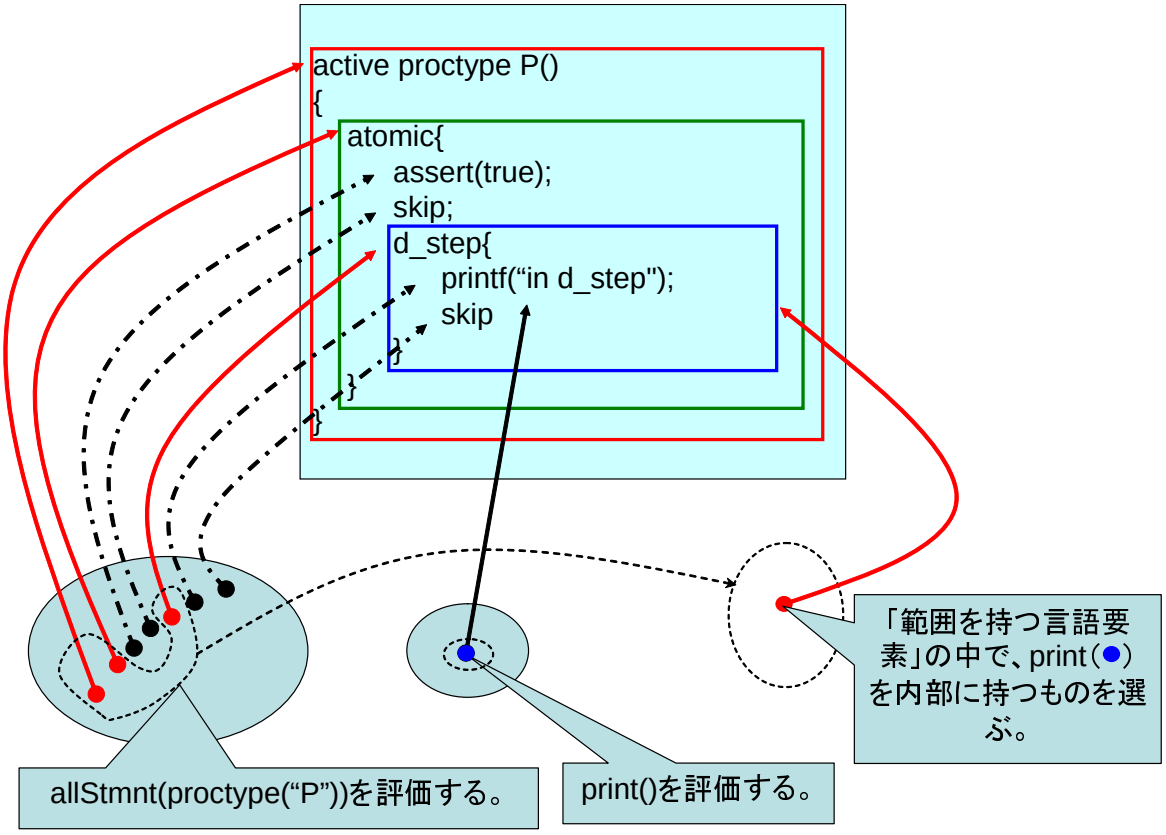

図 5.8: contains の動作

5.4.4 REMOVESCOPE

説明

指定されたポイントカットによって選び出されたジョインポイント中の「範囲を表す言語要素」のジョインポイントを取り除く。

記法

```
removeScope( [pointcut] )
```

[pointcut] には任意のポイントカットを記述する。

使用例

List5.12 と同様の例を用いて、この removeScope の説明を行う。

List 5.18: List5.12 と同様のモデル

```
1 active proctype P()
2 {
3   skip;
4   skip;
5   printf("end");
6 }
```

このモデルに対して、proctype P 中の全てのジョインポイントにアスペクトを作用させたいとき以下のようなアスペクトを用いる。このアスペクトにより List5.19 のように書き換えられる。

```
after : allStmnt(proctype("P")) {#   /* aspect */   #}
```

List 5.19: List5.12 と同様のモデル（ウィーブ後１）

```
1 active proctype P()
2 {
3   skip; /* aspect */
4   skip; /* aspect */
5   printf("end"); /* aspect */
6 }
7  /* aspect */
```

たしかに、proctype P の範囲中の各ジョインポイントの直後にアドバイスを割り込ませることができた。しかし、allStmnt でジョインポイントを絞り込む際に、「範囲を持つ言語要素」も一緒に絞り込まれてしまうため、「ある範囲の各単文に対して、アドバイスを作用させたい時」には、範囲を持つ言語要素」をアドバイスを作用させるジョインポイントの集合から取り除く必要がある。この様なときに、この removeScope を用いる。

List5.18 のモデルに対して、次のアスペクトを作用させることにより 5.20 のように、「範囲内の各単文」だけに対してアドバイスを作用させる。

```
after : removeScope(allStmnt(proctype("P"))) {#   /* aspect */   #}
```

List 5.20: List5.12 と同様のモデル（ウィーブ後２）

```
active proctype P()
{
  skip; /* aspect */
  skip; /* aspect */
  printf("end"); /* aspect */
}
```

注意点

5.5 その他の規則

5.5.1 アスペクトの評価順序

本文法の BNF での aspects は、アスペクトを記述したファイル（アスペクトファイル）を意味している。このアスペクトファイルには、複数のアスペクト (BNF での aspect) を記述することができる。

アスペクトが複数記述された、アスペクトファイルをモデルに作用させる場合は、上に書かれたアスペクトから順に作用されるため、アスペクトを複数書く際には、先に書かれたアスペクトによりモデルがどのように変化するかを考慮してアスペクトを記述する必要がある。

5.5.2 演算子の結合規則

各演算子の結合優先度、結合方向は表 5.6 の通りである。

二項演算子より、単項演算子のほうが結合優先度が高く、二項演算子の結合優先度を高めたいときには括弧でその演算を囲むと結合優先度が高くなる。

また、二項演算子の結合方向は全て、左結合なので、同じ優先度の演算子が複数並んだ場合には、一番左側から評価される。

表 5.6: 演算子の結合優先度、結合方向

結合優先度		演算子	種類	結合方向
（高い）	1	allStmnt	単項演算子	なし
↑	1	removeScope	単項演算子	なし
↑	2	contains	二項演算子	左結合
↑	3	and(&&)	二項演算子	左結合
（低い）	4	or(||)	二項演算子	左結合

5.5.3 セパレータの扱い

promela には、文と文の境目を表すセパレータが 2 種類あり、”;” と”->”である。この二つのセパレータの意味は同じであるが、文が次の文と関連がある場合には”->”そうでないときは”;”を用いることが多い。

本研究での、言語処理系は入力された promela を XML の内部モデルに変換して処理を行うので、文と文の間に使われていたセパレータが”;” であったか、”->”であったかという、「見た目」に関する情報はこの変換で保存されないため、ウィーブ後のモデルのセパレータは全て”;”と出力される。

第6章　実装

6.1　言語処理系の概要

本節では、言語処理系の概要を説明する。言語処理系の全体図を図 6.1 に示す。

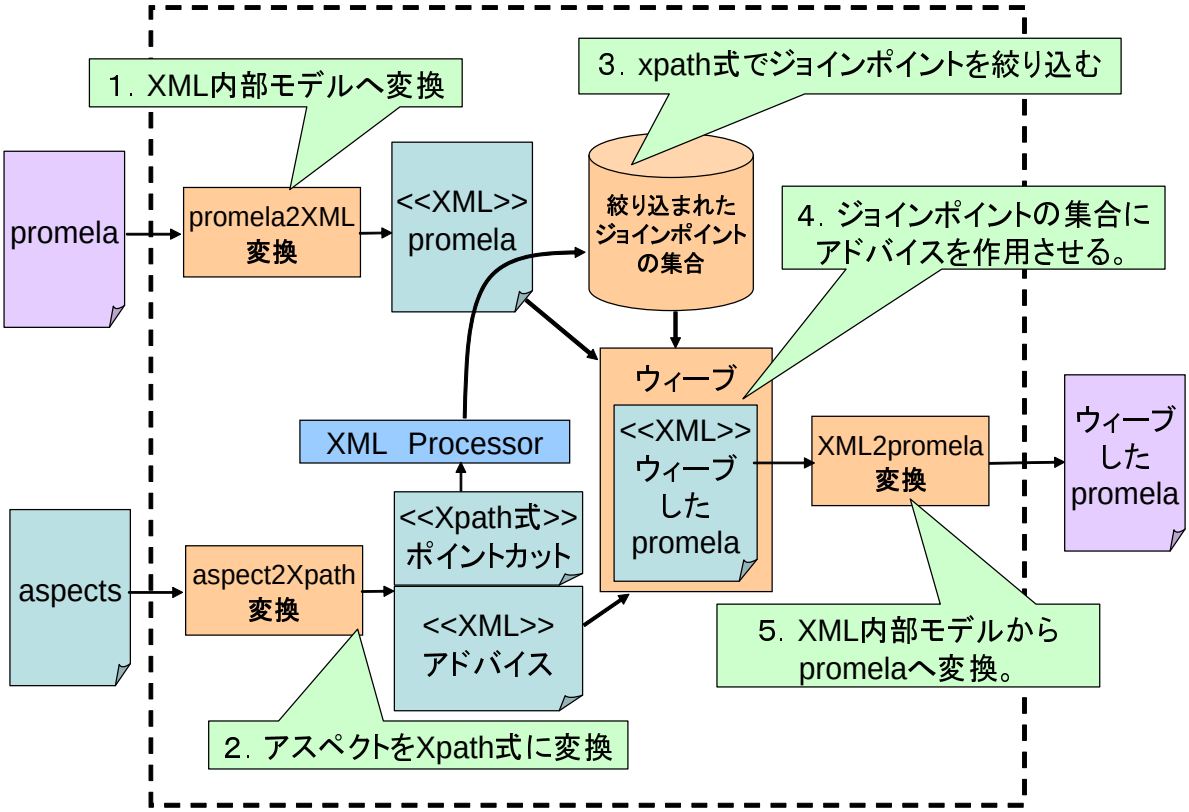

図 6.1: 言語処理系の全体図

言語処理系は、大きく３つの処理系からなるそれは次の通りである。

1. promela から処理系の XML モデルへの変換系
2. アスペクト文法から Xpath 式、XML への変換を行い XML 内部モデル上でのウィーブを行う合成系
3. 合成が終わった XML モデルから promela への変換系

この３つの処理系が 6.1 のような手順で処理を行う。この１～５の実装について、次節より説明を行う。なお、この５つの処理と実際のソースコードの対応は表 6.1 の通りである。

表 6.1: 各処理とソースコードの対応

	処理の内容	ソースコードの対応
1	promela から XML 内部モデルへの変換	p2x.jay,p2x.flex
2	アスペクトから Xpath 式への変換	a2x.jay,a2x.flex
3	ジョインポイントの絞込み	
4	XML 内部モデルへのアドバイスの作用	
5	XML 内部モデルから promela への変換	x2p.java

なお、jay,flex という拡張子が付いたソースコードは構文、字句解析器を生成するための生成系への入力ファイルである。よって正確には、その生成系が生成した、構文、字句解析器が表中のそれぞれの処理を行う。

6.2 言語処理系設計思想

本節では、本処理系がこの設計に至った理由について述べる。まず、本処理系の特徴である、アスペクトのウィーブする際に promela を内部モデルに変換に、内部モデル上でウィーブを行う方式について、この方式を採用するに至った理由を述べる。次に、内部モデルとしてＸＭＬを採用した理由を述べる。

6.2.1 内部モデルを用いる理由

promela に対して、アスペクトをウィーブをする際に、内部モデルに変換せずにそのままウィーブを行う場合、次の３つの処理を並行して行う必要がある。

1. promela の字句解析、構文解析
2. アスペクトの字句解析、構文解析
3. promela に対する、アスペクトのウィーブ

この方式では、１，２，３を個別に考えて実装を行うということができない。つまり、３つの処理のうちの１つの実装を行う際に、その実装が残りの２つの処理にどのような影響を与えるかを考慮しながら実装しなければならず、実装の困難さがその１点に集中してしまう。(図 6.2)

また、再利用性の面から見てもこの３つの処理が入り混じるような方式を避ける必要があった。

そこで、１，２，３の各処理を個別に行える方法を考えたところ、各処理は、入力に対して、「自分のやるべき仕事」を行いその結果を処理結果として出力する方式を考案した。(図 6.3)

この方式ならば、各実装は「入力」と「自分がやるべき仕事」だけを考えればいいので、実装の困難さが一点に集中することはない。

しかしながら、この方式を実現するには、「処理結果」を表現するための、各処理間の共通の表現形式が必要であった。そこで、内部モデルというものを定義して、各処理は、その内部モデルに対応する処理結果を出力することにより、この問題を解決しようと考えた。

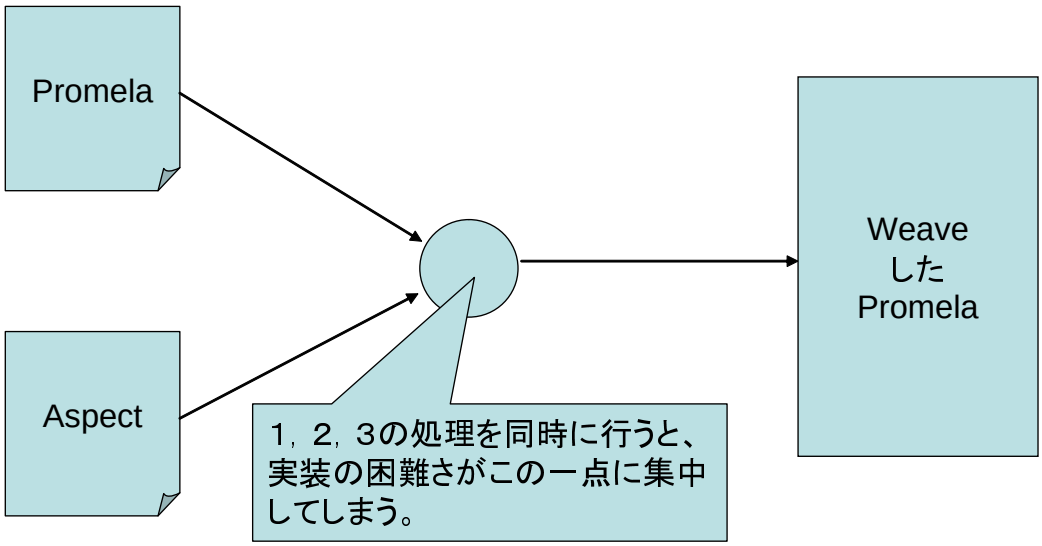

図 6.2: 避けたい処理方式

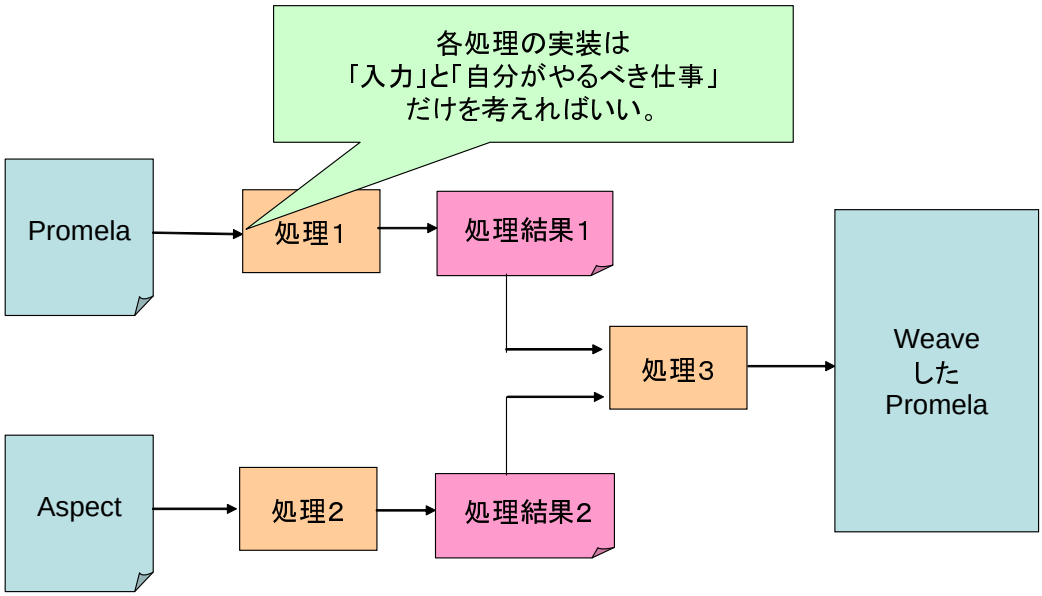

図 6.3: 処理の分割

6.2.2 内部モデルに XML を用いる理由

各処理間の共通表現形式として内部モデルを定義する際に、その内部モデルが持つべき性質は次の 2 点である。

- 本文法は、promela の言語要素に注目したものであり、promela から内部モデルに変換した際に、その promela の各コードに対して、そのコードがどの言語要素なのかを識別するための属性を与えられる必要がある。
- 本文法の「範囲の指定の二つの意味」のうち意味 2 は「範囲を持つ言語要素」を指定した場合には、その範囲中の全てのジョインポイントを指定する」という意味であり、範囲の中にジョイポイントの集合が存在するため、それを表現するための内部モデルは集合、包含関係を扱える必要があった。

XML はこの 2 つの性質を備えている。また、これに加え内部モデルを XML にすることにより、次のような利点がある。

- XPath 式を用いた、XMLProcessor によるモデル中の条件探索機能を用いることが出来る。
- XML は標準化されている規格であり、XSLT という XML モデルの変換用言語が用意されているため、本研究での内部モデルを他の処理系への入力として利用できる可能性がある。
- java で実装する際、XML の生成、操作は java の API を経由して行われるため、XML のバージョンがあがった場合にも、API がそのバージョンに対応すれば、生成される XML の互換性が保障される。
- 実装時にデバッグをする際に、モデルがテキストデータで表現されているため、デバッガなどのツールを使わずに内部モデルの状態を知ることができる。

このように、XML は「内部モデルが持つべき性質」を持ち、かつ付帯的な利点があったため、本研究では内部モデルとして XML を採用した。

6.3 promela から XML への変換

この処理を行うのは p2x.jay,p2x.flex によって生成される、構文、字句解析器である。p2x.jay は promela の BNF を構文規則とし、BNF での各言語要素に対応する XML ノードを生成する。このノードを生成する際に、promela での各コードが、どの言語要素に対応しているかという属性情報を各ノードの属性に保持する。また、promela の BNF の全ての言語要素を XML ノードとする必要はない、なぜならば XML ノードと XML ノードの間が、XML モデル上のジョインポイントとなるため、各ノードは文 (BNF での stmnt) でなくてはならない。仮に式 (expression) を XML ノードとしてしまった場合、式と式の間にアスペクトを割り込ませることが出来てしまうため、文法上正しい位置にアスペクトを作用させることが出来なくなってしまう。よって XML の 1 ノードが、promela での 1 文 (stmnt) と対応する。

6.4 アスペクトから Xpath 式への変換

アスペクトから Xpath 式への変換は、表 6.1 のように a2x.jay,a2x.flex によって生成される構文、字句解析器が行う。XML 内部モデルには、promela の BNF 上での言語要素名が属性として XML ノードに保持

してある他にも、そのノード名自体が言語要素名に対応している。Xpath を用いて XML 文書中の任意のノード名のノードの集合を取り出すには、次のような Xpath 式を用いる。この例は、ノード名”proctype”のノードを取り出す例である。

```
//proctype
```

また、これに属性の情報を加えてノードを絞り込みたい場合には、属性を指定するための@を用いる。次の例は、ノード名”proctype”のノードで、属性”procname”が”P”であるノードを取り出すための例である。

```
//proctype[@procname='P']
```

また、属性名には、その一部に含む文字列という形の指定も出来る。これは xpath の述語である。contains を用いる。次のように指定すると属性名に”P”を含む proctype のノードを XML 文書から取り出すことができる。

```
//proctype[contains(@procname,'P')]
```

アスペクト文法の BNF(p.28) のポイントカットの primitive は、ほとんどこのような xpath 式に変換される。

ただし、primitive のうち、引数に文字列を取るポイントカットにおいて、正規表現（regexp) が指定された場合には、xpath に加えて独自実装の正規表現探索が用いられる。

Xpath のバージョン 2.0 においては、Xpath 式内に正規表現を用いることが出来るため、このような実装は必要ないのだが、本処理系で用いている xpath の処理系は、sun の標準の XpathAPI であるため、対応する Xpath のバージョンは 1.0 であり、正規表現には対応していないためこのような独自実装を行ったが、sun 提供の XpathAPI が Xpath2.0 に対応すればこの部分は、必要なくなるので、基本的には Xpath でノードの絞込みを行っているといえる。

6.5 ジョインポイントの絞込み

ポイントカットのうち primitive に関しては、ほとんど前節で説明した、xpath 式により XML ノードを取り出し、それのアスペクトを作用させるジョインポイントの集合に追加することにより実現している。

ポイントカットには primitive 以外に、and,or,contains,allStmnt,removeScope があるが、これらに関しては、Xpath 式ではなく、java の Collection を用いて集合に対する演算を行っている。

6.6 アドバイスの作用

アドバイスは、ポイントカットの評価を全て終え、ジョインポイントを絞り込み終わった後に作用される。アドバイスの作用は、ジョインポイントの集合から、ノードを一つづつ取り出し、各ノードに対して文法で指定されたアドバイスに対応する処理メソッドを呼び出すことにより実現している。

文法のアドバイスと、その処理を行っているメソッドの対応は表 6.2 の通りである。

表 6.2: アドバイスと処理メソッドの対応

アドバイス	メソッド
before	appendBefore(<ジョンポイント>,<アドバイス>)
after	appendAfter(<ジョンポイント>,<アドバイス>)
around	appendAround(<ジョンポイント>,<アドバイス>)

6.7 XML から promela への変換

本処理系での XML モデルは、生成から内部操作まで全て DOM(Document Object Model) の API を用いている。それは、XML の包含関係やノードの集合演算を行う際に、こちらの API を利用したほうが、実装を行いやすいためである。

XML から promela への変換も、DOM の API を用いて実装することも出来るが、変換ルールを記述するように、変換のためのコードを実装したかったため、イベントドリブンに処理が進む SAX(Simple Api for Xml) のほうが DOM よりこの処理には向いているため、この実装は SAX を用いている。

第7章　適用例

7.1　適用例１：同一の検証対象に対して、異なった検証を行う際への適用

本節では次の List.7.1 のモデルに対して様々な検証を行う例を示す。

List 7.1: 検証モデル

```
mtype = {msg1,msg2,msg3,msg4,msg5}
chan ch = [2] of {mtype};
mtype rcvMsg;

active  proctype P()
{

STATE1:
  {
    printf("STATE1\n");
    ch?rcvMsg;
    if
      ::rcvMsg == msg1 -> goto STATE1
      ::rcvMsg == msg2 -> goto STATE2
      ::rcvMsg == msg3 -> goto STATE3
      ::else->goto STATE1
    fi
  };

STATE2:
  {
    printf("STATE2\n");
    ch?rcvMsg;
    if
      ::rcvMsg == msg1 -> goto STATE1
      ::rcvMsg == msg2 ->
          if
            ::true -> goto STATE1
            ::true -> goto STATE3
          fi
      ::rcvMsg == msg3 -> goto STATE3
      ::else->goto STATE1
    fi
  };

STATE3:
  {
    printf("STATE3\n");
    ch?rcvMsg;
    if
      ::rcvMsg == msg1 -> goto STATE1
      ::rcvMsg == msg2 -> goto STATE2
      ::rcvMsg == msg3 ->
          if
            ::true-> goto STATE1
            ::true-> goto STATE2
          fi
      ::else->goto STATE1
```

```
49        fi
50     }
51  }
52
53  active proctype Q()
54  {
55     do
56        :: ch ! msg1
57        :: ch ! msg2
58        :: ch ! msg3
59        :: ch ! msg4
60     od
61
62  }
```

このモデルは、プロセスQがランダムな値をチャネルに書き込み、プロセスPがそれを受信して状態遷移を行うモデルである。プロセスPには、状態が3つ存在し、各状態をラベルで表現することにより、状態遷移を goto 文で実現している。プロセスPは基本的には、msg1,msg2,msg3 を受信することを意図して作られているが、外部環境（プロセスQ）がこれ以外の値を送信することもあるので、その場合は、STATE1,STATE2,STATE3 のどの状態にプロセスがあったとしても、STATE1 に状態遷移するように作られている。このモデルに対して、行う検証は以下の通りである。

1. チャネルの受信値のアサーションによるチェック
2. else ガードの付け替えによる、考慮外のメッセージ受信の検出
3. STATE1 の具象化
4. STATE2,3 の抽象化

7.1.1 検証１：チャネル受信値のアサーションによるチェック

プロセスPは、基本的には msg1,msg2,msg3 を受信することを意図しているが、それ以外の値も受信することがある。チャネル受信直後にチャネル受信変数をアサーションにより検査することにより、この状態を捕捉できる。このような検証を行うための検証モデルをアスペクトにより生成する。

この検証を行いたい場合には、チャネル ch の受信のジョインポイントをポイントカットによって絞込み after アドバイスでコードを割り込ませれば良いので、次のようなアスペクトでこの検証モデルを生成できる。

```
after : chan("ch","?*","") {#
    assert( rcvMsg == msg1 || rcvMsg == msg2 || rcvMsg == msg3 );
#}
```

今回のモデルでは、チャネル ch の受信はプロセスPのみが、行うが仮に他のプロセスが受信を行うのだが、アサーションはプロセスPの受信直後にのみ埋め込みたい場合には次のようにアスペクトを指定すればよい。このアスペクトは、proctype ポイントカットで proctypeP の範囲をブロックとして絞込み allStmnt によって、その範囲の全てのジョインポインを絞り込むため、このポイントカットと and 演算を行うことにより、アスペクトを作用させる範囲をプロセスP に局所化できる。

```
after : allStmnt(proctype("P")) && chan("ch","?*","") {#
    assert( rcvMsg == msg1 || rcvMsg == msg2 || rcvMsg == msg3 );
#}
```

7.1.2 検証２：else ガードの付け替えによる、考慮外のメッセージ受信の検出

プロセスPは、外部環境により msg1,msg2,msg3 以外も受信する場合があると、先で述べたがどのようなときに、これ以外のメッセージを受信するかを確かめたいときに else のガードが邪魔になる。つまり、else のガードがなければ考慮外の値の受信をしたときに if 文で実行がブロックされるため、その状態を検出できるが、else のガードがあるために、どんな値を受信しても if 文はブロックする事がない。

アスペクトを用いて、この考慮外の値を受信した際にそれを検出できるような検証モデルを生成する。アスペクトは、option ポイントカットで絞り込んだジョインポイントのうち、内部に else を含むような option のジョインポイントだけを絞り込めばよいので contains を使ってこれを実現する。

その様にして、絞り込んだジョインポイントに対して around アドバイスを作用させることにより option を書き換える。

```
around : option() contains else(){#
::else -> assert(false)
#}
```

7.1.3 検証３：STATE1 の具象化

このモデルの STATE1 と STATE2,3 を比較すると、STATE2,3 においては、各状態で msg2,msg3 を受信すると自身以外の状態へのランダムな遷移をするように実装されている。STATE1 も、最終的にはそのような遷移を実装したいのだか、現状ではとりあえず STATE2,STATE3 の振る舞いに注目するため、STATE1 に関してはとりあえず、自己遷移するように実装されていたとする。

この場合に、STATE1 を STATE2,3 のように具象化することを考えたときにモデルを書き換える必要があるが、アスペクトを用いることにより、元々のモデルを保持したまま、アスペクトの追加だけでこれを実現できる。

この場合の検証モデルは、STATE1 内の自身への遷移を具象化のコードに置き換えればよいのでアスペクトは、goto ポイントカットによりまず STATE1 へ遷移するような goto のジョインポイントを選び出す。次に label ポイントカットを用いて STATE1 の範囲のジョインポイントを選び出し allStmnt によりその範囲内の全てのジョインポイントを取得するそして、そのジョインポイントの集合と先ほどの goto のジョインポイントを and 演算することにより、目的のジョインポイントを選び出す。

```
around : goto("STATE1") && allStmnt(label("STATE1")) {#
if
::true -> goto STATE2
::true -> goto STATE3
fi
#}
```

7.1.4 検証４：STATE2,3 の抽象化

今度は、逆に STATE2,3 の抽象化を考える。STATE2,3 にはそれぞれ、msg2,msg3 を受信した際に自身以外の状態へランダムに遷移する if 文を実装済みであるが、これを自己遷移に書き換えることにより STATE1 のような抽象化されたモデルで検証を行いたい。

このような場合には、次のようなアスペクトを用いればよい。

```
around : (option() contains expr("rcvMsg == msg2"))
                                    && allStmnt(label("STATE2")){#
                ::rcvMsg == msg2 -> goto STATE2;
#}

around : (option() contains expr("rcvMsg == msg3"))
                                    && allStmnt(label("STATE3")){#
                ::rcvMsg == msg3 -> goto STATE3;
#}
```

このアスペクトの動作は次の通りである。まず、option ポイントカットで取り出してきたジョインポイントのうち expr ポイントカットに指定されいるような評価式を内部に含むものを取り出す。これだけでは、意図した状態以外のガードも書き換えてしまうので allStmnt と label ポイントカットによりアスペクトが作用する範囲を絞り込み、意図したジョインポイントに対して around アドバイスを用いてコードを書き換える。

7.2 適用例２：類似しているが、異なる検証対象のモデルを効率的に記述する際への適用

本節では、本研究の文法と言語処理系の適用例として、リアルタイムオペレーティングシステムの仕様である OSEK/VDX における、タスクと資源に関する検証を行う際に、記述済みのモデルから異なった検証対象のモデルをアスペクトにより生成する。

7.2.1 OSEK/VDX

OSEK/VDX は自動車向けのリアルタイムオペレーティングシステムの仕様であり。マルチタスクソフトウェアである。OSEK/VDX におけるタスクは優先度を持つことができ、この優先度をもとにスケジュー

リングが行われる。タスクには、基本タスクと拡張タスクがあり、基本タスクは３状態、拡張タスクは４状態の状態遷移をする。今回の検証では、基本タスクをもちいるため、タスクは３つの状態を持ちそれは図 7.1 の通りである。また、資源は 2 つの状態を持し、資源の状態遷移は図 7.1 の通りである。

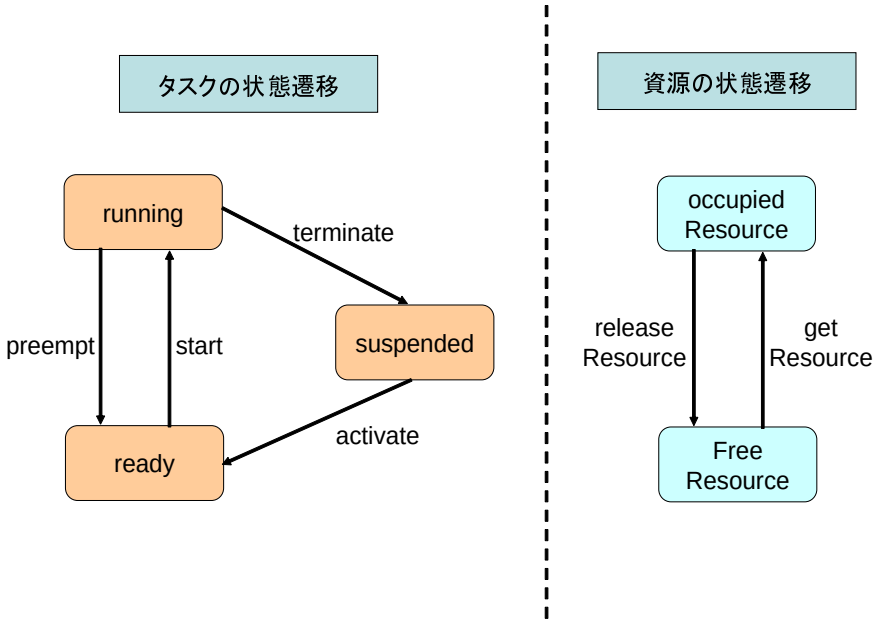

図 7.1: OSEK/VDX における基本タスクと資源の状態遷移

7.2.2 検証モデル

今回、この OSEK/VDX における「タスク２つ、資源１つ」の複合状態のモデルがすでに存在している際に、このモデルから「タスク２つ、資源なし」の複合状態のモデルをアスペクトにより生成する。

「タスク２つ、資源１つ」の複合状態のモデルは、普通に考えるとタスクがそれぞれ３つの状態を持ち、資源が２つの状態を持つため、その複合状態は３Ｘ３Ｘ２＝１８（状態）となる。この１８状態から、OSEK の仕様での制約から到達しない遷移、検査を行うことが無理な遷移を取り除いたものが今回用いるモデルである。モデルは付録 B(p.80) に掲載する。このモデルに対して、タスクが資源を取得している状態への遷移を全て取り除くことにより、「タスク２つ、資源１つ」のモデルから「タスク２つ資源なし」のモデルへモデルを変更する。

変更が必要な遷移を、図 7.2 に示す。この図中の各状態の中の文字列はタスクの状態を示し、括弧の中は資源状態を示す。Ｆならばそのタスクは資源を占有していない、Ｏならばそのタスクが資源を占有している状態である。

この図からわかるように、取り除くべき遷移の遷移先は次の表 7.1 の通りである。では、promela のモデルに対してどのようなアスペクトを記述すればよいかを考える。なお promela での「タスク２つ、資源１つ」のモデルは、付録 B List.B.1 に掲載してある。

この promela でのモデルは、タスクの状態をラベルで表現し、ある状態への遷移を goto 文を用いて表現したものである。例えば、タスクＡ、タスクＢ共に suspended 状態、資源を非占有の状態を表したコードは以下の通りである。

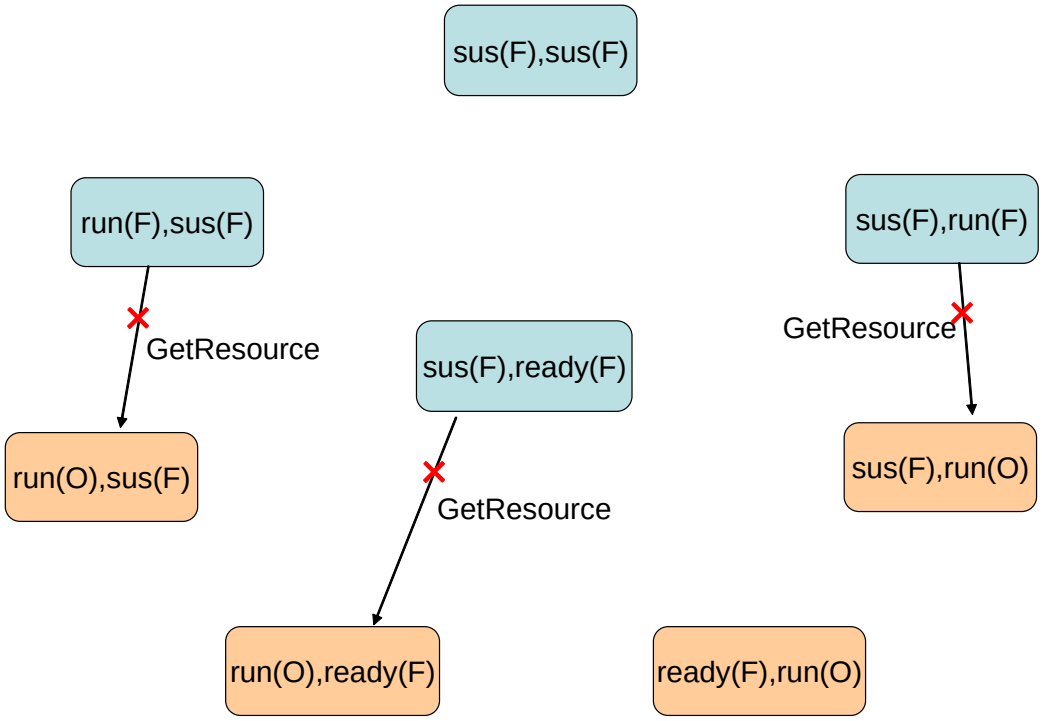

図 7.2: モデルより取り除く遷移

表 7.1: 取り除く遷移の遷移先

	タスクA		タスクB		promela でのラベル名
	状態名	資源	状態名	資源	
1	running	占有	suspended		runOcc_susFree
2	running	占有	ready		runOcc_readyFree
3	suspended		running	占有	susFree_runOcc

```
susFree_susFree:

  atomic{
    printf("susFree_susFree\n");
    if
      ::ActivateTask(ownTaskIDA)->goto runFree_susFree
      ::ActivateTask(ownTaskIDB)->goto susFree_runFree
    fi
  };
```

promelaでのモデルは、どの状態に関しても、このような記述となっているため、ラベル「runOcc_susFree」、「runOcc_readyFree」、「susFree_runOcc」への goto 文を含むセミコロン２つから始まる option をモデル中から取り除けば、資源なしのモデルを生成できる。よって、今回のモデル生成のためのアスペクトは次のように定義できる。

```
around : option() contains goto("runOcc_susFree") {# /* aspect1 */ #}
around : option() contains goto("runOcc_readyFree") {# /* aspect2 */ #}
around : option() contains goto("susFree_runOcc") {# /* aspect3 */ #}
```

このアスペクトを、promela のモデルに作用させると、資源占有状態への遷移を含む option は全て、コメントに置き換わる。ウィーブ後の promela を付録 B ListB.3 に示す。このモデルでは確かに、資源占有状態への遷移がコメントに入れ替わっている。

7.3 適用例に対する考察

7.3.1 適用例１

ここでは、適用例１のうち「検証１：チャネル受信値のアサーションによるチェック」について、本研究での提案手法とそれ以外の手法について比較を行い、考察したいと思う。

今回、例題としてあつかった検証モデルでは、プロセスＰはチャネルから受信した値を一度、rcvMsg という大域変数に書き込み、if 文でその変数を参照し遷移先の判断を行っていた。

このモデルの構造に着目すると、アスペクトを使わずにも不変表明により検証を行えることが分かる。不変表明とは、proctype を新たに宣言し、プロセスの実行中変わることがない論理式を表明することにとり、その論理式を満足しない状態を検出する手法である。

例えば、先ほどのモデルにおいてチャネル受信用変数 rcvMsg が msg5 にならないことを、モデルの実行中保証したい場合は、次のような不変表明を記述する。これにより、もし rcvMsg が msg5 となったときには、表明の論理式が false となり、アサーション違反により、その結果と反例を spin は出力する。

```
active proctype inv()
{
     assert( rcvMsg != msg5 )
}
```

この不変表明を付加したモデルで検証を行いアサーション違反が起きなければそのモデルの実行において rcvMsg が msg5 にならない事を保障できる。

この不変表明による検証方法と、本研究の提案手法を比較する。

不変表明は、Spin が検証において全てのプロセスの複合状態を作り出すため、この不変表明のアサーションが全ての実行のタイミングで割り込まれるため、不変性を保障できるというものである。

全てのタイミングで割り込まれるということは、チャネル操作が行われていない文においてもこのアサーションが割り込まれるが、そもそもチャネルからの受信がない限り今回のモデルにおいては rcvMsg の値は変化しない。

一方、本研究での提案手法は、チャネルの操作の直後のみに注目しアサーションを実行するので、無駄な検査が行われない。

実際に、この二つを定量的に比較するため上記の不変表明を使った場合と、次のアスペクトにより生成したモデルの検証を行い状態数の比較を行った。

```
after : chan("ch","?*","") {#
    assert( rcvMsg != msg5 )
#}
```

アサーションの中の論理式は、プロセスQが msg1〜msg4 までのメッセージを書き込むことから今回のモデルでは常に満たされるため、アサーション違反は起きない。よって、検証器は全状態を探索するため、その状態空間を比較することにより、状態数から見た２つの手法の検討が行える。各手法の状態数は表 7.2 の通りである。

表 7.2: 本研究の提案手法と不変表明による検証時の状態数比較

	アスペクトを用いた場合	不変表明を用いた場合
stored	925	1748
matched	732	1376
transitions(=stored+matched)	1657	3124

transitions の値が、アスペクトを用いた場合には、不変表明の半分程度に抑えられている。

不変表明はモデルのどの複合状態でも、論理式が成り立つことが保障できるが、その分状態数」という面でコストがかかる。今回のように、アサーションにより確認したい場所が明確な場合は、本研究の提案手法は有効である。

また、不変表明を用いる方法は、プロセス数が増えるとその複合状態の全てのタイミングで表明の検査を行う必要があるため状態数が非常に大きくなってしまう、よって今回の例以上に並行度が高くなる場合にも、本研究の提案手法は有効であると考察する。

7.3.2 適用例２

適用例２では、異なる検証対象のモデルを類似したモデルからアスペクトを用いて生成できる事を示した。

今回の例では、状態遷移図を元に切り取るべき遷移が明確で合ったため、このようなモデルの生成を実現できた。

本研究での提案手法は、モデルに対してアスペクトを作用させるときのポイントカットを用いてその作用する範囲を指定するため、モデル上に何らかの特徴がなければ、アスペクトを合成することができない。

今回の例の場合では、

- ラベル名に状態名を用い goto 文により状態遷移を実現していた。
- 各状態が atomic 文で囲まれていたため、状態の絞込みが可能である。

といった特徴があったため、このようなモデル生成を実現できた。

本研究の提案手法を、用いる場合には、モデル中に特徴をもたせ、ポイントカットでの指定を行いやすい形でモデルを作成しておくことにより、より有効に本手法を用いることができると考察する。

第8章　まとめ

8.1　まとめ

本研究では、モデル検査器 SPIN を用いた検証における promela での検証モデルにおいて、検証目的の変更によるモデルの横断的変更をアスペクト指向技術を用いて実現することを目的とした。

モデルの横断的な変更を扱うためには、promela の様々な言語要素に対してアスペクトを作用させる必要があり、既存のアスペクト指向のジョインポイントモデルの方式を promela へ適用できないかを考察したが、promela には「意味上の範囲」と「コード上の範囲」を一致させるような仕組みがないため、これは困難であった。

そのため、独自のジョインポイント指定方式を考案し、これは promela の言語要素での「範囲を持つ言語要素」に対して 2 つの意味を持たせることにより、様々な範囲を絞り込むというものであった。また、promela での検証対象のモデル化においてチャネルは、同期や排他制御などの重要な仕組みをモデル化するために用いられることが多いため、チャネルを指定するポイントカットに関しては、表現力の高い文法としてこれをまとめた。

そして、この文法を処理するための言語処理系を実装するために、言語処理系内部で XML による内部モデルにより処理を進める処理方式を考案し、これを実装した。

最後に、提案した文法、言語処理系の有効性を確認するために、実際の promela のモデルにたいしてアスペクトを作用させ、これを考察し本研究の有効性を確認した。

8.2　今後の課題

8.2.1　本研究の文法におけるアドバイス

本研究の文法におけるアドバイスは、本言語処理系を用いて様々なアスペクトの作用を実現するために、アドバイスに対する文法上のチェックは行わないような仕様となっている。そのためアドバイスとして文法上間違っている文を書いてウィーブした際には、ウィーブ後のモデルは文法上間違っているモデルとなってしまう問題点がある。

この問題点を解決するためには、まず、アスペクトに指定されたポイントカットの種類によって記述することが出来るアドバイスというものを定義する必要がある。これが出来れば、言語処理系の合成系の構文解析器生成の際に、その定義に対応した構文規則を追加した入力を構文解析器生成系に行うことにより、本問題を解決できる。

8.2.2　扱えないモデル

本研究で、実装した言語処理系は promela の BNF を元に処理を行う。そのためこの BNF に従っていない promela モデルは扱うことができない。モデル検査器 Spin は、この BNF に従っていない文法でもある

程度の範囲は許容してコンパイルエラーを出さない。この BNF に従っていないが、Spin では、エラーにならないモデルに対してアスペクトを作用させたいときはそのモデルを、厳密に BNF に従う形に書き直さなければならない。本言語処理系もこのようなモデルに対応させるには、Spin の内部の実装がどの範囲の文法を許容するかを厳密に定義する必要がある。

8.3 今後の展望

本研究では、検証モデルにおける横断的関心事を扱うために、言語レベルでの解決方法を提案した。今後の展望としては、この文法、言語処理系用いた有効的な検証手法を提案する必要がある。

第9章　最後に

9.1　謝辞

本研究を進めるにあたり、多大なるご指導を賜りました、岸知二特任教授、青木利晃特任准教授、片山卓也教授に感謝を申し上げます。また、本研究の適用例２として用いた検証モデルの提供をしていただいた、青木研究室、山崎真吾君に感謝いたします。

最後に、本研究を進める上で様々な議論や質問に快く応じてくださった、岸研究室、青木研究室、片山研究室、デファゴ研究室の皆様にこの場で感謝を申し上げたいと思います。ありがとうございました。

参考文献

[1] Gerard J.Holzmann, "The spin model checker: Primer and reference manual", Addison-Wesley Pub , September , 2003

[2] Gerard J.Holzmann, "コンピュータプロトコルの設計法", 株式会社カットシステム 1994, 11

[3] E. M. Clarke, Orna Grumberg, Doron Peled, "Model Checking", MIT Press 2000, 1, 7

[4] E.W.Dijkstra, "Guarded Commands,Nondeterminacy and Formal Derivation of Programs", Communications of the ACM, Volume 18, Issue 8, pp.453-457

[5] C.S.R.Hoare, "Communicating sequential processes", Communications of the ACM, Volume 21, Issue 8, pp.666-677

[6] 五月女 健治, "JavaCC コンパイラコンパイラ for Java", テクノプレス 2003,10

[7] A. V. エイホ, R. セシィ, J. D. ウルマン, 原田 賢一, "コンパイラ I　原理・技法・ツール" , サイエンス社 1990,10

[8] Ramnivas Laddad, "Aspectj in Action: Practical Aspect-Oriented Programming", Manning Pubns Co 2003,7,1

[9] 長瀬 嘉秀, 天野 まさひろ, 鷲崎 弘宜, 立堀 道昭, "AspectJ によるアスペクト指向プログラミング入門", ソフトバンクパブリッシング , 2004,4,16

[10] Erik T. Ray, 山本 和彦, 梶浦 正規, 中原 晃司, 豊田 公児, "入門 XML", オライリー・ジャパン , 2001,09

[11] Gerard J.Holzmann, "Spin - Formal Verification", (http://spinroot.com/)

[12] James Clark, "XML Path Language (XPath)", World wide web consortium , (http://www.w3.org/TR/xpath)

[13] Gerwin Klein, "JFlex - The Fast Scanner Generator for Java" , (http://jflex.de/)

[14] Axel T.Schreiner, "jay (Language Processing)", (http://www.cs.rit.edu/ ats/projects/lp/doc/jay/package-summary.html)

[15] AOSD Steering Committee, "Aspect-Oriented Software Development Community & Conference :: AOSD", (http://www.aosd.net/)

[16] The Eclipse Foundation, "The AspectJ Project", (http://www.eclipse.org/aspectj/)

[17] jboss.org, ”jboss.org:community driven JBoss AOP”,
(http://labs.jboss.com/jbossaop/)

付 録A　promelaのBNF

```
spec    : module [ module ] *

module  : utype         /* user defined types */
        | mtype         /* mtype declaration  */
        | decl_lst      /* global vars, chans */
        | proctype      /* proctype declaration */
        | init          /* init process       */
        | never         /* never claim        */
        | trace         /* event trace        */
        | c_code '{' ... C ... '}'
        | c_decl '{' ... C ... '}'
        | c_state string string [ string ]
        | c_track string string

proctype: [ active ] PROCTYPE name '(' [ decl_lst ]')'
          [ priority ] [ enabler ] '{' sequence '}'

init    : INIT [ priority ] '{' sequence '}'

never   : NEVER '{' sequence '}'

trace   : TRACE '{' sequence '}'

utype   : TYPEDEF name '{' decl_lst '}'

mtype   : MTYPE [ '=' ] '{' name [ ',' name ] * '}'

decl_lst: one_decl [ ';' one_decl ] *

one_decl: [ visible ] typename  ivar [',' ivar ] *

typename: BIT   | BOOL | BYTE  | PID
        | SHORT | INT  | MTYPE | CHAN
        | uname /* user defined type names (see utype) */

active  : ACTIVE [ '[' const ']' ]      /* instantiation */

priority: PRIORITY const        /* simulation priority */

enabler : PROVIDED '(' expr ')' /* execution constraint */

visible : HIDDEN | SHOW

sequence: step [ ';' step ] *

step    : stmnt [ UNLESS stmnt ]
        | decl_lst
        | XR varref [',' varref ] *
        | XS varref [',' varref ] *
```

```
ivar    : name [ '[' const ']' ] [ '=' any_expr | '=' ch_init ]

ch_init : '[' const ']' OF '{' typename [ ',' typename ] * '}'

varref  : name [ '[' any_expr ']' ] [ '.' varref ]

send    : varref '!' send_args          /* normal fifo send */
        | varref '!' '!' send_args      /* sorted send */

receive : varref '?' recv_args          /* normal receive */
        | varref '?' '?' recv_args      /* random receive */
        | varref '?' '<' recv_args '>'  /* poll with side-effect */
        | varref '?' '?' '<' recv_args '>'      /* ditto */

poll    : varref '?' '[' recv_args ']'  /* poll without side-effect */
        | varref '?' '?' '[' recv_args ']'      /* ditto */

send_args: arg_lst | any_expr '(' arg_lst ')'

arg_lst  : any_expr [ ',' any_expr ] *

recv_args: recv_arg [ ',' recv_arg ] *  |  recv_arg '(' recv_args ')'

recv_arg : varref | EVAL '(' varref ')' | [ '-' ] const

assign  : varref '=' any_expr   /* standard assignment */
        | varref '+' '+'        /* increment */
        | varref '-' '-'        /* decrement */

stmnt   : IF options FI         /* selection */
        | DO options OD         /* iteration */
        | ATOMIC '{' sequence '}'       /* atomic sequence */
        | D_STEP '{' sequence '}'       /* deterministic atomic */
        | '{' sequence '}'      /* normal sequence */
        | send
        | receive
        | assign
        | ELSE                  /* used inside options */
        | BREAK                 /* used inside iterations */
        | GOTO name
        | name ':' stmnt        /* labeled statement */
        | PRINT '(' string [ ',' arg_lst ] ')'
        | ASSERT expr
        | expr                  /* condition */
        | c_code [ c_assert ] '{' ... C ...'}'
        | c_expr [ c_assert ] '{' ... C ...'}'

c_assert: '[' ... C ... ']'

options : ':' ':' sequence [ ':' ':' sequence ] *

andor   : '&' '&' | '|' '|'

binarop : '+' | '-' | '*' | '/' | '%' | '&' | '^' | '|'
        | '>' | '<' | '>' '=' | '<' '=' | '=' '=' | '!' '='
        | '<' '<' | '>' '>' | andor
```

```
unarop  : '~' | '-' | '!'

any_expr: '(' any_expr ')'
        | any_expr binarop any_expr
        | unarop any_expr
        | '(' any_expr '-' '>' any_expr ':' any_expr ')'
        | LEN '(' varref ')'    /* nr of messages in chan */
        | poll
        | varref
        | const
        | TIMEOUT
        | NP_                   /* non-progress system state */
        | ENABLED '(' any_expr ')'              /* refers to a pid */
        | PC_VALUE '(' any_expr ')'             /* refers to a pid */
        | name '[' any_expr ']' '@' name        /* refers to a pid */
        | RUN name '(' [ arg_lst ] ')' [ priority ]

expr    : any_expr
        | '(' expr ')'
        | expr andor expr
        | chanpoll '(' varref ')'       /* may not be negated */

chanpoll: FULL | EMPTY | NFULL | NEMPTY

string  : '"' [ any_ascii_char ] * '"'

uname   : name

name    : alpha [ alpha | const | '_' ] *

const   : TRUE | FALSE | SKIP | number [ number ] *

alpha   : 'a' | 'b' | 'c' | 'd' | 'e' | 'f'
        | 'g' | 'h' | 'i' | 'j' | 'k' | 'l'
        | 'm' | 'n' | 'o' | 'p' | 'q' | 'r'
        | 's' | 't' | 'u' | 'v' | 'w' | 'x'
        | 'y' | 'z'
        | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
        | 'G' | 'H' | 'I' | 'J' | 'K' | 'L'
        | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R'
        | 'S' | 'T' | 'U' | 'V' | 'W' | 'X'
        | 'Y' | 'Z'

number  : '0' | '1' | '2' | '3' | '4' | '5'
        | '6' | '7' | '8' | '9'
```

付 録B　OSEK/VDX

B.1　状態遷移図

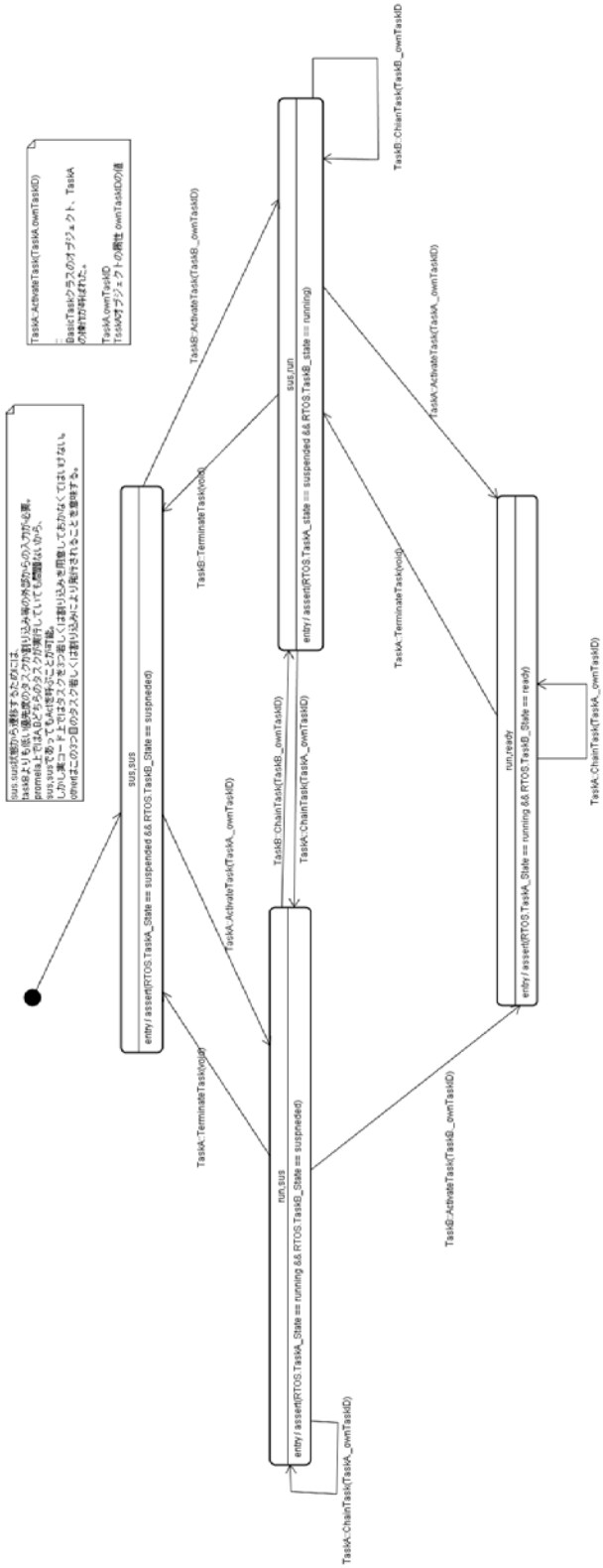

図 B.1: タスク２、資源なしの複合状態

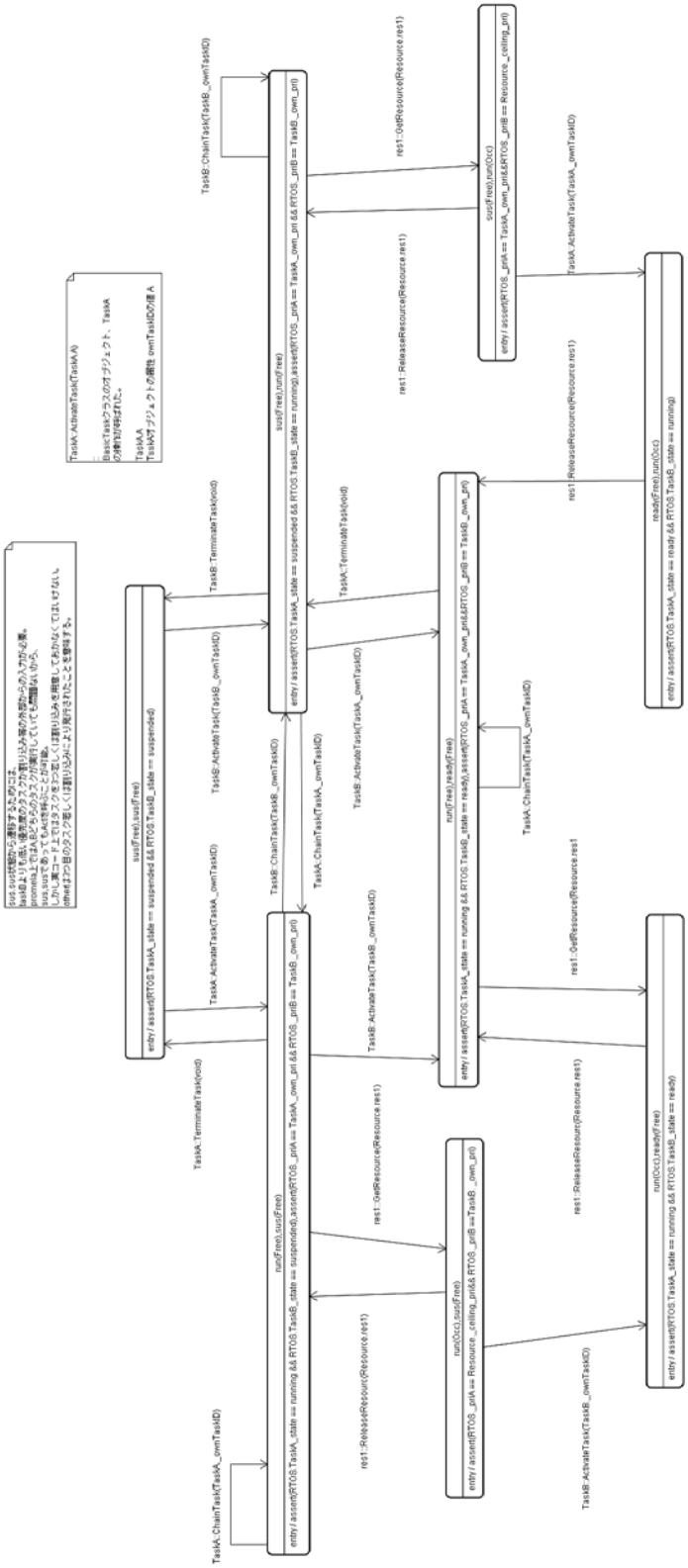

図 B.2: タスク２、資源１の複合状態

B.2 ウィーブ前の promela

List B.1: タスク２つ資源１つのモデル

```
#include "osekoslib.spin"

mtype = {A,B};
mtype = {Resource};
mtype ownTaskIDA,ownTaskIDB;
int _priA,_priB,own_priA,own_priB,ceiling_pri;

proctype task2_resource(){

mtype res1;

susFree_susFree:

  atomic{
    printf("susFree_susFree\n");
    if
      ::ActivateTask(ownTaskIDA)->goto runFree_susFree
      ::ActivateTask(ownTaskIDB)->goto susFree_runFree
    fi
  };

runFree_susFree:
  atomic{
    printf("runFree_susFree\n");
    assert(_priA == 10 && _priB == 5);
    if
      ::TerminateTask(ownTaskIDA)->goto susFree_susFree
      ::ActivateTask(ownTaskIDB)->goto runFree_readyFree
      ::ChainTask(ownTaskIDA,ownTaskIDA)->goto runFree_susFree
      ::ChainTask(ownTaskIDB,ownTaskIDA)->goto susFree_runFree
      ::GetResource(res1,ownTaskIDA)->goto runOcc_susFree
    fi
  };

runFree_readyFree:
  atomic{
    printf("runFree_readyFree\n");
    assert(_priA == 10 && _priB == 5);
    if
      ::TerminateTask(ownTaskIDA)->goto susFree_runFree
      ::ChainTask(ownTaskIDA,ownTaskIDA)->goto runFree_readyFree
      ::GetResource(res1,ownTaskIDA)->goto runOcc_readyFree
    fi
  };

susFree_runFree:
  atomic{
    printf("susFree_runFree\n");
    assert(_priA == 10 && _priB == 5);
    if
      ::TerminateTask(ownTaskIDB)->goto susFree_susFree
      ::ActivateTask(ownTaskIDA)->goto runFree_readyFree
      ::ChainTask(ownTaskIDB,ownTaskIDA)->goto susFree_runFree
      ::ChainTask(ownTaskIDA,ownTaskIDB)->goto runFree_susFree
      ::GetResource(res1,ownTaskIDB)->goto susFree_runOcc
    fi
  };

runOcc_susFree:
  atomic{
    printf("runOcc_susFree\n");
```

```
62      assert(_priA == 20 && _priB == 5);
63      if
64        ::ReleaseResource(res1,ownTaskIDA)->goto runFree_susFree
65        ::ActivateTask(ownTaskIDB)->goto runOcc_readyFree
66      fi
67    };
68
69
70  runOcc_readyFree:
71    atomic{
72      printf("runOcc_readyFree\n");
73      assert(_priA == 20 && _priB == 5);
74      if
75        ::ReleaseResource(res1,ownTaskIDA)->goto runFree_readyFree
76      fi
77    };
78
79  susFree_runOcc:
80    atomic{
81      printf("susFree_runOcc\n");
82      assert(_priA == 10 && _priB == 20);
83      if
84        ::ReleaseResource(res1,ownTaskIDB)->goto susFree_runFree
85        ::ActivateTask(ownTaskIDA)->goto readyFree_runOcc
86      fi
87    };
88
89  readyFree_runOcc:
90    atomic{
91      printf("readyFree_runOcc\n");
92      assert(_priA == 10 && _priB == 20);
93      if
94        ::ReleaseResource(res1,ownTaskIDB)->goto runFree_readyFree
95      fi
96    }
97
98  }
99
100 init{
101   own_priA = 10;
102   own_priB = 5;
103   _priA = own_priA;
104   _priB = own_priB;
105   ceiling_pri = 20;
106
107
108   ownTaskIDA = A;
109   ownTaskIDB = B;
110   run task2_resource()
111 }
```

List B.2: osek のシステムシステムコール (osekoslib.spin)

```
1   int _pri,own_pri;
2   #define ActivateTask(Act)\
3           printm(Act);      \
4           printf(" Activate\n")
5
6   #define TerminateTask(Ter)\
7           printm(Ter);      \
8           printf(" Terminate\n")
9
10
11  #define ChainTask(Act,Ter)\
12          printf("ChainTask\n");\
13          printm(Ter);\
```

```
14          printf(" Terminate\n");\
15          printm(Act);\
16          printf(" Activate\n")
17
18
19  #define GetResource(Res,Tsk)\
20          printm(Res);\
21          printf("Occupied Resource\n");\
22          if\
23          ::Tsk == A->_priA = ceiling_pri\
24          ::Tsk == B->_priB = ceiling_pri\
25          ::else->_pri = ceiling_pri\
26          fi
27
28
29
30  #define ReleaseResource(Res,Tsk)\
31          printm(Res);\
32          printf("Release Resource\n");\
33          if\
34          ::Tsk == A->_priA = own_priA\
35          ::Tsk == B->_priB = own_priB\
36          ::else->_pri = own_pri\
37          fi
```

B.3 ウィーブ後の promela

なお、ウィーバにてウィーブする前に、プリプロセッサにてマクロを展開しているため、osekoslib.spin のマクロが展開されている。

List B.3: ウィーブ後の promela

```
1  int _pri ,own_pri;
2  mtype={A,B}
3  mtype={Resource}
4  int _priA ,_priB,own_priA,own_priB,ceiling_pri;
5  proctype task2_resource() {
6    mtype res1;
7  susFree_susFree:atomic{printf("susFree_susFree\n");
8                         if
9                           ::printm(ownTaskIDA);
10                             printf(" Activate\n");
11                             goto runFree_susFree;
12                          ::printm(ownTaskIDB);
13                             printf(" Activate\n");
14                             goto susFree_runFree;
15                        fi;
16 };
17 runFree_susFree:atomic{printf("runFree_susFree\n");
18                        assert(_priA==10&&_priB==5);
19                        if
20                          ::printm(ownTaskIDA);
21                             printf(" Terminate\n");
22                             goto susFree_susFree;
23                          ::printm(ownTaskIDB);
24                             printf(" Activate\n");
25                             goto runFree_readyFree;
26                          ::printf("ChainTask\n");
27                             printm(ownTaskIDA);
28                             printf(" Terminate\n");
29                             printm(ownTaskIDA);
30                             printf(" Activate\n");
```

```
31                                goto runFree_susFree;
32                             ::printf("ChainTask\n");
33                                printm(ownTaskIDA);
34                                printf(" Terminate\n");
35                                printm(ownTaskIDB);
36                                printf(" Activate\n");
37                                goto susFree_runFree;
38                                /* aspect1 */ fi;
39  };
40  runFree_readyFree:atomic{printf("runFree_readyFree\n");
41                            assert(_priA==10&&_priB==5);
42                            if
43                              ::printm(ownTaskIDA);
44                                 printf(" Terminate\n");
45                                 goto susFree_runFree;
46                              ::printf("ChainTask\n");
47                                 printm(ownTaskIDA);
48                                 printf(" Terminate\n");
49                                 printm(ownTaskIDA);
50                                 printf(" Activate\n");
51                                 goto runFree_readyFree;
52                                 /* aspect2 */ fi;
53  };
54  susFree_runFree:atomic{printf("susFree_runFree\n");
55                          assert(_priA==10&&_priB==5);
56                          if
57                            ::printm(ownTaskIDB);
58                               printf(" Terminate\n");
59                               goto susFree_susFree;
60                            ::printm(ownTaskIDA);
61                               printf(" Activate\n");
62                               goto runFree_readyFree;
63                            ::printf("ChainTask\n");
64                               printm(ownTaskIDA);
65                               printf(" Terminate\n");
66                               printm(ownTaskIDB);
67                               printf(" Activate\n");
68                               goto susFree_runFree;
69                            ::printf("ChainTask\n");
70                               printm(ownTaskIDB);
71                               printf(" Terminate\n");
72                               printm(ownTaskIDA);
73                               printf(" Activate\n");
74                               goto runFree_susFree;
75                               /* aspect3 */ fi;
76  };
77  runOcc_susFree:atomic{printf("runOcc_susFree\n");
78                        assert(_priA==20&&_priB==5);
79                        if
80                          ::printm(res1);
81                             printf("Release Resource\n");
82                             if
83                               ::ownTaskIDA==A;
84                                  _priA=own_priA;
85                               ::ownTaskIDA==B;
86                                  _priB=own_priB;
87                               ::else;
88                                  _pri=own_pri;
89                             fi;
90                             goto runFree_susFree;
91                             /* aspect2 */ fi;
92  };
93  runOcc_readyFree:atomic{printf("runOcc_readyFree\n");
94                          assert(_priA==20&&_priB==5);
95                          if
96                            ::printm(res1);
```

```
97                                  printf("Release Resource\n");
98                                  if
99                                    ::ownTaskIDA==A;
100                                       _priA=own_priA;
101                                   ::ownTaskIDA==B;
102                                       _priB=own_priB;
103                                   ::else;
104                                       _pri=own_pri;
105                                 fi;
106                                 goto runFree_readyFree;
107                           fi;
108 };
109 susFree_runOcc:atomic{printf("susFree_runOcc\n");
110                         assert(_priA==10&&_priB==20);
111                         if
112                           ::printm(res1);
113                               printf("Release Resource\n");
114                               if
115                                 ::ownTaskIDB==A;
116                                     _priA=own_priA;
117                                 ::ownTaskIDB==B;
118                                     _priB=own_priB;
119                                 ::else;
120                                     _pri=own_pri;
121                               fi;
122                               goto susFree_runFree;
123                           ::printm(ownTaskIDA);
124                               printf(" Activate\n");
125                               goto readyFree_runOcc;
126                         fi;
127 };
128 readyFree_runOcc:atomic{printf("readyFree_runOcc\n");
129                           assert(_priA==10&&_priB==20);
130                           if
131                             ::printm(res1);
132                                 printf("Release Resource\n");
133                                 if
134                                   ::ownTaskIDB==A;
135                                       _priA=own_priA;
136                                   ::ownTaskIDB==B;
137                                       _priB=own_priB;
138                                   ::else;
139                                       _pri=own_pri;
140                                 fi;
141                                 goto runFree_readyFree;
142                           fi;
143 };
144 }
145 init {
146   own_priA=10;
147   own_priB=5;
148   _priA=own_priA;
149   _priB=own_priB;
150   ceiling_pri=20;
151   ownTaskIDA=A;
152   ownTaskIDB=B;
153   run task2_resource();
154 }
```