2004年度 修士論文

# 小規模制御系向けLMNtal処理系の設計と実装

提出日: 2005年2月2日
指導　: 上田 和紀 教授

早稲田大学大学院理工学研究科 情報・ネットワーク専攻
学籍番号 : 3603U147-0

矢島 伸吾

## 概 要

階層グラフの書き換えに基づく並行言語モデル LMNtal は，その目的の一つとして「大規模から極小まで様々な計算環境を統一的に扱えるスケーラブルなソフトウェア基盤」となることを掲げている．プログラミング言語としての LMNtal 処理系は現在，スタンドアロン環境に加え分散など大規模計算への対応が進められている．その一方で，小規模計算への対応は全く考えられていない．

小さな計算環境の一つとして，組み込み機器が挙げられる．情報家電などのユビキタス環境をはじめ，組み込み分野の重要性は今後益々高まっていくだろう．そのような組み込み計算環境と LMNtal の関係を明らかにすることは，LMNtal のスケーラビリティを論じる上でも重要である．

本研究では，組み込み環境の特徴を，

- メモリリソースが少ない
- 制御対象（センサ，アクチュエータの入出力）がある
- 処理に優先度がある

ものととらえた上で，

- 少ないメモリリソースにおいて動作する
- LMNtal 上でセンサ，アクチュエータ，タイマが扱える
- 入力，出力の優先処理が可能

という目標を満たす LMNtal 処理系を設計，実装することを通して，LMNtal 上における入出力制御の表現方法や，優先度つき処理の実現方法について考察する．加えて，センサから一定間隔毎に値を読み込むためのタイマ処理，カメラから読み込んだ画像を LMNtal 上で表現し，扱う方法についても考察する．

本研究の成果として，汎用組込み制御システム eyebot 上における LMNtal 処理系の実装を行い，LMNtal プログラムにより各種センサ，モータ，カメラなどを制御することに成功した．また，それらの制御を通常の LMNtal ルール適用処理に優先して行なうことができた．

これにより，LMNtal における制御や優先度の処理の一方式が，それが実際に動作する環境とともに提案された．処理系のプログラムサイズは 54KB に収まり、アトムとリンクのデータ構造は最大 77 倍効率化した。プログラムを記述し，実際に動作させられる環境があることは，今後これらの分野についての議論を深めていく上での助けとなるだろう．今後，LMNtal と組み込み分野の関係を議論する上でのたたき台としての処理系を提供することも，本研究の目的である．

# 目 次

**第 1 章　研究の目的と背景** **1**
1.1　LMNtal の現状と組み込み分野の発展 . . . . . 1
1.2　組み込みへの応用における現 LMNtal 処理系の問題点 . . . . . 1
1.3　組み込み向け LMNtal 処理系の実装 . . . . . 2
1.4　関連研究 . . . . . 3
1.5　本論文の構成 . . . . . 3

**第 2 章　LMNtal 言語解説** **5**
2.1　Atom とリンク . . . . . 5
2.2　ルールと適用 . . . . . 6
2.3　膜と階層化 . . . . . 7
2.4　LMNtal の構文 . . . . . 8
2.5　プロセス文脈 . . . . . 9
2.5.1　プロセス文脈の意味 . . . . . 9
2.5.2　自由リンク制約 . . . . . 10
2.5.3　プロセス文脈の出現規則 . . . . . 10
2.6　ルール変数 . . . . . 11
2.7　型付プロセス文脈 . . . . . 12
2.8　用語集 . . . . . 12

**第 3 章　Java 版 LMNal 処理系概説** **14**
3.1　基本方針 . . . . . 14
3.1.1　中間コード . . . . . 14
3.1.2　実行の流れとスタック . . . . . 16
3.1.3　proxy . . . . . 17

**第 4 章　eyebot 仕様解説** **19**
4.1　eyebot 概要 . . . . . 19
4.2　eyebot 仕様 . . . . . 19
4.3　開発環境 . . . . . 22

**第 5 章　eLMNtal の設計** **24**
5.1 言語の制限 . . . . . 24
5.2 処理の流れ . . . . . 26
5.2.1 タスクのデータ構造と実行 . . . . . 26
5.2.2 全体像 . . . . . 26
5.2.3 バイナリデータの仕様 . . . . . 27
5.2.4 入力データの生成 . . . . . 27
5.2.5 プログラムの実行 . . . . . 30
5.2.6 出力データ . . . . . 31
5.3 制御 . . . . . 31
5.3.1 システムアトム . . . . . 31
5.3.2 システムアトムの意味 . . . . . 34
5.4 実行の順序と優先度 . . . . . 34

**第 6 章　eLMNtal の実装** **36**
6.1 アトム，膜，リンクの管理 . . . . . 36
6.1.1 膜の管理 . . . . . 36
6.1.2 アトムとリンクの表現 . . . . . 37
6.1.3 整数の表現 . . . . . 38
6.1.4 ポインタタグ . . . . . 38
6.1.5 マクロ対ビットフィールド . . . . . 39
6.1.6 アトム管理クラスの実装 . . . . . 39
6.1.7 findatom の実現 . . . . . 40
6.2 ルール適用処理とシステム構成 . . . . . 41
6.2.1 システム構成 . . . . . 41
6.2.2 通常のルール適用の流れ . . . . . 41
6.2.3 コマンド型システムアトムの処理 . . . . . 43
6.2.4 timer . . . . . 44
6.2.5 timer の実装の理由と解説 . . . . . 45
6.2.6 ２つのスタックと生成型システムアトムの処理 . . . . . 45
6.2.7 ルール適用の中断 . . . . . 46
6.2.8 proxy の処理 . . . . . 47
6.3 コンバータ関連の実装 . . . . . 47
6.3.1 整数関係の処理変更 . . . . . 47
6.3.2 生成型システムアトムの関連ルールの取得 . . . . . 47
6.3.3 jump 命令の処理 . . . . . 48
6.4 システムアトム一覧 . . . . . 48

**第 7 章　サンプルプログラム** **51**
7.1 プログラムの実行に用いる機体 . . . . . 51
7.2 tasktest.lmn . . . . . 53
7.3 wind.lmn . . . . . 53
7.4 thermo.lmn . . . . . 55
7.5 redcar.lmn . . . . . 56

**第 8 章　考察・検証** **59**
8.1 アトム，リンク表現のメモリ効率化の検証 . . . . . 59
8.2 制御表現の代案 . . . . . 60
8.3 処理の優先度とリアルタイム性について . . . . . 61

**第 9 章　まとめと今後の課題** **63**
9.1 まとめ . . . . . 63
9.2 今後の予定 . . . . . 64

**謝辞** **66**

**参考文献** **67**

**Appendix.A 中間コード対応** **69**
A.1 Java 版処理系と eLMNtal の中間コード対応表 . . . . . 69

**Appendix.B ソースコード** **73**
B.1 ランタイム . . . . . 73
B.1.1 atom.cpp . . . . . 73
B.1.2 atom.h . . . . . 77
B.1.3 eyebotstub.h . . . . . 78
B.1.4 eyeio.cpp . . . . . 78
B.1.5 eyeio.h . . . . . 79
B.1.6 inst.h . . . . . 79
B.1.7 lmntal.cpp . . . . . 81
B.1.8 lmntal.h . . . . . 84
B.1.9 lmntal_const.h . . . . . 87
B.1.10 mem.cpp . . . . . 88
B.1.11 mem.h . . . . . 93
B.1.12 task.cpp . . . . . 94
B.1.13 task.h . . . . . 111
B.2 コンバータ . . . . . 113
B.2.1 atomid.cpp . . . . . 113

B.2.2 atomid.h . . . . . 114
B.2.3 bin2lmn.cpp . . . . . 115
B.2.4 gental.cpp . . . . . 119
B.2.5 splitlmn.cpp . . . . . 120
B.2.6 template.aid . . . . . 121
B.2.7 translmn.cpp . . . . . 122
B.3 実行用シェルスクリプト . . . . . 130
B.3.1 eyeconv.sh . . . . . 130
B.3.2 runeye.sh . . . . . 131
B.3.3 runpc.sh . . . . . 131

# 図 目 次

2.1 図とテキストによるグラフ構造の表現 . . . . . 5
2.2 Append：グラフ構造に対するルール適用の例 . . . . . 6
2.3 膜によるグラフ構造の階層化とルールの適用 . . . . . 7
2.4 膜から飛び出すリンクが膜の自由リンク . . . . . 8
2.5 データ構造（左）と，それに適用できるルール（右） . . . . . 9
2.6 プロセス文脈の例 . . . . . 11

3.1 スタックの動作 . . . . . 17
3.2 ２つの proxy . . . . . 18
3.3 proxy がない場合 . . . . . 18

4.1 eyebot を用いたロボットの構築例 . . . . . 20
4.2 eyebot 表面 . . . . . 21
4.3 eyebot 裏面 . . . . . 21

5.1 入力データの生成 . . . . . 28
5.2 画像の２次元 grid 表現 . . . . . 33

6.1 膜構造 . . . . . 37
6.2 アトム管理機構のしくみ . . . . . 40
6.3 アトム，リンクのメモリ表現 . . . . . 41
6.4 システム構成 . . . . . 42
6.5 jump 命令による変数番号の付け替え . . . . . 48

7.1 eyebot 二輪走行車横面 . . . . . 51
7.2 eyebot 二輪走行車前面 . . . . . 52
7.3 eyebot 二輪走行車中身 . . . . . 52
7.4 風上に向かって進む車 . . . . . 53
7.5 sharp 製 PSD センサ GP2D12 の出力 (データシートより) . . . . . 54
7.6 温度計回路図 . . . . . 55

# 表 目 次

2.1 LMNtal 構文 . . . . . 8

# 第1章　研究の目的と背景

## 1.1　LMNtalの現状と組み込み分野の発展

階層グラフの書き換えに基づく並行言語モデルLMNtal[1] [2] [3]は，その目的の一つとして「大規模から極小まで様々な計算環境を統一的に扱えるスケーラブルなソフトウェア基盤」となることを掲げている．プログラミング言語としてのLMNtalは，スタンドアロン環境における処理系の構築が完了し，分散環境への対応[8]が進められている最中である．このようにLMNtalの大規模計算への応用が研究されている一方，LMNtalの小規模計算への対応は現在のところ全く考えられていない．

小さな計算環境の一つとして，組み込み機器が挙げられる．ユビキタス環境，情報家電など，「パソコン以外の」「小さな」コンピュータの存在は近年益々身近なものとなり，これからもさらに重要なものとなっていくだろう．これら組み込み計算機の特徴としては，

- メモリリソースが少ない
- 制御対象（センサ，アクチュエータの入出力）がある
- 処理に優先度がある

などが挙げられる．LMNtalにおいてこれらの特徴をどう扱えばよいかは自明でなく，その扱い方について議論，考察することはLMNtalをスケーラブルなソフトウェア基盤足らしめるために重要である．

## 1.2　組み込みへの応用における現LMNtal処理系の問題点

現在のLMNtal処理系[4] [5] [6] [7]はJavaで記述され，その大きさはjarファイルだけで293KBに達する．それに加えてJavaランタイム環境が必要となり，これはメモリリソースの少ない組み込み環境に載せるには大きすぎる．コンパクトなLMNtal処理系を実装しようとした場合，プログラムサイズはどれくらいま

で縮小することが可能なのかを確かめることは，LMNtal の小規模計算環境への適応性を確かめる上で重要である．

また，組み込み機器の主な役割としてセンサなどから入力を読み取り，何らかの計算をして，モータやサーボなどのアクチュエータを動作させるという制御がある．プログラミングの観点から言えば，これらの制御は低レベルなポートアクセスを行なったり，提供されたライブラリ関数を呼んだりすることにより行なわれると思われるが，LMNtal で制御を書く場合にはどのように表現すればよいかはわかっていない．LMNtal におけるセンサやアクチュエータの表現方法には様々な方式が考えられるが，そのなかで軽く，高速で，意味的にも LMNtal とマッチし，処理系の改変を少なくすませられる方式を見つけ出すことが必要である．

優先度については，現在の Java 処理系は「内側の膜を優先して行なう」方式に基づいている．子膜のルールほど優先して実行され，また同一膜内のルール適用の優先度は規定がない（実装上，デフォルトの実行においては最初に書いたルールほど優先される）．しかし，組み込み機器においては優先的に行ないたい処理があることが多い．たとえば「Xmsec たったのでモータの駆動を止めたい」「ボタンが押されたので〜を処理したい」など，タイマ処理やイベント駆動，センサ値による制御分岐などは，通常の（待ち状態時に行なわれるような）処理よりも優先して行ないたい．これらの優先処理を行なう機構をどのように実現するかについて考える．本論文では，リアルタイム処理に関しては深い議論を行なっていないが，この優先度機構に時間の概念を組み合わせることにより，リアルタイム処理の実装につなげていくことが可能であると思われる．

## 1.3 組み込み向け LMNtal 処理系の実装

これらの問題点を議論する為に有効なアプローチの１つは，実際に組み込み環境上で動作する処理系を作成し，それらを作る過程を通して様々な問題点を明らかにするとともに，出来上がった処理系を使って様々な制御プログラムを記述することを通して LMNtal における良い制御の表現方法のあり方を探っていくという方法だろう．

本研究では，小規模な制御システム上で動作する LMNtal 処理系（eLMNtal と命名）を設計，実装することを通じて，LMNtal における制御や優先度のあり方を考えるとともに，今後の議論の礎となることを目指す．また，小規模即ちメモリを食わない LMNtal 処理系の実装方法についても検討する．

小規模な制御システムとして，本研究では汎用組み込み制御システム eyebot [10] [11] を用いた．eyebot には，

- アナログ，デジタル入力，サーボ，モータ，カメラなど，多彩なインターフェースを持っている

- C++で開発が可能で，上記センサなどはすべて提供されたC ライブラリを用いて制御が可能である
- ROM512KB（OS に 128KB, プログラムに 128KB * 3 つ），RAM1MB と組み込みにしては大きなメモリを積んでいる

という特徴があり，組み込み向け LMNtal の開発用機材としては適当である．eyebot への実装を通して，LMNtal を動作させる為に必要となるハードウェアリソースについても推し量ることができるだろう．

## 1.4 関連研究

LMNtal は GHC, KL1 などの並行論理型言語や，CHR などの制約処理言語といった言語研究の流れの中で，Gamma などの多重集合計算や膜による階層化の概念などを加え，それらの計算モデルを包含し，統一的に扱えるような言語として提唱された．Java 版 LMNtal 処理系の構築においては，C 言語による KL1 処理系である KLIC[9] や，CHR の処理系の実装 [20] [21] [22] に影響を受けた．

また，組み込みシステム上への言語処理系の実装を行った研究として，LEGO Mindstorms の制御モジュール RCX [15] 上に Scheme 処理系を載せることで制御プログラムの対話型開発・実行環境を構築した研究 [18] や，RCX 上で動作する Java 処理系の多くの実装がある．

組み込み制御に関する研究は非常に数多く，小規模なメモリ下におけるリアルタイム制御のためのシステム (RTOS など) やアルゴリズムなどが活発に議論されている [17].

## 1.5 本論文の構成

次章以降の本論文の構成は以下のとおりである．

- **第 2 章**
  LMNtal 言語の解説と Java 版処理系について解説する
- **第 3 章**
  LMNtal の Java 版処理系の大まかな処理内用について述べる
- **第 4 章**
  eyebot の仕様や使用例を紹介する
- **第 5 章**
  組み込み向け LMNtal 処理系 eLMNtal の全体設計について解説する

- **第 6 章**
  第 5 章の解説に基づき，eLMNtal の内部実装について詳しく記す
- **第 7 章**
  サンプルプログラムの紹介を通して，eLMNtal におけるプログラムの記法や実現した制御の内容について述べる
- **第 8 章**
  Java 版処理系とのメモリ効率の比較・検証や，制御表現の代案との比較，優先度とリアルタイム性についての考察などについて記す
- **第 9 章**
  まとめと今後の課題について述べる
- **Appendix**
  付録として，Java 版と eLMNtal の中間コードの対応を記し，eLMNtal 関連の全ソースコードを添付する

# 第2章　LMNtal言語解説

本章では，LMNtal(Linked Multisets of Nodes transformation language) 言語仕様などについて軽く触れる．この章の記述は上田の LMNtal 発表論文 [2][3] 及びその後の議論に基づき，私の理解の及ぶ範囲で再構成したものである．しかし，＝やプロセス文脈の解釈など，私の主観を交えた部分もあることを最初に断っておく．

## 2.1　Atom とリンク

Atom は，LMNtal のグラフ構造の基本ノードであり，Atom の名前と 0 個以上のリンクを持つ．リンクは Atom を 1 対 1 接続する．また，Atom のもつリンクには順序があり，例えばリンク X,Y をもつ名前 A の Atom と，リンク Y,X をもつ名前 A の Atom は別物として扱われる．

Atom とリンクによりグラフ構造が形成される．次に例を示す．

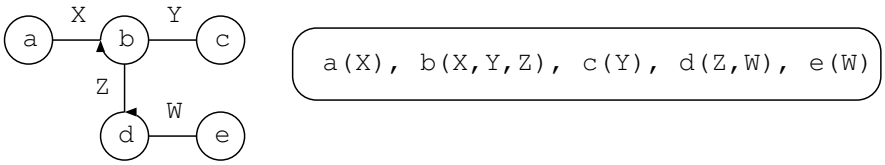


図 2.1: 図とテキストによるグラフ構造の表現

この図は LMNtal のグラフ構造の，構文（2.4 参照）に即したテキスト表現と図示したものの対応を表している．構文がそのままグラフィカルな表現に対応していることが分かるだろう．丸が Atom を，丸の中の文字が Atom 名を，Atom をつなぐ線がリンクを，リンクのそばの文字がリンク変数名を表している．丸についている矢印は，Atom のもつリンクの順序を表している（例えば，b の持つリンクは X,Y,Z の順序であることを示している）．

リンクの実体は論理変数である．しかし，KL1 や CHR のように，リンクが「具体化」されることは無い（リンク X に整数 3 が代入される，といったことは無い）．また，リンクは方向を持たない無方向リンクである．

## 2.2 ルールと適用

ルールは，グラフ構造のある特定の部分を書き換える規則である．例を挙げて説明する．

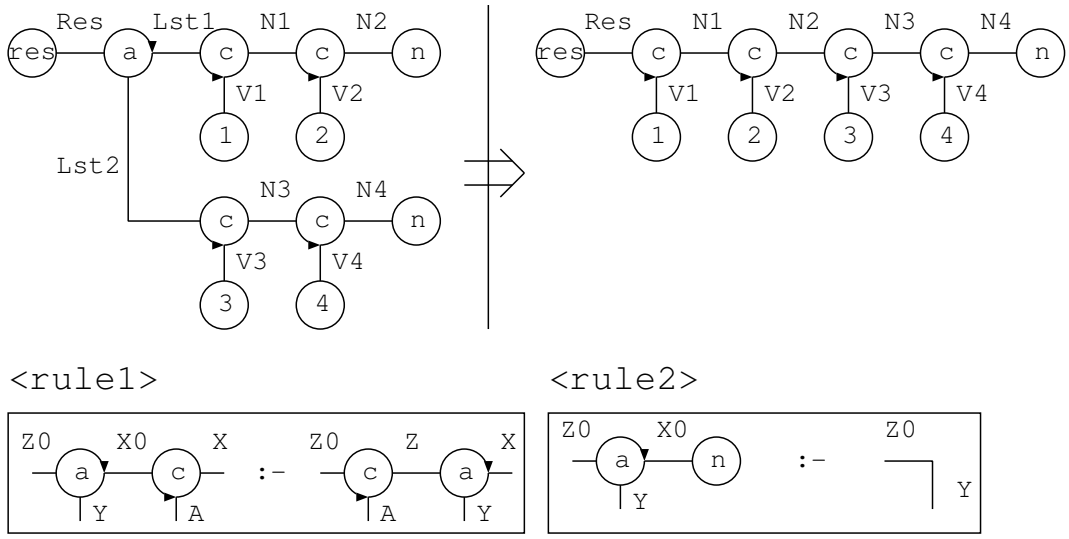


図 2.2: Append：グラフ構造に対するルール適用の例

図 2.2 のテキスト表現を以下に示す．

```
{
    append(Lst1, Lst2, Res), res(Res),
    c(V1, N1, Lst1), c(V2, N2, N1), n(N2), no1(V1), no2(V2),
    c(V3, N3, Lst2), c(V4, N4, N3), n(V4), no3(V3), no4(V4),
    ( append(X0,Y,Z0), c(A,X,X0) :- c(A,Z,Z0), append(X,Y,Z) ),
    ( append(X0,Y,Z0), n(X0) :- Y=Z0 )
}
```

これは，２つのリストを連結する「Append」の例であり，LMNtal の動作の解説によく用いられる．Atom c のリンクによるつながりがリストを表し，1,2,3,4 がその要素である．リストの終端が Atom n を用いて表されている．append は図中では a と略してある．

図の左上のグラフが初期状態であり，これにルールが適用されて右上の状態になる．まず，初期状態のグラフには rule1 が適用できる．rule1 は append と，その１番目のリンクと繋がっている c に対して，リンクの付け替えを行うことによりグラフ構造を変更する．直感的には，グラフ図上において a と c を「入れ替える」処理と言えるだろう．ルール中のリンク変数名は，ルール中で２つのリンク

が繋がっていることを示すために便宜的につけられたものであり,「リンク変数名がちがうからこのルールは適用できない」といった事は無い.

rule1 を 2 回適用すると，Atom append はリンク N2 の間の位置に移動する．このとき，append の 1 番目のリンクと n が接続しているため，rule2 が適用できる．rule2 の適用により append と n が消去されると共に，append の 2 番目と 3 番目のリンクが接続される．これで，グラフの構造は図の右上のようになった.

## 2.3 膜と階層化

膜は，ノードの多重集合を保持する仕組みである．膜は膜をノードとしてもつことができるため，膜を用いて階層的なグラフ構造を実現することができる．また，ルールは膜に所属し，膜内のグラフ構造に対して書換えを行う．以下に例を示す.

```
{ { {a(X)}, b(X,Y), ({a(X)} :- {d(X)}) }, c(Y) }
```

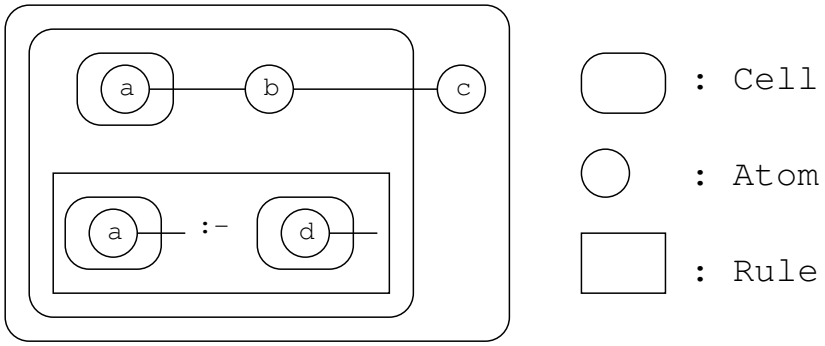


図 2.3: 膜によるグラフ構造の階層化とルールの適用

このように，膜は任意個の膜，Atom，ルールを持つことができる．膜，Atom，ルールをあわせて「プロセス」と呼ぶ．また，膜そのもの（テキスト表現における { } ）のことを compound，膜内にある全てのプロセスを含む膜全体のことを Cell と呼び分ける.

ちなみに，この図のルールは適用可能である．膜内に適用可能なルールが無いとき，その膜を「安定 (stable) 状態」にある膜と呼ぶ．この例では，初期状態においては Atom a のある膜のみ安定状態で，b のある膜，c のある膜はどちらも安定状態でない．ルール適用後は，a,b,c のある膜はどれも安定状態にある．LMNtal プログラムが実行されるということはルールが適用されるということであり，適用できるルールが無くなった（全ての膜が安定状態になった）ら実行終了である.

## 2.4 LMNtalの構文

LMNtal の主な構文は以下のように定義されている．

| プロセス P | | | | プロセステンプレート T | | | |
|---|---|---|---|---|---|---|---|
| P | ::= | 0 | (null) | T | ::= | 0 | (null) |
| | \| | p($\vec{X}$) | (atom) | | \| | p($\vec{X}$) | (atom) |
| | \| | P,P | (composition) | | \| | T,T | (composition) |
| | \| | {P} | (cell) | | \| | {T} | (cell) |
| | \| | {P}/ | (stable cell) | | \| | {T}/ | (stable cell) |
| | \| | (T :- T) | (reaction rule) | | \| | @p | (rule variable) |
| | | | | | \| | $p(FVspec) | (process variable) |

表 2.1: LMNtal 構文

構文中の X はリンク（リンク変数）を表す．$\vec{X}$ は順序のあるリンク X の列を表している．X は大文字から始まる文字列で表される．

p は名前を表す．p は Atom 名のほか，ルール変数名，プロセス文脈名（後述）としても使われる．数も名前の一種である（数も Atom として扱われる）．p は大文字以外から始まる文字列で表される．

ルール内の構文 T は，P とは異なる．まず，ルール中にルールを含むことは出来ない．したがって，実行中に新しいルールを生成することは出来ない．さらに，プロセス文脈とルール変数（後述）をもつ．

リンクは，P 中に１回か２回出現する．１回しか出現しないリンクは，P 中にリンク先の無いリンクである．２回出現するということは，リンクの両端が共に P 中にあるということである．P 中に１回しか出現しないリンクを，P の自由リンクと呼ぶ．

膜{P}の自由リンクとは，P の自由リンクのことであるとする．これは，直感的には膜から飛び出しているリンク，ということができる．

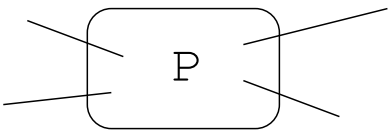


図 2.4: 膜から飛び出すリンクが膜の自由リンク

また，ルールの左辺はリンクの単一化規則 $X = Y$ をもつことができる．これにより，X と Y のリンク先がリンクで接続される[1]

[1]正式な定義では，＝は唯一の予約名であり，データ構造中にいつでも生成可能であるなど，難しい定義がなされている．しかし，私はその有効性について理解していないため，今のところ＝はルール右辺にのみ現れ，ルール適用時に即，リンクの単一化が行われるという前提で話をすすめることにする．

## 2.5 プロセス文脈

### 2.5.1 プロセス文脈の意味

プロセス文脈は,「膜をノードとして扱う」ために必要なものである．ルールにおいて，膜が持つ自由リンクを扱いたい（その自由リンクが具体的に膜内のどのAtom に繋がっているか，といったことを意識せずにルールを書きたい）場合にプロセス文脈が用いられる．次に例を示す.

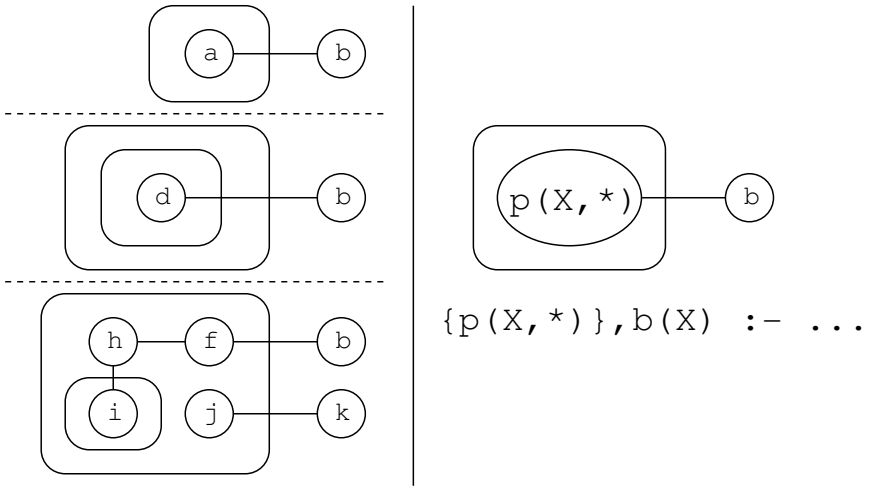


図 2.5: データ構造（左）と，それに適用できるルール（右）

左側の 3 つのデータ構造はどれも，右のルールにマッチする．膜内のデータがどんなに複雑であっても，それをプロセス文脈で置き換えることで，意識せずにルールを記述することができる.

プロセス文脈は,「膜内にある膜，Atom で，ルール中で特定されないもの全て」を表す．例えば,

```
{a(), b(), c(), {d()}}
```

というデータ構造に対して,

```
{a(), $p()} :-
```

が適用されたとき，プロセス文脈`$p()` が表すものは,

```
b(), c(), {d()}
```

である．a() はルール中で特定されているので，プロセス文脈には含まれないことに注意する.

### 2.5.2 自由リンク制約

プロセス文脈$p(FVspec) の FVspec（自由リンク制約）は,「プロセス文脈$p の自由リンク」を表す．膜のもつ自由リンクではなく,「プロセス文脈$p が表す膜, Atom の中で，１回しか現れないリンク」であることに注意する[2]．

FVspec の構文は次のように表される．[ ] はオプショナル，* は０回以上の反復を表す．[3].

[, ]FVspec ::= [, X]*[, *]

X はリンク名であり，*は X で特定されないリンクが 0 個以上存在しても良いことを示す．以下に例を挙げる．

- $p(X,Y)　$p の自由リンクは X,Y のみ.
- $p(X,*)　$p の自由リンクに X が含まれる．その他の自由リンクがあっても良い.
- $p()　$p は自由リンクを持たない.
- $p(*)　$p の自由リンクはあっても無くてもいい.

### 2.5.3 プロセス文脈の出現規則

プロセス文脈は，ルール左辺の膜内に高々１回，出現することができる．例えば，ルールの左辺に３つの膜がある場合，その３つの膜の中に１つずつプロセス文脈を書くことができる．そのとき，それらのプロセス文脈名は同じものであってはならない．図にして説明する．

図 2.6 の左側があらわすデータ構造

```
{ { { f(), a(X) }, b(X,Y), e(W) }, c(Y,Z), d(Z,W) }
```

は，図の右側があらわすルール

```
{ { { $p(), a(X0) }, b(X0,Y0), $q(X) }, c(Y0,Y), $r(X,*) }
```

[2] こう定義することによって，{a(X), {b(X)}} というデータ構造に {$p(X),{b(X)}} がマッチできるようになる．$p には b(X) が含まれないため，リンク X は a(X) がもつ１つだけ，つまり X は$p の自由リンクとなる．

[3] この定義は正式な定義ではない．LMNtal の海外発表論文 [3] では，FVspec は「+で連結された，自由リンクのリスト（空でも良い）で，末尾に不特定多数の自由リンクを表す*をつけることができる，というような意味の自然言語による定義になったようだ．この論文中の定義との差は，, と+の差だけである．

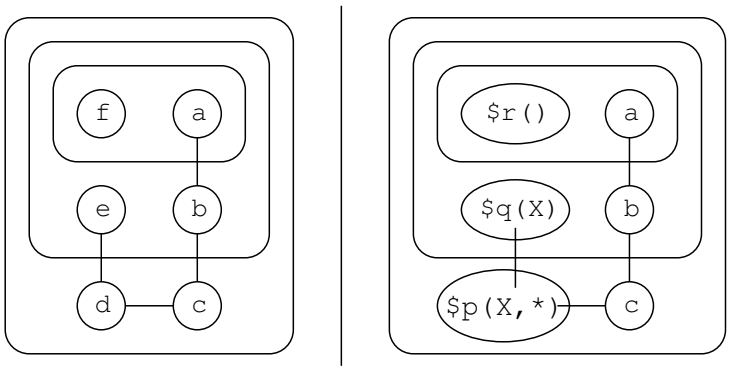


図 2.6: プロセス文脈の例

にマッチする．このとき，$p() は f() を，$q(X) は e(W) を，$r(X,Y) は d(Z,W) を表している．

ルール左辺に現れたプロセス文脈は，必ずルール右辺でも 1 回現れなければならない．これにより,「膜内の全ての膜，Atom を，別の膜の中に移動する」といった処理が可能になる．同時に，自由変数をもつプロセス文脈を消去することは出来ない．ただし，FVspec が空であるプロセス文脈は，ルール右辺で出てこなくても良い．出てこない場合，そのプロセス文脈が表す膜，Atom は全て消去される．

最後に 1 つ注意事項を挙げておく．例えば`{a()}:-...`のように「プロセス文脈の無い膜」を扱うルールは，`{a(),b()}`のように「ルールで記述されない膜や Atom を持つ膜」にはマッチしない．この例では，このルールに適用可能なのは`{a()}`だけで，膜内に b() など余計なものが入る場合にはマッチできない.「余計なものの入っている可能性のある膜」にルールを適用したい場合はプロセス文脈を用いる必要がある．逆に，プロセス文脈のある膜は余計なものが無い膜にもマッチすることができる（プロセス文脈は空でもかまわない）[4].

## 2.6 ルール変数

ルール変数は，膜内の全てのルールにマッチする変数である．プロセス文脈と同じように，ルール左辺にある膜ごとに高々 1 回出現する．ルール変数を用いることで，ルールを膜から膜へ移動することができる．

ルール変数によるルール移送

```
{a(), @p}, {b(), @q}, {c()} :- {a()}, {b()}, {c(), @p, @q}
```

[4]最後にもう 1 つ意見を．プロセス文脈という名前であるが，プロセス文脈にはルールは含まれない．よって，本来はこの名前は「ノード変数」とでもするのが正しいのではないだろうか？

このルールでは，a() のある膜内の全てのルールと，b() のある膜内の全てのルールが，c() のある膜内に移動している．このように，複数の膜のルールが合わさってひとつになるようなルールを書くことができる．しかし，一度一緒になったルールが再度分かれることはない（膜内のルールを２つに分ける方法は無く，ルール変数で全てまとめて扱うことしか出来ないため）．

プロセス文脈のときと同じように，ルール変数の無い膜は，ルールをもつ膜にはマッチできない．逆に，ルール変数のある膜は，ルールを持つ膜にももたない膜にもマッチできる（ルール変数は空でも良い）．また，ルール変数は右辺に出現する必要は無い（膜内のルールを消去するルールがかける）．

## 2.7 型付プロセス文脈

LMNtal の意味や文法を損なうことなく整数などを扱う為に，型付プロセス文脈という概念が提唱された．整数を「整数という型を持ったプロセス文脈」と考えることによって，LMNtal との意味的な整合性をとる．詳しくはここでは述べないが，略記法と併用することによって，Java 版 LMNtal 処理系においてはリンクをあたかも整数のように扱うことができる．例えば，

$$a(2), b(3).a(X), b(Y) :- c(X + Y)$$

というプログラムの実行結果は c(5) となる．

上のプログラムにおいては，まず+(2, 3, c) というアトムが生成され，アトム+ に対してシステムが持つルールが適用されることで c(5) という形に変換された．

このように右辺にアトムを生成する方法のほかに，ガードを用いた計算も行なうことが可能である．たとえば，以下のように記述する．

$$a(2), b(3).a(X), b(Y) :- int(X), int(Y), int(Z), Z = X + Y | c(Z).$$

このように，ガード（ルール適用における条件の記述）の位置に演算を書くことも可能である．上記の例の場合，Z というリンクの先に繋がるプロセス文脈（実体は５という名前の整数アトム）を生成し，それを c とリンクさせている．

ガードなどの記述に関しては，LMNtal の Web ページに若干の記載があるが，使用法の詳しいドキュメントは整備されていない．これもまた早急な整備が求められる．

## 2.8 用語集

最後に，この論文中で今後用いる LMNtal の用語について定義しておく．

- **親膜・子膜・先祖膜・子孫膜**
  膜 A の中に膜 B が入っている時，膜 B は膜 A の子膜と呼び，膜 A は膜 B の親膜と呼ぶ．また，親膜をたどっていったときに現れる膜を先祖膜，子膜をたどっていったときに現れる膜を子孫膜と呼ぶ．特に，膜をｎ回たどった時に現れる膜を,「ｎ代先祖の膜」「ｎ代子孫の膜」と呼ぶ．また，親方向にたどることを「上にたどる」，子方向にたどることを「下にたどる」と呼ぶ．

- **root 膜・膜ツリー**
  グラフ構造全体をあらわす，全ての膜の先祖膜である膜を root 膜と呼ぶ．また，root 膜以下の膜の階層構造（木構造で表される）を膜ツリーと呼ぶ．

- **マルチセット Atom**
  リンクをもたない Atom．または，ルール中においてリンクをたどることで見つけることが出来ない Atom．

- **ルールセット**
  膜内にある全てのルールの多重集合．

- **simple process**
  Atom の正式名称．長いのであまり使われない．

- **compound process**
  Cell の正式名称．長いのであまり使われない．

- **膜・Cell・compound**
  膜のもつ内容物を含む，膜全体を Cell と呼ぶ．また，膜そのもの（構文中における { }）を compound と呼ぶ．日本語で「膜」という場合その両方をさすが，compound の意味で使われることはあまり無いだろう．

- **実行**
  適用できるルールがなくなるまでルールを適用すること．

- **ファンクタ**
  アトムの名前および arity を含んだ名称．アトムの識別に用いる．

# 第3章　Java版LMNal処理系概説

eLMNtal の設計，実装の基本方針は，Java 版 LMNtal 処理系の小規模化と制御処理機構，優先度機構の付加である．本章では，Java 版 LMNtal 処理系の仕様について触れる．

## 3.1　基本方針

LMNtal における実行とは，膜，アトム，リンク，ルールからなるデータ構造に対してルールを適用することである．Java 処理系では膜，アトム，リンク，ルールはそれぞれオブジェクトであり，それらの相互参照によりデータ構造が形成される．ルールとはそのデータ構造を参照，変形させる手続きである．LMNtal 処理系は，適用可能なルールがなくなるまでルールの適用を試みることを繰り返し，最終的なデータ構造を出力する．

### 3.1.1　中間コード

Java 版処理系では，ルールは「中間コード」と呼ばれる命令列で表される．例えば

```
a(X), b(X) :- c
```

というルールがあった場合，このルールは

(1) ルールのある膜から a_1(arity:1, 名前 a のアトム) を１つ見つける (findatom)

(2) a の１番目のリンクのリンク先アトムを取得し，そのアトムの１番目のリンクと繋がっていることを確認する (deref)

(3) 取得したアトムの名前が b_1(arity:2, 名前 b のアトム) であることを確かめる (func)

(4) a_1, b_1 を消去し，c_0 を生成，ルールのある膜に追加する．

という手続きにコンパイルされる．この「データ構造に対する個々の操作」を表すのが中間コードであり，すべてのデータ構造の生成，消去，改変操作は中間コードの解釈を通じて行なわれる．Java 版処理系では，実行時にデバッグオプションをつけることでこの「中間コード」を表示することができる．上記のルールの中間コードとしては，

```
Compiled Rule ( a(X) b(X) :- c )
        --atommatch:
                spec            [2, 3]
        --memmatch:
                spec            [1, 3]
                findatom    [1, 0, a_1]
                deref       [2, 1, 0, 0]
                func            [2, b_1]
                jump            [L100, [0], [1, 2], []]
        --body:L100:
                spec            [3, 4]
                commit          [( a(X) b(X) :- c )]
                dequeueatom     [1]
                dequeueatom     [2]
                removeatom      [1, 0, a_1]
                removeatom      [2, 0, b_1]
                newatom     [3, 0, c_0]
                enqueueatom     [3]
                freeatom        [1]
                freeatom        [2]
                proceed         []
```

という出力が得られる．

より詳しく見ていくと，例えば `deref[2,1,0,0]` という命令は「1 番目の変数に入っているアトムの 0 番目のリンクが，リンク先アトムの 0 番目のリンクとリンクしているならば，２番目の変数に格納する」という命令である．ルール適用時には，アトム，膜，リンクなどを格納しておく変数が用いられ，命令の引数には変数番号，アトム名，リンク番号などが指定される．

LMNtal 処理系で使われる中間命令は数百個ある．この詳細なドキュメントはなく，Java 番処理系のソースコード（非公開）のコメントとして記述されるのみである．処理系の内部仕様を解説したドキュメントの整備が望まれる．

### 3.1.2 実行の流れとスタック

処理系は，まずLMNtalプログラムを読み込み，それをコンパイルする．ここで，初期データ構造は「最初に１度だけ実行される，無から全てのデータ構造を生成するルール」としてコンパイルされる．(LMNtalにおいてはルールがルールを生成することが許されている)．こうして全てがルールにコンパイルされた後，ルール適用が行なわれる．

効率のよいルール適用を行い，正しく「適用できるルールがない」ことを判別する為に，Java処理系では「実行スタック」と「実行膜スタック」という２種類のスタックを用いたルール適用制御方式を取っている．

処理系ごとに１つの「実行膜スタック」を定義し，そこには「未適用ルールがある膜」が積まれる．処理系は，実行膜スタックの先頭の膜（本膜）からアトムを取り出し，その膜のルール適用を試みる．全てのルールの適用に連続して失敗した場合，本膜内のルールに適用可能なものはないとし，実行膜スタックから取り除く．もし子膜を含むルール適用に成功した場合，その子膜を実行膜スタックに追加する．というのが基本方針である．

実行膜スタックにおいては子膜は親膜より上にpushされており，実行膜スタックが空になった（root膜がpopされた）ならルール適用が終了したとみなす，というのが基本方針である．

処理の高速化のために，膜ごとに１つの「実行スタック」を定義する．実行スタックには「その膜のルールの適用がまだ試みられていないアトムへの参照」が積まれ，処理系は「本膜の実行スタックからアトムを取り出し，そのアトムに対し本膜のルールの適用を試みる」処理を行なう．ルール適用に失敗したアトムは，「本膜の親膜の実行スタック」に積まれる．ルール適用に成功したアトムは「そのアトムが所属する膜の実行スタック」に積みなおされる．実行スタックが空になるまでこれを繰り返し，実行スタックが空になったら本膜を実行膜スタックから取り除く．こうすることで無駄なルール適用を省き，処理を高速化することが可能である．Java処理系では現在，最適化オプションをつけることで実行スタックを用いた処理が可能となっているが，デフォルトでは実行膜スタックを用いた処理のみである．

実行膜スタックにおける,「あるルールがこの膜内に適用可能か」を確かめる処理を「膜主導」のルール適用処理と呼び，実行スタックを用いた「このアトムにこの膜のルールが適用可能か」を確かめる処理を「アトム主導」のルール適用処理と呼ぶ．膜主導は，最初にある名前のアトムを膜内から取得する処理（findatom命令）を行なう必要があり，この処理は負荷が大きいため，これを避けるためにアトム主導ルール適用処理が考案された．

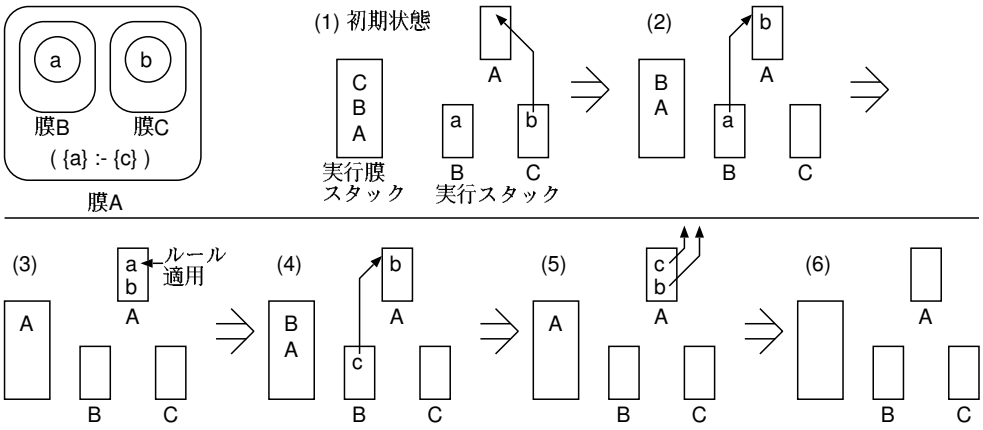


図 3.1: スタックの動作．(1) 膜 C(本膜) の実行スタックから pop されたアトム b にルール適用失敗 (膜 C にはルールがないため)，よって b を膜 A(親膜) の実行スタックに push する．膜 C の実行スタックが空なので膜 C を実行膜スタックから pop．(2) (1) と同様に a を移動し膜 B を pop．(3) a に膜 A のルール ({a}:-{c}) を適用．ルール右辺の c を膜 B の実行スタックに追加し，膜 B を活性化．(4) (1) と同様に c を移動，膜 B を pop．(5) (1) と同様に b,c を pop し，膜 A も pop するが，A の親膜は無いので b,c はどこにも push しない．(6) 実行膜スタックが空なので終了．

### 3.1.3 proxy

Java 処理系は分散など，複数の計算主体において実行されることを念頭において設計されている．その 1 つとして，膜をまたがるリンクの扱いがあげられる．

Java 処理系では，膜をまたがるリンクがある場合，2 リンクを直接リンクさせるのではなく，膜の内側，外側にそれぞれ特殊なアトム inside proxy, outside proxy を中継させる (図 3.2).

膜をまたがるリンクを直接接続した場合，例えば膜 A にあるアトム a(X) をアトム c(X) に書き換えたい場合，X のリンク先の膜 B にあるアトム b(X) のリンクも書き換えなければならない．分散処理などにおいて膜 B が異なる計算主体に存在したとすると，アトム b(X) のリンクを書き換えるためだけに膜 B との通信を行なわなければならない (図 3.3)．対して，proxy を中継させている場合は，通常のアトムのリンク先は必ず同じ膜内（親膜方向に出るアトムなら inside proxy, 子膜に入るアトムなら outside proxy) にあるため，膜内のルール適用を行なう時に他の膜内にアクセスする必要がなくなる．また，proxy はプロセス文脈の引数を扱う時や，プロセス文脈ごと移動する場合のリンクの再接続，リンクのリンク先が同一膜内にあるかの判別，などにも用いられている．

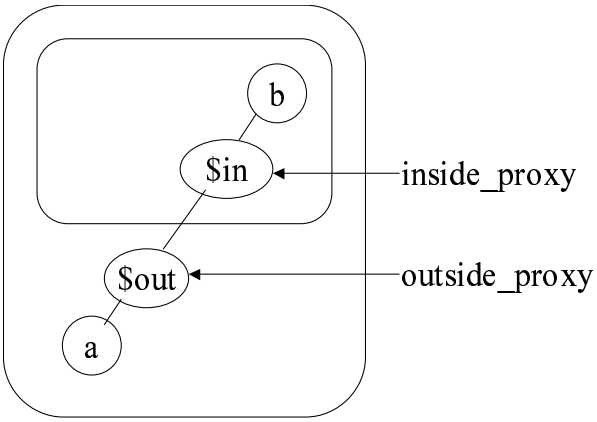


図 3.2: ２つの proxy

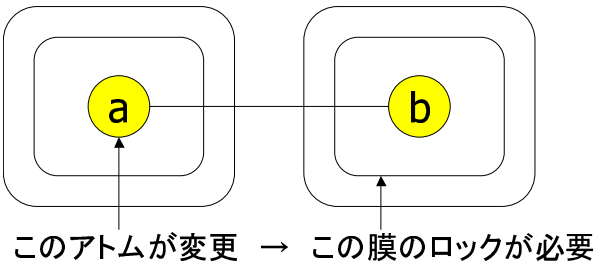


図 3.3: proxy がない場合

# 第4章　eyebot仕様解説

## 4.1　eyebot概要

eyebotは，Western Australia大学において開発され，Joker Robotics社などから販売されている，ロボット制御のためのコントローラーである．eyebotはカメラをはじめとした多彩なインターフェースを備え，二足歩行ロボットやサッカーロボットの構築にも対応しうる (図4.1) スペックを備えるとともに，扱いやすいプログラム開発環境が提供されており，教育，研究用途に適したシステムである．また，組み込み制御機器にしては高速なCPUに大容量のメモリ，デバッグに有益な液晶画面などを搭載しているため，言語処理系などの複雑なシステムも比較的開発，搭載しやすい．本研究では，小規模な組み込み制御システムの一例としてeyebotを取り上げ，eyebot上で動作するLMNtal処理系を実装することを通して組み込み制御とLMNtalの関係について考える．

## 4.2　eyebot仕様

eyebotの主なハードウェア仕様は以下のとおりである (図4.2, 4.3参照)．

- 512KBのROM，1MBのRAM
- 25MHz 32bit CPU (motorola 68332)
- シリアルポート（プログラムのダウンロード，PCとの通信用）
- ２つのモータ出力，エンコーダ入力
- 最大12個のサーボ出力
- ８個のデジタル入力，デジタル出力，アナログ入力
- カメラ (80*60または172*144画素，24bitフルカラー or 8bitグレースケール)
- 液晶画面（128*64画素）
- 無線通信（オプション．PCまたは他のeyebotとの通信）

図 4.1: eyebot を用いたロボットの構築例

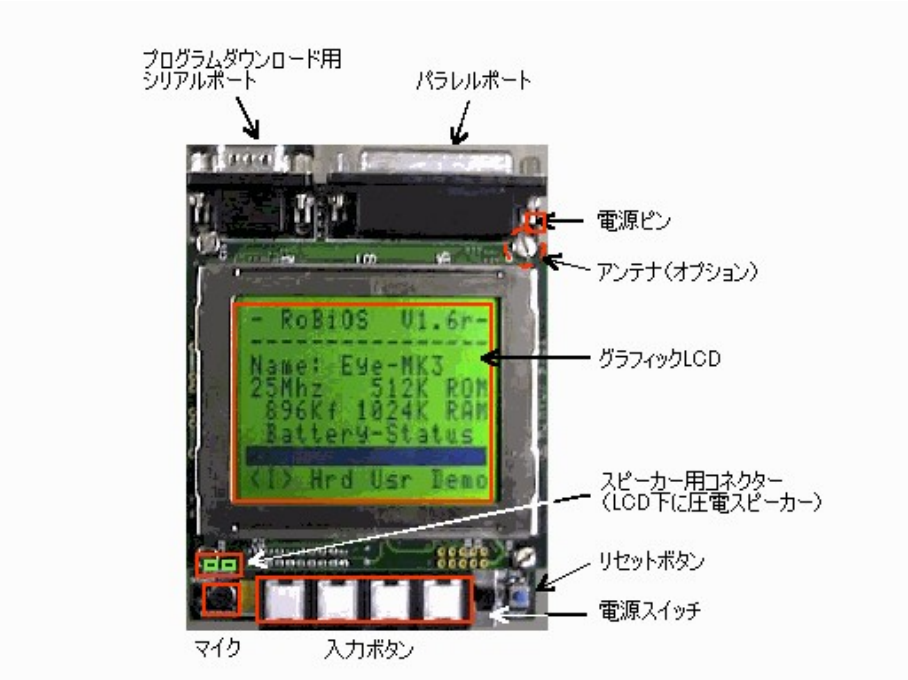


図 4.2: eyebot 表面

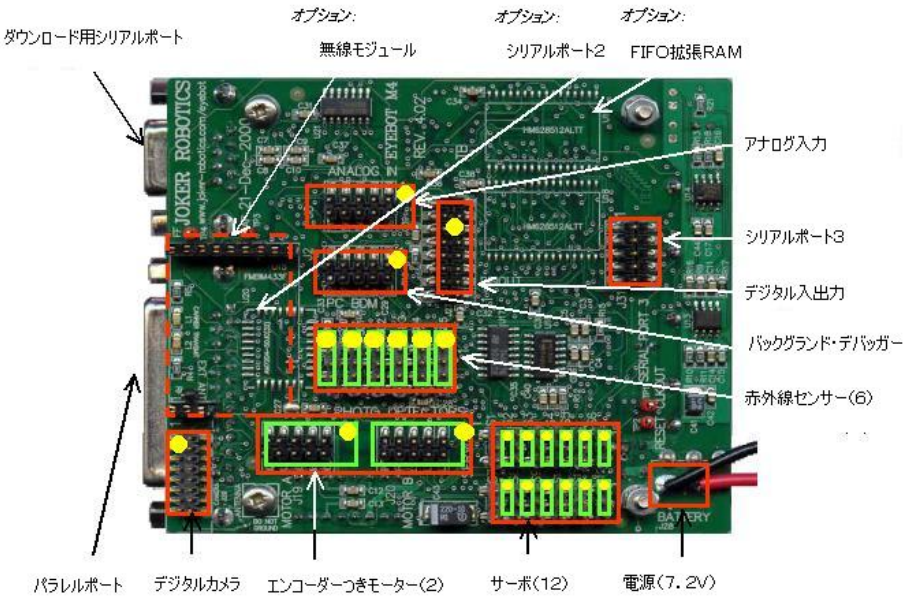


図 4.3: eyebot 裏面

- ４つのボタン
- スピーカ，マイク

これらの多彩なインターフェースが 10cm 四方，高さ 3cm 程の大きさにコンパクトにまとめられている．

ソフトウェア面では，上記インターフェースを簡単に扱えるような C ライブラリ群が提供されており，C 言語からそれらの関数を呼び出すことで，容易に制御プログラムを記述することが可能である．

タイマに関しては，10msec 刻みのタイマが提供されており，タイマカウントの取得，指定秒数の Wait，指定秒数毎の割り込みハンドラの呼び出し機能が提供されている．

また，マルチタスク機能が標準で備わっており，プリエンプティブ，協調モードによるマルチタスクを行なうことができる．タスク間の排他制御のためのセマフォも提供されている．

## 4.3 開発環境

eyebot 用の開発環境として，68k 用 gcc クロスコンパイラ，及び処理用シェルスクリプト，各種ライブラリが提供されている．これらをインストールした後，

- C 言語で記述したプログラムを eyebot 用にクロスコンパイル
- 生成したバイナリをシリアルケーブル経由で eyebot に転送
- eyebot 上でそのプログラムを実行

というのが eyebot のプログラム開発の主な流れである．コマンドラインとしては

```
> gcc68 hoge.c -o hoge.hex  # クロスコンパイル
> dl hoge.hex               # eyebot にシリアル転送
```

のように行なう．gcc68, dl はともに，68k 用 gcc, /dev/ttyS0 などを用いてコンパイル，シリアル転送を行なうシェルスクリプトである．

プログラミング言語としては C 言語のほかに C++言語，アセンブラが使え，それぞれ g++68, gas68 というシェルスクリプトが用意されている．

C 言語におけるプログラム例を以下に示す．

```
#include "eyebot.h"
int main(){
LCDPrintf("hello world!");
return 0;
}
```

このように，eyebot.h をインクルードしている以外は，通常の C 言語と全く同じように記述することができる．eyebot.h 以下には各種制御用ライブラリの定義や定数などが記されており，上記の LCDPrintf（eyebot の液晶上に文字を表示する printf）のように多くのライブラリ関数を使用することが可能である．

また，上記では Linux 版について説明したが，Windows 版の処理系も提供されている．Windows 版においても全く同じように開発を行なうことが可能である．

eyebot は PC からダウンロードしたプログラムのセーブ，ロードが可能である．512KB ある ROM のうち，128KB はシステム (OS) 領域として使われており，残りの 384KB は 128KB ずつ 3 つの領域に分割され，それぞれの領域にダウンロードしたプログラムを焼きこむことが可能である．それらのプログラムはロードして実行することが可能である．

ここで重要なのは，プログラムの静的な最大サイズは 128KB までであること，システムの処理（eyebot ライブラリから呼び出されるルーチンを含む）は 128KB のシステム領域に含まれており，ライブラリ関数を呼んでもそれほどプログラムサイズは変化しないことである．

eyebot の仕様，ライブラリ関数などについては，オンラインドキュメント [11] が整備されているので，そちらもあわせて参照して欲しい．

# 第5章　eLMNtalの設計

本章では，eyebot 上で動作する LMNtal 処理系 eLMNtal の設計方針について記す.

## 5.1　言語の制限

LMNtal の全ての機能を実装するのは，いたずらに処理速度を下げ，処理系のサイズを大きくするだけである．低速な CPU，少ないメモリであることが多い組み込みシステム上で動作させる為には，言語仕様を限定することで高速かつコンパクトな処理系を構築する必要がある.

eLMNtal の設計にあたっては，言語仕様に以下のような制限を設けた.

- root 膜は 1 個以上の子膜（タスク膜と呼ぶ）を持つ．アトム，ルールは持たない.
- タスク膜内には子膜，アトム，ルールが記述できる．また，タスク膜の子膜内にルールは記述できない．タスク間をまたがるリンクも記述できない.
- ルール中にルール変数は（タスク膜の子膜にはルールがないので）書けない.
- プロセス文脈は，必ず両辺で 1 回ずつ出現する（線形である）.
- プロセス文脈の引数は指定できない．必ず$p という引数指定のない形で記述する.
- 型付きプロセス文脈は整数のみ許可する．演算は全てガード中で行なう．また，整数値の有効範囲は符号付 30bit とする.
- 整数の findatom は許さない．つまり 2(2) といった構造は定義できない.

root 膜内に複数の（ルールを持つ）膜を設け，それらを個別に分離し独立性のあるタスクとして扱うことで，マルチタスクを実現した．各タスク膜のルールの適用を平等に行なうことで，擬似的にタスクの並行動作を行なわせるのが狙いである.

また，タスクの子膜内のルール記述を禁止することで，煩雑な実行膜スタック，実行スタックを消去した．タスク膜直下にしかルールがないため，タスク膜のルールを１つずつ適用し（膜主導），全てのルール適用に失敗すればその時点でそのタスクの実行は終了となる．全てのタスク膜が終了した時点でプログラムは停止する．

プロセス文脈のコピー，複製はそれぞれ copymem, dropmem という中間命令を実装すれば可能である．また，プロセス文脈の引数については notfunc 命令を実装すれば可能かもしれないが，現在未実装である．引数無しのプロセス文脈であってもかなりの記述力があるため，組み込み用としては十分であろうと判断した．また，natoms(膜内のアトムの個数を返す）は実装してあるため，$p[] という記述を許可することは可能である．

これだけの制限を設けても膜の記述を可能とした（タスク膜を flat LMNtal にしなかった）のは，膜を記述可能にすることで複雑なデータ構造を記述可能としたかったからである．

型付きプロセス文脈については，整数のみ許可する．演算は必ずガード中で行なわれ，ルール右辺に+などは記述できない．Java 版処理系では右辺の+などはシステムの組み込みルールにより処理されるが，eLMNtal はその機構を省いた．また，有効範囲は後述するアトム保持機構のために符号付 30bit とする．浮動小数点もアトム保持機構との兼ね合いから省略した．

これらの制限によって，eLMNtal 用 LMNtal プログラムは以下のような形式で書かれる．

```
{
        a, b, c(X), {d(X)}, {{e}}. # タスク０のデータ構造
        c(X), {d(X)} :- f. # タスク０のルール群
        a, b :- g.
}{
        a(2), a(3), a(5).
        a(X), a(Y) :- int(X), int(Y), int(Z), Z = X + Y | a(Z).
}{
        ...
}
...
```

## 5.2 処理の流れ

### 5.2.1 タスクのデータ構造と実行

タスクは，大まかに以下のデータを保持する．

- タスク内のアトム
- タスク内の膜構造
- ルール

これらの詳しい実装は次章で解説する．タスク，アトム管理，膜管理はそれぞれクラスで実装される．

タスクはメソッド react を持つ．react は１つのルールのルール適用を行なうメソッドである．react は前回成功終了したなら最初のルールを適用し，前回失敗ならその次のルールを適用する．未適用のルールがもうないなら stable（安定）フラグを立てる．メインループは各タスクの react を１回ずつ順に実行することで擬似的な並行動作を実現し，全てのタスクが stable になるまでそれを続ける．

### 5.2.2 全体像

プログラミングからコンパイル，実行までの流れを解説する．実行の流れは以下のようになる．

(1) 前項のようなデータ構造保持機構，ルール適用機構をもつ，eyebot 上で動作するプログラム eLMNtal を実装し，eyebot 上に転送し，実行しておく．

(2) シリアル転送により，LMNtal プログラムをコンパイルし，バイナリデータ化したもの（初期化ルールと通常のルール）を eyebot に送る．eyebot 側では eLMNtal がバイナリデータを読み込み，ルール領域に保存する．

(3) 初期化ルールを実行，ついで react の呼び出しを開始する．

(4) 実行が終了したら，タスクごとのアトムデータ，膜データを PC にシリアル転送する．

(5) PC はアトムデータ，膜データから LMNtal のデータ構造を再現して表示する．

### 5.2.3 バイナリデータの仕様

バイナリデータとして，以下のようなデータが eyebot に渡される．

- タスクの個数 (1 word)
- N0: タスク０の初期化ルールのデータ長 (1 word)
- 初期化ルールのデータ列（N0 word)
- N1: タスク 0 の１番目のルールのデータ長 (1 word)
- １番目のルールのデータ列 (N1 word)
- ...

データ長 0 は終端を表す．ルールのデータ列は，中間命令列をバイナリに落としたものである．

### 5.2.4 入力データの生成

入力データの生成は，図 5.1 のような流れで行なわれる．この一連の実行はシェルスクリプト [19] で記述され、ユーザは

```
> eyeconv.sh filename[.lmn]
```

のように容易に行うことができる。
この流れの中の各プログラムで行なわれている処理は以下のとおりである．

- **splitlmn**
  LMNtal プログラムにおける，各タスク内の LMNtal 記述を temp_taskXX.lmn という名前のファイルとして生成する．
- **lmntal.jar**
  Java 版 LMNtal 処理系本体である．temp_taskXX.lmn を Java 版処理系でデバッグオプションをつけてコンパイルし，中間コードを吐かせている．
- **translmn**
  Java 版処理系が吐いた中間コードを字句解析及び変換し，ssv テキストデータ（ただの整数値の羅列）に変換し，temp_taskXX.tmp という名前のファイルに落とす．
- **gental**
  temp_taskXX.tmp を順に読み込み，一つのバイナリデータとして結合する．

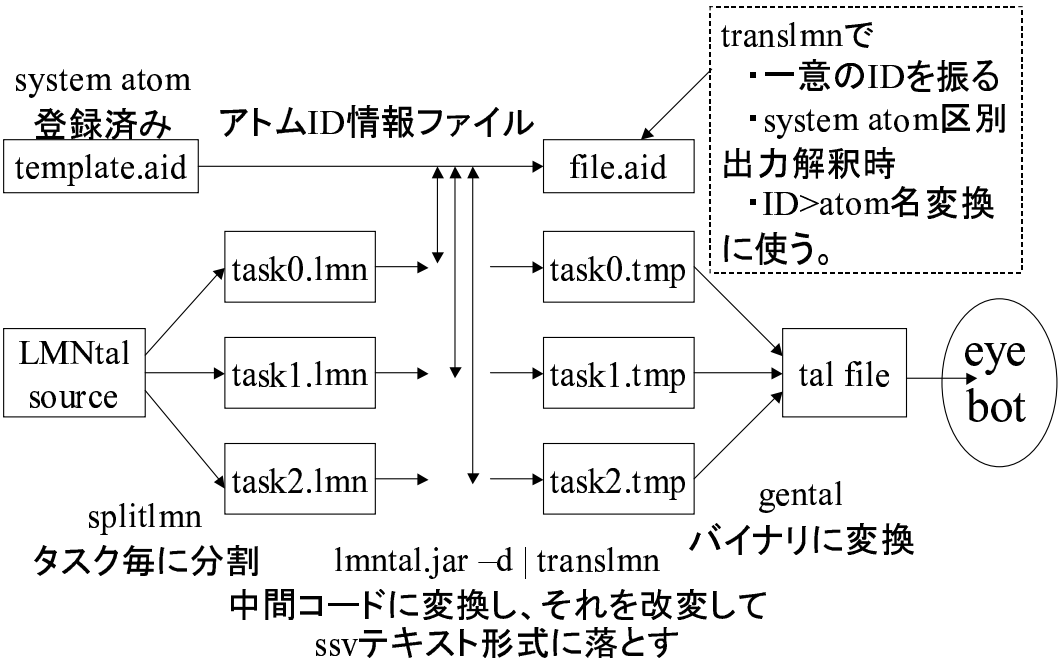


図 5.1: 入力データの生成

上記のように，LMNtal プログラムの字句解析，中間コードへの変換は Java 版 LMNtal 処理系を利用している．これは，LMNtal プログラムの字句解析は非常に複雑な処理であり，実装，改変を行なうのは非常に困難である為であるが，この利用により proxy を使わざるをえないなどの問題も発生する．

中間コードと.tmp ファイルの中身の例を以下に示す．

元 LMNtal ファイル

```
{ a(X), b(X) :- c. }
```

中間コード

```
spec            [1, 3]
findatom    [1, 0, a_1]
deref       [2, 1, 0, 0]
func            [2, b_1]
jump            [L100, [0], [1, 2], []]
spec            [3, 4]
commit          [( a(X) b(X) :- c )]
dequeueatom     [1]
dequeueatom     [2]
removeatom      [1, 0, a_1]
removeatom      [2, 0, b_1]
newatom     [3, 0, c_0]
enqueueatom     [3]
freeatom        [1]
freeatom        [2]
proceed         []
```

.tmp (変換されたテキストデータ)

```
28
4 1 0 65539
1 2 1 0 0
20 2 65541
201
30 1 0
30 2 0
31 3 0 7
35 1
35 2
204
```

template.aid はアトムのファンクタを記してあるファイルである．eLMNtal ではファンクタは整数値 ID（必ず奇数）で管理される．プログラムごとに.aid ファイルを複製し，translmn はその aid ファイルにプログラム固有のアトム名を追加していくことで，ID の管理を行なっている．template.aid に最初から含まれているのは組み込みのシステムアトム名であり，translmn は中間コード中に含まれるアトム名がシステムアトム名であればそのファンクタ ID を用い，未登録のアトムであれば新しいファンクタ ID を割り当てる．割り当てたファンクタ ID は aid ファイルにも追加され，以降のタスクの変換時にも用いられる．こうすることで

全タスクを通して同じアトム名のアトムには同じファンクタ ID を振ることができ，これは今後，タスク間の複雑なデータ通信を実装する場合に有益である．

aid ファイルの中身の例を以下に示す.

```
9
pix_7 70001
$in_2 2ffff
$out_2 2fffd
analog0_1 1e001
...
dig_outall_1 1f033
picture_remove_1 1f035
a_1 10003
b_1 10005
c_0 7
```

template.aid は巻末付録に記載してある．

### 5.2.5 プログラムの実行

前項で示したように eyeconv.sh を用いてコンパイルを行うと、(filename).tar というバイナリファイルが生成される。これをシリアルケーブル経由で eyebot に転送する。eyebot 側では eLMNtal プログラムをあらかじめ実行しておき、eLMNtal は実行の最初でシリアルケーブルからのプログラムの読み込みを行う．

eyebot へのプログラム転送はシェルスクリプト runeye.sh を用いて

```
> runpc.sh filename[.tal]
```

のように行う。

なお、eLMNtal はデバッグ用途のため，入出力デバイスやタイマなど eyebot の機能を用いない LMNtal プログラムは PC 上においても実行可能である．eLMNtal のソースコードは，コンパイル時に PC を define することで PC 用のプログラムとしてコンパイル可能である．

PC での実行も同様に

```
> runpc.sh filename[.tal]
```

で実行可能である。

### 5.2.6 出力データ

eLMNtal はルール適用終了後，タスクごとのアトム構造，膜構造をシリアルケーブル経由で出力する．アトムのデータとして出力されるのは，アトムの存在するアドレス，アトム情報（ファンクタと膜 ID），リンク内容（リンク先アトムのアドレスとリンク番号）である．

プログラム bin2lmn はそれらの情報を読み取り，アドレス値をもとにリンク構造を再現し，LMNtal プログラムの形に変換する．また aid ファイルを参照して，アトムのファンクタ値からアトム名を割り出し，ソースプログラムとの整合性を取る．bin2lmn は略記などを全く用いない LMNtal プログラムを出力するので，出力結果を Java 版処理系に通すことで読みやすい出力結果に変換している．

## 5.3 制御

### 5.3.1 システムアトム

LMNtal 上において制御を扱うために，本処理系では「システムにあらかじめ特別なアトム名を定義しておき，その名前のアトムの生成時に特別な処理（制御処理）を行なう」という方式を取った．この方式は単純であるため，高速であるだけでなく，コンパイラが生成した中間コードにほとんど手を加えることなく実現可能であるというメリットがある．これは，コンパイル部分に Java 版処理系を利用するために重要なだけでなく，今回提唱する方式が容易に Java 版処理系にも応用，実装可能であることを意味している．

この「システム組み込みのアトム」をシステムアトムと呼ぶことにする．システムアトムを役割上，生成型システムアトムとコマンド型システムアトムの 2 種類に区別して考える．それらの特徴と役割は，

- **生成型**
  主にセンサなどの入力デバイスから読み込んだ値を保持するためのアトム．例えば analog1(N) など（N:整数）．コマンド型システムアトムにより生成される．

- **コマンド型**
  このアトムが生成された時，システムは特別な処理（生成型システムアトムの生成や，モータなどの出力デバイスへの出力など）を行なう．コマンド型システムアトム（とそれとリンクするアトム）は生成後すぐに消去される．

である．以下に使用例を述べる．

**get**

コマンド型システムアトム get は，生成型システムアトムの即時生成を行なう．get と生成型システムアトムがリンクする形で必ず生成される．

```
get(analog1).
```

このような構造が生成された時，この構造は生成後（生成を行なったルールの適用が終わった直後）アナログセンサ１から値 M を読み，analog1(M) というアトムを生成する．元の構造 get(analog1) は消去される．

これは，見た目としては get が（アナログセンサ１から読み込んだ）整数値に置き換わったと考えることができるだろう．

**timer**

一定時間毎にセンサの値をチェックしたい，などの用途のために，タイマが必要である．LMNtal におけるタイマ処理は,「生成型システムアトムを一定時間毎に生成する」という処理で実現することにした．それを行なうタイマ機構に情報を登録するには，コマンド型システムアトム timer を用いる．書き方は以下のように行なう．

```
analog1(timer(50))
```

timer の第１リンクには生成間隔を表す整数値，第２リンクには生成型システムアトムが繋がる．この構造が生成された場合，直ちに消去され，生成型システムアトムの functor と生成間隔がタイマ機構に登録される．指定された生成間隔ごとに，これが登録されたタスクに生成型システムアトムが（この場合はアナログセンサ１から読み込んだ値とともに）生成される．

見た目としては，timer(50) が整数値に置き換わった生成型システムアトムが一定時間毎に生成される，と見ることができるだろう．

### タスク間通信

タスク間で情報をやり取りするために，コマンド型システムアトム send, 生成型システムアトム recv を用いる．

タスクには，プログラムに記述された順に 0,1,2... と ID がふられているとする．send,recv はタスク ID と送受信する整数値をリンクに持ち，たとえばタスク 0 に send(1, 50) が生成された場合，直ちに消去され，タスク 1 に recv(0, 50) が生成される．recv のもつ 0 は send が生成されたタスクの ID である．

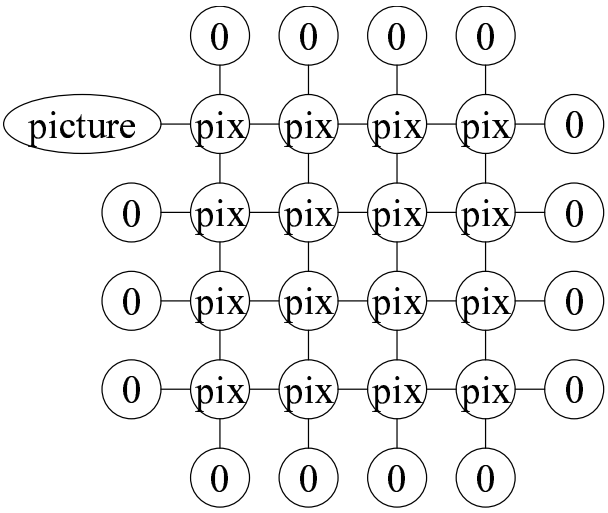


図 5.2: 画像の 2 次元 grid 表現

### 画像の扱い

eyebot にはカメラが付属しており，フルカラーの画像を取得することができる．カメラも生成型システムアトム及びそれとリンクするデータの形式で表すことにした．取得した画像データは 2 次元のグリッドで表現し，その各画素は以下のアトムで表現される．

```
pix(PX, PY, NX, NY, R, G, B)
```

PX, PY, NX, NY はそれぞれ，上下左右画素とリンクしている．端の画素の場合は整数値 0 とリンクする．R, G, B はその画素の RGB 値を表す整数値である．左上画素の PX は，生成型システムアトム picture とリンクする．他の生成型システムアトムと同様に，picture(get) といった構造を生成することで，図 5.2 のようなアトム構造を得ることができる．なお，この図は省略してあるが，実際には縦 60，横 80 個の画素からなる grid が生成される．(低解像度での画像取得にのみ対応している為，60*80 である)

画像処理では for ループによって順に画素を見ていくだけでなく、輪郭抽出や差分取得など上下左右の隣接画素を任意に参照したいことが多い [16]．そのような用途において grid による表現は記述性、高速性の両面において有理である。

他にも，画素をマルチセットとして管理し，それらを膜で囲むなどの仕様も検討したが，そのように表現した場合，目的の画素を 60*80 の 4800 個のアトムから毎回 findatom をしなくてはならない．ルールに複数の画素が含まれる場合は計算量は指数的に増加する．あるアトムを起点として，そのアトムからリンクする画素を指定することができる本方式は，計算量を抑える為に有効である．ちなみに，eLMNtal では pix に対する findatom 命令はあえて必ず失敗するように実

装してある．

また，このように 2 次元の grid をアトムとして扱うとデータ構造が複雑，巨大になるため，LMNtal における 2 次元配列の高速，小規模な実装方法が現在研究されているが，まだ良い実装は発見されていない．

### 5.3.2 システムアトムの意味

システムアトムは「特別な処理を行なう」が，これは LMNtal のモデルとしてはどのような意味になるだろうか．例えば analog1(get) は以下のように解釈することで，LMNtal の意味を損なわない実装であると捉えることができるだろう．

- root 膜に，アナログセンサ 1 を表すアトム dev_analog1(N) がある．N はリアルタイムに更新されるセンサ値である．
- root 膜には以下のルールがかかれている．

```
{$p, analog1(get)}, dev_analog1(N) :- int(N) |
{$p, analog1(N)}, dev_analog1(N).
```

このように，アナログセンサ本体を表すアトムと，タスクにコマンド型システムアトムが生成された場合に適用されるルールが root 膜に存在すると考えれば，意味的にも整合性は取れている実装ということができる．

同様に，タイマは以下のように考えられる．

```
{analog1(timer(50)}.
{$p, analog1(timer(X))} :- {$p, timer_analog1(X)}.
{$p, timer_analog1(X)}, dev_analog1(N) :-
        int(N), （前回の適用から X*10msec たっている） |
        {$p, timer_analog1(X), analog1(N)}, dev_analog1(N).
```

ガードは LMNtal の明確な意味は定義されておらず，このような条件でも書けるとすれば，このように解釈することで timer の意味付けができると考えられる．

## 5.4 実行の順序と優先度

システムアトムに対するルール適用は，通常のルール適用よりも優先して実行する．現在は未実装であるが，ユーザがシステムアトム（優先的に処理されるアトム）を定義することによって，複雑な一連のルール適用を優先的に行なうこと

も可能となるだろう．一連の処理の優先実行を行なうことができれば，あとはどのタスクを優先的に react させるかを決定する機構を実装すればリアルタイム処理に近い処理が実現可能となるだろう.（リアルタイム処理の為にはルールの適用にかかる時間など，更なる情報が必要となる）.

システムアトムを高速に処理する為に，以下のような処理を行なっている.

- コマンド型システムアトムは，newatom される時に保持され，ルール適用終了後に直ちに解釈される.
- 生成型システムアトムは，タスクの実行アトムスタックに積まれ，実行アトムスタックが空でないタスクが 1 つでもある場合，メインループは実行アトムスタックに積まれた生成型システムアトムに対し（通常のルール適用に優先して）ルール適用を行う.
- 生成型システムアトムに対するルール適用において，ルールの先頭の命令が，その生成型システムアトムの findatom である場合，変数にそのアトムを入れ，findatom の次の命令からルール適用処理を行なう（これによって，擬似的なアトム主導ルール適用が行なわれる．また，この処理が行なわれるように，生成型システムアトムはルールの先頭に書くことが推奨される）.
- 生成型システムアトムを含むルールは，あらかじめタスクごと，生成型システムアトムごとに配列として保持しておき，適用試行が必要なルールがわかるようにする.

これにより，システムアトムの処理は通常のルール適用に対してかなりの高速化が見込まれる.

# 第6章　eLMNtalの実装

本章では，eLMNtal の実装について，特にデータ保持機構，およびシステムアトムの処理方法，タイマの実装などについて解説を行なう．

## 6.1　アトム，膜，リンクの管理

### 6.1.1　膜の管理

Java 版では膜オブジェクトは atom の生成，消去その他様々な処理を担う重要かつ大きなクラスだったが，eLMNtal ではただのツリー構造である．膜は膜構造体の配列として表現され，膜構造体 (2 word) の中には以下の情報が入る (図 6.3 参照)

- 親膜の index 値 (8bit)
- 次の兄弟膜への index 値 (8bit)
- 前の兄弟膜への index 値 (8bit)
- 代表子膜の index 値 (8bit)
- 膜内に含まれるアトムの個数 (16bit)

親兄弟，子膜は，配列の index 値で表される．この値はアトムのもつ膜 ID としても使われている．この情報による膜の相互参照により，以下の図 6.1 のようなデータ構造が形作られる．

膜を 8bit の index で表現することで，アトム情報の領域も削減することができた．また，8bit のため膜配列の上限は 255 個までである（膜 ID:0 は空き領域を表す）．

なお，膜管理機構および以下に述べるアトム管理機構はタスクごとに１つ生成される．

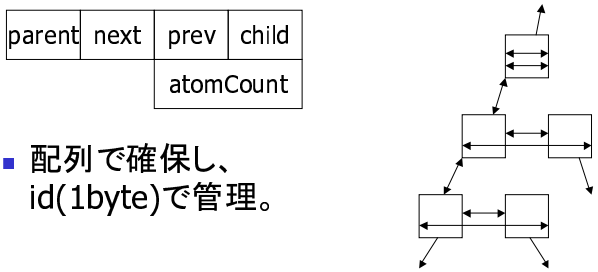


図 6.1: 膜構造

### 6.1.2 アトムとリンクの表現

Java 版処理系では，アトム，リンクはオブジェクトで表現され，ファンクタも文字列が用いられていた．これではメモリの使用量が多すぎるため，eLMNtal ではアトムやリンクをできるだけ少ないメモリで表現することを考えた．

eLMNtal では，N-arity のアトムを「1 word(32bitCPU なので 4byte) のアトム情報+ N word のリンク情報」の N+1 word で管理する．アトムが保持すべき情報としては，functor(アトム名と arity)，所属する膜があるが，これを 1 word で管理する為に 1 word の領域を分割して，ビット単位で情報管理を行なう．

1 word のアトム領域には以下の情報が格納される．

- 所属する膜の ID (8bit)
- アトムの functor（24bit)
  - アトムの arity(8bit)
  - アトムの識別 ID（16bit)

ここで，アトムの識別 ID は必ず奇数であるという制限を設ける．これによってアトム情報領域の最下位ビットは必ず 1 となる．これは findatom の実装（後述）において使われる．

同様に，リンク毎に 1 word のリンク領域には以下の情報が格納される．

- リンク先アトムのリンク番号 (8bit)
- リンク先アトムのアドレス (24bit)

eLMNtal における「アトムへの参照」は,「atom のあるアドレスのアドレス値」として表現する．これは，アドレスなどを含めた様々な値が WORD 型の変数の中に入れられることが多いためである．アトムにアクセスしたい場合は，アドレス値をポインタ型にキャストして用いる．

eyebot のメモリアドレスは 24bit で表される．そこで eLMNtal では，1word の領域にアドレス値を格納した時に余る上位 8bit の領域に，リンク先アトムの何番目のリンクとリンクしているか，という情報を格納する．この情報は deref 命令などで使われる．

### 6.1.3 整数の表現

Java 版処理系においては，整数は functor として整数値を持つアトムとして表現される．しかし，制御の場面などにおいては整数を用いる機会が多く，また 1 word の整数値のために 1 word のリンク (双方向)，整数アトムのアトム情報領域を確保するのは無駄であることから，リンク領域に整数値を直接入れる方式をとった．この方式の有効性が顕著に表れるのは画像 grid の生成においてである．

画像 grid において，r,g,b 値をすべてアトム（1 word のアトム情報，1 word のリンク) で表現した場合，各画素ごとに 3 * 2 word の領域が必要である．また，端画素に接続するアトム 0 用の領域も必要となる．これらを全て pix(8 word) のリンク領域に押し込めることによって省略することで，必要なメモリサイズは

$$\frac{8 * 4800}{8 * 4800 + 2 * (3 * 4800 + 2 * (60 + 80))} = \frac{38400}{67600} = 0.568$$

より，57%程度に縮小することが可能となる.（また，rgb 値は各 8bit で表現できる為，演算時の bit 演算の手間を惜しまないならば pix を 6 word で表現することも可能である.）

### 6.1.4 ポインタタグ

整数をリンク領域に押し込めるため，ポインタタグと呼ばれる方式を用いる．これは KLIC 処理系 [9] においても用いられていた方式である．eyebot の場合には 32bit CPU であるため，メモリは 4byte 単位で扱われる．つまり，アドレスは必ず 4 の倍数であるため，下位 2bit は必ず 0 である．この下位 2bit をフラグとして利用し，格納されているデータがアドレスか整数データなのかの判別を行なう.

まず，最下位 bit が 1 であればデータ，0 であればアドレスとする．ここでデータかどうかの判別を行なう為，残りの 31bit は自由に扱うことができるが，eLMNtal では整数は上位の 30bit（符号付）で表現し，下位 2bit はフラグ領域（整数なら 01 が格納される）とした．フラグ領域を 2bit より増やし，いろいろなデータ表現（文字型，文字列型など）として扱うことも考えられるが，現在 eLMNtal ではデータ型として整数のみを想定している.

以降，アトムを指すリンクをリンク型リンク，整数が格納されているリンクを整数型リンクと呼ぶ.

### 6.1.5 マクロ対ビットフィールド

これらアトム，リンク，アトムの参照などは全て WORD 型の領域に格納される．これは領域節約のためにメモリ領域が複数の意味で扱われ，さらにそのそれぞれの意味において bit 操作が行なわれるためである．これらの領域の操作は，マクロを用いて行なう．マクロは lmntal.h（巻末に記載）に定義してある．

ビットフィールドを用いることも考えたが，複数の型でアクセスすることを考えると難しい．例えば型 A の A.hoge(8bit) と型 B.hoge(8bit) が同じ bit を指すことが保証されない（ビットフィールドで定義されたデータの各 bit を，メモリのどの位置に割り付けるかは処理系依存である）ため，今回のような複雑な処理には向かない．また，ビットフィールドは整数型のみしか扱えず，キャストしてポインタとして扱うといった用途には不向きである．

### 6.1.6 アトム管理クラスの実装

Java 処理系においては，アトム，リンクはオブジェクトであり，生成は new を用いて行なわれていた．しかし，eLMNtal におけるアトム生成としてこの方式を使うとなると，数 byte の領域確保のために毎回 new, delete を呼ぶことになる．これでは

- アトム領域自体を小さくしても，malloc の管理領域が大量に必要となる．
- 毎回生成，消去を行なうのは遅い
- いろいろな大きさのアトムの生成，消去を繰り返すのはメモリのフラグメンテーションを引き起こす

といった問題がある．そこで，eLMNtal では「arity 毎に一定量のメモリをあらかじめ確保しておき，必要に応じてそれを割り当てる」という方式を取る．図で表すと以下の図 6.2 のようになる．

メモリを一定量確保し，それを割り付ける方式は pooled allocation と呼ばれる．[12] こうすることで，毎回 new を行なう遅さとメモリ管理領域の無駄から解放される．さらに，arity 毎にメモリを分けることで，割り当てるメモリ量が（確保した領域単位で）一定となり，アトムをレコードのように管理することが可能となる．

arity 毎の atom を管理するクラス Atoms は以下の処理を行なう．

- メモリ領域が不足していれば一定量 new し，arity+1 の大きさのノードに分割して空きノードリストにつなげる．
- アトムが生成されたなら，freelist の先頭のノードをリストから取り出して返す．

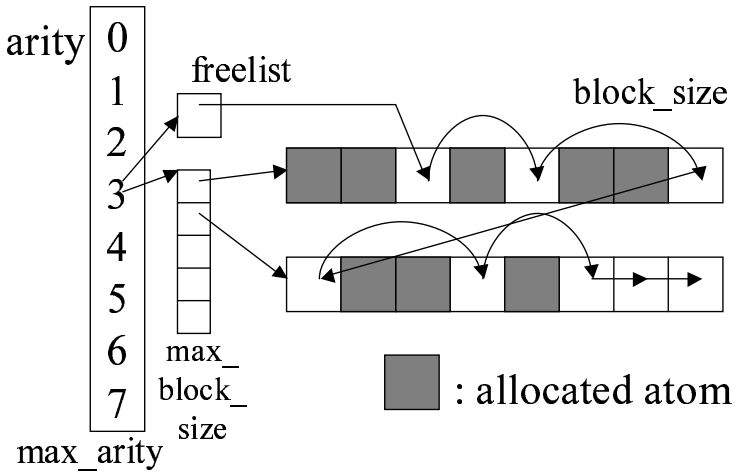


図 6.2: アトム管理機構のしくみ

- アトムが消去されたなら，freelist の先頭につなげる．

アトム管理クラス AtomManager は，arity の個数（+ proxy，pix 用）の Atoms クラスを保持し，アトムの生成，消去時に適する Atoms の生成，消去メソッドを呼び出す．

メモリはあらかじめ全量を確保するのではなく，freelist が空になったら一定量確保することを繰り返す．こうすることでメモリの無駄を省いている．(freelist は最初はなので，たとえば arity 7 のアトムのないプログラムであれば arity 7 用のアトム領域は全く確保されない)．

メモリブロックは，eLMNtal においては arity に関わらず一定量を確保している．今後 GC を実装する時，メモリブロックの消去によるフラグメンテーションがおこならないようにすることを考えてこのような実装にした．

アトム情報，リンク型リンク，整数型リンク，freenode，アトムへの参照を図で表すと図 6.3 のようになる．

### 6.1.7 findatom の実現

LMNtal の実行時間におけるネックとなるのが，findatom 命令（ある膜からある functor のアトムを 1 つ取り出す）である．LMNtal のルール適用の必要時間のほとんどは findatom の処理時間にかかっていると言っても良い（膜のロックなどがない場合)．

eLMNtal では,「アトム管理機構のブロックを，実アトムも空きノードも含めて頭からなめ，条件に合うアトムを返す」という方式を取っている．freelist は，アトム情報が入るべきところ（アトム先頭 word) に次の空きノードへの参照を入れることで実現されるため，空きノードの先頭 word の最下位 bit は必ず 0 である．

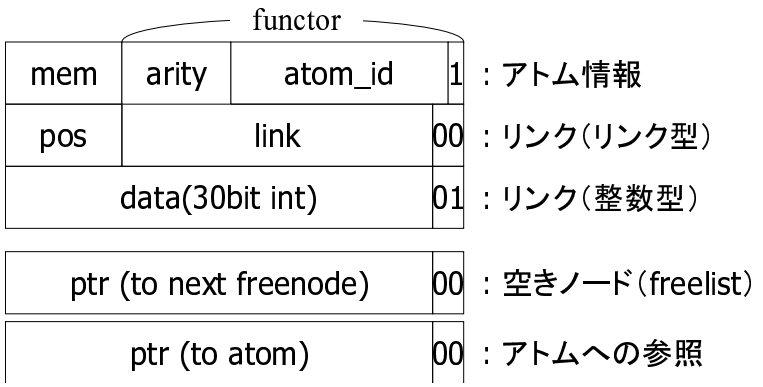


図 6.3: アトム，リンクのメモリ表現

対して，実アトムの最下位 bit は（functor が必ず奇数である為）必ず 1 である．findatom 命令はまず arity でどの Atoms クラスを見るかを判別し，ついでメモリブロックの各ノードに対して最下位 bit を見て空きノードかどうかを判別し，その後膜 ID および functor を見て条件に合うアトムがあれば返す．

Java 処理系では findatom 処理は，要求に合うアトムリストの反復子を返すことで行なわれた．候補を１つ見つけたら，ルール適用メソッドを再帰呼び出しし，適用が失敗終了した場合には次のアトム候補を取り出して再帰呼び出しを繰り返す．候補が全てなくなった時点でルール適用は失敗終了する．

eLMNtal においては反復子として，メモリブロックのどの位置まで見たかという構造体を定義した．これにより，要求に合うアトム候補の取得後ルール適用が失敗した場合，この情報を元にメモリの次の位置から検索を再開する．

## 6.2 ルール適用処理とシステム構成

### 6.2.1 システム構成

eLMNtal のルール適用にかかわるルール構成は図 6.4 のように表される．

タスクは膜管理クラス，アトム管理クラス，ルールのほかに，生成型システムアトムが積まれる実行アトムスタック，他タスクなどからタスクにアトムを生成したい場合に用いる生成待ちスタックを持つ．これらの詳しい使い方は後述する．タスクはこの他に，タスクが安定状態かを表すフラグや，生成型システムアトムと関係するルールのリスト等を持つ．

### 6.2.2 通常のルール適用の流れ

タスクはメソッド react を持つ．react を呼び出すとタスク内のルールが１つ選ばれ，ルール適用用 private メソッド interpret が呼ばれる．選ばれるルールは変

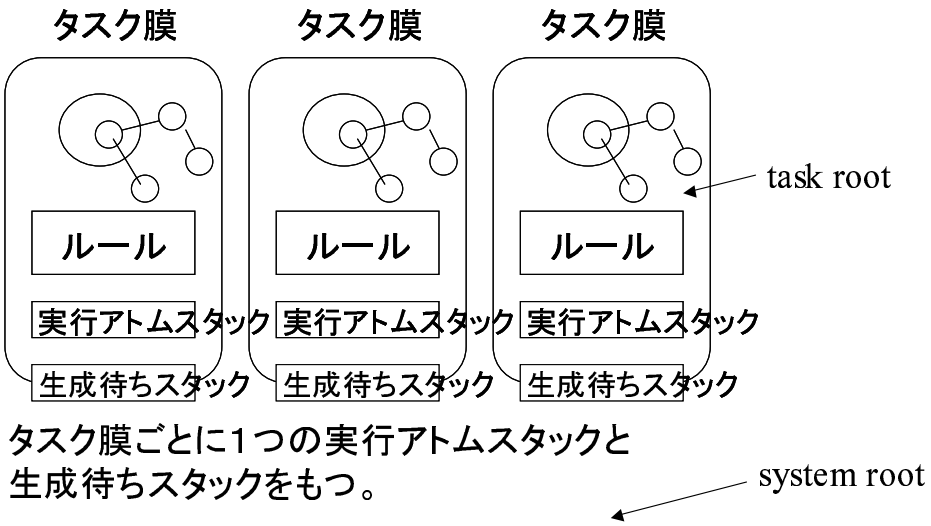


図 6.4: システム構成

数 int nextreact により管理され，nextreact は次のように変化する．

- 初期値は 0
- interpret が成功終了したなら 0 にリセット
- 失敗終了したなら+1 する．未適用のルールがなければ安定状態フラグをたて，0 にリセットする

メソッド interpret は適用するルール番号，次に読む命令の index（プログラムカウンタ）を受け取るメソッドで，次のような処理を行なう．

- ルール番号で指定されたルール配列の，pc 番目の位置に格納されている命令を読み出す．
- その命令番号ごとに固有の処理を行なう．
  - その命令の引数を pc+1 番目以降から読み出し，その情報に従ってデータ構造の確認，アトムの生成消去，リンク張り替えなどの処理を行なう．
  - 一時変数に値を格納する命令であれば，変数に値を格納する．
  - pc を次の命令の位置まで増やす（どれだけ増やすかは命令による）
  - データ構造が条件を満たさないなど，ルール適用に失敗したなら-1 を返す．proceed（ルール適用終了）時のみ 0 を返す．ルール適用を続行するなら switch 文を break し，while ループにより繰り返す．

一時変数はタスクが持つ WORD 型の配列として定義され，命令によってアトムへの参照，リンク型リンク，整数型リンク，膜 ID，が入る．整数値は整数型リンクとして扱うことで,（必ず最下位 bit が 1 であるので）アトムへの参照と区別することができる．

findatom, findmem 等の命令は，まず反復子を定義し，１つアトムを取得することができたらそのアトムを変数に代入して interpret を（pc を次の命令にセットして）再帰呼び出しする．成功終了ならそのまま成功終了し，失敗終了なら次の候補を反復子を用いて見つけ出し，再帰呼び出しを繰り返す．候補がなくなれば終了である．

### 6.2.3 コマンド型システムアトムの処理

コマンド型システムアトム（get, timer や出力デバイスの制御など）の処理は，以下のように行なう．

(1) interpret 中，newatom 命令によりアトムが生成された時，そのアトムがコマンド型システムアトムであれば，アトムへの参照を保持しておく．保持にはタスクが持つ専用の配列を用いる．

(2) interpret が成功終了した場合，上記の専用配列に積まれているコマンド型システムアトムのそれぞれについて，専用の処理を行なう．

- get であれば，リンク先の生成型システムアトムと，適切な整数値（センサ入力などから取得）をリンクさせて，実行アトムスタックに積む．get は消去．
- timer であれば，リンクする生成型システムアトムや整数値などを，タイマ情報（後述）としてタイマ処理機構に登録する．timer やそれとリンクするアトムは消去．
- leftmotor など，出力デバイスの制御を行なうアトムの場合は，リンクする整数値情報をもとに eyebot の API を呼ぶ．leftmotor などはその場で消去．
- send の場合は，指定タスクの生成待ちアトムスタックに，整数値，このタスクの ID，recv を表すファンクタ，の順に積む．send は消去する．

ルール適用の直後に必ずこの処理が行なわれるため，コマンド型システムアトムの処理は全てのルール適用に優先して行なわれることになる．

### 6.2.4 timer

eyebot には，最小 10ms ごとに呼び出される割込みハンドラ関数を呼び出す機能がある．eLMNtal では，割込みハンドラを 10ms 毎に呼び出し，そこ中でグローバル定義した変数に変更を加えることでタイマを実現する．

タイマ処理のために，以下の構造体の配列をグローバル定義する．

```
typedef struct{
    WORD count;
    WORD border;
    WORD atomFunc;
    int taskNo;
    int once;
} TIMER;
```

各メンバの意味は以下のとおり．

- count：初期値０，10ms 毎に割り込みハンドラによってインクリメントされるカウンタ．
- border：count がこの値を超えたら生成型システムアトムを生成する，という閾値．
- atomFunc：生成する生成型システムアトムの functor
- taskNo：どのタスクにアトムを生成するか．
- once：timeronce 用フラグ．

timer が生成された後のコマンド型システムアトム処理時には，count のリセット，border, atomfunc, taskNo, once の設定が行なわれる．

上記配列のほかに,「count > border である要素が存在する」ことを示すフラグ interruptFlag をグローバル定義する．割り込みハンドラ内で 10ms 毎に行なわれる処理は,

- 全ての TIMER 構造体の count 値をインクリメントする
- TIMER 構造体を頭からチェックし，count > border なものを見つけたら interruptFlag を立てる．

である．interruptFlag がたっている場合，メインループはルール適用 (react) 処理を行なわずにタイマによるアトム生成を行なう．メインループで行なわれる処理は,

- count > border である要素を見つける（複数ある場合もある）
- count −= border する.（count > border であることもありうるため）
- taskNo の指すタスクの生成待ちアトムスタックに，atomFunc を push する.
- once フラグがたっているならこの要素を無効にする（border = 0xffffffff にすることで無効にできる）

である.

### 6.2.5 timer の実装の理由と解説

割り込みハンドラ内とメインループで 2 回 border, count の比較を行なっているのは，割込みハンドラ内の処理を簡潔かつ排他制御のいらない形にする為に必要である.

また，eyebot にはシステムのタイマ値を取得する API があるが，これを用いずに割り込みハンドラを用いた理由としては，ルール適用途中であっても，commit（左辺&ガード成功）前に interruptFlag が立っていたら中断終了し，タイマ処理を優先実行したいためである.

### 6.2.6 2 つのスタックと生成型システムアトムの処理

通常は各タスクの react 処理を順番に行なっているメインループにおいて，interruptFlag が立っている，もしくはスタックに空でないものがある場合には，通常 react に優先してそれらが処理される.

まず，interruptFlag が立っているならばタイマ処理を行い，生成アトムは指定タスクの生成待ちスタックに積まれる．その後，2 つのスタックの処理がタスク 0 から順に行なわれる．その処理は以下のように行なわれる.

- 生成待ちアトムスタックに入っている（生成型システムアトムの）fanctor を取り出し，適切な値（センサ値など）を設定して，実行アトムスタックに積む．fanctor が recv の場合には，生成した recv アトムとリンクさせる値を，生成待ちアトムスタックから pop する.
- 実行待ちアトムスタックからアトムを 1 つ pop し，そのアトムに関連するルールを 1 つずつ適用する．その時ルール先頭がその functor, タスクルート膜に対する findatom であったなら，変数にアトムの参照を代入し，interpret を pc = 4 から実行する．これによってアトム主導ルール適用を行なっている.

これにより，生成型システムアトムは生成後，他の通常のルール適用に優先して処理される．また，タスク ID の若いものから順に処理を行なっているため，最初に書いた膜ほど優先度が高いと言うことができる．

生成型システムアトムに関連するルールの情報は，ルールなどと一緒にバイナリデータ入力として読み込む．生成型システムアトムは 0xe001 以上の連続したアトム ID(functor の arity 以外の部分）を持ち，(アトム ID - 0xe000 / 2) とすることで添え字に変換することが可能となるようにした．

以上の解説をまとめて，ルール適用ループの処理内容をアルゴリズムとして記述すると以下のようになる．

```
do{
  for(各タスク){
    if(タイマ割り込み発生 ||
       スタックが空でない) break;
    タスクのルールを１つ適用試行
  }
  if(タイマ割り込み発生)
    アトム functor を生成待ちスタックに追加
  if(スタックが空でない){
    for(各タスク){
      生成待ちスタックに積まれたアトムを生成
      実行スタックに積まれたアトムの優先実行
    }
  }
}while(全てのタスクが stable になるまで);
```

### 6.2.7 ルール適用の中断

interrupt メソッドは引数として lock を受け取る．これが 0 であれば，ルール適用を途中で中断しても良いことを表す．commit 命令解釈時に 1 にセットし，それ以降（右辺におけるデータ構造変更処理の途中）では中断を禁止する．

命令解釈の switch 文の手前で interruptFlag をチェックし，もし 1 であれば中断終了を行なう．中断終了時は nextReact は変更されない．(次回 react 時には同じルールの適用が試みられる)．こうすることで，より早く timer 処理を行なうことが可能となる．

なお，生成型システムアトムに対するルール適用は中断終了しない．これは，lock をはじめから 1 として interpret を呼ぶことで実装される．

### 6.2.8 proxy の処理

proxy 関連の処理は，ほぼ Java 版 LMNtal 処理系と同じことを行なっている．ルール中に出現する proxy を star という名前に変え，ルール右辺において膜構造の確定後に，star を適切な proxy に再構築する．詳しい処理については省略するので，巻末に記載したプログラム中のコメントや，Java 版処理系を参照してほしい．

## 6.3 コンバータ関連の実装

### 6.3.1 整数関係の処理変更

translmn において，newatom のファンクタとして整数アトムが指定された場合，そのまま newatom として出力する代わりに，新命令 newatom_int を出力する．これは，eLMNtal 中においては引数からは int なのか functor なのかがわからないためである．

atom と int, link 型と int 型は最下位ビットから判別可能であるが，functor が int か整数かを判別する方法はない．Java 版処理系では functor が変数に積まれる場合があるが，それらは getfunc, eqfunc など，主に最適化のために使われる命令であり，デフォルトの出力ではほとんど生成されない．唯一の例外は newatomindirect 命令で使うための functor を取得する為に使われる getfunc 命令であるが，これらの命令は整数の生成用途にのみ使われるため，変数には functor の代わりに整数型リンクを入れることにした．これによって functor が変数に積まれることはなくなり，その他の命令は整数か否かで場合分けによって処理することが可能となった．

### 6.3.2 生成型システムアトムの関連ルールの取得

生成型システムアトムと関連するルールを取得する為に，translmn 中で入力データを内部データ構造に変換した後，各ルールの findatom 命令を探し，そこで取得するアトムの functor が生成型システムアトムであれば，そのルール番号と functor を対応づける．その情報もルールなどと一緒に出力している．

生成型システムアトムは必ずアトムと整数の対になっており，整数から findatom される事はないため，必ず findatom される（deref 等によりリンクをたどって参照されることはありえない）．

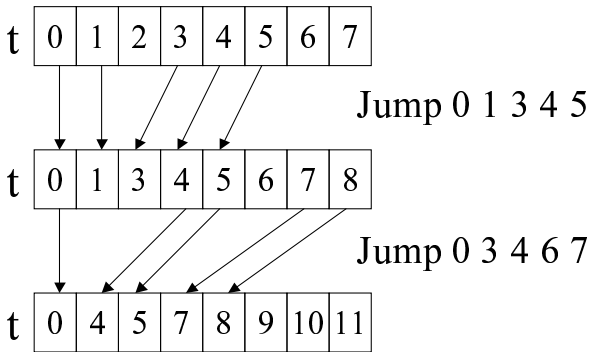


図 6.5: jump 命令による変数番号の付け替え

### 6.3.3 jump 命令の処理

Java 版処理系では，左辺の処理，ガードの処理，右辺の処理，の間で jump 命令による変数番号の付け替えが行なわれている．これは，アトム主導方式などを用いる場合，左辺の処理が複数あって，ガード，右辺の処理が共通である場合などに有効であるが，eLMNtal においては冗長な命令である．

jump 命令を処理する為に，translmn は「付け替える値」を保持する配列を持つ．例えば中間コード中で 3 とかかれていれば，配列 t[3] の値を出力することで，jump 命令の代わりとする．jump 命令による変数番号の付け替えの例を図 6.5 に示す．

jump の i 番目の引数の値が arg[i] のとき，t[i] を t[arg[i]] で上書きする．引数として出てこなかった要素には，今までに出てきた最大値以上の値を設定する．こうすることで jump 命令による命令の引数の付け替えを消去できた．

## 6.4 システムアトム一覧

現在定義されているシステムアトムの一覧を以下に示す．
生成型システムアトムを以下に記す．

- **analog0(X)**
  X にはアナログセンサ値 (10bit) が繋がる．他に analog1...analog7 まで．
- **dig_in4(X)**
  X はデジタル入力値 (0 or 1)．dig_in7 までの 4 入力．
- **quadleft(X), quadright(X)**
  X はエンコーダ値．左右 2 つのエンコーダに対応．
- **key(X)**
  eyebot の KEYRead API で返されるボタン入力値を取得する．

- **picture(X)**
  画像の２次元 grid を生成，取得する．

- **tick(X)**
  X は必ず 0．timer によって，ただ一定時間毎にアトムを生成したい時に用いる．センサによらない周期タスクを実現するために有効だろう．

コマンド型システムアトムを以下に記す．

- **get(X)**
  接続する生成型システムアトムに値を設定し，実行アトムスタックに積む．

- **timer(N, X)**
  N*10ms 毎に生成型システムアトム X をこのタスクに生成するようにタイマ処理機構（TIMER 配列）に設定する．

- **timeronce(N, X)**
  timer に加えて once フラグも立てる．１回アトム生成後に TIMER から除去される．

- **timerkill(N, X)**
  N, X で指定された情報が TIMER 配列に格納されていたら消去する．

- **leftmotor(X), rightmotor(X)**
  左右モータ出力を値 X に設定する．

- **putint(X), putchar(X)**
  X を整数，または文字として LCD 画面に出力する．

- **dig_out0(X)**
  デジタル出力値を X に設定する．dig_out7 までの８出力に対応．

- **dig_outall(X)**
  dig_out0 から dig_out7 までの出力値を，8bit の値 X に一括設定．あまり多く（8 個以上？）の dig_out を連続して処理するとエラーが出るため．

- **servo7(X)**
  ７番目のサーボコネクタに接続したサーボの角度を X に指定する．モータはサーボ 1,2, エンコーダはサーボ 3〜6 と競合するため使えないようにした．servo12 まで対応．

- **quadleft_reset, quadright_reset**
  左右エンコーダの値を０にリセットする．

- **send(T, N)**
  タスク T にアトム recv(T0, N) を生成する．T0 は send のあるタスクの番号.

- **picture_remove(X)**
  picture のリンク先のデータ構造を X に接続すると，接続する画像データ grid を一括消去することができる.

他に以下のアトムが組み込みで定義されている.

- **inside_proxy(X,Y), outside_proxy(X,Y), star_proxy(X,Y)**
  proxy アトム関係.

- **pix(PX, PY, NX, NY, R, G, B)**
  画像 grid における，画素を構成するアトム.

# 第7章　サンプルプログラム

本章では，eLMNtal 上で実行可能な制御プログラムを通して，eLMNtal で実現可能な処理について解説を行う．

## 7.1　プログラムの実行に用いる機体

制御対象として，eyebot を用いて二輪走行車を作成した (図 7.1, 7.2, 7.3 参照)．本章で扱うサンプルプログラムはすべてこの機体で実行可能である．(走行機能を用いるのは wind.lmn および redcar.lmn のみ)

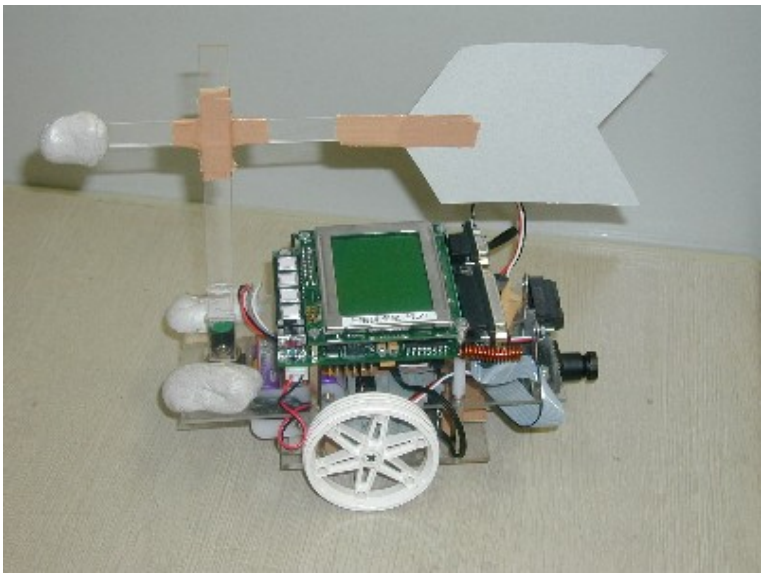

図 7.1: eyebot 二輪走行車横面．上にあるのは wind.lmn で用いる矢羽である．これで風を受け，風上の方向を判別する．

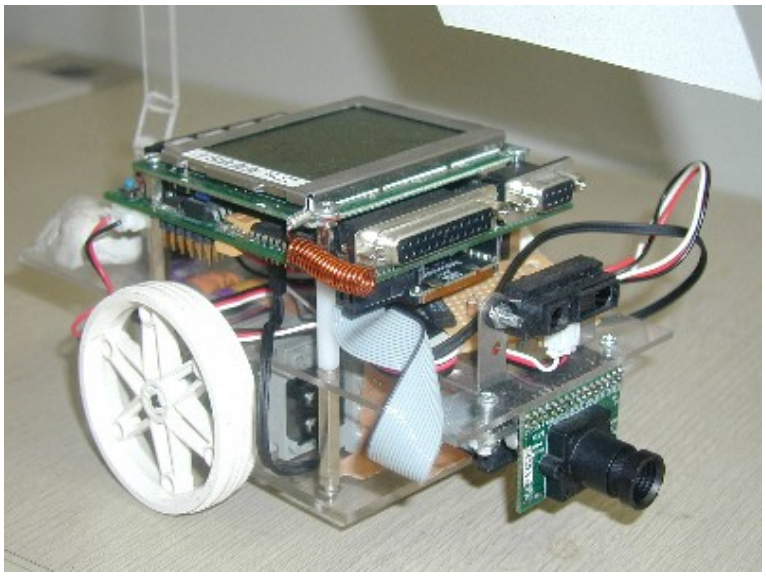

図 7.2: eyebot 二輪走行車前面．前面にみえるセンサは，上が PSD(距離) センサ，下がカメラ

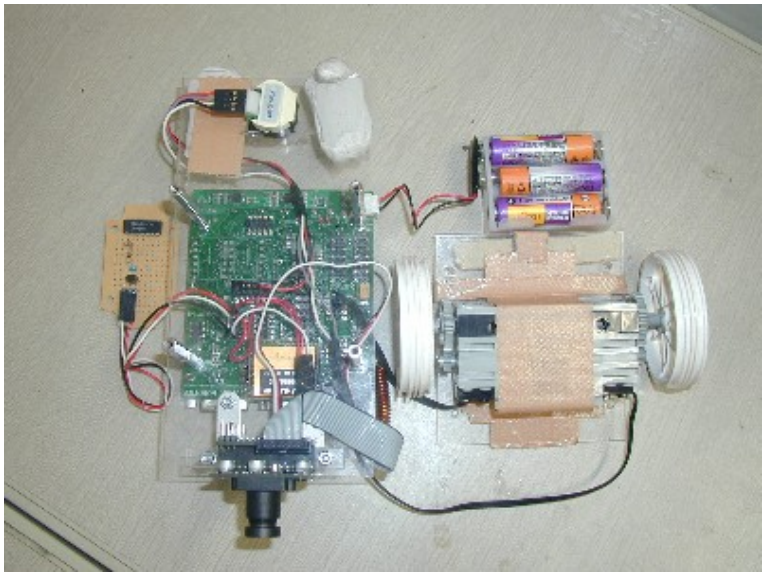

図 7.3: eyebot 二輪走行車中身．右はモータ部，モーター，タイヤは LEGO Mindstorm 用のパーツを流用している．トルクを上げるためのギアボックスも LEGO のパーツで組んでいる．左は自作の温度計．中央上に見える丸い部品はロータリーエンコーダ．これに矢羽を固定する．

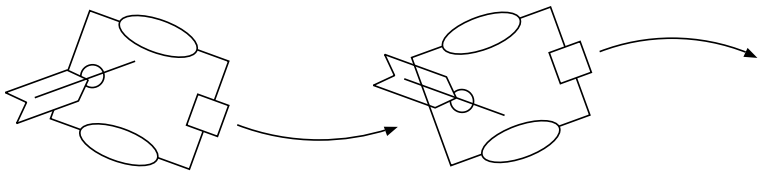

図 7.4: 風上に向かって進む車

## 7.2 tasktest.lmn

```
{
    append(c(1,c(2,c(3,n))),c(4,c(5,n)),result).
    append(X,Y,Z), n(X)      :- Y=Z.
    append(X,Y,Z), c(A,X1,X) :- c(A,Z1,Z), append(X1,Y,Z1).
}, {
    {{{a(X)}, b(X, Y)}, c(Y), send(T)}, {recv(T), a(Z), {c(Z)}}.
    {send(T), $p, @p}, {recv(T), $q, @q} :- {@p}, {$p, $q, @q}.
}{i(2).
    i(X) :- X < 10, Y = X*2 | i(Y).
}.
```

このプログラムは特に意味のあるプログラムではないが，eLMNtal において複数のタスク内において，膜や整数，リンクを使った LMNtal プログラムが正常に動くことを示している．このプログラムの実行結果は

```
{result_1(c_3(1,c_3(2,c_3(3,c_3(4,c_3(5,n_1))))))},
{{}, {a_1(_34), c_1(_30), {b_2(_24,_30), {a_1(_24)}}, {c_1(_34)}}},
{i_1(16)}
```

となる．

## 7.3 wind.lmn

「風上に向かって走る車」のためのプログラムである．左右２つのモータ，及び１つのエンコーダ (岩通アイセック EC202A050A 2 相矩形波 50pulse/round) を用い，エンコーダの先に風見鶏をつけるなどして風上がわかるようにしたマシンを作成する．また，アナログセンサ３には距離センサを接続し，前方に向けて設置しておく．

このプログラムを実行すると、図 7.4 のように eyebot 二輪走行車が矢羽の示す風上の方向へ向かって進む。

```
{
    quadleft_reset, quadleft(timer(3)), analog3(timer(3)).
    quadleft(X) :- int(X), int(Y),
        Y = (((X + 1000100) mod 200) / 2) - 50 | wind(Y).
    wind(Y) :- int(Y), int(VL), int(VR), VL = 50 + Y, VR = 50 - Y |
        leftmotor(VL), rightmotor(VR).
    analog3(X) :- int(X), X > 500 |
        end, analog3(timerkill(3)), quadleft(timerkill(3)).
    analog3(X) :- int(X), X =< 500 | .
    end :- leftmotor(0), rightmotor(0).
}
```

このプログラムではまず，30msec 毎にエンコーダ値及びアナログ値の取得を行なうようにタイマを設定し，エンコーダ値をリセットする（このとき風上を向けておくこと）．

１つ目のルールでは取得したエンコーダ値から風向きを割り出し，wind(Y) というアトムを生成している．２つ目のルールで wind(Y) の値を元にモータの出力を設定し，風上方向に軌道修正しながら走るようにしている．

アナログセンサは距離センサからの入力値を読み込み，前方の距離を測るために用いている．シャープの距離センサ GP2D12 では 10cm で 600, 25cm で 300 程度の値を返す (図 7.5 参照)．３つ目のルールでは，もしアナログセンサ値が 500 より大きい（障害物に接近した）ならばタイマを消去し，end を生成している．４番目のルールはアナログセンサ値が小さい場合に，不要な analog3 を消去している．

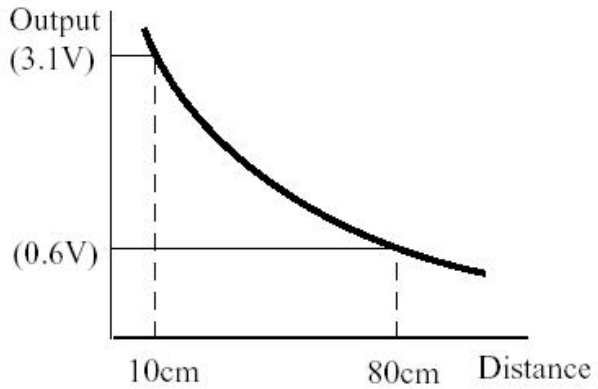


図 7.5: sharp 製 PSD センサ GP2D12 の出力 (データシートより)

アトム end は，左右のモータを止めるコマンドを生成している．なぜ３番目のルールで行なわないかというと，３番目のルールでタイマを消去すると同時にモータを止めた場合，その時点で未解釈の wind が残っており，それの解釈によってモータが再度動き出してしまう為である．２番目と３番目のルールを（長すぎるため）２つに分けたのが原因であるが，wind をユーザ定義のシステムアトムと

して定義する機能を実装すればこの問題は解決する．wind が end より優先して処理されることは（wind を処理するルールのほうが上にかかれているため）保証される．

## 7.4 thermo.lmn

次のプログラムは，温度センサから値を読み取り，それを LCD 画面に出力するプログラムである．温度センサは温度センサ素子 LM35DZ とオペアンプ等を組み合わせて自作したもので，0 ℃で 0V, 40 ℃で 4V の電圧が出力される．eyebot ではアナログセンサは 0V で 0，4V で 1000 を返すため，アナログセンサ値を 25 で割れば温度を求めることができる．

温度センサの回路図を図 7.6 に載せる．アナログ電子回路の作成にあたっては [13] [14] を参考にした．

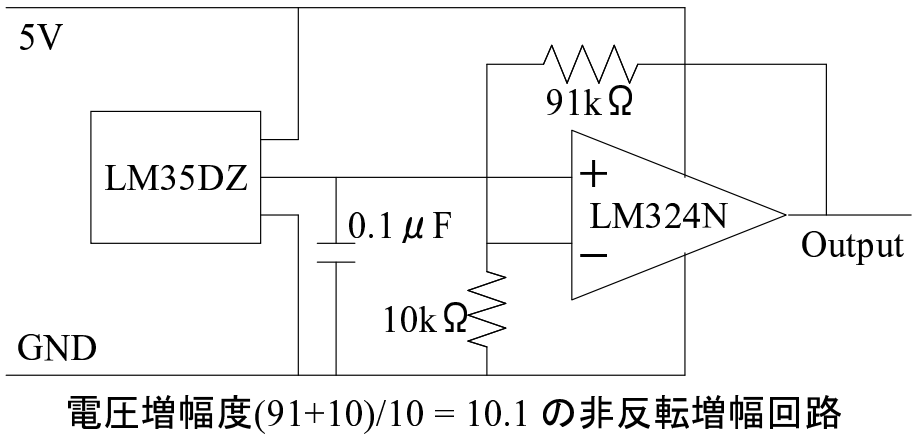


図 7.6: 温度計回路図

```
{
    buf(c(84, c(101, c(109, c(112, c(58, n))))))),
    analog2(timer(25)).

    analog2(X1), analog2(X2), analog2(X3), analog2(X4), buf(B) :-
        int(X1), int(X2), int(X3), int(X4), int(X),
        X = (X1 + X2 + X3 + X4) / 4 | data(X), buf(Y), putstr(B, Y).

    Z = putstr(c(A, Z1)) :- int(A) | Z = c(A, putstr(Z1)), putchar(A).
    Z = putstr(n) :- Z = n, next.

    next, data(X) :- int(X), int(X1), int(X2),
        X1 = X / 25, X2 = ((X * 10) / 25) mod 10 |
        putint(X1), putchar(46), putint(X2), putchar(10).
}
```

最初の行は文字列「Temp:」という文字列を表現している，リスト構造である．2，3 番目のルールが文字列の出力を行なっている．

2 行目でタイマがセットされ，250msec 毎に analog2 が生成される．1 番目のルールでは，analog2 が 4 つ溜まったところで値の平均をとり，出力を行なっている．4 番目のルールは，2，3 番目のルールで「Temp:」が出力された後に実行され，アナログセンサ値を 25 で割った値，小数点第一位の値，改行を生成，出力している．

このようにすることで，システム的に文字列が組み込まれていなくても擬似的に文字列を表示することは可能である．

## 7.5 redcar.lmn

このプログラムは，画像取得を行い，赤いものが映っている方向に走っていく車を走らせる．2 つのモータをもつ車を作成し，カメラが前方を写すように設置して実行する．

```
{
# motor drive
# 0: not found, 1: go, 2: turn left a bit. 3 turn right a bit.
        recv(1, 0) :- leftmotor(50), rightmotor(-50),
                        putint(0), putchar(10), tick(timeronce(10)).
        recv(1, 1) :- leftmotor(50), rightmotor(50),
                        putint(1), putchar(10), tick(timeronce(20)).
        recv(1, 2) :- leftmotor(-50), rightmotor(50),
                        putint(2), putchar(10), tick(timeronce(5)).
        recv(1, 3) :- leftmotor(50), rightmotor(-50),
                        putint(3), putchar(10), tick(timeronce(5)).

        tick(0) :- leftmotor(0), rightmotor(0).
}
{
        picture(get), countred.
        picture(X), countred :- picture(Y), genl(X, Y, 0).

# generate counter

        genl(T, Left, C), pix(T, PY, NX, NY, R, G, B) :-
                int(C), int(C1), C1 = C + 1 |
                pix(Left, Count, T1, NY, R, G, B), genc(NX, T1, C1),
                count(Count, PY, 0, left).
        genc(T, Left, C), pix(T, PY, NX, NY, R, G, B) :-
                int(C), C < 40, int(C1), C1 = C + 1 |
                pix(Left, PY, T1, NY, R, G, B), genc(NX, T1, C1).
        genc(T, Left, C), pix(T, PY, NX, NY, R, G, B) :-
                int(C), C =:= 40, int(C1), C1 = C + 1 |
                pix(Left, Count, T1, NY, R, G, B), genr(NX, T1, C1),
                count(Count, PY, 0, center).
```

```
        genr(T, Left, C), pix(T, PY, NX, NY, R, G, B) :-
                int(C), C < 79, int(C1), C1 = C + 1 |
                pix(Left, PY, T1, NY, R, G, B), genr(NX, T1, C1).
        genr(T, Left, C), pix(T, PY, NX, NY, R, G, B) :-
                int(C), C =:= 79 |
                pix(Left, Count, NX, NY, R, G, B),
                count(Count, PY, 0, right).

# count red pixel
        count(T, Prev, Count, No), pix(PX, T, NX, NY, R, G, B) :-
                int(R), int(G), int(B),
                R > (G * 2), R > (B * 2), R > 64,
                int(Count), int(Count1), Count1 = Count + 1 |
                count(NY, T1, Count1, No), pix(PX, Prev, NX, T1, R, G, B).
# lower priority rule . if not red
        count(T, Prev, Count, No), pix(PX, T, NX, NY, R, G, B) :-
                count(NY, T1, Count, No), pix(PX, Prev, NX, T1, R, G, B).

# sum data
        count(N, P, C, left)   :- int(N), N =:= 0 | N = P, redleft(C).
        count(N, P, C, center) :- int(N), N =:= 0 | N = P, redcenter(C).
        count(N, P, C, right)  :- int(N), N =:= 0 | N = P, redright(C).

# send data to task 1
# 0: not found, 1: go, 2: turn left a bit. 3 turn right a bit.
        redleft(L), redcenter(C), redright(R) :-
                int(L), int(C), int(R), C > 20, C >= L, C >= R |
                send(0, 1), continue.
        redleft(L), redcenter(C), redright(R) :-
                int(L), int(C), int(R), L > 20, L >= C, L >= R |
                send(0, 2), continue.
        redleft(L), redcenter(C), redright(R) :-
                int(L), int(C), int(R), R > 20, R >= L, R >= C |
                send(0, 3), continue.
        redleft(L), redcenter(C), redright(R) :-
                int(L), int(C), int(R), L =< 20, C =< 20, R =< 20 |
                send(0, 0), continue.

# end
        finish, picture(X) :- picture_remove(X).
# continue
        continue, picture(X) :- picture_remove(X), next.
        next :- picture(get), countred.
}
```

画像を読み込み，picture を起点として genl(c, r) を右に移動させながら，3 つの count アトムを画像データ構造の中に割り込ませている．これらはそれぞれ，画像の左端，中央，右端の画素列において赤いものを写している画素の数をカウントしながら，画像の下方に進んでいく．赤かどうかの判別には,「赤成分が青成分，緑成分の二倍以上大きく，さらに 64 より大きい値である」かどうかをしらべるというアルゴリズムを用いた．count が下端まで来た（カウントが終了した）ら，

赤画素数の合計値を持つ redleft, redcenter, redright というアトムを生成する．

redleft, redcenter, redright の 3 アトムの比較を行い，赤いものが映っているか，それは右よりか左よりか中央に映っているか，を判別し，別タスクにその情報を送っている．タスク 0 においては recv したその情報を元に，モーターを動かしている．(あまり別タスクにする意味はない)

画像処理の 1 ルーチンが全て終わったら，画像を消去し，また生成して，同じ処理を繰り返している．

# 第8章　考察・検証

## 8.1　アトム，リンク表現のメモリ効率化の検証

アトム，リンクを前述のように設計することで，どれくらい必要メモリを効率化することができたのかを検証する．検証には以下のプログラムを用いた．

```
{
    gen(100010). # or gen(10).

    gen(N) :- int(N), int(N1), N =\= 0, N1 = N - 1 | gen(N1), a(N).
    gen(0) :- stop.
    stop :- stop.
}
```

a(N) (N: 整数アトム) という構造を 10 個作った時と 100010 個作った時の使用メモリから，構造 a(N) に必要なメモリ領域を割り出す．

まず，eLMNtal においては a(N) という構造に必要なメモリ領域は 2word = 8byte である．（メモリ保持機構や pooled allocation による予備領域を考えても，それほどは変わらないであろう）．Java 版処理系においては，10 個作った時と 100010 個作った時で，実行時に確保されるメモリ領域を ps aux コマンドで表示し，その差から求める．

ps aux の実行結果は以下のようになった．

```
USER       PID %CPU %MEM   VSZ  RSS TTY
yajima   20592 99.9  1.2 218496 13276 pts/1 # 10 のとき
yajima   20751 43.0  7.4 219548 77032 pts/1 # 100010 のとき
```

実行環境は，

- CPU : PentiumM 1.5GHz
- OS : Vine Linux3.0
- Memory : 1GB

である．

メモリが 1GB のときに使用メモリが 6.2 %向上した．10 万個で 62MB メモリ使用量が増加したことになるので，1 構造あたり 620byte のメモリを使用していることがわかる．ここで確保されたメモリは，アトム a，整数型アトム，それらのリンクオブジェクトのために用いられている．

上記の結果では，8 対 620 で約 77 倍メモリ効率が向上している．これはプログラムによっても若干は変化するが，概ねデータ構造のメモリ使用量を数十分の 1 にすることができたといえるだろう．

## 8.2 制御表現の代案

今まで説明した eLMNtal の実装とは別に,「センサを膜で表現し，その中に整数型アトムが生成されている」という実装も考えた．この実装アトムをコマンドとして扱う現在の方式（これは手続きプログラム的である）よりも LMNtal 的な表現である．たとえば，次のように書くことを考える．

```
{
    {ここにアナログ値が生成，analog(50) など}@analog1．# 名前つき膜
    { analog(50)，$p}@analog1 :- ...
}
```

実際には 1 行目はプログラム中に記述せず，もともとあるものとして考える．この表現には以下のような欠点があるため廃案となった．

- 膜の名前を記述する機構が現在 LMNtal 処理系にない（分散型 LMNtal 処理系が計算ノードの指定のために同様の記法を用いるが，コンパイラになんらかの変更が必要になるだろう）．また，名前つきの膜を指定する中間命令を実装したとして，アトムの探索，消去，再生成などの一連の中間コードをフックしてセンサ用の特別な処理を行うのは手間が大きい．アトムの生成を捕捉し，ルール適用後に処理を行う現在の方式のほうが高速かつ容易に行うことができる．

- タスク膜ごとに 1 つ，アナログセンサを表す膜があるのはおかしい．今回はタスク切替えを手動で行っているが，システム組み込みのプリエンプティブマルチタスク機能を用いた場合には，ロック機構が必要になり，自分のタスクの子膜が他のタスクによってロックされることになる．これは避けたい事象である．対して eLMNtal の実装では，コマンド型システムアトムの処理を root 膜が行うようにし，同時に 1 つのコマンド型システムアトムしか処理しないようにすれば，排他制御の問題は解決できる．

また，現在の実装は意味的には，root 膜に（アトムでなく）名前つきの膜が存在すると考えることもできるだろう．しかしその場合,（意味上はルート膜に存在すると仮定する）ルールはより複雑なものになると予想される．

また，以下のようにガードでアナログ値やタイマを表現する方式も考えた．

```
hoge :- ANALOG1 > 500 | fuga
hoge :- TIMER =:= 10 | fuga
```

これらは意味的には，以下の記述の省略形であると考える．

```
hoge, analog1(ANALOG1) :- int(ANALOG1), ANALOG1 > 500 |
        analog1(ANALOG1), fuga.
hoge, timer(TIMER) :- int(TIMER), TIMER =:= 10 | timer(0), fuga.
```

この実装には以下の理由のため廃案になった．

- いつアナログ値をチェックすればいいのかわからない．イベント駆動を実現するにはよさそうな記法だが，eyebot にはアナログセンサ値によってアクティベートされるプロセスを実現する方法がない.（10ms ごとのタイマ割り込みハンドラ内でセンサ値のチェックをすれば可能かもしれない.）
- コンパイラを流用するためには，eLMNtal 用ガード記述を消去した上で Java 版処理系でコンパイルを行い，eLMNtal 側でも（ガード記述情報に基づく）特殊な処理を行う必要があり，手を加える部分が大きい．

意味的に不整合になりそうであるのと，処理系改編の手間のおおきさからこの実装方法は見送ったが，有効性などについては再考の余地があると思われる．

## 8.3 処理の優先度とリアルタイム性について

前述のように，システムアトムの処理は優先して行われる．ユーザが（優先度の指定のために）任意のアトムをシステムアトムであると定義できるようにすれば，複雑な一連の処理も優先的に実行することが可能となる（サンプルプログラム wind.lmn 参照）．１タスクに優先実行すべき処理が（同時に）1つしか含まれないと仮定すると，複数のタスクが同時に優先実行したい状態においてどのタスクから実行を行うかを扱うかを制御できれば，ある程度のリアルタイム性を確保することができる．

リアルタイム性において求められるのは，

- 決められた時間に処理が開始できる．

- イベント発生から決められた時間内に処理が開始できる．

ことである．eLMNtal では，timer によって決められた時間に処理を開始することを，timer でセンサ値アトムを生成し，それに対するルール適用のガード内で値を比較（イベントが発生しているかどうか）することによって，イベント駆動を実現している．問題は，タスク切替えや割り込み禁止状態時における遅延をどのように処理するかである．

ルール適用において，右辺の適用（データ構造の変更）を途中で終了することはできない（データ構造の破壊につながる）．しかし，ルール適用の実行時間のほとんどは左辺の findatom にかかっているため，右辺の処理（割り込み禁止）の時 v 間は長くない．左辺の処理を中断可能にすることで，割り込みの禁止時間による遅延を最小限にとどめることができる．

どのタスクを優先実行するかは，タスクごと（処理ごと）に制限時間を設けた上で，現在提唱されている様々なタスクスケジューリングアルゴリズムが適用可能であるだろう．

タスクの制限時間をルールの命令数やステップ数で擬似的に表現するのではなく，実時間（秒）で指定する場合には，ルールの適用時間を予測する必要がある．ここで問題となるのはやはり findatom 命令で，この命令は eLMNtal の実装においては，アトムの検索時間はタスク内にある同じ arity のアトムの個数に影響し，それ以降のルール適用にかかる時間はタスク内の同じ functor のアトムの個数に影響する（それ以降のルール適用が必ず成功する，もしくは失敗するときはすぐに失敗することがわかる場合はそれほどおおきなオーバーヘッドにはならない）．findatom, findmem 以外の命令はほぼ一定時間で終了するため，ある程度の実行時間予測をすることは可能である．

リアルタイム制御を，

- 一連の処理（タスク）にかかる時間
- タスクの優先度と制限時間

という情報を元に行う場合，findatom の実行時間を制限もしくは予想する方法を考える，または findatom を行わない実行方法を考えることが必要となるだろう．

Java 版処理系における実行膜，実行アトムスタックがうまく機能し，かつルール左辺が単一のアトムグループであるならば，findatom を原則的に行わない実装とすることができるかもしれない．しかし eLMNtal においては生成型システムアトムと整数型はかならず独立したアトムグループとして生成され，左辺にそれ以外の情報を指定したい場合にはどうしても複数のアトムグループになってしまう．また，Java 版処理系においてはアトム管理は膜ごとに，アトム名ごとのアトムリストをハッシュで引くことが可能であるので，eLMNtal における「findatom における，同じ arity のアトム個数に依存する検索時間」はかからないようになっている．

# 第9章　まとめと今後の課題

## 9.1　まとめ

本研究により，小規模な組み込みシステム eyebot 上で動作する LMNtal 処理系 eLMNtal が設計，実装された．eLMNtal では，

- 複数タスクの擬似的な並行動作
- タスク膜内における，膜，プロセス文脈の一部機能を含む複雑な LMNtal プログラムの解釈実行
- 各種入出力デバイスの制御
- カメラからの画像取得と画素アトムによる 2 次元 grid 形式への展開
- タイマによる一定間隔のセンサ値読み込みと処理起動
- メモリ必要量を効率化したアトム，リンク，膜の管理機構
- システムアトム（読み込んだセンサ値の処理）の優先実行と，通常のルールの中断処理
- リンク領域に整数アトムを押し込めることによるメモリ使用量の効率化

などの機能が実現された．これによって，7 章であげたような，センサ入力から処理を経てアクチュエータに出力する，という処理を一定時間ごとに繰り返すような動作を LMNtal で記述し，ある程度高速に動作させることに成功した．また，アトムとリンクのメモリ使用量は，条件によっては 77 分の 1 程度に抑えることができた．さらに、eLMNtal の静的なプログラムサイズは 54KB であり、プログラムサイズも 5 分の 1 以下に縮小することができた．

本処理系によって LMNtal における制御処理と優先実行の一方式を提唱したことで，制御とリアルタイム処理に関する様々な問題，

- タスクに対する制限時間や優先度の指定，制御法
- ルール適用時間の静的な解釈の必要性と，findatom の処理時間とアトム個数の関係

- eLMNtal 式の制御表現の問題点と，これ以外の表現法

に関する足掛かりを提供することができた.

## 9.2 今後の予定

eLMNtal は一応一通りの機能の実装を行うことができた．今後はこれを元に，eyebot に限らず幅広い制御系に適用可能な一般的な方式や意味，理論について議論していきたい．また，eLMNtal はコンパイラ部分は Java 処理系を利用しており，コンパイラレベルでの大きな変更や最適化はできなかった．中間言語の interpret 解釈とデータ構造の変更による計算進行は本質的には明らかに組み込み向きではないため，文法的には flat LMNtal など大きく制限を設けても，高度なコンパイルを行うことでより高速な制御処理を行えるような処理系の実装についても考えていく必要がある.

今後の研究課題としては，以下のようなものがあげられる.

- eLMNtal に対する実時間制御機能の追加
- LMNtal の言語仕様のうち，本当に制御に必要な機能はどの部分かを割り出し，組み込みに特化した LMNtal 言語サブセットの提唱
- flat LMNtal(をさらに制限した言語) による，データ構造の動的変更によらない高速な制御処理方式の設計
- プリエンプティブなマルチタスクに対応する処理系（eLMNtal でも極力意識して実装を行っているが，これを厳密に実装する）

また，eLMNtal に対して今後追加していきたい機能としては

- KEYGet や OSGetCharRS232，OSWait など，イベントがあるまでサスペンドしてしまうような API の eLMNtal 中での表現（マルチタスク化により実現可能かもしれない）
- root 膜にルールを書くことによる，高度なタスク間通信の実現
- ユーザが任意のアトムをシステムアトム（優先実行されるアトム）として指定可能にすることで，一連の（複数ルールによる）複雑な処理を優先的に実行可能にする
- 画像用領域の一括生成，消去，および画像データのリフレッシュ（現在は画素アトムも通常のアトム管理機構を流用している.）

- 中間言語レベルでの最適化（newlink, relink の展開や，getfunc と newatomindirect の消去など）

などがあげられる．

# 謝辞

本研究を進めるにあたり、御指導を頂きました上田和紀教授に厚く御礼申し上げます。

議論を通じて様々なご意見を頂きました上田研究室の方々に感謝いたします。特に加藤紀夫氏には、設計方針において多大なご指摘、ご助言を頂きました。ここに心からの感謝を捧げたいと思います。

2005年2月　矢島 伸吾

# 参考文献


[1] 上田和紀，加藤紀夫： 言語モデル LMNtal．コンピュータソフトウェア，Vol.21，No.2

[2] 上田和紀, 加藤紀夫：Programming with Logical Links, 日本ソフトウェア科学会第 19 回大会論文集, 2002.

[3] Kazunori Ueda and Norio Kato：Programming with Logical Links, Design of the LMNtal language, In Proc. 3rd Asian Workshop on Programming Languages and Systems (APLAS 2002), pp.115-126, 2002.

[4] 矢島伸吾，永田貴彦，加藤紀夫，上田和紀：LMNtal プロトタイプ処理系の設計と実装，日本ソフトウェア科学会第 20 回記念大会論文集，pp. 21-25, 2003.

[5] 矢島伸吾：LMNtal Runtime の Java による設計と実装, 早稲田大学卒業論文, 2003.

[6] 原耕司, 水野謙, 矢島伸吾, 永田貴彦, 中島求, 加藤紀夫, 上田和紀：LMNtal 処理系および他言語インタフェースの設計と実装．SWoPP2004, p.157, 2004.

[7] 水野兼，永田貴彦，加藤紀夫，上田和紀：LMNtal ルールコンパイラにおける内部命令の設計．情報処理学会第 66 回全国大会, 2004.

[8] 中島求，加藤紀夫，水野謙，上田和紀：LMNtal 分散処理系の設計と実装．日本ソフトウェア科学会第 21 回大会論文集，pp.3B-2, 2004.

[9] 関田大吾：Inside KLIC Version 1.0, KLIC Task Group/AITEC/JIPDEC, 1998.
`http://www.klic.org/software/klic/inside/master.html`

[10] Thomas Bräunl：Embedded Robotics. Springer-Varlag, 2003.

[11] Thomas Bräunl：Eyebot - Online Documentation.
`http://www.ee.uwa.edu.au/~blaunl/eyebot/`

[12] James Noble, Charles Weir：Small Memory Software. Addison-Wesley Pub (Sd), 2000.

[13] 小川実吉，幾島康夫：温度センサとセンサ IC の基礎知識. トランジスタ技術 2003/12, pp. 143–158

[14] 畔津明仁，三宅和司，川田章弘：はじめてのパーツ選びと定数設計．トランジスタ技術 2004/6, pp. 115–145

[15] Jiro Eto：legOS 解説.
http://www.ylw.mmtr.or.jp/~arcadia/legos/

[16] 井上誠喜, 林正樹, 三谷公二, 八木伸行, 中須英輔, 奥井誠人：C 言語で学ぶ実践画像処理．オーム社, 1999.

[17] Khawar M. Zuberi, Kang G. Shin：EMERALDS: A Small-Memory Real-Time Microkernel. IEEE Transactions on Software Engineering, Vol.27, No.10, 2001/10.

[18] 湯浅太一：Lego Mindstorms 制御プログラムの対話型開発・実行環境．IPSJ Programming Symposium 2004.

[19] 木村孝通, 平林浩一：Bourne Shell 自習テキスト. 1993/6
http://flex.ee.uec.ac.jp/texi/sh/sh.html

[20] S. Abdennadher： Operational Semantics and Confluence of Constraint Propagation Rules, Third International Conference on Principles and Practice of Constraint Programming, CP'97, Schloss Hagenberg, Austria, Springer LNCS, 1997

[21] Holzbaur C., Banda M.G.de la, Jeffrey D., Stuckey P.J.： Optimizing Compilation of Constraint Handling Rules, in Proceedings of the International Joint Conference on Logic Programming (ICLP'01), pp.74-89, 2001.

[22] Holzbaur C., Fruehwirth T.： A PROLOG Constraint Handling Rules Compiler and Runtime System, Applied Artificial Intelligence, 14(4), 369-388, 2000.

# Appendix.A 中間コード対応

## A.1 Java版処理系とeLMNtalの中間コード対応表

eLMNtalの中間コードはJava版処理系の出力を利用しているが、eLMNtal処理系では使わない命令や、整数をリンクに押し込めたことにより場合分けによる処理が必要となった命令など、変更を加えた部分がある。ここでは、Java版処理系の全中間命令が、eLMNtalの命令とどのように対応しているかを一覧で示す。

```
アトムに関係する出力する基本ガード命令 (1--5)
- 1 deref     [-dstatom(I/A), srcatom(A), srcpos, dstpos]
-- src は必ず Atom(int から引くことはない)。dst は int or atom。
- 2 derefatom [-dstatom(I), srcatom(A), srcpos]
-- ガードのみ、整数参照用のみに使用。src は必ず atom.
- 3 dereflink [-dstatom, srclink, dstpos]
-- 未使用
- 4 findatom  [-dstatom(A), srcmem, funcref(A)]
-- funcref に整数が指定されることはない。(2(X) などは文法的に許さない)

膜に関係する出力する基本ガード命令 (5--9)
- 5 lockmem    [-dstmem, freelinkatom]
-- ロックはせず、膜の取得のみを行う。
- 6 anymem     [-dstmem, srcmem]
-- ロックはせず、mm.findmem を呼び出し膜を取得するのみ。
- 7 lock       [srcmem]
-- 未使用(アトム主導テスト用)
- 8 getmem     [-dstmem, srcatom]
-- 未使用(アトム主導テスト用)
- 9 getparent  [-dstmem, srcmem]
-- 未使用(アトム主導テスト用)

膜に関係する出力しない基本ガード命令 (10--19)
- 10 testmem    [dstmem, srcatom(A)]
-- ★ srcatom はアトムのみ?(のはず)。
- 11 norules    [srcmem]
-- 消去(必ず成功する)
- 12 nfreelinks [srcmem, count]
-- 未使用($p(*) のみ許可なので生成されない命令)
- 13 natoms     [srcmem, count]
-- ★ int もカウントする?(一応する)(natoms, 0 以外に呼ばれることある?)
- 15 nmems      [srcmem, count]
-- そのまま。
- 17 eqmem      [mem1, mem2]
-- そのまま。
- 18 neqmem     [mem1, mem2]
-- ロックを使わないので、この命令を生成するようにコンパイラを変更する。
- 19 stable     [srcmem]
-- 未使用(非対応)

アトムに関係する出力しない基本ガード命令 (20-24)
- 20 func     [srcatom(I/A), funcref]
-- getfunc[tmp,srcatom];loadfunc(_int)[tmpfunc,funcref];eqfunc[tmp,tmpfunc] に展開。
- 21 notfunc  [srcatom, funcref]
```

-- 未使用。プロセス文脈の明示的な自由リンク用。
- 22 eqatom [atom1, atom2]
-- atom(addr) が等しいか比較（★ atom が int のことはない？）java 版ではオブジェクトの比較を行っている。
- 23 neqatom [atom1, atom2]
-- 同上
- 24 samefunc [atom1, atom2]
-- getfunc[func1,atom1];getfunc[func2,atom2];eqfunc[func1,func2] に展開。

ファンクタに関係する命令 (25--29)~
★未実装。func 関係は最適化のみしかつかわない？~
追記：getfunc は演算結果を生成する時に使う。★暫定：getfunc は link をそのまま返し、その link は allocatomindirect でそのまま link 型整数 atom としてコピーされる？

- 25 dereffunc [-dstfunc(I), srcatom(A), srcpos]
-- derefatom[dstatom,srcatom,srcpos];getfunc[dstfunc,dstatom] に展開。★必ず int?
- 26 getfunc [-func, atom(I/A)]
-- ★場合分け。アトムの時は atomID を、int のときは signed int 値を vars[func] に代入
- 27 loadfunc [-func, funcref(I/A)]
-- ★ INT 命令 loadfunc_int 追加。vars[func] に funcref を格納。
- 28 eqfunc [func1, func2]
-- そのまま。
- 29 neqfunc [func1, func2]
-- そのまま。

アトムを操作する基本ボディ命令 (30--39)
- 30 [local]removeatom [srcatom(I/A), srcmem]
-- 膜のアトム個数を-1 するだけ。proxy のときはなにもしない。(★ int の時は増減する？(一応する))。★第３引数 funcref は、int の時は渡されない。消去する。
- 31 [local]newatom [-dstatom(I/A), srcmem, funcref(I/A)]
-- ★ INT 命令 newatom_int 追加。dstatom は atom のとき atom, int のとき link.
- 32 [local]newatomindirect [-dstatom, srcmem, func]
-- 未使用。
- 33 [local]enqueueatom [srcatom]
-- 未使用。実行アトムスタックは使用しない。
- 34 dequeueatom [srcatom]
-- 未使用。
- 35 freeatom [srcatom(I/A)]
-- int のときはなにもしない（命令自体の消去は、free されるのが atom か int か命令列からはわかりにくいためできない)。atom のときは am.freeatom を呼ぶ。

- 36 [local]alterfunc [atom, funcref]
-- 未使用。最適化用。
- 37 [local]alterfuncindirect [atom, func]
-- 未使用。最適化用。

アトムを操作する型付き拡張用命令 (40--49)
- 40 allocatom [-dstatom(I), funcref(I)]
-- ★ INT のみ。-dstatom には int(link 形式）が格納される。
- 41 allocatomindirect [-dstatom, func]
-- ★ INT のみ？getfunc で取得した func から atom を作る。★暫定：func(link) を dstatom(link) にコピーするのみ。

- 42 [local]copyatom [-dstatom(I), mem, srcatom(I)]
-- ★ INT のみ。値をコピーする。
- 43 local addatom [dstmem, atom]
-- 未使用。最適化用。

膜を操作する基本ボディ命令 (50--59)
- 50 [local]removemem [srcmem, parentmem]
-- そのまま。parentmem つかわない？
- 51 [local]newmem [-dstmem, srcmem]
-- そのまま。
- 52 allocmem [-dstmem]
-- 未使用。最適化用。
- 53 newroot [-dstmem, srcmem, node]
-- 未使用。分散処理用。

```
- 54 movecells                [dstmem, srcmem]
-- アトム移動は 1 個 1 個。膜移動はツリーの書き換えで対応。
- 55 enqueueallatoms          [srcmem]
-- 未使用。実行アトムスタック使わない。
- 56 freemem                  [srcmem]
-- removemem 済みであること。
- 57 [local]addmem                    [dstmem, srcmem]
-- removemem 済みの膜を dstmem に移動（膜ツリー書き換え）する。
- 58 [local]enqueumem                 [srcmem]
-- 未使用。実行膜スタックを使わない。
- 59 [local]unlockmem                 [srcmem]
-- 未使用。ロック使わない。
- 60 [local]setmemname                [dstmem, name]
-- 未使用。
```

予約 (61--62)

リンクに関係する出力するガード命令 (63--64)

```
- 63 getlink   [-link,atom(A),pos]
-- atom のみ。int のリンク先が取得されることはない。
- 64 alloclink [-link,atom(A/I),pos]
-- ★場合分け。link の時はコピーするだけ、atom の時は link 生成。
```

リンクを操作するボディ命令 (65--69)

```
- 65 [local]newlink     [atom1, pos1, atom2, pos2, mem1]
-- alloclink[link1,atom1,pos1]; alloclink[link2,atom2,pos2];
unifylinks[link1,link2,mem1] に展開。

- 66 [local]relink      [atom1, pos1, atom2, pos2, mem]
-- alloclink[link1,atom1,pos1]; getlink[link2,atom2,pos2];
unifylinks[link1,link2,mem] に展開。
- 67 [local]unify       [atom1, pos1, atom2, pos2]
-- getlink[link1,atom1,pos1]; getlink[link2,atom2,pos2];
unifylinks[link1,link2,mem] に展開。
- 68 [local]inheritlink [atom1, pos1, link2, mem]
-- 未使用。最適化用。
- 69 [local]unifylinks  [link1, link2, mem]
-- ★場合分け。両方 atom なら相互接続、片方が int なら link 中に押し込める。両方 int ならエラー。
```

自由リンク管理アトム自動処理のためのボディ命令 (70--74)

```
- 70 removeproxies          [srcmem]
- 71 removetoplevelproxies  [srcmem]
- 72 insertproxies          [parentmem,childmem]
- 73 removetemporaryproxies [srcmem]
-- proxy 関連は Java と同じ処理を行う。MembraneManager の対応するメソッドを呼ぶ。
```

ルールを操作するボディ命令 (75--79)

```
- 75 [local]loadruleset [dstmem, ruleset]
- 76 [local]copyrules   [dstmem, srcmem]
- 77 [local]clearrules  [dstmem]
-- 全て未使用。ルールはタスク膜に固定。
```

型付きでないプロセス文脈をコピーまたは廃棄するための命令 (80--89)

```
- 80 recursivelock            [srcmem]
- 81 8recursiveunlock          [srcmem]
-- 未使用。ロック無し。
- 82 copymem          [-dstmap, dstmem, srcmem]
-- 未使用。プロセス変数は線形（両辺に 1 度ずつしか出ない）
- 83 dropmem                  [srcmem]
-- 未使用。プロセス変数は$p のみ、線形とするなら使わない命令。$p[] を許し、
プロセス変数が左辺にのみ出現する（右辺で消える）ことを許すなら、
dropmem, および proxy_star の個数を数える機構を実装する必要がある。
```

制御命令 (200--209)

```
-  react       [ruleref,         [memargs...], [atomargs...], [varargs...]]
-  jump        [instructionlist, [memargs...], [atomargs...], [varargs...]]
-  commit      [ruleref]
```

```
- resetvars   [[memargs...], [atomargs...], [varargs...]]
- changevars  [[memargs...], [atomargs...], [varargs...]]
- spec        [formals,locals]
- proceed
- stop
- branch      [instructionlist]
- loop        [[instructions...]]
- run         [[instructions...]]
- not         [instructionslist]
```

型検査のためのガード命令 (220-229
- 220 isground [-natomsfunc, link, srcset]
-- 非対応。
- 221 isunary [atom]
-- ★未使用？int かどうか調べるときは必ず isint を使う？実装するには場合分けが必要。
- 225 isint [atom]
-- 変数の 1bit めを調べて判別。
- 226 isfloat, 227 isstring, 228 getclass
-- 未使用。

整数用の組み込みボディ命令 (400--419+OPT)
- 400 iadd [-dstintatom, intatom1, intatom2]
-- 2 つの int(link) を受けとって、演算結果を変数に int(link) で格納する。
- 401 isub, 402 imul, 403 idiv, 404 ineg, 405 imod
-- iadd に同じ。
- 410 inot, 411 iand, 412 ior, 413 ixor
-- ビット演算用？

整数用の組み込みガード命令 (420--429+OPT)
- 420 ilt, 421 ile, 422 igt, 423 ige, 424 ieq, 425 ine [intatom1, intatom2]
-- 大小比較演算。受け取るのは int(link)

＜以下、非対応な命令＞

型付きプロセス文脈を扱うための追加命令 (216--219)
- 216 eqground, 217 copyground, 218 removeground, 219 freeground

分散拡張用の命令 (230--239)
- 230 getruntime, connectruntime

アトムセットを操作するための命令 (240--249)
- 240 newset, 241 addatomtoset
-- 基底項プロセス関係で使う。

浮動小数点用命令 (600--640)

# Appendix.B ソースコード

以下にソースコードを記載する。

- runtime system：2572 行
- 中間言語コンバータ：1122 行
- その他 shellscript, makefile 等

で合計 3700 行程度のプログラムを記述した。

## B.1 ランタイム

### B.1.1 atom.cpp

```
 1 #include "atom.h"
 2 #include "lmntal.h"
 3 #include "eyeio.h"
 4
 5 ////////////////////////
 6 // コンストラクタ・デストラクタ
 7
 8 AtomManager::AtomManager(void)
 9 {
10     atoms = new (Atoms *)[MAX_ARITY];
11
12     for(int i=0; i < MAX_ARITY; i++){
13         atoms[i] = new Atoms(i, MAX_BLOCK, BLOCK_SIZE);
14     }
15     proxies = new Atoms(2, MAX_BLOCK, BLOCK_SIZE); // in, out, star は 2 引数アトム
16     pixes = new Atoms(7, 120, 1024); // 画像 3 枚
17 }
18
19 AtomManager::~AtomManager(void)
20 {
21     for(int i=0; i < MAX_ARITY; i++){
22         delete atoms[i];
23     }
24     delete proxies;
25     delete pixes;
26     delete[] atoms;
27 }
28
29 Atoms::Atoms(int arity, int max_block, int block_size){
30     Atoms::arity = arity;
31     atom_size = arity + 1;
32     maxBlock = max_block;
33     blockSize = block_size;
```

```
34      block = new (WORD*)[maxBlock+1]; // 末尾：番兵
35      dbg("block ptr:%x\n", (WORD)block);
36      for(int i=0; i < maxBlock+1; i++) block[i] = NULL;
37      nBlock = 0;
38      freelist = 0;
39  }
40
41  Atoms::~Atoms(void)
42  {
43      for(int i=0; i < nBlock; i++){
44          if(block[i] != NULL) delete[] block[i];
45      }
46      delete[] block;
47  }
48
49  ////////////////////////////////////////////
50  // atoms メソッド
51
52  WORD Atoms::newAtom(int func, int mem)
53  {
54      WORD atom;
55
56      if(freelist == 0){
57          addBlock(); // todo: 例外処理
58      }
59      atom = freelist;
60      freelist = *((WORD *)freelist);
61
62      *((WORD*)atom) = mem << 24 | (func & 0xffffff);
63
64  // dbg("new atom %8x on arity:%d, mem:%d, %8x\n", *((WORD*)atom), arity, mem, atom);
65      return atom; // block のなかの空き領域のアドレスが WORD 値で返る
66  }
67
68  void Atoms::freeAtom(WORD atom) // ptr to freeing atom
69  {
70      WORD temp;
71      temp = atom;
72      *(WORD *)atom = freelist;
73      freelist = temp;
74  }
75
76  void* Atoms::addBlock()
77  {
78      int i;
79
80      if(nBlock < maxBlock){
81          block[nBlock] = new WORD[blockSize];
82      }
83
84      if(block[nBlock] == NULL) return NULL;
85
86      for(i=0; i <= blockSize - atom_size*2; i += atom_size){
87          block[nBlock][i] = (WORD)(&(block[nBlock][i + atom_size]));
88      }
89      block[nBlock][i] = 0; // 末尾
90      freelist = (WORD)(&(block[nBlock][0]));
91
92      return (void*)block[nBlock++];
93  }
94
95  // 見つからない時は 0 を返す。見つかったらアドレスを WORD で返す。
96  WORD Atoms::next(AtomIt* ait)
97  {
98      int i;
99      WORD a;
100
```

```
101     while(block[ait->block] != NULL){// block がなくなるまで進める
102
103         a = (WORD)(&(block[ait->block][ait->index]));
104         // 次の領域にポインタを進める
105         ait->index += atom_size;
106         if(ait->index > blockSize - atom_size){
107             ait->index = 0;
108             ait->block++;
109         }
110         if(((*((WORD*)a)) & 0x1 == 1) && ((*((WORD*)a)) == ait->target)){
111             return a;
112         }
113     }
114     return 0;
115 }
116
117 // movecells 用、srcmem 内のアトムを dstmem に移動する。
118 // 効率的な方法は、この実装（タスク膜ごとのアトム管理）だとなさそう。
119 // 頻繁に呼ぶようなら膜ごとにアトム管理したほうがいい。
120 void Atoms::moveAtoms(BYTE dstmem, BYTE srcmem)
121 {
122     for(int i=0; block[i] != NULL; i++){
123         for(int j=0; j < blockSize; j+=atom_size){
124             // addr < 0x00ffffff より、空き領域の時は 0(!=srcmem) になる
125             if(((block[i][j] >> 24) & 0xff) == srcmem){
126                 block[i][j] =
127                     (((WORD)dstmem & 0xff) << 24) | (block[i][j] & 0x00ffffff);
128             } // srcmem に所属するアトムは dstmem 所属に書き換える
129         }
130     }
131 }
132
133 void Atoms::dumpAtom(){
134
135     int i, j, k;
136     WORD a;
137     for(i=0; block[i] != NULL; i++){
138         for(j=0; j <= blockSize - atom_size; j+=atom_size){
139             a = (WORD)(&(block[i][j]));
140
141             if((*(WORD*)a) & 0x1 == 1){ // 実アトム（freenode でない）なら
142                 putword(atom_size + 1);
143                 putword(*(WORD*)a);
144                 putword(a);
145                 for(k=1; k <= arity; k++) putword(((WORD*)a)[k]);
146             }
147         }
148     }
149 }
150
151 void Atoms::debugout(){
152
153     int i, j, k;
154     WORD a;
155     for(i=0; block[i] != NULL; i++){
156         for(j=0; j <= blockSize - atom_size; j+=atom_size){
157             a = (WORD)(&(block[i][j]));
158
159             if((*(WORD*)a) & 0x1 == 1){ // 実アトム（freenode でない）なら
160                 dbg("%x@%x>", *(WORD*)a, a);
161                 for(k=1; k <= arity; k++) dbg("%x, ", ((WORD*)a)[k]);
162                 dbg("\n");
163             }
164         }
165     }
166 }
167
```

```
168 ////////////////////////////////
169 // AtomManager メソッド
170
171 WORD AtomManager::newAtom(int functor, int mem)
172 {
173     if(functor & 0x01 != 1){
174         dbg("atom name must be odd number.\n");
175         return 0;
176     }
177
178     // pix か？
179     if(functor == PIX_7){
180         return pixes->newAtom(functor, mem);
181     // proxy か？
182     }else if(FUNC_NAME(functor) > PROXY_BORDER){
183       dbg("proxy newed\n");
184         return proxies->newAtom(functor, mem);
185     }else{
186         WORD arity;
187         arity = FUNC_ARITY(functor);
188         return atoms[arity]->newAtom(functor, mem);
189     }
190 }
191
192 void AtomManager::freeAtom(WORD atom)
193 {
194     if(ATOM_GETFUNC(atom) == PIX_7){
195         dbg("pix removed\n");
196         pixes->freeAtom(atom);
197     }else if(ATOM_GETNAME(atom) > PROXY_BORDER){ // proxy なら
198         proxies->freeAtom(atom);
199     }else{
200         WORD arity;
201         arity = ATOM_GETARITY(atom);
202         atoms[arity]->freeAtom(atom);
203     }
204 }
205
206 void AtomManager::initIt(AtomIt* ait, WORD functor, WORD mem)
207 {
208     ait->index = 0;
209     ait->block = 0;
210     ait->arity = FUNC_ARITY(functor);
211     ait->target = mem << 24 | (functor & 0xffffff);
212 }
213
214 WORD AtomManager::next(AtomIt* ait)
215 {
216     if((ait->target & 0x00ffffff) == PIX_7){
217         LCDPrintf("findatom pix_7 called.\n");
218         return 0;
219     }else if((ait->target & 0x0000ffff) > PROXY_BORDER){
220         return proxies->next(ait);
221     }else{
222         return atoms[ait->arity]->next(ait);
223     }
224 }
225
226 void AtomManager::dumpAtom()
227 {
228     for(int i=0; i < MAX_ARITY; i++){
229         atoms[i]->dumpAtom();
230     }
231     proxies->dumpAtom();
232     pixes->dumpAtom();
233
234     putword(0); // 末尾 size = 0
```

```
235 }
236
237 void AtomManager::moveAtoms(BYTE dstmem, BYTE srcmem)
238 {
239     for(int i=0; i < MAX_ARITY; i++){
240         atoms[i]->moveAtoms(dstmem, srcmem);
241     }
242     proxies->moveAtoms(dstmem, srcmem); // いるのか？正しく動くのか？
243     // pixes->moveAtoms(dstmem, srcmem); // root 膜のみなのでいらないはず。
244 }
```

## B.1.2 atom.h

```
 1 #ifndef ATOM_H
 2 #define ATOM_H
 3
 4 #include "lmntal.h"
 5 /*
 6 atom info
 7   0x00000001 : 必ず 1 (ポインタ (必ず 0) との区別)
 8   0x0000fffe : atom id（上とあわせて必ず奇数、32k 種類）
 9   0x000f0000 : arity
10     // 上 3 つを合わせて広義の atom id とする。
11   0x00f00000 : flag ( reserved )
12   0xff000000 : mem
13
14 link
15   0x00ffffff : ptr to atom
16   0x0f000000 : linked atom's link no.
17   0xf0000000 : flag ( reserved )
18  or
19   0x00000001 : flag (1 : data)
20   0xfffffffe : data (int 31bit)
21
22 freelist
23   0x00000001 : 必ず 0（生ポインタ）(必ず 1 な atom と区別)
24   0xffffffff : ptr to next free node.
25 */
26
27 typedef struct{
28     int index;
29     int block;
30     int arity;
31     WORD target; // 見つけるアトム名（mem&arity&id）
32 } AtomIt;
33
34 ////////////////////////////
35 // クラス定義
36
37
38 class Atoms{
39 public:
40     Atoms(int arity, int max_block, int block_size);
41     ~Atoms(void);
42     WORD    newAtom(int id, int mem);
43     void    freeAtom(WORD atom);
44     WORD    next(AtomIt* it);
45     void    dumpAtom();
46     void    debugout();
47     void    moveAtoms(BYTE dstmem, BYTE srcmem);
48 private:
49     void*   addBlock();
50
51     int     arity;
```

```
52      int     atom_size;
53      WORD**  block;
54      int     nBlock;
55      WORD    freelist;
56
57      int maxBlock;
58      int blockSize;
59  };
60
61  // atoms[max_arity] ですむ？> atomManager.addatom() とかしたい。(arity の自動処理)
62  class AtomManager{
63  public:
64      AtomManager(void);
65      ~AtomManager(void);
66      WORD    newAtom(int id, int mem);
67      void    freeAtom(WORD atom);
68      void    initIt(AtomIt* it, WORD atomid, WORD mem);
69      WORD    next(AtomIt* it);
70      void    dumpAtom();
71      void    moveAtoms(BYTE dstmem, BYTE srcmem);
72
73  // todo to private
74      Atoms *pixes;
75
76
77  private:
78      Atoms **atoms;
79      Atoms *proxies;
80      int maxArity;
81  };
82
83  #endif
```

## B.1.3 eyebotstub.h

```
 1  // dummy for eyebot library
 2
 3  #define OSExit(i) exit((i))
 4  #define LCDPrintf(f, ...) fprintf(stderr, f, ##__VA_ARGS__)
 5  #define KEYGet()  1
 6
 7  #define OSGetAD(x) 0
 8
 9  #define MOTORDrive(h, x)
10  #define SERVOSet(h, x)
11  #define QUADRead(h) 0
12  #define QUADReset(h)
13  #define OSReadInLatch(x) 0
14  #define OSWriteOutLatch(x, y, z)
15
16  #define OSWait(x)
17  #define LCDPutInt(x)
18  #define LCDPutChar(x)
19
20  #define CAMGetColFrame(x, y)
21  #define CAMInit(x)
```

## B.1.4 eyeio.cpp

```
1  #include "lmntal.h"
2  #include "eyeio.h"
3
```

```
 4  BYTE getbyte(){
 5  #ifdef PC
 6      return (BYTE)fgetc(stdin);
 7  #else
 8      char c;
 9      OSRecvRS232(&c, SERIAL1);
10      return (BYTE)c;
11  #endif
12  }
13
14  WORD getword(){
15      WORD w;
16      w = (WORD)getbyte() << 24;
17      w |= (WORD)getbyte() << 16;
18      w |= (WORD)getbyte() << 8;
19      w |= (WORD)getbyte();
20      return w;
21  }
22
23  void putbyte(BYTE b){
24  #ifdef PC
25      putchar(b);
26  #else
27      OSSendCharRS232((char)b, SERIAL1);
28  #endif
29  }
30
31  void putword(WORD w){
32      putbyte(w >> 24 & 0xff);
33      putbyte(w >> 16 & 0xff);
34      putbyte(w >> 8 & 0xff);
35      putbyte(w & 0xff);
36  }
37
```

### B.1.5 eyeio.h

```
1  extern void putbyte(BYTE b);
2  extern void putword(WORD w);
3  extern BYTE getbyte();
4  extern WORD getword();
```

### B.1.6 inst.h

```
 1  #ifndef INST_H
 2  #define INST_H
 3
 4  // 命令列の値を定義。値は本処理系と原則同じようにする。
 5  // eyebot システム命令は 1000 番以降
 6  #define FAIL        -1
 7
 8  #define UNDEF       0
 9  // アトムに関係する出力する基本ガード命令 (1--5)
10  // dereflink
11  #define DEREF       1
12  #define DEREFATOM   2
13  #define FINDATOM    4
14
15  // 膜に関係する（変数に）出力する基本ガード命令 (5--9)
16  // lockmem, lock, getmem, getparent
17  #define LOCKMEM         5
18  #define ANYMEM          6
```

```
19 // ロックはせず膜取得のみを行う。
20
21 // 膜に関係する出力しない基本ガード命令 (10--19)
22 // norules, nfreelinks, stable
23 #define TESTMEM     10
24 #define NATOMS      13
25 #define NMEMS       15
26 #define EQMEM       17
27 #define NEQMEM      18 // ロックをしないなら必要。
28
29 // アトムに関係する出力しない基本ガード命令 (20-24)
30 // notfunc
31 #define FUNC        20
32 #define EQATOM      22
33 #define NEQATOM     23
34 #define SAMEFUNC    24
35
36 // ファンクタに関係する命令 (25--29)
37 // dereffunc
38 #define GETFUNC     26
39 #define LOADFUNC    27
40 #define EQFUNC      28
41 #define NEQFUNC     29
42
43 // アトムを操作する基本ボディ命令 (30--39)
44 // enqueueatom, dequeueatom, alterfunc, alterfuncindirect, newatomindirect
45 #define REMOVEATOM  30
46 #define NEWATOM     31
47
48 // INT をつくる NEWATOM は NEWATOM_INT に変換する
49 #define NEWATOM_INT     39
50
51 #define FREEATOM    35
52
53 // アトムを操作する型付き拡張用命令 (40--49)
54 // addatom
55 #define ALLOCATOM   40
56 #define ALLOCATOMINDIRECT   41
57 #define COPYATOM    42
58
59 // 膜を操作する基本ボディ命令 (50--59)
60 // allocmem, newroot, enqueueallatoms, unlockmem, setmemname
61 #define REMOVEMEM   50
62 #define NEWMEM      51
63 #define MOVECELLS   54
64 #define FREEMEM     56
65 #define ADDMEM      57
66
67 // 予約 (60--62) リンクに関係する出力するガード命令 (63--64)
68 #define GETLINK     63
69 #define ALLOCLINK   64
70
71 // リンクを操作するボディ命令 (65--69)
72 // inheritlink
73 #define NEWLINK     65
74 #define RELINK      66
75 #define UNIFY       67
76 #define UNIFYLINKS  69
77
78 // 自由リンク管理アトム自動処理のためのボディ命令 (70--74)
79 #define REMOVEPROXIES   70
80 #define REMOVETOPLEVELPROXIES   71
81 #define INSERTPROXIES   72
82 #define REMOVETEMPORARYPROXIES  73
83
84 // ルールを操作するボディ命令 (75--79)
85 // loadruleset, copyrules, clearrules, loadmodule
```

```
 86
 87 // 型付きでないプロセス文脈をコピーまたは廃棄するための命令 (80--89)
 88 // recursivelock, recursiveunlock
 89 #define COPYMEM     82
 90 #define DROPMEM     83
 91
 92 // 予約 (90--99)
 93
 94 ////////////////////////////////////////////
 95 // 以下、制御命令
 96
 97 // 制御命令 (200--209)
 98 // jump, resetvars, changevars, spec, stop, branch, loop, run, not
 99 #define REACT       200
100 #define COMMIT      201
101 #define PROCEED     204
102 // commit 以下の命令は書き換えを行なう。割り込み禁止との関係で使える？
103
104 // 組み込み機能に関する命令（仮） (210--219)
105 // inline, builtin
106
107 // 型付きプロセス文脈を扱うための追加命令 (216--219)
108 // eqground
109
110 // 型検査のためのガード命令 (220--229)
111 // isground, getclass, getclassfunc, isintfunc
112 #define ISUNARY     221
113 #define ISINT       225
114
115 ////////////////////////////////////////
116 // 以下、演算用
117 // ファンクタ関係、浮動小数点は省略
118
119 // 整数用の組み込みボディ命令 (400--419)
120 #define IADD    400
121 #define ISUB    401
122 #define IMUL    402
123 #define IDIV    403
124 #define INEG    404
125 #define IMOD    405
126 #define INOT    410
127 #define IAND    411
128 #define IOR     412
129 #define IXOR    413
130
131 // 大小比較
132 #define ILT     420
133 #define ILE     421
134 #define IGT     422
135 #define IGE     423
136 #define IEQ     424
137 #define INE     425
138
139 ////////////////////////////////////////////
140 // eyebot 用システム関数 (1000-)
141
142 #endif
```

## B.1.7 lmntal.cpp

```
1 #include "lmntal.h"
2 #include "task.h"
3 #include "atom.h"
4 #include "mem.h"
```

```
 5 #include "eyeio.h"
 6
 7
 8 ////////////////////////////////////////
 9 // メイン
10
11 TIMER timer[MAX_TIMER];
12 int interruptFlag;
13
14 #ifndef PC
15
16 // グローバル変数定義
17 EYEHANDLE eyeHandle;
18
19 // タイマハンドラ
20 void timerHandler()
21 {
22     int i;
23
24     // カウンタを 1 ずつ増加
25     for(i=0; i < MAX_TIMER; i++) timer[i].count++;
26
27     // 閾値越すものがあったらフラグをたてて終了
28     interruptFlag = 0;
29     for(i=0; i < MAX_TIMER; i++){
30         if(timer[i].count >= timer[i].border){
31             interruptFlag = 1;
32             break;
33         }
34     }
35 }
36
37 #endif
38
39 int main(int argc, char** argv)
40 {
41     dbg("lmntal main start\n");
42
43     int taskcount;
44     WORD w, len, len2;
45     Task *tasks; // タスク配列
46
47
48     // timer の初期化
49     for(int i=0; i < MAX_TIMER; i++){
50         timer[i].count = 0;
51         timer[i].border = 0xffffffff; // 絶対越えない
52         timer[i].atomFunc = 0;
53         timer[i].taskNo = 0;
54         timer[i].once = 0;
55     }
56
57 #ifdef PC
58     // アドレスの offset を表示。。
59         // プログラムがマップされるメモリ空間があまり変わらないと信じて。
60     // lmntal.h の LINK_OFFSET の値がこれと等しくなるようにする。
61         // atom のアドレスの上位 8bit は LINK_OFFSET と同じでなければならない。
62
63     WORD *dummy;
64     dummy = new WORD;
65     dbg("link offset = %8x, address upper 8bit = %8x\n",
66            LINK_OFFSET, (WORD)dummy & 0xff000000);
67     delete dummy;
68 #else
69     // eyebot 初期化
70     OSInitRS232(SER115200, RTSCTS, SERIAL1);
71     OSFlushInRS232(SERIAL1);
```

```
72
73      // ハンドルの初期化
74      eyeHandle.leftmotor = MOTORInit(LEFTMOTOR);
75      eyeHandle.rightmotor = MOTORInit(RIGHTMOTOR);
76      eyeHandle.quadleft = QUADInit(QUAD_LEFT);
77      eyeHandle.quadright = QUADInit(QUAD_RIGHT);
78      eyeHandle.servo7 = SERVOInit(SERVO7);
79      eyeHandle.servo8 = SERVOInit(SERVO8);
80      eyeHandle.servo9 = SERVOInit(SERVO9);
81      eyeHandle.servo10 = SERVOInit(SERVO10);
82      eyeHandle.servo11 = SERVOInit(SERVO11);
83      eyeHandle.servo12 = SERVOInit(SERVO12);
84
85      // timerHandler の登録
86      eyeHandle.th = OSAttachTimer(1, timerHandler);
87
88      // カメラの初期化
89      CAMInit(NORMAL);
90
91  #endif
92
93      LCDPrintf("press button to load tal.\n");
94      KEYGet();
95
96      // 初期化コマンド読み込み
97      w = getword();
98
99      for(int i=0; i < w; i++){
100         // 初期化コマンド未実装。w はおそらく 0。何もせず終了。
101         getword(); // 読み飛ばし
102     }
103
104     // タスク生成・データ読み込み
105     taskcount = (int)getword(); // タスク数読み込み
106
107     tasks = new Task[taskcount];
108     Task::tasks = tasks;
109
110     for(int i=0; i < taskcount; i++){ // 各タスクの初期化
111         tasks[i].setId(i);
112         if(tasks[i].loadRules() != 0){
113             dbg("task[%d] loadRules failed.\n", i);
114             return -1;
115         }
116     }
117
118     LCDPrintf("press button to start\n");
119     KEYGet();
120
121     // 初期化ルールの適用
122     for(int i=0; i < taskcount; i++){
123         if(tasks[i].reactInitRule() != 0){
124             dbg("task[%d] reactInitRule failed.\n", i);
125             return -1;
126         }
127     }
128
129     dbg("******** initialize finished. ********\n\n");
130
131     // メインループ
132     int ret, endflag;
133     do {
134         endflag = 1;
135         for(int i=0; i < taskcount; i++){
136             if(interruptFlag == 1 || Task::interruptAtomCount > 0){
137                 endflag = 0;
138                 break;
```

```
139             }
140             ret = tasks[i].react();
141             if(!tasks[i].isStable()) endflag = 0;
142         }
143         // タイマ割り込み処理
144         if(interruptFlag == 1){ // タイマ割り込み発生
145             for(int j=0; j < MAX_TIMER; j++){
146                 if(timer[j].count > timer[j].border){
147                     timer[j].count -= timer[j].border;
148 
149                     // todo: 前に作ったアトムがまだ残ってるならなにもしない
150                     // 現在の仕様だと、前作ったアトムのアドレスがわからない
151                     // つかわないアトムを消すルールを書きましょう。
152 
153                     // タスクにアトム生成（ロックして行う？）
154                     Task::interruptAtomCount++;
155                     tasks[timer[j].taskNo].genAtom(timer[j].atomFunc);
156 
157                     if(timer[j].once == 1){ // 1 回だけのときは消去
158                         tasks[timer[j].taskNo].decTimerCount();
159                         timer[j].count = 0;
160                         timer[j].border = 0xffffffff;
161                         timer[j].atomFunc = 0;
162                         timer[j].taskNo = 0;
163                         timer[j].once = 0;
164                     }
165                 }
166             }
167         }
168 
169         // 割り込みアトム処理
170         if(Task::interruptAtomCount > 0){ // 割りこみアトムがある
171             dbg("interrupt occurred.\n");
172             for(int j = 0; j < taskcount; j++){
173                 // 割り込みアトム処理。ないタスクはすぐ帰ってくる
174                 tasks[j].reactInterruptAtom();
175             }
176         }
177 
178     }while(!endflag);
179 
180     dbg("**************react finished.**************\n");
181 
182     // 表示
183     LCDPrintf("press button to output data\n");
184     KEYGet();
185 
186     // タスク数出力
187     putword((WORD)taskcount);
188 
189     // 各タスクを出力
190     for(int i=0; i < taskcount; i++) tasks[i].dump();
191 
192     return 0;
193 }
```

## B.1.8 lmntal.h

```
1  #ifndef LMNTAL_H
2  #define LMNTAL_H
3 
4  // 命令列定義
5  #include "inst.h"
6  // 定数定義
```

```
 7 #include "lmntal_const.h"
 8
 9 ////////////////////////////////////////////
10 // for eyebot (for PC : define PC)
11 #ifndef PC
12 #include "eyebot.h"
13
14 #define LINK_OFFSET 0x0
15
16 #define dbg(f, ...)
17
18 typedef struct{
19     MotorHandle leftmotor;
20     MotorHandle rightmotor;
21     QuadHandle quadleft;
22     QuadHandle quadright;
23     // servo1-4 は encoder と競合、servo5,6 はコンパスと競合
24     ServoHandle servo7;
25     ServoHandle servo8;
26     ServoHandle servo9;
27     ServoHandle servo10;
28     ServoHandle servo11;
29     ServoHandle servo12;
30     TimerHandle th;
31 } EYEHANDLE;
32
33 extern EYEHANDLE eyeHandle;
34
35 ////////////////////////////////////////////
36 // for PC
37
38 #else
39
40 #include <stdio.h>
41 #include <stdlib.h>
42 typedef unsigned char BYTE;
43
44 // eyebot ライブラリのダミー
45 #include "eyebotstub.h"
46
47 // for GCC only? 可変長引数のマクロ
48 #define dbg(f, ...) fprintf(stderr, f, ##__VA_ARGS__)
49
50 // PC では new したメモリアドレスが 0x00ffffff を超えるので、0xff000000 部分を変更
51 // for win #define LINK_OFFSET 0x0a000000
52 #define LINK_OFFSET 0x08000000
53
54 typedef BYTE colimage[62][82][3];
55
56 #endif
57
58 ////////////////////////////////////////////
59 // 以下共通
60
61 typedef struct{
62     WORD count;
63     WORD border;
64     WORD atomFunc; // 生成するアトムのファンクタ名
65     int taskNo;  // 生成先タスク番号
66     int once;
67 } TIMER;
68
69 extern TIMER timer[MAX_TIMER];
70 extern int interruptFlag;
71
72 ////////////////////////////////////////////
73 // 以下マクロ定義
```

```
 74
 75 // atom は、対象アトムへのポインタを WORD 型にキャストした値で表すとする。
 76 // アトム領域への値の SET は newatom 時にしか行なわれない
 77
 78 // atom の functor を返す
 79 #define ATOM_GETFUNC(atom) (*((WORD*)(atom)) & 0x00ffffff)
 80 // atom の name を返す
 81 #define ATOM_GETNAME(atom) (*((WORD*)(atom)) & 0x0000ffff)
 82 // atom の arity を返す
 83 #define ATOM_GETARITY(atom) ((*((WORD*)(atom)) & 0x00ff0000) >> 16)
 84 // atom の所属膜 ID を返す
 85 #define ATOM_GETMEM(atom) ((*((WORD*)(atom)) & 0xff000000) >> 24)
 86
 87 // リンク atom1.pos1 の値が atom2.pos2 を指すようにする
 88 #define APOS_SETLINK(atom1, pos1, atom2, pos2) \
 89     (((WORD*)(atom1))[(pos1)+1] = \
 90     ((atom2) & 0x00ffffff | ((pos2)<<24) & 0xff000000))
 91 // リンク atom.pos に INT 型の値 i を格納する
 92 #define APOS_SETINT(atom, pos, i) \
 93     ((((WORD*)(atom))[(pos)+1]) = (i)<<2 | 0x01)
 94
 95 // リンク atom.pos のリンク先アトムを返す
 96 #define APOS_GETATOM(atom, pos) \
 97     (((((WORD*)(atom))[(pos)+1]) & 0x00ffffff) | LINK_OFFSET)
 98 // リンク atom.pos のリンク先リンク番号を返す
 99 #define APOS_GETPOS(atom, pos) \
100     (((((WORD*)(atom))[(pos)+1]) & 0xff000000) >> 24)
101
102 // リンク値をそのまま返す
103 #define APOS_GETLINK(atom, pos) \
104     (((WORD*)(atom))[(pos)+1])
105 // atom.pos を指すリンクを新規作成
106 #define APOS_ALLOCLINK(atom, pos) \
107     (((atom) & 0x00ffffff) | (((WORD)(pos) & 0xff) << 24))
108
109 /*
110 // リンク下位 2bit: x0=リンク, 01=INT, 11=STRING?(UNDEFINED)
111 // リンク atom.pos がリンクなら 0, データが格納されているなら 1 を返す
112 #define APOS_ISDATA(atom, pos) \
113     ((((WORD*)(atom))[(pos)+1]) & 0x00000001)
114 // リンク atom.pos に INT 型データが格納されているなら 1, それ以外なら 0 を返す
115 #define APOS_ISINT(atom, pos) \
116     (((((WORD*)(atom))[(pos)+1]) & 0x00000002) == 0x01 ? 1 : 0)
117 */
118
119 // リンクがリンクか？それともデータか？
120 #define LINK_ISLINK(link) (!((link) & 0x00000001))
121 #define LINK_ISINT(link) (!(((link) & 0x00000003) ^ 0x01))
122
123 #define LINK_GETATOM(link) (((link) & 0x00ffffff) | LINK_OFFSET)
124 #define LINK_GETPOS(link) (((link) >> 24) & 0xff)
125
126 // link1 が表すリンク先が、link2 が表すリンク先を指すようにする
127 #define LINK_SETLINK(link1, link2) \
128     ((((WORD*)LINK_GETATOM(link1))[LINK_GETPOS(link1) + 1]) = link2)
129
130 // LINK_GETLINK で得たリンクを整数値に直す (符号付 30bit 値)
131 #define LINK2INT(link) \
132     (((signed int)(link)) >> 2)
133 // INT 値を LINK 形式に変換 (allocatom 用)
134 #define INT2LINK(i) \
135     ((WORD)((i) << 2 | 0x01))
136
137
138
139 #endif
```

### B.1.9 lmntal_const.h

```
1  // 各種設定値
2  // MAX_MEM < 255
3  // MAX_ARITY <= 16
4  // MAX_TASK < 100
5
6  #define MAX_ARITY 16
7  #define MAX_MEM 250
8  #define BLOCK_SIZE 32
9  #define MAX_BLOCK 16
10 #define MAX_VARS 64
11 #define MAX_RULES 32
12 #define MAX_TASK 16
13 #define MAX_TIMER 16
14
15 #define EXECSTACK_LEN 16
16 #define GENSTACK_LEN 16
17 #define MAX_SYSATOM_INRULE 8
18
19 typedef unsigned int WORD;
20
21 #define FUNC_ARITY(func) (((func) >> 16) & 0xff)
22 #define FUNC_NAME(func) ((func) & 0xffff)
23
24 // システムアトム func（funcname は必ず奇数！）
25 #define PIX_7 0x70001
26
27 // システムアトム（アクティブ）：e000 以上
28 #define SYSATOM_BORDER 0xe000
29
30 // アクティブシステムアトムの個数（タスクが持つ配列長の決定）
31 // 奇数のみ、続き番号のこと。
32 #define MAX_ACTIVE_SYSATOM 18
33 #define ANALOG0_1 0x1e001
34 #define ANALOG1_1 0x1e003
35 #define ANALOG2_1 0x1e005
36 #define ANALOG3_1 0x1e007
37 #define ANALOG4_1 0x1e009
38 #define ANALOG5_1 0x1e00b
39 #define ANALOG6_1 0x1e00d
40 #define ANALOG7_1 0x1e00f
41 #define DIG_IN4_1 0x1e011
42 #define DIG_IN5_1 0x1e013
43 #define DIG_IN6_1 0x1e015
44 #define DIG_IN7_1 0x1e017
45 #define QUADLEFT_1 0x1e019
46 #define QUADRIGHT_1 0x1e01b
47 #define KEY_1 0x1e01d
48 #define RECV_2 0x2e01f
49 #define PICTURE_1 0x1e021
50 #define TICK_1 0x1e023
51
52 // ルールと関係付けられないシステムアトム（非アクティブ）：f000 以上
53 #define NONACTIVE_SYSATOM_BORDER 0xf000
54 #define GET_1 0x1f001
55 #define TIMER_2 0x2f003
56 #define TIMERONCE_2 0x2f005
57 #define TIMERKILL_2 0x2f007
58 #define LEFTMOTOR_1 0x1f009
59 #define RIGHTMOTOR_1 0x1f00b
60 #define PUTINT_1 0x1f00d
61 #define PUTCHAR_1 0x1f00f
62 #define DIG_OUT0_1 0x1f011
63 #define DIG_OUT1_1 0x1f013
64 #define DIG_OUT2_1 0x1f015
65 #define DIG_OUT3_1 0x1f017
```

```
66 #define DIG_OUT4_1 0x1f019
67 #define DIG_OUT5_1 0x1f01b
68 #define DIG_OUT6_1 0x1f01d
69 #define DIG_OUT7_1 0x1f01f
70 #define SERVO7_1 0x1f021
71 #define SERVO8_1 0x1f023
72 #define SERVO9_1 0x1f025
73 #define SERVO10_1 0x1f027
74 #define SERVO11_1 0x1f029
75 #define SERVO12_1 0x1f02b
76 #define QUADLEFT_RESET_0 0x0f02d
77 #define QUADRIGHT_RESET_0 0x0f02f
78 #define SEND_2 0x2f031
79 #define DIG_OUTALL_1 0x1f033
80 #define PICTURE_REMOVE_1 0x1f035
81
82 // 未実装：VW 制御、PSD、パラレル
83 // todo : 無線
84
85 // Proxy: fffa(65530) 以上
86 #define PROXY_BORDER 0xfffa
87 #define INSIDE_PROXY 0x02ffff
88 #define OUTSIDE_PROXY 0x02fffd
89 #define STAR_PROXY 0x02fffb
90
91
92
93
```

## B.1.10 mem.cpp

```
 1 #include "mem.h"
 2 #include "atom.h"
 3 #include "eyeio.h"
 4
 5 MembraneManager::MembraneManager(){
 6     // すべて freelist に。
 7
 8     mem = new MEMBRANE[MAX_MEM];
 9
10     // mem[1] : task root mem[0] : NULL
11     mem[1].parent = 0;
12     mem[1].next = 1;
13     mem[1].prev = 1;
14     mem[1].child = 0;
15     mem[1].atomCount = 0;
16
17     freelist = 2;
18     for(int i=1; i < MAX_MEM - 1; i++){
19         mem[i].next = i+1;
20         mem[i].parent = 0; // 空き領域は parent=0（使わない？）
21     }
22     mem[MAX_MEM - 1].next = 0; // freelist の最後は 0
23     mem[MAX_MEM - 1].parent = 0;
24 }
25
26 MembraneManager::~MembraneManager(void)
27 {
28     delete[] mem;
29 }
30
31 ////////////////////////////////////
32 // 以下メソッド
33
```

```
34
35  // 新しい膜をつくり、id を返す（親膜指定しない）
36  BYTE MembraneManager::newMem()
37  {
38      BYTE m;
39      m = freelist;
40      freelist = mem[freelist].next;
41
42      mem[m].atomCount = 0;
43
44      return m;
45  }
46
47  // id で指定された膜を膜ツリーから除去し、freelist に戻す。
48  void MembraneManager::freeMem(BYTE id)
49  {
50    if(mem[id].parent != 0){
51      dbg("unremoved mem %d cannot free.", id);
52    }
53      mem[id].next = mem[freelist].next;
54      mem[freelist].next = id;
55  }
56
57  // 膜 id を膜 parent の子として膜ツリーに追加する。
58  // id はツリーから除かれていること。(newMem か removeMem の直後)
59  void MembraneManager::addMem(BYTE id, BYTE parent)
60  {
61      BYTE m0, m1;
62
63      dbg("add %d to %d's child.\n", id, parent);
64
65      if(mem[parent].child == 0){ // 子がないとき
66          mem[parent].child = id;
67          mem[id].parent = parent;
68          mem[id].next = id;
69          mem[id].prev = id;
70      }else{
71          m0 = mem[parent].child;
72          m1 = mem[m0].next;
73
74          mem[id].parent = parent;
75          mem[id].prev = m0;
76          mem[m0].next = id;
77          mem[id].next = m1;
78          mem[m1].prev = id;
79      }
80      dbg("added mem %d 's parent is %d\n", id, mem[id].parent);
81      mem[id].child = 0;
82  }
83
84  // srcmem の子膜を dstmem の下に移動する。
85  void MembraneManager::moveMems(BYTE dstmem, BYTE srcmem)
86  {
87      BYTE child1, child2, next1, next2;
88
89      child1 = mem[dstmem].child;
90      child2 = mem[srcmem].child;
91
92      if(child2 != 0){ // 移動する子膜がある
93          mem[child1].parent = dstmem;
94          if(child1 == 0){ // dst 子膜なし
95              mem[dstmem].child = child2;
96          }else{ // dst 子膜あり＞連結
97              next1 = mem[child1].next;
98              next2 = mem[child2].next;
99              mem[child1].next = next2;
100             mem[next2].prev = child1;
```

```
101             mem[child2].next = next1;
102             mem[next1].prev = child2;
103         }
104     }
105 }
106
107 // 膜 id を膜ツリーから除去する（どこにも属さない膜にして、$p として移動可にする）
108 void MembraneManager::removeMem(BYTE id)
109 {
110     BYTE m0, m1, mp;
111
112     if(mem[id].parent == 0){
113         dbg("removemem : mem %d belongs nowhere.\n", id);
114     }
115
116     // 兄弟膜リストから自分を削除、自分が子膜代表なら親膜が別の子膜を指すように
117     mp = mem[id].parent;
118     m0 = mem[id].prev;
119     m1 = mem[id].next;
120     if(m0 == id){ // 最後の膜
121         mem[mp].child = 0; // 子膜なし
122     }else{
123         if(mem[mp].child == id){ // 消す膜が子代表のとき
124             mem[mp].child = m1;
125         }
126         mem[m0].next = m1;
127         mem[m1].prev = m0;
128     }
129     mem[id].parent = 0; // 所属膜なしにする。二重消去防止
130 }
131
132 // 子膜連続取得用反復子の初期化
133 void MembraneManager::initIt(MemIt* it, BYTE parent)
134 {
135     it->parent = parent;
136     it->index = mem[parent].child;
137 }
138
139 BYTE MembraneManager::next(MemIt* it)
140 {
141     BYTE ret;
142
143     ret = it->index;
144     if(it->index != 0){
145         it->index = mem[it->index].next;
146         if(it->index == mem[it->parent].child) // 一周した
147             it->index = 0; // 次は null を返す。
148     }
149     return ret;
150 }
151
152 void MembraneManager::dumpMem(BYTE memID)
153 {
154     BYTE c;
155     putbyte('\xff'); // 膜開始
156     putbyte(memID);  // 膜 ID
157 dbg("output memID : %d, child : %d\n", memID, mem[memID].child);
158     c = mem[memID].child;
159
160     if(c != 0){ // 子膜があるとき
161         do{
162             dumpMem(c); // 再帰出力
163             c = mem[c].next; // c に兄弟膜をセット
164         }while(c != mem[memID].child); // c が未出力なら繰り返し
165     }
166     putbyte('\x00'); // 膜終了
167 }
```

```
168
169 ////////////////////////////////////////////
170 // 以下、proxy 関連
171 // see Java 処理系 AbstractMembrane.java
172
173 #define ATOM_RENAME(atom, newfunc) \
174     ((*(WORD*)(atom)) = ((*(WORD*)(atom)) & 0xffff0000) | newfunc & 0xffff)
175
176 // 膜 mem の余分な Proxy を除去
177 void MembraneManager::removeProxies(BYTE thismem, AtomManager &am)
178 {
179   WORD outside, a0, a1;
180   AtomIt ait;
181
182   am.initIt(&ait, OUTSIDE_PROXY, thismem);
183
184   while((outside = am.next(&ait)) != 0){
185     a0 = APOS_GETATOM(outside, 0);
186
187     // outside のリンク先が子膜でないとき
188     if(mem[ATOM_GETMEM(a0)].parent != thismem){
189       a1 = APOS_GETATOM(outside, 1);
190
191       if((ATOM_GETFUNC(a1) == INSIDE_PROXY) ||
192      (ATOM_GETFUNC(a1) == OUTSIDE_PROXY &&
193       mem[ATOM_GETMEM(a1)].parent != thismem)){
194     // この膜を通過して親膜に出ていくリンク、もしくは
195     // この膜を通過して無関係な膜に入っていくリンクを消去
196     WORD link1, link2;
197     link1 = APOS_GETLINK(outside, 0);
198     link2 = APOS_GETLINK(a1, 0);
199
200     // proxy に直結する整数アトムはないとする。
201     // outside, a1 のリンク先を直結し、outside, a1 を消去
202     LINK_SETLINK(link1, link2);
203     LINK_SETLINK(link2, link1);
204     am.freeAtom(outside);
205     am.freeAtom(a1);
206       }else{
207     // 上記条件以外の outside アトムは自由リンクなので
208         // 名前を star に改名する。
209     ATOM_RENAME(outside, STAR_PROXY);
210       }
211     }
212   }
213   // 残っているすべての insideproxy を star にする。
214   am.initIt(&ait, INSIDE_PROXY, thismem);
215   while((a0 = am.next(&ait)) != 0){
216     ATOM_RENAME(a0, STAR_PROXY);
217   }
218 }
219
220 // 本膜において、2つの outside 経由で他の膜に入るリンクを除去
221 void MembraneManager::removeTopLevelProxies(BYTE thismem, AtomManager &am)
222 {
223   AtomIt ait;
224   WORD outside, a0, a1, a2;
225
226   am.initIt(&ait, OUTSIDE_PROXY, thismem);
227
228   while((outside = am.next(&ait)) != 0){
229     a0 = APOS_GETATOM(outside, 0);
230     // outside のリンク先が子膜でない
231     if(mem[ATOM_GETMEM(a0)].parent != thismem){
232       // ATOM_GETMEM(a0) != 0 はいらない？
233       a1 = APOS_GETATOM(outside, 1);
234       if(ATOM_GETFUNC(a1) == OUTSIDE_PROXY){
```

```
235       a2 = APOS_GETATOM(a1, 0);
236       if(mem[ATOM_GETMEM(a2)].parent != thismem){
237         // ATOM_GETMEM(a2) != 0 はいらない？
238         WORD link1, link2;
239         link1 = APOS_GETLINK(outside, 0);
240         link2 = APOS_GETLINK(a1, 0);
241         LINK_SETLINK(link1, link2);
242         LINK_SETLINK(link2, link1);
243         am.freeAtom(outside);
244         am.freeAtom(a1);
245       }
246         }
247       }
248     }
249   }
250
251   // 子膜の star を取得、それとそのリンク先を改名 or 除去
252   void MembraneManager::insertProxies(BYTE thismem, BYTE childmem, AtomManager &am)
253   {
254     AtomIt ait;
255     WORD star, oldstar;
256
257     am.initIt(&ait, STAR_PROXY, childmem);
258
259     while((star = am.next(&ait)) != 0){
260       oldstar = APOS_GETATOM(star, 0);
261       if(ATOM_GETMEM(oldstar) == childmem){ // 膜内の新しい局所リンク？
262         WORD link1, link2;
263         link1 = APOS_GETLINK(star, 1);
264         link2 = APOS_GETLINK(oldstar, 1);
265         LINK_SETLINK(link1, link2);
266         LINK_SETLINK(link2, link1);
267         am.freeAtom(star);
268         am.freeAtom(oldstar);
269       }else{
270         // star を inside_proxy に改名
271         ATOM_RENAME(star, INSIDE_PROXY);
272
273         // 反対側のアトムがこの膜のアトムなら outside_proxy に改名
274         if(ATOM_GETMEM(oldstar) == thismem){
275       ATOM_RENAME(oldstar, OUTSIDE_PROXY);
276       // newlink(oldstar, 0, star, 0) は要らない？
277         }else{
278       // 通過するだけなら outside, star をこの膜に新規作成
279       WORD outside, newstar;
280       WORD link1, link2;
281
282       outside = am.newAtom(OUTSIDE_PROXY, thismem);
283       newstar = am.newAtom(STAR_PROXY, thismem);
284
285       // newlink
286       link1 = APOS_ALLOCLINK(outside, 1);
287       link2 = APOS_ALLOCLINK(newstar, 1);
288       LINK_SETLINK(link1, link2);
289       LINK_SETLINK(link1, link2);
290       // relink
291       link1 = APOS_ALLOCLINK(newstar, 0);
292       link2 = APOS_GETLINK(star, 0); // oldstar get
293           LINK_SETLINK(link1, link2);
294       LINK_SETLINK(link2, link1);
295       // newlink
296       link1 = APOS_ALLOCLINK(star, 0);
297       link2 = APOS_ALLOCLINK(outside, 0);
298       LINK_SETLINK(link1, link2);
299       LINK_SETLINK(link2, link1);
300         }
301       }
```

```
302   }
303 }
304
305 // 最後に本膜に残った star を削除
306 void MembraneManager::removeTemporaryProxies(BYTE thismem, AtomManager &am)
307 {
308   AtomIt ait;
309   WORD star, outside;
310
311   am.initIt(&ait, STAR_PROXY, thismem);
312
313   while((star = am.next(&ait)) != 0){
314     outside = APOS_GETATOM(star, 0); // star の先は必ず star or outside
315
316     // リンク張り替えて消去
317     WORD link1, link2;
318     link1 = APOS_GETLINK(star, 1);
319     link2 = APOS_GETLINK(outside, 1);
320     LINK_SETLINK(link1, link2);
321     LINK_SETLINK(link2, link1);
322     am.freeAtom(star);
323     am.freeAtom(outside);
324   }
325 }
326
```

## B.1.11 mem.h

```
 1 #ifndef MEM_H
 2 #define MEM_H
 3
 4 #include "lmntal.h"
 5 #include "atom.h"
 6
 7 // 領域節約の為、膜は通し番号 ( < 255) で管理。
 8 // mem[0] : NULL の代わり。 mem[1] : root 膜
 9 typedef struct mem_t{
10     BYTE parent;
11     BYTE next;
12     BYTE prev;
13     BYTE child;
14     short atomCount;
15 } MEMBRANE;
16
17 typedef struct{
18     BYTE index;
19     BYTE parent;
20 } MemIt;
21
22 class MembraneManager{
23 public:
24     MembraneManager(void);
25     ~MembraneManager(void);
26     BYTE newMem();
27     void freeMem(BYTE id);
28     void addMem(BYTE id, BYTE parent);
29     void removeMem(BYTE id);
30     void initIt(MemIt* it, BYTE parent);
31     BYTE next(MemIt* it);
32     void dumpMem(BYTE memID);
33     void moveMems(BYTE dstmem, BYTE srcmem);
34 // proxy 操作関連。atom は直接アクセスと am を通しての操作のみ。
35     void    removeProxies(BYTE thismem, AtomManager &am);
36     void    removeTopLevelProxies(BYTE thismem, AtomManager &am);
```

```
37      void    insertProxies(BYTE thismem, BYTE childmem, AtomManager &am);
38      void    removeTemporaryProxies(BYTE thismem, AtomManager &am);
39
40      short getAtomCount(BYTE id) { return mem[id].atomCount; }
41      short incAtomCount(BYTE id) { return ++(mem[id].atomCount); }
42      short decAtomCount(BYTE id) { return --(mem[id].atomCount); }
43
44
45      MEMBRANE *mem;
46  private:
47      BYTE freelist;
48  };
49
50  #endif
```

## B.1.12 task.cpp

```
 1  #include "lmntal.h"
 2  #include "task.h"
 3  #include "eyeio.h"
 4
 5  Task::Task(void){
 6      for(int i=0; i < MAX_RULES; i++) rules[i] = NULL;
 7      execStackHead = 0;
 8      genStackHead = 0;
 9      nextreact = 0;
10      remainRuleFlag = 1;
11      timerCount = 0;
12  }
13
14  Task::~Task(void){
15      for(int i=0; i < MAX_RULES; i++){
16          delete[] rules[i];
17      }
18      for(int i=0; i < MAX_ACTIVE_SYSATOM; i++){
19          if(relatedRule[i].count > 0){
20              delete[] relatedRule[i].buf;
21          }
22      }
23  }
24
25  void Task::dump(){
26      am.dumpAtom();
27      mm.dumpMem(1);
28  }
29
30  // 実行アトムスタック（および生成待ちアトムスタック？）に積まれているアトムに処理。
31  int Task::interruptAtomCount = 0;
32
33  // タスク間通信のとき、他タスクの生成待ちスタックにアクセスするために必要
34  Task * Task::tasks;
35
36
37  ////////////////////////////////////////////////////////////////////
38  // 以下ルール関係
39
40  // ルールを読み込み
41  int Task::loadRules()
42  {
43      int i, j;
44      WORD w;
45
46      // 初期化ルール読み込み
47      w = getword(); // データ長
```

```
48      initrule = new int[w];
49      for(i=0; i < w; i++){
50          initrule[i] = (int)getword();
51      }
52
53      // タスク膜内のルールの読み込み
54      for(i=0; i < MAX_RULES; i++){
55          w = getword();
56          if(w == 0) break;
57
58          rules[i] = new int[w+1];
59          for(j=0; j < w; j++){
60              rules[i][j] = (int)getword();
61          }
62          rules[i][j] = UNDEF; // 末尾 UNDEF=0 にしておく
63      }
64      if(i==0){ // 1 個もルールが無いタスクには、必ず失敗するルールを作って入れる
65          rules[0] = new int[1];
66          rules[0][0] = FAIL;
67      }
68
69      // システムアトムに関連するルール情報を読み込み
70      for(i=0; i < MAX_ACTIVE_SYSATOM; i++){
71          relatedRule[i].count = getword();
72          if(relatedRule[i].count != 0){
73              relatedRule[i].buf = new int[relatedRule[i].count];
74              for(j = 0; j < relatedRule[i].count; j++){
75                  relatedRule[i].buf[j] = getword();
76              }
77          }
78      }
79      return 0;
80  }
81
82  // システムアトムを扱う
83  int Task::reactsysatom()
84  {
85      int j;
86      WORD atom, w, value, task;
87      dbg("sysatomcount: %d\n", sysatom_count);
88
89      for(int i=0; i < sysatom_count; i++){
90          atom = sysatom_buf[i];
91          dbg("sysatombuf[%d]=%x, value=%x\n", i, atom, ATOM_GETFUNC(atom));
92          switch(ATOM_GETFUNC(atom)){
93          case GET_1: // get_1
94              // get を消し、リンク先アトムに値を設定して、実行スタックに積む
95              // リンク先アトムに値設定
96              w = APOS_GETATOM(atom, 0);
97              getSysValue(w);
98              // 実行スタックに積む
99              interruptAtomCount++;
100             execStack[execStackHead++] = w;
101             dbg("get: interruptAtomCount: %d\n", interruptAtomCount);
102             // get は消去
103             mm.decAtomCount(ATOM_GETMEM(atom));
104             am.freeAtom(atom);
105             dbg("get\n");
106             break;
107
108         case TIMERONCE_2:
109         case TIMER_2: // timer_2
110             w = APOS_GETATOM(atom, 1); // analog など
111             value = LINK2INT(APOS_GETLINK(atom, 0));
112
113             for(j = 0; j < MAX_TIMER; j++){
114                 // timer の空きノードを探す
```

```
115                 if(timer[j].border == 0xffffffff) break;
116             }
117             if(j == MAX_TIMER){
118                 dbg("timer overflew\n");
119                 return -1;
120             }
121             // タイマに登録
122             timer[j].count = 1; // 0xffffffff を越さないように、1 から始める
123             timer[j].border = value;
124             timer[j].taskNo = id; // このタスクの ID
125             timer[j].atomFunc = ATOM_GETFUNC(w);
126 
127             // 1 回だけで消すか
128             if(ATOM_GETFUNC(atom) == TIMERONCE_2){
129                 LCDPrintf("regist timeronce");
130                 timer[j].once = 1;
131             }else{
132                 timer[j].once = 0;
133             }
134 
135             // アトムの消去
136             am.freeAtom(atom);
137             am.freeAtom(w);
138             mm.decAtomCount(ATOM_GETMEM(atom));
139             mm.decAtomCount(ATOM_GETMEM(w));
140             LCDPrintf("timer regist\n");
141 
142             // タイマカウントを増やす
143             timerCount++;
144             break;
145 
146         case TIMERKILL_2: // kill_2 timer で登録されたノードを消去
147             w = APOS_GETATOM(atom, 1); // analog など
148             value = LINK2INT(APOS_GETLINK(atom, 0));
149 
150             // w, value が登録されてるノードを見付けたら消す
151             for(j=0; j < MAX_TIMER; j++){
152 //              LCDPrintf("%d=%d,%d=%d,%d=%d\n",timer[j].atomFunc, ATOM_GETFUNC(w),
153 //                        timer[j].border, value, timer[j].taskNo, id);
154 
155                 if(timer[j].atomFunc == ATOM_GETFUNC(w) &&
156                    timer[j].border == value &&
157                    timer[j].taskNo == id){
158                     // 見付けたら border を 0xffffffff（空き）にする
159                     timer[j].count = 0;
160                     timer[j].border = 0xffffffff;
161                     timer[j].atomFunc = 0;
162                     timer[j].taskNo = 0;
163 
164                     // タイマカウントを減らす
165 
166                     timerCount--;
167                     break;
168                 }
169             }
170 
171             // アトムの消去
172             am.freeAtom(atom);
173             am.freeAtom(w);
174             mm.decAtomCount(ATOM_GETMEM(atom));
175             mm.decAtomCount(ATOM_GETMEM(w));
176 
177             break;
178         case PUTINT_1:
179             w = APOS_GETLINK(atom, 0);
180             LCDPutInt(LINK2INT(w));
181             mm.decAtomCount(ATOM_GETMEM(atom));
```

```
182             am.freeAtom(atom);
183             break;
184         case PUTCHAR_1:
185             w = APOS_GETLINK(atom, 0);
186             LCDPutChar((char)LINK2INT(w));
187             mm.decAtomCount(ATOM_GETMEM(atom));
188             am.freeAtom(atom);
189             break;
190         case QUADLEFT_RESET_0:
191             QUADReset(eyeHandle.quadleft);
192             mm.decAtomCount(ATOM_GETMEM(atom));
193             am.freeAtom(atom);
194             break;
195         case QUADRIGHT_RESET_0:
196             QUADReset(eyeHandle.quadright);
197             mm.decAtomCount(ATOM_GETMEM(atom));
198             am.freeAtom(atom);
199             break;
200         case LEFTMOTOR_1:
201             w = APOS_GETLINK(atom, 0);
202             MOTORDrive(eyeHandle.leftmotor, LINK2INT(w));
203             mm.decAtomCount(ATOM_GETMEM(atom));
204             am.freeAtom(atom);
205             break;
206         case RIGHTMOTOR_1:
207             w = APOS_GETLINK(atom, 0);
208             MOTORDrive(eyeHandle.rightmotor, LINK2INT(w));
209             mm.decAtomCount(ATOM_GETMEM(atom));
210             am.freeAtom(atom);
211             break;
212         case SERVO7_1:
213             w = APOS_GETLINK(atom, 0);
214             SERVOSet(eyeHandle.servo7, LINK2INT(w));
215             mm.decAtomCount(ATOM_GETMEM(atom));
216             am.freeAtom(atom);
217             break;
218         case SERVO8_1:
219             w = APOS_GETLINK(atom, 0);
220             SERVOSet(eyeHandle.servo8, LINK2INT(w));
221             mm.decAtomCount(ATOM_GETMEM(atom));
222             am.freeAtom(atom);
223             break;
224         case SERVO9_1:
225             w = APOS_GETLINK(atom, 0);
226             SERVOSet(eyeHandle.servo9, LINK2INT(w));
227             mm.decAtomCount(ATOM_GETMEM(atom));
228             am.freeAtom(atom);
229             break;
230         case SERVO10_1:
231             w = APOS_GETLINK(atom, 0);
232             SERVOSet(eyeHandle.servo10, LINK2INT(w));
233             mm.decAtomCount(ATOM_GETMEM(atom));
234             am.freeAtom(atom);
235             break;
236         case SERVO11_1:
237             w = APOS_GETLINK(atom, 0);
238             SERVOSet(eyeHandle.servo11, LINK2INT(w));
239             mm.decAtomCount(ATOM_GETMEM(atom));
240             am.freeAtom(atom);
241             break;
242         case SERVO12_1:
243             w = APOS_GETLINK(atom, 0);
244             SERVOSet(eyeHandle.servo12, LINK2INT(w));
245             mm.decAtomCount(ATOM_GETMEM(atom));
246             am.freeAtom(atom);
247             break;
248         case DIG_OUTALL_1:
```

```
249             w = APOS_GETLINK(atom, 0);
250             OSWriteOutLatch(0, ~0xff, LINK2INT(w) & 0xff);
251             mm.decAtomCount(ATOM_GETMEM(atom));
252             am.freeAtom(atom);
253             break;
254         case DIG_OUT0_1:
255             w = APOS_GETLINK(atom, 0);
256             OSWriteOutLatch(0, ~0x01, (LINK2INT(w) & 1) << 0);
257             mm.decAtomCount(ATOM_GETMEM(atom));
258             am.freeAtom(atom);
259             break;
260
261         case DIG_OUT1_1:
262             w = APOS_GETLINK(atom, 0);
263             OSWriteOutLatch(0, ~0x02, (LINK2INT(w) & 1) << 1);
264             mm.decAtomCount(ATOM_GETMEM(atom));
265             am.freeAtom(atom);
266             break;
267         case DIG_OUT2_1:
268             w = APOS_GETLINK(atom, 0);
269             OSWriteOutLatch(0, ~0x04, (LINK2INT(w) & 1) << 2);
270             mm.decAtomCount(ATOM_GETMEM(atom));
271             am.freeAtom(atom);
272             break;
273         case DIG_OUT3_1:
274             w = APOS_GETLINK(atom, 0);
275             OSWriteOutLatch(0, ~0x08, (LINK2INT(w) & 1) << 3);
276             mm.decAtomCount(ATOM_GETMEM(atom));
277             am.freeAtom(atom);
278             break;
279         case DIG_OUT4_1:
280             w = APOS_GETLINK(atom, 0);
281             OSWriteOutLatch(0, ~0x10, (LINK2INT(w) & 1) << 4);
282             mm.decAtomCount(ATOM_GETMEM(atom));
283             am.freeAtom(atom);
284             break;
285         case DIG_OUT5_1:
286             w = APOS_GETLINK(atom, 0);
287             OSWriteOutLatch(0, ~0x20, (LINK2INT(w) & 1) << 5);
288             mm.decAtomCount(ATOM_GETMEM(atom));
289             am.freeAtom(atom);
290             break;
291         case DIG_OUT6_1:
292             w = APOS_GETLINK(atom, 0);
293             OSWriteOutLatch(0, ~0x40, (LINK2INT(w) & 1) << 6);
294             mm.decAtomCount(ATOM_GETMEM(atom));
295             am.freeAtom(atom);
296             break;
297         case DIG_OUT7_1:
298             w = APOS_GETLINK(atom, 0);
299             OSWriteOutLatch(0, ~0x80, (LINK2INT(w) & 1) << 7);
300             mm.decAtomCount(ATOM_GETMEM(atom));
301             am.freeAtom(atom);
302             break;
303         case SEND_2:
304         {
305             int arg1, arg2;
306             arg1 = LINK2INT(APOS_GETLINK(atom, 0)); // 生成先タスク
307             arg2 = LINK2INT(APOS_GETLINK(atom, 1)); // 送る値
308
309             // 他の ID のタスクの生成待ちスタックにじかに積む
310             tasks[arg1].genStack[tasks[arg1].genStackHead++] = (WORD)arg2; // 値
311             tasks[arg1].genStack[tasks[arg1].genStackHead++] = (WORD)id; // id
312             tasks[arg1].genStack[tasks[arg1].genStackHead++] = RECV_2; // ファンクタ
313
314             // 割り込みアトム数を＋ 1
315             interruptAtomCount++;
```

```
316
317             mm.decAtomCount(ATOM_GETMEM(atom));
318             am.freeAtom(atom);
319             break;
320         }
321         case PICTURE_REMOVE_1:
322             // 必ず左上端の pix と接続しているものとする。
323             // 全ての pix を消去
324             WORD p, l, temp;
325             p = APOS_GETATOM(atom, 0);
326
327             while(1){
328                 l = APOS_GETATOM(p, 2);
329                 while(LINK_ISLINK(APOS_GETLINK(l, 2))){ // 右端にくるまで
330                     temp = APOS_GETATOM(l, 2); // NX のリンク先
331                     am.freeAtom(l); // l を消去。
332                     // 生成時に atom 数増やさなかったので、消すときも減らさない
333                     l = temp; // 右隣を新しい l とする
334                 }
335                 am.freeAtom(l); // 右端を消す
336
337                 if(LINK_ISINT(APOS_GETLINK(p, 3))) break; // 最下行なら終了
338                 temp = APOS_GETATOM(p, 3); // 下隣を新しい p とする。旧 p は消去
339                 am.freeAtom(p);
340                 p = temp;
341             }
342             // 左下端の pix を消去
343             am.freeAtom(p);
344             // 最後に picture_remove 自身を消去
345             mm.decAtomCount(ATOM_GETMEM(atom));
346             am.freeAtom(atom);
347             break;
348         default:
349             dbg("unknown sysatom %x\n", ATOM_GETFUNC(atom));
350             break;
351         }
352     }
353     return 0;
354 }
355
356 // アトム ID に応じて、センサなどから値を読み込み、設定する。
357 // atom はタスクルート膜に生成済み、かつリンク先を上書きしてよいアトム
358 int Task::getSysValue(WORD atom)
359 {
360     WORD link1, link2;
361
362     link1 = APOS_ALLOCLINK(atom, 0);
363
364     switch(ATOM_GETFUNC(atom)){
365     case ANALOG0_1:
366         link2 = INT2LINK(OSGetAD(0));
367         break;
368     case ANALOG1_1:
369         link2 = INT2LINK(OSGetAD(1));
370         break;
371     case ANALOG2_1:
372         link2 = INT2LINK(OSGetAD(2));
373         break;
374     case ANALOG3_1:
375         link2 = INT2LINK(OSGetAD(3));
376         break;
377     case ANALOG4_1:
378         link2 = INT2LINK(OSGetAD(4));
379         break;
380     case ANALOG5_1:
381         link2 = INT2LINK(OSGetAD(5));
382         break;
```

```
383     case ANALOG6_1:
384         link2 = INT2LINK(OSGetAD(6));
385         break;
386     case ANALOG7_1:
387         link2 = INT2LINK(OSGetAD(7));
388         break;
389     case DIG_IN4_1:
390         link2 = INT2LINK((OSReadInLatch(0) >> 4) & 1);
391         break;
392     case DIG_IN5_1:
393         link2 = INT2LINK((OSReadInLatch(0) >> 5) & 1);
394         break;
395     case DIG_IN6_1:
396         link2 = INT2LINK((OSReadInLatch(0) >> 6) & 1);
397         break;
398     case DIG_IN7_1:
399         link2 = INT2LINK((OSReadInLatch(0) >> 7) & 1);
400         break;
401     case QUADLEFT_1:
402         link2 = INT2LINK(QUADRead(eyeHandle.quadleft));
403         break;
404     case QUADRIGHT_1:
405         link2 = INT2LINK(QUADRead(eyeHandle.quadright));
406         break;
407     case PICTURE_1:
408         link2 = getPicture();
409         LINK_SETLINK(link2, link1);
410         break;
411     case TICK_1:
412         link2 = INT2LINK(0);
413         break;
414     default:
415         dbg("getSysValue: failed. atomname: %d\n", ATOM_GETNAME(atom));
416         return -1;
417     }
418     LINK_SETLINK(link1, link2);
419     return 0;
420 }
421
422 // 画像を取得し、pix を生成する
423 // 返り値：picture と接続する pix.link0
424 int Task::getPicture()
425 {
426     static colimage colbuf;
427     static WORD line[80];
428
429     int x, y, link1, link2;
430     WORD w, temp, pix00;
431
432     // 画像取得
433     CAMGetColFrame(&colbuf, 0);
434     LCDPrintf("image captured\n");
435
436     // 左上端の pix を保持しておく
437     pix00 = temp = am.newAtom(PIX_7, 1);
438
439     for(y = 0; y < 60; y++){
440         for(x = 0; x < 80; x++){
441             // 左
442             link1 = APOS_ALLOCLINK(temp, 0);
443             if(x == 0){
444                 link2 = INT2LINK(0); // 左端
445                 LINK_SETLINK(link1, link2);
446             }else{
447                 link2 = APOS_ALLOCLINK(line[x-1], 2); // 左隣の NX
448                 LINK_SETLINK(link1, link2);
449                 LINK_SETLINK(link2, link1);
```

```
450             }
451             // 上
452             link1 = APOS_ALLOCLINK(temp, 1);
453             if(y == 0){
454                 link2 = INT2LINK(0); // 上端
455                 LINK_SETLINK(link1, link2);
456             }else{
457                 link2 = APOS_ALLOCLINK(line[x], 3); // 上隣の NY
458                 LINK_SETLINK(link1, link2);
459                 LINK_SETLINK(link2, link1);
460             }
461             // 右
462             link1 = APOS_ALLOCLINK(temp, 2);
463             if(x == 79){
464                 link2 = INT2LINK(0); // 右端
465                 LINK_SETLINK(link1, link2);
466             }
467             // 下
468             link1 = APOS_ALLOCLINK(temp, 3);
469             if(y == 59){
470                 link2 = INT2LINK(0); // 下端
471                 LINK_SETLINK(link1, link2);
472             }
473             LINK_SETLINK(APOS_ALLOCLINK(temp, 4), INT2LINK(colbuf[y+1][x+1][0]));
474             LINK_SETLINK(APOS_ALLOCLINK(temp, 5), INT2LINK(colbuf[y+1][x+1][1]));
475             LINK_SETLINK(APOS_ALLOCLINK(temp, 6), INT2LINK(colbuf[y+1][x+1][2]));
476
477             // 前行保持バッファに格納
478             line[x] = temp;
479             // 次の pix を生成
480             temp = am.newAtom(PIX_7, 1);
481         }
482     }
483     // 最後の 1 個作りすぎた pix を消去（毎ループ if を使うより 1 個消した方が早そう）
484     am.freeAtom(temp);
485
486     return APOS_ALLOCLINK(pix00, 0);
487 }
488
489
490 ////////////////////////////////////////////////////////
491 // 以下 react 関係
492
493 // 割り込みアトムの処理。生成アトムスタックにアトム ID その他が入っていれば処理。
494 // 実行アトムスタックにアトムが入っていれば、関係するルールを適用する
495 // 実行アトムスタックのアトム処理を行った場合、ルール全ての再適用を行うように
496 // 各フラグの再初期化を行う。
497
498 int Task::reactInterruptAtom()
499 {
500     int i, j, index;
501     WORD atom, w;
502
503     // 生成待ちアトムの処理
504     while(genStackHead != 0){
505         // 登録されてるファンクタ名で分岐
506         w = genStack[--genStackHead];
507
508         switch(w){
509         case RECV_2:
510         {
511             WORD link1, link2;
512             // さらに 2word 読み出す
513             link1 = INT2LINK(genStack[--genStackHead]); // id
514             link2 = INT2LINK(genStack[--genStackHead]); // 値
515             // アトムを生成して値を登録
516             atom = am.newAtom(w, 1);
```

```
517             mm.incAtomCount(1);
518             LINK_SETLINK(APOS_ALLOCLINK(atom, 0), link1);
519             LINK_SETLINK(APOS_ALLOCLINK(atom, 1), link2);
520             // 実行アトムスタックに積む
521             execStack[execStackHead++] = atom;
522             break;
523         }
524         default:
525             // タスクルート膜（1）にアトム生成
526             atom = am.newAtom(w, 1);
527             mm.incAtomCount(1);
528             // 適切な値をセンサなどから読み出し
529             getSysValue(atom);
530             // 実行アトムスタックに積む
531             execStack[execStackHead++] = atom;
532             break;
533         }
534     }
535
536     if(execStackHead != 0){ // 実行アトムスタックに積まれている場合。
537         // 各種フラグの初期化
538         nextreact = 0;
539         remainRuleFlag = 1;
540
541         while(execStackHead != 0){
542             // 取り出す
543             // todo: これに積まれてるアトムが直前のルール適用によって消されてるかもしれない？
544             // もしくは書き換わっているかもしれない？
545             // システムアトムを扱う普通のルールを書いた場合、普通のルールの適用が先に行われる
546             // と、アトムが消される（or 消されて別の場所に生成される）ことがある。注意。
547             atom = execStack[--execStackHead];
548             interruptAtomCount--;
549
550             // アトムに関連づけられたルールを適用
551             dbg("reactinterruptatom: %x\n", ATOM_GETFUNC(atom));
552             index = (ATOM_GETNAME(atom) - SYSATOM_BORDER) / 2;
553             dbg("index:%x, name=%x\n", index, ATOM_GETNAME(atom));
554             for(j=0; j < relatedRule[index].count; j++){
555                 dbg("count:%d, buf[%d]=%d\n", relatedRule[index].count, j,
556                     relatedRule[index].buf[j]);
557                 react(relatedRule[index].buf[j], atom);
558             }
559             dbg("react finished.\n");
560         }
561     }
562     return 0;
563 }
564
565 // 初期化ルールを実行
566 int Task::reactInitRule()
567 {
568     vars[0] = 1;
569     sysatom_count = 0;
570     int ret = interpret(initrule, 0, 1);
571     if(ret == 0){
572         reactsysatom();
573         nextreact = 0;
574         return 0;
575     }else{
576         return -1;
577     }
578 }
579
580
581
582 // ルール適用成功なら０、失敗なら正、適用可能ルールが無いなら-1 を返す
583 // タイマ割り込みによって中断されたなら-2 を返す。
```

```
584 // task[i].remainRule() が偽の時にはこの関数は呼ばれないことを保証する。
585 // lock（interpret 第 3 引数）を 0 にして呼んでやることで、割り込み時の中断を許す。
586 int Task::react()
587 {
588     // vars : member of Ruleset (variable buf)
589     int ret;
590 
591     vars[0] = 1;
592     sysatom_count = 0; // sysatom 個数を 0 にリセット
593     ret = interpret(rules[nextreact], 0, 0);
594     if(ret == 0){ // 成功
595         reactsysatom(); // sysatom[] に登録されてるアトムを扱う
596 dbg("rule %d react.\n\n", nextreact);
597         nextreact = 0;
598         return 0;
599     }else if(ret == -2){ // interpret が割り込み終了した
600         return -2; // 割込中段なら-2 を返す。nextreact は変化しない。
601     }else{ // 失敗
602 dbg("rule %d failed.\n", nextreact);
603         nextreact++;
604         if(rules[nextreact] == NULL){ // 全ルール失敗
605             nextreact = 0; // 0 に戻しておく
606             remainRuleFlag = 0;
607             return -1; // ルールもうなし＞-1 を返す
608         }else{ // まだルール残ってる
609             return nextreact; // 正の数を返す
610         }
611     }
612 }
613 
614 // ルール番号を指定して実行。
615 // もし rules[n] が
616 // [0] : FINDATOM, [2] : 1(task root), [3] : ATOM_GETFUNC(atom)
617 // のときは、vars[[1]] に atom を入れて、pc=4 から開始する。
618 // 割り込みアトムの処理に使われるので、最初から lock=1 で interpret を呼び出す
619 int Task::react(int n, WORD atom)
620 {
621     int ret;
622 
623     dbg("react for interrupt called. ruleNo:%d, atomfunc:%x\n", n, ATOM_GETFUNC(atom));
624 
625     vars[0] = 1;
626     sysatom_count = 0; // sysatom 個数を 0 にリセット
627 
628     if(atom != 0 && // initrule 用に使うときは 0 指定
629        rules[n][0] == FINDATOM &&
630        rules[n][2] == 0 &&
631        rules[n][3] == ATOM_GETFUNC(atom)){
632         dbg("findatom optimize succeeded.\n");
633         // findatom の手間を省く
634         vars[1] = atom;
635         ret = interpret(rules[n], 4, 1);
636     }else{ // 先頭が findatom(atom) じゃなきゃ普通に実行
637         ret = interpret(rules[n], 0, 1);
638     }
639     if(ret == 0){ // 成功
640         reactsysatom(); // sysatom[] に登録されてるアトムを扱う
641 dbg("rule %d react.\n\n", nextreact);
642         nextreact = 0;
643         return 0;
644     }else{ // 失敗
645 dbg("rule %d failed.\n", nextreact);
646         return nextreact; // 正の数を返す
647     }
648 }
649 
650 // rule[pc] は命令列が入っているので、index>=1。
```

```
651 #define ARG(index) (rule[pc+(index)])
652
653 // pc:プログラムカウンタ
654 int Task::interpret(int *rule, int pc, int lock)
655 {
656     // 再帰するのでローカル変数は少ないほうがいい
657     while(rule[pc] != UNDEF){
658
659         // タイマ割り込みがあれば中断終了する。
660         if((interruptFlag == 1) && (lock == 0)){
661             return -2; // 中断終了
662         }
663
664         dbg("interpret %d, pc=%d\n", rule[pc], pc);
665
666         switch(rule[pc]){
667         case FAIL:
668             // 必ず失敗する命令（ルールの無いタスクに置く）
669             return -1;
670         case DEREF: // [-dstatom, srcatom, srcpos, dstpos]
671             //  src は必ず Atom。Int から引くことはない。dst は Atom と int で場合分け。
672             // とりあえずリンクを代入
673             vars[ARG(1)] = APOS_GETLINK(vars[ARG(2)], ARG(3));
674             // 整数ならそのまま。アトムならリンク先アトムを取得
675             if(LINK_ISINT(vars[ARG(1)])){
676                 if(ARG(4) != 0) return -1; // 整数なのに謎な pos 指定
677             }else{ // LINK_ISLINK
678                 if(APOS_GETPOS(vars[ARG(2)], ARG(3)) != ARG(4)) return -1;
679                 vars[ARG(1)] = LINK_GETATOM(vars[ARG(1)]);
680             }
681             pc+=5;
682             break;
683         case DEREFATOM: // [-dstatom or -link, srcatom, srcpos]
684              // たぶん Int のみ、ガードのみ？
685             // 整数なら link 形式で代入。アトムならアトムを入れる。＜怪しい仕様
686             vars[ARG(1)] = APOS_GETLINK(vars[ARG(2)], ARG(3));
687             if(LINK_ISLINK(vars[ARG(1)])){
688                 vars[ARG(1)] = LINK_GETATOM(vars[ARG(1)]);
689 dbg("derefatom : nomal atom derefed.\n");
690             }
691             pc+=4;
692             break;
693         case FINDATOM: // [-dstatom, srcmem, funcref]
694                 // funcref は Int ではない。
695             // 指定 ID・膜のアトムを見つけてきて interpret 再帰呼び出し
696             {
697                 AtomIt ait;
698                 WORD atom;
699
700                 am.initIt(&ait, ARG(3), vars[ARG(2)]);
701                 while((atom = am.next(&ait)) != 0){
702                     vars[ARG(1)] = atom;
703                     if(interpret(rule, pc+4, lock) == 0){
704                         return 0;
705                     }
706                 }
707                 return -1;
708             }
709             break; // 到達しない
710         case LOCKMEM: // [-dstmem, freelinkatom]
711           // ロックせず膜取得のみを行う。
712           vars[ARG(1)] = ATOM_GETMEM(vars[ARG(2)]);
713           dbg("lockmem: get mem %d. parent is %d\n", vars[ARG(1)], mm.mem[vars[ARG(1)]].parent);
714           pc+=3;
715           break;
716         case ANYMEM: // [-dstmem, srcmem]
717           // 子膜を１つずつ取得。
```

```
718           {
719             MemIt mit;
720             WORD mem;
721
722             mm.initIt(&mit, vars[ARG(2)]);
723             while((mem = mm.next(&mit)) != 0){
724               vars[ARG(1)] = mem;
725               if(interpret(rule, pc+3, lock) == 0){
726             return 0;
727               }
728             }
729             return -1;
730           }
731           break;
732         case TESTMEM: // [dstmem, srcatom(A?)]
733             if(ATOM_GETMEM(vars[ARG(2)]) != vars[ARG(1)]){
734                 return -1;
735             }
736             pc+=3;
737             break;
738         case NATOMS: //[srcmem, count]
739             if(mm.mem[vars[ARG(1)]].atomCount != ARG(2)){
740                 return -1;
741             }
742             pc+=3;
743             break;
744         case NMEMS: // [srcmem, count] 子膜の数を数える。
745         {
746             BYTE child, head, count;
747
748             child = mm.mem[vars[ARG(1)]].child;
749             if(ARG(2) == 0){ // 子膜なしかどうか調べたいとき
750                 if(child != 0) return -1;
751             }else{ // 子膜の個数を知りたいとき（使われない?）
752                 head = mm.mem[child].next;
753
754                 for(count = 1; head != child; count++){
755                     head = mm.mem[head].next;
756                 }
757                 if(ARG(2) != count) return -1;
758             }
759             pc+=3;
760             break;
761         }// 正常終了
762         case EQMEM: // [mem1, mem2]
763             if(vars[ARG(1)] != vars[ARG(2)]) return -1;
764             pc+=3;
765             break;
766         case NEQMEM: // [mem1, mem2]
767             // Java 版ではロック取得可能かで判別。ロックなしなら必要な命令。
768             if(vars[ARG(1)] == vars[ARG(2)]) return -1;
769             pc+=3;
770             break;
771         case FUNC: //[srcatom, funcref]
772             dbg("hoge");
773              // いずれ getfunc, loadfunc, eqfunc に展開
774             if(LINK_ISINT(vars[ARG(1)])){
775                 if(LINK2INT(vars[ARG(1)]) != ARG(2)){
776                     dbg("func(int): %x, %x\n", LINK2INT(vars[ARG(1)]), ARG(2));
777                     return -1; // 整数が入ってるとき
778                 }
779             }else if(ATOM_GETFUNC(vars[ARG(1)]) != ARG(2)){
780                 dbg("func: arg1=%x, arg2=%x\n", ATOM_GETFUNC(vars[ARG(1)]), ARG(2));
781                 return -1;
782             }
783             pc+=3;
784             break;
```

```
785         case EQATOM: // [atom1, atom2]
786             // atom1 と atom2 が同じ。int が渡されることはない?
787             if(vars[ARG(1)] != vars[ARG(2)]) return -1; // 参照先 atom のアドレス値を比較
788             pc+=3;
789             break;
790         case NEQATOM: // [atom1, atom2]
791             if(vars[ARG(1)] == vars[ARG(2)]) return -1;
792             pc+=3;
793             break;
794 // func 関係は未実装。atomID と intfunc を区別する方法も未確定。
795 // 一応 int を 2 倍の値（偶数）で保持すれば区別をつけることはできる。
796 // 非最適化時にはこれらは代入されないことは保証されている？
797 //      case SAMEFUNC: // [atom1, atom2]
798              // getfunc, getfunc, eqfunc に展開
799         case GETFUNC: // [-func, atom(I)]
800             // 整数計算結果の link から、allocatomindirect に渡す為の funcref 取得。
801             if(!LINK_ISINT(vars[ARG(2)])){ // 整数値以外が来たらエラー
802                 dbg("getfunc notint atom error\n");
803                 return -1;
804             }else{
805                 vars[ARG(1)] = vars[ARG(2)];
806             }
807             pc+=3;
808             break;
809 /*
810              // int のときは整数値を取得、atom のときは atomID を取得
811              if(LINK_ISINT(vars[ARG(2)])){ // atom のアドレスの時は 1bit 目が 0 なので false.
812                //もっといいマクロ名がほしい。＞ LINK_ISINT
813                vars[ARG(1)] = LINK2INT(vars[ARG(2)]);
814              }else{ // atom のとき
815                vars[ARG(1)] = ATOM_GETFUNC(vars[ARG(2)]);
816              }
817              pc+=3;
818              break; */
819         case LOADFUNC: // [-func, funcref(A/I)]
820         case EQFUNC:
821         case NEQFUNC:
822 dbg("...func instruction %d called.\n", ARG(0));
823             return -1;
824         case REMOVEATOM: //[srcatom, srcmem]
825              // 名前 funcref のアトム srcatom を、srcmem から取り除く
826              if(LINK_ISINT(vars[ARG(1)])){ // int のとき
827                mm.decAtomCount(vars[ARG(2)]);
828              }else if(ATOM_GETNAME(vars[ARG(1)]) < PROXY_BORDER){ // proxy でないアトムの時
829                mm.decAtomCount(vars[ARG(2)]);
830              }
831             pc+=3;
832             break;
833         case NEWATOM: // [-dstatom, srcmem, funcref(A)]
834              // int の生成は NEWATOM_INT でおこなう。
835
836             vars[ARG(1)] = am.newAtom(ARG(3), vars[ARG(2)]);
837             if(ATOM_GETNAME(vars[ARG(1)]) < PROXY_BORDER){
838
839                 mm.incAtomCount(vars[ARG(2)]); // 膜のアトム数を 1 増やす
840                 if(ATOM_GETNAME(vars[ARG(1)]) > NONACTIVE_SYSATOM_BORDER){
841                     // システムアトムが生成された場合、保持しておく
842                     dbg("sysatom add %x\n", ATOM_GETFUNC(vars[ARG(1)]));
843                     sysatom_buf[sysatom_count++] = vars[ARG(1)];
844                 }
845             }
846             pc+=4;
847             break;
848         case NEWATOM_INT: // [-dstlink, funcref(I)]
849           vars[ARG(1)] = INT2LINK(ARG(2));
850           pc+=3;
851           break;
```

```
852         case FREEATOM: //[srcatom(A)]
853             if(LINK_ISLINK(vars[ARG(1)])){
854                 am.freeAtom(vars[ARG(1)]);
855             }
856             pc+=2;
857             break;
858         case ALLOCATOM: // [-dstatom, funcref(I)]
859             // 一時的に使われる定数アトムを生成= LINK 形式で作る。
860             vars[ARG(1)] = INT2LINK(ARG(2));
861             pc+=3;
862             break;
863         case ALLOCATOMINDIRECT: // [-dstatom(I), func(LINK)]
864             // getfunc が返したリンクをそのままコピー。演算結果を右辺で使う用？
865             if(!LINK_ISINT(vars[ARG(2)])){
866                 return -1;
867             }else{
868                 vars[ARG(1)] = vars[ARG(2)];
869             }
870             pc+=3;
871             break;
872         case COPYATOM: // [-dstatom(I), mem, srcatom(I)]
873         // int 値のコピーを行なう。
874             if(LINK_ISINT(vars[ARG(3)])){
875                 vars[ARG(1)] = vars[ARG(3)];
876             }else{
877 dbg("int 以外の copymem 不可\n");
878                 return -1;
879             }
880             pc+=4;
881             break;
882         case REMOVEMEM: // [srcmem, parentmem] srcmem を parentmem の子膜から除去
883             // 親膜情報は要らない？
884             mm.removeMem(vars[ARG(1)]);
885             pc+=3;
886             break;
887         case NEWMEM: // [-dstmem, srcmem]
888             vars[ARG(1)] = mm.newMem();
889             mm.addMem(vars[ARG(1)], vars[ARG(2)]);
890             pc+=3;
891             break;
892         case MOVECELLS: // [dstmem, srcmem] srcmem は直後に廃棄される。srcmem!=0
893             am.moveAtoms(vars[ARG(1)], vars[ARG(2)]); // src 膜から dst 膜にアトムを移動
894             mm.moveMems(vars[ARG(1)], vars[ARG(2)]); // src 膜の子膜を dst 膜に移動
895             pc+=3;
896             break;
897         case FREEMEM: // [srcmem] free されるのは remove された MEM のみ？
898             mm.freeMem(vars[ARG(1)]);
899             pc+=2;
900             break;
901         case ADDMEM: // [dstmem, srcmem] remove 済み srcmem を dstmem に追加
902             mm.addMem(vars[ARG(2)], vars[ARG(1)]);
903             pc+=3;
904             break;
905 // リンク関係
906         case GETLINK: // [-link, atom(A), pos]
907             // リンク atom.pos を取得. atom が int のことはない。
908             vars[ARG(1)] = APOS_GETLINK(vars[ARG(2)], ARG(3));
909             pc+=4;
910             break;
911         case ALLOCLINK: // [-link, atom(A/I), pos]
912             if(vars[ARG(2)] & 0x01 == 1){ //リンク型データが入っている時
913                 if(ARG(3) != 0) dbg("alloclink: unexpected argument.\n");
914                 vars[ARG(1)] = vars[ARG(2)]; // そのまま
915             }else{
916                 // atom.pos を指すリンクを取得
917                 vars[ARG(1)] = APOS_ALLOCLINK(vars[ARG(2)], ARG(3));
918             }
```

```
919             pc+=4;
920             break;
921         case NEWLINK: // [atom1, pos1, atom2, pos2, mem1]
922             // atom1.link[pos1] と atom2.link[pos2] をつなげる。
923             // alloclink, alloclink, unifylinks
924             {
925                 WORD link1, link2;
926                 //alloclink
927                 if(vars[ARG(1)] & 0x1 == 1){
928                   link1 = vars[ARG(1)];
929                 }else{
930                   link1 = APOS_ALLOCLINK(vars[ARG(1)], ARG(2));
931                 }
932                 // alloclink
933                 if(vars[ARG(3)] & 0x1 == 1){
934                   link2 = vars[ARG(3)];
935                 }else{
936                   link2 = APOS_ALLOCLINK(vars[ARG(3)], ARG(4));
937                 }
938                 // 以下、UNIFYLINKS
939                 if(LINK_ISLINK(link1)){
940                     if(LINK_ISLINK(link2)){ //両方リンク
941                         LINK_SETLINK(link1, link2);
942                         LINK_SETLINK(link2, link1);
943                     }else if(LINK_ISINT(link2)){ // link1 に link2 の値を入れる
944                         LINK_SETLINK(link1, link2);
945                     }
946                 }else if(LINK_ISINT(link1)){
947                     if(LINK_ISLINK(link2)){ //link2 に link1 の値を入れる
948                         LINK_SETLINK(link2, link1);
949                     }else if(LINK_ISINT(link2)){ // 両方数字はエラー
950 dbg("両方数字\n");
951                     }
952                 }else{
953 dbg("unifylinks: link フラグ設定ミス\n");
954                 }
955             }
956             pc+=6;
957             break;
958         case RELINK: //[atom1, pos1, atom2, pos2, mem]
959             // atom1.link[pos1] と、atom2.link[pos2] のリンク先を接続
960             // alloclink, getlink, unifylinks
961             {
962                 WORD link1, link2;
963                 if(vars[ARG(1)] & 0x1 == 1){
964                   link1 = vars[ARG(1)];
965                 }else{
966                   link1 = APOS_ALLOCLINK(vars[ARG(1)], ARG(2));
967                 }
968                 link2 = APOS_GETLINK(vars[ARG(3)], ARG(4));
969                 // 以下、UNIFYLINKS
970                 if(LINK_ISLINK(link1)){
971                     if(LINK_ISLINK(link2)){ //両方リンク
972                         LINK_SETLINK(link1, link2);
973                         LINK_SETLINK(link2, link1);
974                     }else if(LINK_ISINT(link2)){ // link1 に link2 の値を入れる
975                         LINK_SETLINK(link1, link2);
976                     }
977                 }else if(LINK_ISINT(link1)){
978                     if(LINK_ISLINK(link2)){ //link2 に link1 の値を入れる
979                         LINK_SETLINK(link2, link1);
980                     }else if(LINK_ISINT(link2)){ // 両方数字はエラー
981 dbg("両方数字\n");
982                     }
983                 }else{
984 dbg("link1:%x, link2:%x\n", link1, link2);
985 dbg("unifylinks: link フラグ設定ミス\n");
```

```
 986                 }
 987             }
 988             pc+=6;
 989             break;
 990         case UNIFY: //[atom1, pos1, atom2, pos2]
 991             // getlink, getlink, unifylinks
 992             {
 993                 WORD link1, link2;
 994                 link1 = APOS_GETLINK(vars[ARG(1)], ARG(2));
 995                 link2 = APOS_GETLINK(vars[ARG(3)], ARG(4));
 996                 // 以下、UNIFYLINKS
 997                 if(LINK_ISLINK(link1)){
 998                     if(LINK_ISLINK(link2)){ //両方リンク
 999                         LINK_SETLINK(link1, link2);
1000                         LINK_SETLINK(link2, link1);
1001                     }else if(LINK_ISINT(link2)){ // link1 に link2 の値を入れる
1002                         LINK_SETLINK(link1, link2);
1003                     }
1004                 }else if(LINK_ISINT(link1)){
1005                     if(LINK_ISLINK(link2)){ //link2 に link1 の値を入れる
1006                         LINK_SETLINK(link2, link1);
1007                     }else if(LINK_ISINT(link2)){ // 両方数字はエラー
1008 dbg("両方数字\n");
1009                     }
1010                 }else{
1011 dbg("link1:%x, link2:%x\n", link1, link2);
1012 dbg("unifylinks: link フラグ設定ミス\n");
1013                 }
1014             }
1015             pc+=5;
1016             break;
1017         case UNIFYLINKS: // [link1, link2, mem]
1018             // link1 のリンク先と link2 のリンク先にリンクを張る。mem 使わない。
1019             {
1020                 WORD link1, link2;
1021                 link1 = vars[ARG(1)];
1022                 link2 = vars[ARG(2)];
1023 
1024                 if(LINK_ISLINK(link1)){
1025                     if(LINK_ISLINK(link2)){ //両方リンク
1026                         LINK_SETLINK(link1, link2);
1027                         LINK_SETLINK(link2, link1);
1028                     }else if(LINK_ISINT(link2)){ // link1 に link2 の値を入れる
1029                         LINK_SETLINK(link1, link2);
1030                     }
1031                 }else if(LINK_ISINT(link1)){
1032                     if(LINK_ISLINK(link2)){ //link2 に link1 の値を入れる
1033                         LINK_SETLINK(link2, link1);
1034                     }else if(LINK_ISINT(link2)){ // 両方数字はエラー
1035 dbg("両方数字\n");
1036                     }
1037                 }else{
1038 dbg("link1:%x, link2:%x\n", link1, link2);
1039 dbg("unifylinks: link フラグ設定ミス\n");
1040                 }
1041             }
1042             pc+=4;
1043             break;
1044 // proxy 関連
1045         case REMOVEPROXIES: // [srcmem]
1046           mm.removeProxies(vars[ARG(1)], am);
1047           pc+=2;
1048           break;
1049         case REMOVETOPLEVELPROXIES: // [srcmem]
1050           mm.removeTopLevelProxies(vars[ARG(1)], am);
1051           pc+=2;
1052           break;
```

```
1053          case INSERTPROXIES: // [parentmem, childmem]
1054            mm.insertProxies(vars[ARG(1)], vars[ARG(2)], am);
1055            pc+=3;
1056            break;
1057          case REMOVETEMPORARYPROXIES: // [srcmem]
1058            mm.removeTopLevelProxies(vars[ARG(1)], am);
1059            pc+=2;
1060            break;
1061          case COPYMEM: // $p が右辺に最大１つなら使わない。（プロセス文脈は線形）
1062  dbg("copymem called.\n");
1063              return -1;
1064          case DROPMEM: // [srcmem]
1065              // srcmem とその中身を破棄する。再帰的にアトム消去＞膜消去
1066              // {$p[]} :- . など、リンクのないプロセス変数を消す時に使う。
1067              // プロセス変数を$p 固定にするなら、DROPMEM は使われない。
1068              // mm.dropMem(vars[ARG(1)]); // もしやるならこう
1069  dbg("dropmem called.\n");
1070              return -1;
1071
1072  // システム命令
1073          case REACT: // どこでででてくる？
1074              break;
1075          case COMMIT:
1076              lock = 1; // データ構造書き換え時は中断できなくする
1077              pc+=1;
1078              break;
1079          case PROCEED:
1080              return 0; // 正常終了
1081
1082          case ISUNARY: // [atom]
1083              if(ATOM_GETARITY(vars[ARG(1)]) != 1) return -1;
1084              pc+=2;
1085              break;
1086          case ISINT: // [link] アトムがくることはない？
1087              if(!LINK_ISINT(vars[ARG(1)])) return -1;
1088              pc+=2;
1089              break;
1090  // 算術演算
1091          case IADD: // [-dstlink, link1, link2]
1092              vars[ARG(1)] =
1093                  INT2LINK(LINK2INT(vars[ARG(2)]) + LINK2INT(vars[ARG(3)]));
1094              pc+=4;
1095              break;
1096          case ISUB:
1097              vars[ARG(1)] =
1098                  INT2LINK(LINK2INT(vars[ARG(2)]) - LINK2INT(vars[ARG(3)]));
1099              pc+=4;
1100              break;
1101          case IMUL:
1102              vars[ARG(1)] =
1103                  INT2LINK(LINK2INT(vars[ARG(2)]) * LINK2INT(vars[ARG(3)]));
1104              pc+=4;
1105              break;
1106          case IDIV:
1107              if(LINK2INT(vars[ARG(3)]) == 0) return -1;
1108              vars[ARG(1)] =
1109                  INT2LINK(LINK2INT(vars[ARG(2)]) / LINK2INT(vars[ARG(3)]));
1110              pc+=4;
1111              break;
1112          case INEG: //[-dstlink, link]
1113              vars[ARG(1)] = INT2LINK( - LINK2INT(vars[ARG(2)]));
1114              pc+=3;
1115              break;
1116          case IMOD:
1117              if(LINK2INT(vars[ARG(3)]) == 0) return -1;
1118              vars[ARG(1)] =
1119                  INT2LINK(LINK2INT(vars[ARG(2)]) % LINK2INT(vars[ARG(3)]));
```

```
1120            pc+=4;
1121            break;
1122 // ビット演算（どうやって使うのか謎）
1123        case INOT:
1124            vars[ARG(1)] = INT2LINK( ~LINK2INT(vars[ARG(2)]));
1125            pc+=3;
1126            break;
1127        case IAND:
1128            vars[ARG(1)] =
1129                INT2LINK(LINK2INT(vars[ARG(2)]) & LINK2INT(vars[ARG(3)]));
1130            pc+=4;
1131            break;
1132        case IOR:
1133            vars[ARG(1)] =
1134                INT2LINK(LINK2INT(vars[ARG(2)]) | LINK2INT(vars[ARG(3)]));
1135            pc+=4;
1136            break;
1137        case IXOR:
1138            vars[ARG(1)] =
1139                INT2LINK(LINK2INT(vars[ARG(2)]) ^ LINK2INT(vars[ARG(3)]));
1140            pc+=4;
1141            break;
1142 // 比較演算 違う時は失敗終了する
1143        case ILT: //[link1, link2]
1144            if(!(LINK2INT(vars[ARG(1)]) < LINK2INT(vars[ARG(2)]))) return -1;
1145            pc+=3;
1146            break;
1147        case ILE:
1148            if(!(LINK2INT(vars[ARG(1)]) <= LINK2INT(vars[ARG(2)]))) return -1;
1149            pc+=3;
1150            break;
1151        case IGT:
1152            if(!(LINK2INT(vars[ARG(1)]) > LINK2INT(vars[ARG(2)]))) return -1;
1153            pc+=3;
1154            break;
1155        case IGE:
1156            if(!(LINK2INT(vars[ARG(1)]) >= LINK2INT(vars[ARG(2)]))) return -1;
1157            pc+=3;
1158            break;
1159        case IEQ:
1160            if(!(LINK2INT(vars[ARG(1)]) == LINK2INT(vars[ARG(2)]))) return -1;
1161            pc+=3;
1162            break;
1163        case INE:
1164            if(!(LINK2INT(vars[ARG(1)]) != LINK2INT(vars[ARG(2)]))) return -1;
1165            pc+=3;
1166            break;
1167
1168        default:
1169            dbg("invalid instruction: %d\n", rule[pc]);
1170            OSExit(-1);
1171            break;
1172        }
1173    }
1174    dbg("instruction code finished without proceed.\n");
1175    return -1;
1176 }
```

## B.1.13 task.h

```
1 #ifndef TASK_H
2 #define TASK_H
3
4 #include "lmntal.h"
```

```
 5 #include "atom.h"
 6 #include "mem.h"
 7
 8 typedef struct{
 9     int count;
10     int *buf;
11 } RELATEDRULE;
12
13 class Task{
14 public:
15     Task(void);
16     ~Task(void);
17     int react();
18     int react(int n, WORD atom);
19     int interpret(int *rule, int pc, int lock);
20     int loadRules();
21     int remainRule() { return remainRuleFlag; };
22     int isStable() {return ((!remainRuleFlag) && (timerCount == 0)); };
23     void dump();
24     int reactInitRule();
25     int reactInterruptAtom();
26     int setId(int i) { return id = i; };
27     int getPicture();
28
29     static int interruptAtomCount;
30     static Task *tasks;
31
32     // timer で生成されたアトムを生成待ちスタックに登録
33     WORD genAtom(WORD atomFunc) { genStack[genStackHead++] = atomFunc; };
34
35     void decTimerCount() { timerCount--; };
36     // private
37     int timerCount; // 登録してあるタイマの個数
38
39 private:
40     int reactsysatom();
41     int getSysValue(WORD atom);
42
43     int id; // このタスクの ID
44     AtomManager am;
45     MembraneManager mm;
46     int *initrule;
47     int *rules[MAX_RULES];
48     int nextreact; // 次に react で実行するルール。
49     int remainRuleFlag; // 未適用ルールがのこっているか？(割り込みで 0、react 中で 1 に変化)
50
51     // システムアトム用
52     WORD execStack[EXECSTACK_LEN]; // システムアトムのアドレスが入る
53     int execStackHead;
54     WORD genStack[GENSTACK_LEN]; // 生成アトム ID、その他情報が入る
55     int genStackHead;
56     RELATEDRULE relatedRule[MAX_ACTIVE_SYSATOM];
57
58     // interpret 用
59     WORD vars[MAX_VARS]; // アトム領域 (WORD) へのポインタが入る
60     WORD sysatom_buf[MAX_SYSATOM_INRULE]; // interpret 中に newatom された sysatom 格納用
61     int sysatom_count;
62 };
63
64 #endif
```

# B.2 コンバータ

## B.2.1 atomid.cpp

```
 1 #include <stdio.h>
 2 #include <ctype.h>
 3 #include <string.h>
 4
 5 #include "atomid.h"
 6
 7 AtomID::AtomID(void)
 8 {
 9     int i;
10
11     funcID = new FUNCINFO[MAX_FUNC];
12     for(i=0; i < MAX_FUNC; i++) funcID[i].id = 0;
13     next_id = 1;
14 }
15
16 AtomID::~AtomID(void)
17 {
18     delete[] funcID;
19 }
20
21 int AtomID::loadData(const char *filename)
22 {
23     FILE *fp;
24     int i;
25
26     if(NULL == (fp = fopen(filename, "rt"))) return -1;
27
28     // 次に生成する atom の NAME を読み込み
29     if(1 != fscanf(fp, "%x\n", &next_id)) return -1;
30
31
32     for(i=0; i < MAX_FUNC; i++){
33         if(2 != fscanf(fp, "%s %x\n", &funcID[i].name, &funcID[i].id)){
34             break; // 読み込み終了
35         }
36     }
37     for(; i < MAX_FUNC; i++) funcID[i].id = 0; // 残り要素は0
38
39     return 0;
40 }
41
42 int AtomID::saveData(const char *filename)
43 {
44     FILE *fp;
45
46     if(NULL == (fp = fopen(filename, "wt"))) return -1;
47
48     fprintf(fp, "%x\n", next_id);
49     for(int i=0; i < MAX_FUNC; i++){
50         if(funcID[i].id == 0) break; // 出力終了
51         fprintf(fp, "%s %x\n", funcID[i].name, funcID[i].id);
52     }
53     return 0;
54 }
55
56 // id が登録されていれば、対応するファンクタ名を返す
57 char* AtomID::getName(int id)
58 {
59     for(int i=0; i < MAX_FUNC; i++){
60         if(funcID[i].id == 0) return NULL; // 見つからなかった
61         if(funcID[i].id == id){ // 同じ ID のものが見つかった
62             return funcID[i].name; // ファンクタ名を返す
```

```
63          }
64      }
65      return NULL;
66  }
67
68
69  WORD AtomID::funcConv(const char *funcbuf)
70  {
71      int num;
72
73      if(sscanf(funcbuf, "%d", &num) == 1){ // func は数字だった
74          return (WORD)num;
75      }else if(isalpha(funcbuf[0]) || funcbuf[0] == '$'){ // アトム名だった
76          WORD tmp;
77          tmp = (WORD)getID(funcbuf);
78          return tmp;
79      }else{ // 謎
80          return (WORD)-1;
81      }
82  }
83
84  // 整数でない functor の ID を（なければ登録して）返す。
85  int AtomID::getID(const char *funcbuf)
86  {
87      int i, j, arity;
88
89      for(i=0; i < MAX_FUNC; i++){
90          if(funcID[i].id == 0) break;
91          if(strcmp(funcID[i].name, funcbuf) == 0){
92              return funcID[i].id;// 登録されているなら返す
93          }
94      }
95      // 登録されて無いなら新規登録してその値を返す
96      strncpy(funcID[i].name, funcbuf, FUNC_LEN); // functor を登録
97
98      // arity を抽出
99      for(j=strlen(funcbuf); funcbuf[j] != '_'; j--);
100     if(1 != sscanf(funcbuf+j, "_%d", &arity)) return -1;
101
102     funcID[i].id = next_id | (arity << 16);
103
104     next_id += 2; // 必ず奇数
105     return funcID[i].id;
106 }
107
108 int AtomID::isIntFunc(const char *funcbuf)
109 {
110     int num;
111     if(sscanf(funcbuf, "%d", &num) == 1){
112         return -1;
113     }else{
114         return 0;
115     }
116 }
```

## B.2.2 atomid.h

```
1  #ifndef ATOMID_H
2  #define ATOMID_H
3
4  #define MAX_FUNC 1024
5  #define FUNC_LEN 32
6
7  typedef struct{
```

```
 8      char name[FUNC_LEN];
 9      int id;
10  } FUNCINFO;
11
12  #include "../runtime/lmntal_const.h"
13
14  class AtomID{
15  public:
16      AtomID(void);
17      ~AtomID(void);
18
19      WORD funcConv(const char *funcbuf);
20      int isIntFunc(const char *funcbuf);
21
22      int loadData(const char *filename);
23      int saveData(const char *filename);
24      char* getName(int id);
25
26  private:
27      int getID(const char *funbuf);
28      FUNCINFO *funcID;
29      int next_id;
30  };
31
32  #endif
```

## B.2.3 bin2lmn.cpp

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <sys/types.h>
 4  #include <signal.h>
 5  #include <unistd.h>
 6
 7  #include "atomid.h"
 8  #include "../runtime/lmntal_const.h"
 9
10  #define dbg(f, ...) fprintf(stderr, f, ##__VA_ARGS__)
11
12  #define LINK_ISINT(link) (!(((link) & 0x00000003) ^ 0x01))
13  // LINK_GETLINK で得たリンクを整数値に直す (符号付 30bit 値)
14  #define LINK2INT(link) \
15      (((signed int)(link)) >> 2)
16
17  #define MAX_ATOM 20000
18  // 20000 アトムまで対応可能
19
20  typedef struct t_link{
21      WORD key;
22      int value;
23      struct t_link *next;
24  }LINK;
25
26  typedef struct t_atom{
27      int mem;
28      int arity;
29      int name;
30      int addr;
31      int linkno[MAX_ARITY];
32      int intflag[MAX_ARITY]; // 1 なら linkno に格納されてるのはリンクでなく整数。
33  }ATOM;
34
35  class LinkTable{
36  public:
```

```
37      LinkTable(void);
38      ~LinkTable(void);
39      int resist(WORD w);
40      int search(WORD w);
41      int check();
42  private:
43      static int no;
44      LINK *head;
45  };
46
47  unsigned int getWord(){
48      int i;
49
50      i = (getchar() & 0xff) << 24;
51      i = i | (getchar() & 0xff) << 16;
52      i = i | (getchar() & 0xff) << 8;
53      i = i | (getchar() & 0xff);
54      return i;
55  }
56
57  int get_atom(ATOM *a);
58  void output_mem(ATOM *a, int atom_count, AtomID *atomid, int n);
59
60  /* 出力フォーマット
61  WORD n : タスク数
62
63  WORD size, WORD atom, WORD, addr,  WORD link1 ...
64      : size 個分の WORD データが続く。最初 1word は atom、次に atom のあったアドレス、
65          その後にリンクが続く。if size == 0 アトム情報記述終了
66  BYTE 0xff : 膜開始（'{' に対応）
67  BYTE ID : 膜 ID
68  BYTE x : 0x00 この膜終了（'}' に対応）, 0xff 子膜開始（'{' に対応）
69
70  これがタスク個数続く
71  */
72
73
74  int main(int argc, char **argv){
75      ATOM a[MAX_ATOM];
76      WORD w;
77      int task_count, atom_count;
78      AtomID *atomid = NULL;
79
80
81      // aid ファイルが引数に指定されていれば、アトム ID 情報を読み込み
82      if(argc > 1){
83          atomid = new AtomID;
84          if(0 != atomid->loadData(argv[1])){ // 読み込み失敗
85              dbg("cannot open aid file : %s\n", argv[1]);
86              delete atomid;
87              atomid = NULL;
88          }
89      }else{
90          dbg("please specify aid file.\n");
91      }
92
93      // タスク個数を読み込み
94      task_count = (int)getWord();
95
96      for(int i=0; i < task_count; i++){ // 各タスクを出力
97
98          // アトム情報の読み込み。
99          atom_count = get_atom(a);
100
101         // タスク膜開始情報を読み飛ばし。なければエラー
102         if(0xff != getchar()) return -1;
103
```

```
104         printf("{"); // タスク膜開始
105
106         // 膜情報読み込み、膜・アトム出力
107         output_mem(a, atom_count, atomid, 2);
108
109         printf("\n}\n"); // タスク膜終了
110     }
111
112     // 終了処理
113     // 入力に/dev/ttyS0 を指定した場合、普通に終了するとなぜかプロセスが残る。
114     // よって kill で無理矢理強制終了。
115     fflush(stdout); // 出すもん出し切る
116     kill(getpid(), SIGINT);
117
118     return 0;
119 }
120
121 // atom 情報（バイナリ）を読み込んで a に入れる
122 int get_atom(ATOM *a)
123 {
124     WORD w;
125     WORD size;
126     int i, j, k, value, flag;
127     LinkTable lt;
128
129     for(i=0; i < MAX_ATOM; i++){
130         size = getWord();
131         if(size == 0) break;
132         w = getWord();
133         a[i].mem = w >> 24 & 0xff;
134         a[i].arity = w >> 16 & 0xff;
135         a[i].name = w & 0xffff;
136         w = getWord();
137         a[i].addr = w & 0xffffff;
138         for(j=0; j < size - 2; j++){ // link を１つずつ読み込み
139             w = getWord();
140             if(LINK_ISINT(w)){ // リンクが整数値
141                 a[i].linkno[j] = LINK2INT(w);
142                 a[i].intflag[j] = 1; // linkno に入ってるのは整数値
143             }else{ // リンクの時
144                 a[i].intflag[j] = 0; // linkno はリンク番号
145                 value = lt.search((j << 24) | (a[i].addr & 0xffffff));
146                 if(value == 0){ // 自分は未登録だった
147                     a[i].linkno[j] = lt.resist(w); // リンク先を登録,No をもらう
148                 }else{
149                     a[i].linkno[j] = value; // 登録済みならその No を設定
150                 }
151             }
152         }
153     }
154     if(lt.check()) dbg("link connect failed\n");
155     return i;
156 }
157
158 void indent(int n){
159     for(int i=0; i < n; i++) putchar(' ');
160 }
161
162 // 膜の中身を出力、子膜を再帰出力。（自分の膜の{, }は出力しない
163 void output_mem(ATOM *a, int atom_count, AtomID *atomid, int n)
164 {
165     WORD w;
166     int i, j, flag = 0;
167     char *str;
168
169     // 膜を読み込みつつ出力
170     w = getchar(); // まず膜 ID を読み込み
```

```
171     for(i=0; i < atom_count; i++){
172         if(w == a[i].mem){ // この膜の中にあるアトム全て出力
173             if(flag) printf(", "); // 2 回目以降出力前にはカンマ
174             printf("\n"); indent(n); // 字下げをする
175 
176             if(a[i].name > PROXY_BORDER){ // proxy のときは'=' として出力
177                 printf("'='");
178             }else{
179                 str = NULL;
180                 if(atomid != NULL){ // atomid があるとき
181                     str = atomid->getName(a[i].arity << 16 | a[i].name);
182                 }
183                 if(str != NULL){ // ID に対応する名前が見つかった
184                     printf("%s", str);
185                 }else{ // 見つからなかったら ID のまま出力
186                     printf("a%d_%d", a[i].name, a[i].arity);
187                 }
188             }
189             if(a[i].arity > 0){
190                 printf("(");
191                 for(j=0; j < a[i].arity; j++){
192                     if(a[i].intflag[j] == 1){ // 整数値が格納されてる時
193                         printf("%d", a[i].linkno[j]); // 整数アトムを出力
194                     }else{
195                         printf("L%d", a[i].linkno[j]); // リンク番号
196                     }
197                     if(j+1 < a[i].arity) printf(", ");
198                 }
199                 printf(")");
200             }
201             flag = 1; // 2 つめ以降を出力する前にはカンマを出力
202         }
203     }
204 
205     while((w = getchar()) == 0xff){ // 子膜があるとき
206         printf("\n"); indent(n); printf("{"); // タスク膜開始
207         output_mem(a, atom_count, atomid, n + 2); // 子膜を再帰出力
208         printf("\n"); indent(n); printf("}"); // タスク膜終了
209 
210     }
211     if(w != 0){ // 最後が 0 で閉じなかった
212         printf("unexpected w %d read.(must be 0)\n", w);
213     }
214     return;
215 }
216 
217 
218 LinkTable::LinkTable(void)
219 {
220     head = new LINK;
221     head->next == NULL;
222 }
223 
224 int LinkTable::no = 1;
225 
226 LinkTable::~LinkTable(void)
227 {
228     LINK *p, *tmp;
229     p = head->next;
230 
231     while(p != NULL){
232         tmp = p->next;
233         delete p;
234         p = tmp;
235     }
236     delete head;
237 }
```

```
238
239 // リンクを使い切る（head.next == NULL）なら成功。0 を返す
240 int LinkTable::check()
241 {
242     if(head->next == NULL) return 0;
243     else return -1;
244 }
245
246 // No を設定し、リンク先を key に保存。設定値を返す
247 int LinkTable::resist(WORD w)
248 {
249     LINK *p;
250
251     p = new LINK;
252     p->key = w;
253     p->value = no;
254     p->next = head->next;
255     head->next = p;
256
257     return no++;
258 }
259
260 int LinkTable::search(WORD w)
261 {
262     LINK *p, *prev;
263     int i;
264
265     p = head->next;
266     prev = head;
267
268     while(p != NULL){
269         if(p->key == w){ // 他方のリンクが見つかった
270             prev->next = p->next;
271             i = p->value;
272             delete p;
273             return i;
274         }
275         prev = p;
276         p = p->next;
277     }
278     // 見つからなかったら 0 を返す
279     return 0;
280 }
```

## B.2.4 gental.cpp

```
 1 #include <stdio.h>
 2
 3 typedef unsigned int WORD;
 4 // 引数に指定された.tmp ファイルをバイナリに変換して出力
 5
 6
 7 /* 出力ファイルフォーマット
 8 N : 初期化コマンドデータ長
 9 n word data (init command)
10 N : タスクの個数
11
12 N : task1 初期化ルール データ長
13 n word data (task1 init rule)
14 0 : 初期化ルールに関連付けられたシステムアトムなし
15 N : task1 ルール 1 データ長
16 n word data (task1 rule1)
17 ...
18 ID : アクティブシステムアトム ID（0:入力終了）
```

```
19 M : 関連付けられたルールの個数
20 m 個の関連付けられたルール番号
21 ...
22
23 */
24
25 // リトルエンディアンで WORD 値を標準出力に書き出す
26 void putword(WORD w)
27 {
28     fputc(w >> 24 & 0xff, stdout);
29     fputc(w >> 16 & 0xff, stdout);
30     fputc(w >> 8 & 0xff, stdout);
31     fputc(w & 0xff, stdout);
32 }
33
34 int main(int argc, char **argv)
35 {
36     FILE *fp;
37     int i;
38
39     // argc < 3 ならおかしい（task 無し）ので終わる
40     if(argc < 3) return -1;
41
42     // temp_init.tal を読み込んで出力
43     if(NULL == (fp = fopen(argv[1], "rt"))) return -1;
44     fprintf(stderr, "tal2bin: read %s\n", argv[1]);
45
46     // init コードは未実装。長さ 0 とする。
47     putword(0);
48
49     fclose(fp);
50
51     // init コードの後にタスク数を出力
52     putword(argc - 2);
53
54     // タスクごとの tmp ファイルを読み込み、出力
55     for(int i=2; i < argc; i++){
56         int c;
57
58         if(NULL == (fp = fopen(argv[i], "rt"))) return -1;
59         fprintf(stderr, "tal2bin: read %s\n", argv[i]);
60
61         //
62         while(fscanf(fp, "%d", &c) == 1){ // 読み込めてる間
63             putword((WORD)c);
64         }
65         fclose(fp);
66     }
67
68     return 0;
69 }
70
```

## B.2.5 splitlmn.cpp

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // lmn ファイルを各タスク、初期化部ごとに切り分けてファイルに保存。
5
6 int main(int argc, char **argv)
7 {
8     FILE *fp;
9     int c, memcount;
```

```
10      pid_t pid;
11      char taskfile[] = "temp_task00.lmn";
12
13      // 最初のタスク膜開始まで読み込む
14      while((c = getchar()) != '{'){ // タスク開始まで読み込めなければ終了
15          if(c == EOF) return -1;
16      }
17      // 各タスクを temp_taskXX.lmn に書き込む
18      for(int i=0; i < 100; i++){ // 最大１００個
19          taskfile[9] = i / 10 + '0';
20          taskfile[10] = i % 10 + '0';
21
22          if(NULL == (fp = fopen(taskfile, "wb"))) return -1;
23
24
25          // 現在タスク膜開始直後。膜が閉じるまでファイルに書き出し
26          memcount = 1;
27          while(1){
28              c = getchar();
29              if(c == EOF) return -1; // タスク膜が閉じずに EOF ＞エラー
30              else if(c == '{') memcount++;
31              else if(c == '}'){
32                  memcount--;
33                  if(memcount == 0){
34                      fclose(fp);
35                      break; // タスク膜が閉じた
36                  }
37              }
38              fputc(c, fp); // タスク膜を閉じる}以外はファイルに書き出し
39          }
40
41          // 次のタスク膜が始まるなら繰り返し、EOF なら終了
42          while(1){
43              c = getchar();
44              if(c == EOF) break; // ファイル終端ならタスク記述終了
45              else if(c == '{') break; // 次のタスク開始
46              else if(c == '.') break; // タスク記述終了
47          }
48          if(c == EOF || c == '.') break; // タスク記述終了なら抜ける。以降は初期化部
49      }
50
51      // 初期化部を temp_init.lmn に書き込む
52      if(NULL == (fp = fopen("temp_init.lmn", "wb"))) return -1;
53
54      while(EOF != (c = getchar())){// 残りは初期化記述
55          fputc(c, fp);
56      }
57      fclose(fp);
58
59      return 0;
60  }
```

## B.2.6 template.aid

```
1  3
2
3  pix_7 0x70001
4
5  $in_2 0x2ffff
6  $out_2 0x2fffd
7
8  analog0_1 0x1e001
9  analog1_1 0x1e003
10 analog2_1 0x1e005
```

```
11 analog3_1 0x1e007
12 analog4_1 0x1e009
13 analog5_1 0x1e00b
14 analog6_1 0x1e00d
15 analog7_1 0x1e00f
16 dig_in4_1 0x1e011
17 dig_in5_1 0x1e013
18 dig_in6_1 0x1e015
19 dig_in7_1 0x1e017
20 quadleft_1 0x1e019
21 quadright_1 0x1e01b
22 key_1 0x1e01d
23 recv_2 0x2e01f
24 picture_1 0x1e021
25 tick_1 0x1e023
26
27 get_1 0x1f001
28 timer_2 0x2f003
29 timeronce_2 0x2f005
30 timerkill_2 0x2f007
31 leftmotor_1 0x1f009
32 rightmotor_1 0x1f00b
33 putint_1 0x1f00d
34 putchar_1 0x1f00f
35 dig_out0_1 0x1f011
36 dig_out1_1 0x1f013
37 dig_out2_1 0x1f015
38 dig_out3_1 0x1f017
39 dig_out4_1 0x1f019
40 dig_out5_1 0x1f01b
41 dig_out6_1 0x1f01d
42 dig_out7_1 0x1f01f
43 servo7_1 0x1f021
44 servo8_1 0x1f023
45 servo9_1 0x1f025
46 servo10_1 0x1f027
47 servo11_1 0x1f029
48 servo12_1 0x1f02b
49 quadleft_reset_0 0x0f02d
50 quadright_reset_0 0x0f02f
51 send_2 0x2f031
52 dig_outall_1 0x1f033
53 picture_remove_1 0x1f035
```

### B.2.7 translmn.cpp

```
 1 #include <stdio.h>
 2 #include <ctype.h>
 3 #include <string.h>
 4 #include <vector>
 5
 6 using namespace std;
 7
 8 #include "atomid.h"
 9
10 #include "../runtime/lmntal_const.h"
11 #include "../runtime/inst.h"
12
13 #define dbg(f, ...) fprintf(stderr, f, ##__VA_ARGS__)
14
15 #define MAX_INST 1024
16 #define INST_LEN 6
17 #define MAX_SYSATOM 8
18 #define MAX_RELATEDRULE 16
```

```
19
20 typedef struct{
21     int count;
22     WORD ibuf[INST_LEN];
23 } INST;
24
25 typedef struct{
26     int count;
27     WORD atoms[MAX_SYSATOM];
28 } SYSATOM;
29
30
31
32 int readCode(INST *inst, AtomID *atomid);
33 int optimize(INST *inst, int max_inst, int ruleNo, vector<int> *sysrule);
34 int writeCode(INST *inst, int max_inst);
35 int getInst(INST *inst, int pc, AtomID *atomid, char *buf, int *vartable,
36             int instID, int argcount, int withFunc, unsigned rawDataFlag);
37
38 // 標準入力から中間コードを読み込み、標準出力に吐き出す
39
40 int main(int argc, char **argv)
41 {
42     INST inst[MAX_INST];
43     AtomID atomid;
44     int i, pc;
45     SYSATOM sysatom;
46     vector<int> sysrule[MAX_ACTIVE_SYSATOM];
47
48     // funcID の初期化
49     if(-1 == atomid.loadData("temp_aid.aid")){
50         dbg("file load error.\n");
51         return -1;
52     }
53
54     dbg("translmn: now translating...\n");
55
56     // 標準入力からデータ取得、解釈し、inst に格納
57     for(int ruleNo = -1; ; ruleNo++){ // ルール番号計測。初期化ルールは-1.
58         for(i=0; i < MAX_INST; i++) inst[i].count = 0; // 命令列の初期化
59
60         pc = readCode(inst, &atomid);
61         if(pc < 1) break; // ルールがなくなるまで or エラーが起こるまで
62
63         // inst 内に最適化できる部分があればする.
64         // ルールと sysatom の関係
65         optimize(inst, pc, ruleNo, sysrule);
66         // 標準出力に書き出す
67         writeCode(inst, pc);
68     }
69     printf("0\n\n"); // ルール終了
70
71     // 各システムアトムに関係するルールの個数、番号を出力
72     for(i=0; i < MAX_ACTIVE_SYSATOM; i++){
73         printf("%d ", sysrule[i].size());
74         for(vector<int>::iterator it = sysrule[i].begin(); it != sysrule[i].end(); it++){
75             printf("%d ", *it);
76         }
77         printf("\n");
78     }
79
80     // atomID をファイルに書き出して終了
81     if(-1 == atomid.saveData("temp_aid.aid")){
82         dbg("file save error.\n");
83         return -1;
84     }
85
```

```
 86      return 0;
 87  }
 88
 89  // 命令の読み込み
 90  // withFunc != 0 のとき、命令の末尾は funcref. 整数値ではなく文字列を読み込む
 91  // 注意！リンク番号は vartable ではなく生の値を使う！
 92  // rawDataFlag が 1 である引数は生の値を用いる（例：0x06 なら一つ目、２つ目の引数が生の値）
 93  // rawDataFlag の 1bit 目は、命令番号自体をさすため参照されない。
 94
 95  int getInst(INST *inst, int pc, AtomID *atomid, char *buf, int *vartable,
 96              int instID, int argcount, int withFunc, unsigned rawDataFlag)
 97  {
 98      int i, count;
 99      int p = 0; // 現在見ている buf 位置
100      int arg[MAX_VARS];
101      char func[FUNC_LEN];
102
103
104      inst[pc].count = argcount;
105      inst[pc].ibuf[0] = instID;
106
107      // 引数読み込み（withFunc が真なら読み込み個数を-1 する）
108      if(withFunc) count = argcount - 1;
109      else count = argcount;
110
111      for(i = 1; i < count ; i++){
112          while(buf[p] != ' ')p++; // 次のスペースまで読み飛ばし
113          while(buf[p] == ' ')p++; // スペースの次の数字まで読み飛ばし
114          sscanf(buf + p, "%d", &arg[i]);
115
116          // rawDataFlag が設定されている時は生の値を使う
117          if((rawDataFlag >> i) & 0x1) inst[pc].ibuf[i] = arg[i];
118          else                         inst[pc].ibuf[i] = vartable[arg[i]];
119      }
120      if(withFunc){ // 末尾 funcref なら、文字列読み込み
121          while(buf[p++] != ' '); // funcref 前まで飛ばす
122          sscanf(buf + p, "%s", func);
123          inst[pc].ibuf[i] = atomid->funcConv(func);
124      }
125      return 0;
126  }
127
128  // sysatom の取得と命令列の最適化
129  int optimize(INST *inst, int max_inst, int ruleNo, vector<int> *sysrule)
130  {
131      int i, name;
132
133      for(i=0; i < max_inst; i++){
134          switch(inst[i].ibuf[0]){ // 命令コードが
135          case FINDATOM:
136              // findatom される sysatom の ID を格納。
137              name = FUNC_NAME(inst[i].ibuf[3]);
138
139              if(name > SYSATOM_BORDER && name < NONACTIVE_SYSATOM_BORDER){
140                  sysrule[(name - SYSATOM_BORDER) / 2].push_back(ruleNo);
141              }
142              break;
143  // todo: getfunc と newatomindirect のところの最適化。他の部分への影響あり？
144          default:
145              break;
146          }
147      }
148      return 0;
149  }
150
151  // sysatom, 命令列をテキスト形式で標準出力に出力
152  int writeCode(INST *inst, int max_inst)
```

```
153 {
154     int i, j, count = 0;
155     vector<int>::iterator it;
156 
157     // 命令長を出力
158     for(i=0; i < max_inst; i++){
159         count += inst[i].count;
160     }
161     printf("%d\n", count);
162 
163     // 命令列を出力
164     for(i=0; i < max_inst; i++){
165         for(j=0; j < inst[i].count; j++){
166             printf("%d ", inst[i].ibuf[j]);
167         }
168         printf("\n");
169     }
170     printf("\n");
171 
172     return 0;
173 }
174 
175 ////////////////////////////////////////////////////
176 // 標準入力から中間コードを読み込み、inst, atomid に格納。
177 
178 int readCode(INST *inst, AtomID *atomid)
179 {
180     char buf[1024], code[16];
181     char funcbuf[FUNC_LEN];
182     int arg[MAX_VARS], vartable[MAX_VARS]; // arg[0] は使わない
183     int max_vars = 0;
184     int i, num;
185     int pc = 0; // inst[pc] に命令を書き込む
186 
187     // jump で変数 index が再定義されるのに対応する用に使う。
188     for(i=0; i < MAX_VARS; i++){
189         vartable[i] = i;
190     }
191 
192     while(fgets(buf, 1024, stdin) != NULL){ // 読み込み終了まで
193         // \t\t で始まってない行は読み飛ばし
194         if(buf[1] != '\t') continue;
195 
196         if(sscanf(buf, "%s", code) != 1){
197             dbg("code read error\n");
198             return -1;
199         }
200 
201         for(i=0; buf[i] != '\0'; i++){ // [,] の消去
202             if(buf[i] == '[' || buf[i] == ']' || buf[i] == ',') buf[i] = ' ';
203         }
204 
205         // PROCEED  ★ PROCEED を読み込んだら終了。
206         if(strcmp(code, "proceed") == 0){
207             inst[pc].count = 1;
208             inst[pc].ibuf[0] = PROCEED;
209             pc++;
210 
211             return pc; // 終了
212 
213         // COMMIT interpret のロック（割り込み時の中断可否）のために使う
214         }else if(strcmp(code, "commit") == 0){
215             inst[pc].count = 1;
216             inst[pc].ibuf[0] = COMMIT;
217             pc++;
218 
219         // SPEC
```

```
220         }else if(strcmp(code, "spec") == 0){
221             // 変数の使用個数 max_vars の書き換え >信用ならない
222             sscanf(buf, "%*s %*d %d", &max_vars);
223         // JUMP
224         }else if(strcmp(code, "jump") == 0){
225             int top = 0;
226 
227             // L<num>を読み飛ばす
228             while(buf[top++] != 'L');
229             while(isdigit(buf[top])) top++;
230 
231             for(i=0; i < MAX_VARS; i++){
232                 // 数字が出てくるまで飛ばす
233                 while(!isdigit(buf[top])){
234                     if(buf[top] == '\0') goto NEXT;
235                     top++;
236                 }
237                 // 読み込み
238                 sscanf(buf+top, "%d", &arg[i]);
239 
240                 // 数字以外が出てくるまで飛ばす
241                 while(isdigit(buf[top])){
242                     if(buf[top] == '\0') goto NEXT;
243                     top++;
244                 }
245             }
246 NEXT:
247             max_vars = i;
248             vartable[arg[i]]; // temp
249             // 読み込んだ値にしたがって vartable を書き換える。
250             // vartable[n] を vartable[arg[n]] の値に書き換える。
251             for(i=0; i < max_vars; i++){
252                 arg[i] = vartable[arg[i]]; // temp
253             }
254             int max = 0;
255             for(i=0; i < max_vars; i++){
256                 vartable[i] = arg[i]; // 書き換え
257                 if(max < vartable[i]) max = vartable[i]; // 最大値を保存しておく
258             }
259             // これ以降に出現する変数番号は、まだ出てきてない大きさの番号に置き換える
260             for(i = max_vars; i < MAX_VARS; i++){
261                 vartable[i] = i - max_vars + max + 1;
262             }
263 // 出力する命令の開始
264         // DEREF
265         }else if(strcmp(code, "deref") == 0){
266             getInst(inst, pc, atomid, buf, vartable, DEREF, 5, 0, 0x18);
267             pc++;
268         // DEREFATOM
269         }else if(strcmp(code, "derefatom") == 0){
270             getInst(inst, pc, atomid, buf, vartable, DEREFATOM, 4, 0, 0x08);
271             pc++;
272         // FINDATOM
273         }else if(strcmp(code, "findatom") == 0){
274             getInst(inst, pc, atomid, buf, vartable, FINDATOM, 4, 1, 0);
275             pc++;
276         // LOCKMEM
277         }else if(strcmp(code, "lockmem") == 0){
278             getInst(inst, pc, atomid, buf, vartable, LOCKMEM, 3, 0, 0);
279             pc++;
280         // ANYMEM
281         }else if(strcmp(code, "anymem") == 0){
282             getInst(inst, pc, atomid, buf, vartable, ANYMEM, 3, 0, 0);
283             pc++;
284         // TESTMEM
285         }else if(strcmp(code, "testmem") == 0){
286             getInst(inst, pc, atomid, buf, vartable, TESTMEM, 3, 0, 0);
```

```
287             pc++;
288         // NATOMS
289         }else if(strcmp(code, "natoms") == 0){
290             getInst(inst, pc, atomid, buf, vartable, NATOMS, 3, 0, 0x04);
291             pc++;
292         // NMEMS
293         }else if(strcmp(code, "nmems") == 0){
294             getInst(inst, pc, atomid, buf, vartable, NMEMS, 3, 0, 0x04);
295             pc++;
296         // EQMEM
297         }else if(strcmp(code, "eqmem") == 0){
298             getInst(inst, pc, atomid, buf, vartable, EQMEM, 3, 0, 0);
299             pc++;
300         // NEQMEM
301         }else if(strcmp(code, "neqmem") == 0){
302             getInst(inst, pc, atomid, buf, vartable, NEQMEM, 3, 0, 0);
303             pc++;
304         // FUNC
305         }else if(strcmp(code, "func") == 0){
306             getInst(inst, pc, atomid, buf, vartable, FUNC, 3, 1, 0);
307             pc++;
308         // EQATOM
309         }else if(strcmp(code, "eqatom") == 0){
310             getInst(inst, pc, atomid, buf, vartable, EQATOM, 3, 0, 0);
311             pc++;
312         // NEQATOM
313         }else if(strcmp(code, "neqatom") == 0){
314             getInst(inst, pc, atomid, buf, vartable, NEQATOM, 3, 0, 0);
315             pc++;
316         // SAMEFUNC その他 func 関係 : 出てこない？
317
318         // GETFUNC : ★一応出力。最適化で newatomindirect と一緒に消える予定？
319         }else if(strcmp(code, "getfunc") == 0){
320             getInst(inst, pc, atomid, buf, vartable, GETFUNC, 3, 0, 0);
321             pc++;
322         // REMOVEATOM
323         }else if(strcmp(code, "removeatom") == 0){
324             getInst(inst, pc, atomid, buf, vartable, REMOVEATOM, 3, 0, 0);
325             pc++;
326         // NEWATOM ★ int の時は NEWATOM_INT を出力。
327         }else if(strcmp(code, "newatom") == 0){
328             sscanf(buf, "%*s %d %d %s", &arg[1], &arg[2], funcbuf);
329             if(atomid->isIntFunc(funcbuf)){ // int のとき
330                 inst[pc].count = 3;
331                 inst[pc].ibuf[0] = NEWATOM_INT;
332                 for(i=1; i < inst[pc].count - 1; i++){
333                     inst[pc].ibuf[i] = vartable[arg[i]];
334                 }
335                 inst[pc].ibuf[i] = atomid->funcConv(funcbuf);
336             }else{ // atom のとき
337                 inst[pc].count = 4;
338                 inst[pc].ibuf[0] = NEWATOM;
339                 for(i=1; i < inst[pc].count - 1; i++){
340                     inst[pc].ibuf[i] = vartable[arg[i]];
341                 }
342                 inst[pc].ibuf[i] = atomid->funcConv(funcbuf);
343             }
344             pc++;
345         // FREEATOM
346         }else if(strcmp(code, "freeatom") == 0){
347             getInst(inst, pc, atomid, buf, vartable, FREEATOM, 2, 0, 0);
348             pc++;
349         // ALLOCATOM
350         }else if(strcmp(code, "allocatom") == 0){
351             getInst(inst, pc, atomid, buf, vartable, ALLOCATOM, 3, 1, 0);
352             pc++;
353         // ALLOCATOMINDIRECT
```

```
354         }else if(strcmp(code, "allocatom..") == 0 ||
355                  strcmp(code, "allocato..") == 0){
356             getInst(inst, pc, atomid, buf, vartable, ALLOCATOMINDIRECT, 3, 0, 0);
357             pc++;
358         // COPYATOM
359         }else if(strcmp(code, "copyatom") == 0){
360             getInst(inst, pc, atomid, buf, vartable, COPYATOM, 4, 0, 0);
361             pc++;
362         // REMOVEMEM
363         }else if(strcmp(code, "removemem") == 0){
364             getInst(inst, pc, atomid, buf, vartable, REMOVEMEM, 3, 0, 0);
365             pc++;
366         // NEWMEM
367         }else if(strcmp(code, "newmem") == 0){
368             getInst(inst, pc, atomid, buf, vartable, NEWMEM, 3, 0, 0);
369             pc++;
370         // MOVECELLS
371         }else if(strcmp(code, "movecells") == 0){
372             getInst(inst, pc, atomid, buf, vartable, MOVECELLS, 3, 0, 0);
373             pc++;
374         // FREEMEM
375         }else if(strcmp(code, "freemem") == 0){
376             getInst(inst, pc, atomid, buf, vartable, FREEMEM, 2, 0, 0);
377             pc++;
378         // ADDMEM
379         }else if(strcmp(code, "addmem") == 0){
380             getInst(inst, pc, atomid, buf, vartable, ADDMEM, 3, 0, 0);
381             pc++;
382         // GETLINK
383         }else if(strcmp(code, "getlink") == 0){
384             getInst(inst, pc, atomid, buf, vartable, GETLINK, 4, 0, 0x08);
385             pc++;
386         // ALLOCLINK
387         }else if(strcmp(code, "alloclink") == 0){
388             getInst(inst, pc, atomid, buf, vartable, ALLOCLINK, 4, 0, 0x08);
389             pc++;
390         // NEWLINK
391         }else if(strcmp(code, "newlink") == 0){
392             getInst(inst, pc, atomid, buf, vartable, NEWLINK, 6, 0, 0x14);
393             pc++;
394         // RELINK
395         }else if(strcmp(code, "relink") == 0){
396             getInst(inst, pc, atomid, buf, vartable, RELINK, 6, 0, 0x14);
397             pc++;
398         // UNIFY
399         }else if(strcmp(code, "unify") == 0){
400             getInst(inst, pc, atomid, buf, vartable, UNIFY, 5, 0, 0x14);
401             pc++;
402         // UNIFYLINKS
403         }else if(strcmp(code, "unifylinks") == 0){
404             getInst(inst, pc, atomid, buf, vartable, UNIFYLINKS, 4, 0, 0);
405             pc++;
406         // REMOVEPROXIES
407         }else if(strcmp(code, "removeproxies") == 0){
408             getInst(inst, pc, atomid, buf, vartable, REMOVEPROXIES, 2, 0, 0);
409             pc++;
410         // REMOVETOPLEVELPROXIES
411         }else if(strcmp(code, "removetoplev..") == 0){
412             getInst(inst, pc, atomid, buf, vartable, REMOVETOPLEVELPROXIES, 2, 0, 0);
413             pc++;
414         // INSERTPROXIES
415         }else if(strcmp(code, "insertproxies") == 0){
416             getInst(inst, pc, atomid, buf, vartable, INSERTPROXIES, 3, 0, 0);
417             pc++;
418         // REMOVETEMPORARYPROXIES
419         }else if(strcmp(code, "removetemp..") == 0){
420             getInst(inst, pc, atomid, buf, vartable, REMOVETEMPORARYPROXIES, 2, 0, 0);
```

```
421             pc++;
422
423         // COPYMEM, DROPMEM はいらない
424         // REACT はいらない
425
426         // ISUNARY
427         }else if(strcmp(code, "isunary") == 0){
428             getInst(inst, pc, atomid, buf, vartable, ISUNARY, 2, 0, 0);
429             pc++;
430         // ISINT
431         }else if(strcmp(code, "isint") == 0){
432             getInst(inst, pc, atomid, buf, vartable, ISINT, 2, 0, 0);
433             pc++;
434         // IADD
435         }else if(strcmp(code, "iadd") == 0){
436             getInst(inst, pc, atomid, buf, vartable, IADD, 4, 0, 0);
437             pc++;
438         // ISUB
439         }else if(strcmp(code, "isub") == 0){
440             getInst(inst, pc, atomid, buf, vartable, ISUB, 4, 0, 0);
441             pc++;
442         // IMUL
443         }else if(strcmp(code, "imul") == 0){
444             getInst(inst, pc, atomid, buf, vartable, IMUL, 4, 0, 0);
445             pc++;
446         // IDIV
447         }else if(strcmp(code, "idiv") == 0){
448             getInst(inst, pc, atomid, buf, vartable, IDIV, 4, 0, 0);
449             pc++;
450         // INEG
451         }else if(strcmp(code, "ineg") == 0){
452             getInst(inst, pc, atomid, buf, vartable, INEG, 3, 0, 0);
453             pc++;
454         // IMOD
455         }else if(strcmp(code, "imod") == 0){
456             getInst(inst, pc, atomid, buf, vartable, IMOD, 4, 0, 0);
457             pc++;
458         // INOT
459         }else if(strcmp(code, "inot") == 0){
460             getInst(inst, pc, atomid, buf, vartable, INOT, 3, 0, 0);
461             pc++;
462         // IAND
463         }else if(strcmp(code, "iand") == 0){
464             getInst(inst, pc, atomid, buf, vartable, IAND, 4, 0, 0);
465             pc++;
466         // IOR
467         }else if(strcmp(code, "ior") == 0){
468             getInst(inst, pc, atomid, buf, vartable, IOR, 4, 0, 0);
469             pc++;
470         // IXOR
471         }else if(strcmp(code, "ixor") == 0){
472             getInst(inst, pc, atomid, buf, vartable, IXOR, 4, 0, 0);
473             pc++;
474         // ILT
475         }else if(strcmp(code, "ilt") == 0){
476             getInst(inst, pc, atomid, buf, vartable, ILT, 3, 0, 0);
477             pc++;
478         // ILE
479         }else if(strcmp(code, "ile") == 0){
480             getInst(inst, pc, atomid, buf, vartable, ILE, 3, 0, 0);
481             pc++;
482         // IGT
483         }else if(strcmp(code, "igt") == 0){
484             getInst(inst, pc, atomid, buf, vartable, IGT, 3, 0, 0);
485             pc++;
486         // IGE
487         }else if(strcmp(code, "ige") == 0){
```

```
488             getInst(inst, pc, atomid, buf, vartable, IGE, 3, 0, 0);
489             pc++;
490         // IEQ
491         }else if(strcmp(code, "ieq") == 0){
492             getInst(inst, pc, atomid, buf, vartable, IEQ, 3, 0, 0);
493             pc++;
494         // INE
495         }else if(strcmp(code, "ine") == 0){
496             getInst(inst, pc, atomid, buf, vartable, INE, 3, 0, 0);
497             pc++;
498         // 読み飛ばし
499         }else if(strcmp(code, "enqueueatom") == 0){
500         }else if(strcmp(code, "dequeueatom") == 0){
501         }else if(strcmp(code, "copyrules") == 0){
502         }else if(strcmp(code, "norules") == 0){
503         }else if(strcmp(code, "loadruleset") == 0){
504
505         // default
506         }else{
507             dbg("cannot translate code:%s, buf: %s", code, buf);
508         }
509     }
510     return 0;
511 }
```

# B.3 実行用シェルスクリプト

## B.3.1 eyeconv.sh

```
 1 #!/bin/sh
 2
 3 # please set LMNEYE = (executable file dir)
 4
 5 # remove temporary files
 6 rm -f temp_init.lmn temp_task*.lmn temp_init.tmp temp_task*.tmp atomid.dat
 7
 8 # split lmntal file to init & tasks
 9 if ! ${LMNEYE}/splitlmn < ${1%.lmn}.lmn
10 then echo "cannot split $1"; exit;
11 fi
12
13 # copy atomid template
14 cp ${LMNEYE}/template.aid temp_aid.aid
15
16 # convert init
17 java -jar ${LMNEYE}/lmntal.jar -d temp_init.lmn | ${LMNEYE}/translmn > temp_init.tmp
18
19 # convert task
20 for i in $(ls temp_task*.lmn)
21     do
22     java -jar ${LMNEYE}/lmntal.jar -d $i | ${LMNEYE}/translmn > ${i%.lmn}.tmp
23     done
24
25 # concat temp and generate binary tal file
26 tasktmp=$(ls temp_task*.tmp)
27 ${LMNEYE}/gental temp_init.tmp ${tasktmp} > ${1%.lmn}.tal
28
29 mv temp_aid.aid ${1%.lmn}.aid
30
31 if [ $# -lt 2 ]; then
32     rm -f temp*
33 elif [ $2 != noclean ]; then
34     rm -f temp*
```

```
35 fi
36
```

### B.3.2 runeye.sh

```
 1 #!/bin/sh
 2
 3 # run @ eyebot
 4 cat ${1%.tal}.tal > /dev/ttyS0
 5 echo "now waiting output of eyebot"
 6 ${LMNEYE}/bin2lmn ${1%.tal}.aid < /dev/ttyS0 > ${1%.tal}.out
 7 java -jar ${LMNEYE}/lmntal.jar ${1%.tal}.out
 8
 9 if [ $# -lt 2 ]; then
10     rm -f ${1%.tal}.out
11 elif [ $2 != noclean ]; then
12     rm -f ${1%.tal}.out
13 fi
14
```

### B.3.3 runpc.sh

```
 1 #!/bin/sh
 2
 3 # run @ pc
 4 ${LMNEYE}/lmntal < ${1%.tal}.tal | ${LMNEYE}/bin2lmn ${1%.tal}.aid > ${1%.tal}.out
 5 java -jar ${LMNEYE}/lmntal.jar ${1%.tal}.out
 6
 7 if [ $# -lt 2 ]; then
 8     rm -f ${1%.tal}.out
 9 elif [ $2 != noclean ]; then
10     rm -f ${1%.tal}.out
11 fi
```