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

Title	並行自己反映計算の宣言的記述に関する研究
Author(s)	石川, 洋
Citation	
Issue Date	1998-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/851
Rights	
Description	Supervisor:二木 厚吉, 情報科学研究科, 博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

博 士 論 文

並行自己反映計算の宣言的記述に関する研究

指導教官　二木 厚吉 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

石川 洋

1998 年 1 月 16 日

Copyright © 1998 by Hiroshi Ishikawa

要 旨

並行・分散システムにおいて自己反映計算を考慮した並行自己反映計算は，計算資源管理機構をその計算資源上で稼働するアプリケーションに対応したモジュールとして扱うことが可能である．そのため信頼性，効率性が高く，変更が容易な並行・分散システムを構築するための有用な枠組として認識されている．この枠組を利用してオペレーティングシステム，並行/並列オブジェクト指向言語，計算資源管理といった分野への応用例が提案されている．このようなシステムをより信頼性高く実装するためには，システムの正確な仕様記述が要求される．また，そのようなシステムの正確な仕様記述があれば，その記述からシステムの性質が解析できるようになると考えられる．

本研究では並行自己反映計算の形式仕様を，並行システムの動的な振舞いを宣言的に記述可能である書き換え論理に基づき，関数型言語および実行可能な処理系を持つ仕様記述言語を用いて与えた．形式仕様として操作的意味を与えている．本研究で与えた仕様は仕様記述言語の処理系によって実行可能であるので，例題を記述，実行することにより，その記述の妥当性を確認し，さらに，その記述を基にシステムの有意な性質の解析も試みている．本研究で考察した並行自己反映計算の例は以下のものである．

- 並行自己反映計算を構築する枠組のひとつであるグループワイド アーキテクチャ(Group-Wide Architecutre, GWA) について考察している．GWA のアクターモデルに基づいたモデル化に着目し，関数型言語を用いてメタレベルシステムの操作的意味を与え，その記述に基づいてオブジェクト移送の例題を実行した．さらにその記述を使用し，メタレベルシステムの正当性について証明を与えた．

- 有機的プログラミング言語 GAEA の操作的意味を，GAEA 開発メンバーからの意見を参考にしながら，仕様記述言語を用いて与えた．その記述に基づいて自律ロボットの例題の実行した．さらに，その記述を使用し，同期処理，マルチプロセス処理について考察した．

本研究では，書き換え論理に基づいた並行自己反映計の操作的意味記述は，システムの複雑な動作を理解したり，システムの性質を解析するための有用な手段として期待できることを示している．

目 次

1 序論 **1**

2 準備 **4**

2.1 並行自己反映計算 4

2.1.1 自己反映計算 4

2.1.2 並行自己反映計算の枠組 6

2.2 書き換え論理 8

2.2.1 書き換え論理の定義 8

2.2.2 書き換え論理による並行計算システムの表現例 10

2.3 仕様記述 11

3 グループワイド アーキテクチャの宣言的記述 **13**

3.1 アクターモデル 13

3.1.1 アクターモデル 13

3.1.2 書き換え論理のアクターモデルとの対応 15

3.2 アクターによる GWA の書き換え論理に基づく記述 16

3.2.1 GWA 16

3.2.2 メタレベルアーキテクチャ 17

3.2.3 自己反映計算の実現 19

3.3 書き換え論理によるモデル化 21

3.3.1 項の定義 21

3.3.2 書き換え規則の例 24

3.3.3 自己反映計算の実現について 25

3.4 例題 — オブジェクト移送 — 26
3.4.1 問題の設定 26
3.4.2 アクターモデルの場合 26
3.4.3 本実装の場合 28
3.4.4 メタレベルシステムの正当性 32

4 有機的プログラミング言語 GAEA の操作的意味 39
4.1 有機的プログラミング言語 GAEA 39
4.2 有機的プログラミング言語 GAEA の操作的意味 51
4.2.1 Maude 51
4.2.2 有機的プログラミング言語 GAEA の操作的意味 54
4.2.3 操作的意味その 1 56
4.2.4 操作的意味その 1 の反省 64
4.2.5 操作的意味その 2 65
4.3 自律ロボットの例題 72
4.4 考察 72
4.4.1 待ち状態の表現 72
4.4.2 同期処理 73
4.4.3 他プロセスへの作用 79
4.4.4 言語仕様の考察 80

5 関連研究 83
5.1 自己反映計算の操作的意味 83
5.2 並行計算と自己反映計算 84
5.3 書き換え論理と仕様記述 85
5.4 書き換え論理と自己反映計算 86

6 結論および今後の課題 88
6.1 結論 88
6.1.1 グループワイドアーキテクチャ関連 88
6.1.2 GAEA 関連 89
6.2 今後の課題 90

6.2.1 グループワイドアーキテクチャ関連 90
6.2.2 GAEA 関連 91

謝辞 **94**

参考文献 **95**

本研究に関する発表論文 **100**

Appendix A **102**

Appendix B **163**

図 目 次

2.1 リフレクティブなシステム: (a) 概念図 [28], (b) リフレクティブタワー . . . 6
2.2 並行自己反映計算: (a)IBA,(b)GWA 7
2.3 銀行口座の例 11

3.1 メッセージ送信の差異: (a) 逐次オブジェクト指向モデル,(b) アクターモデル 14
3.2 メタレベルのアクターとメッセージの流れ [45] 18
3.3 ベースレベルシステムからメタレベルシステムへのメッセージ送信 20
3.4 メタレベルシステムからベースレベルシステムへのメッセージ送信 20
3.5 基本的なデータタイプの宣言 22
3.6 オブジェクトとメッセージの定義 23
3.7 コンフィギュレーションの定義 24
3.8 Task Handler オブジェクトの書き換え規則 24
3.9 Task Handler オブジェクトの書き換え規則 (Customer オブジェクトの場合) 25
3.10 グループ (ノードアクター) の定義 27
3.11 Migrator アクターの振舞いの定義 27
3.12 オブジェクト移送に関する書き換え規則 29
3.13 例題のデータ 30
3.14 例題の実行例 32

4.1 Prolog と GAEA のデータベース 41
4.2 環境の操作 42
4.3 GAEA によるプログラム例 47
4.4 GAEA での節定義 47
4.5 通常の Prolog で書き直した場合 47

4.6 節定義を含むセルの定義方法 . . . 48
4.7 未処理のゴール列 . . . 48
4.8 agent/2 のボディ部 . . . 49
4.9 環境 (セル名のスタック, 左が先頭) . . . 49
4.10 セル proceed の一番目の節定義 . . . 49
4.11 環境の変化 . . . 50
4.12 セル want の節定義 . . . 50
4.13 GAEA の仕様 (始めの部分) . . . 52
4.14 ソート宣言の例 . . . 52
4.15 サブソート宣言の例 . . . 52
4.16 オペレータ宣言 . . . 52
4.17 変数宣言 . . . 53
4.18 等式宣言 . . . 53
4.19 書き換え規則宣言 . . . 53
4.20 計算状態の遷移 . . . 55
4.21 セルの定義 . . . 56
4.22 プロセスの定義その 1 . . . 56
4.23 システムの状態 . . . 57
4.24 分類規則 . . . 58
4.25 主要規則その 1 . . . 60
4.26 主要規則その 2 . . . 61
4.27 述語 push(Cell) に対応する書き換え規則 . . . 62
4.28 述語 subst_cell(Old,New) に対応する書き換え規則 . . . 64
4.29 述語 cv_write(Cell,CV,V) に対応する書き換え規則 . . . 64
4.30 プロセスの定義その 2 . . . 66
4.31 セルやプロセスに含まれる項の定義 . . . 66
4.32 プロセスの遷移 . . . 67
4.33 セル名の取り出し . . . 68
4.34 プロセスの初期状態 . . . 68
4.35 セル名が獲得できた状態 . . . 68

4.36 セル名が獲得できなった状態 68
4.37 節定義の獲得 69
4.38 節定義を得るための状態 69
4.39 節定義が得られた状態 69
4.40 節定義が得られなかった状態 70
4.41 節定義のための構成子 70
4.42 ユニフィケーション成功 70
4.43 ユニフィケーション失敗 71
4.44 次のセル名に移る 71
4.45 組み込み述語適用の準備 71
4.46 組み込み述語push/1 に対応する書換え規則 73
4.47 述語 cv_write(Cell,CV,V) に対応する書き換え規則 73
4.48 プロセス'p1 の項表現 74
4.49 プロセス'p2 の項表現 74
4.50 ロボット'r1 が交差点にいる場合の状態表現 74
4.51 セル'want に格納されている一番目の節定義 75
4.52 述語find/1 の定義 75
4.53 組み込み述語cv-rev/3 の呼出し 75
4.54 ロボット'r2 のモードが変化 75
4.55 交差点に誰も存在しない状態 76
4.56 セル'want に格納されている二番目の節定義 76
4.57 プロセス'p1 の変化 77
4.58 プロセス'p2 の変化 77
4.59 組み込み述語cv-write/3 に対応する状態遷移規則 78
4.60 状態遷移規則適用前の状態 78
4.61 状態遷移規則適用後の状態 78
4.62 述語 push(Cell,P2) に対応する書き換え規則 80
4.63 本研究の項設計に基づく環境変化後のバックトラック 81
4.64 GAEA システムでの環境変化後のバックトラック 82

6.1 組み込み述語cv-write/3 に対応する条件付き状態遷移規則 92

6.2　組み込み述語cv-write/3 に対応する条件のない状態遷移規則 93

表 目 次

3.1 アクターモデルとオブジェクト指向モデルとの間の用語の対応 16
3.2 記法の定義 33
3.3 遷移関係の定義 35

4.1 同期処理機構に関連するセル変数操作述語 43

第 1 章

序論

近年の様々な研究により，並行・分散システム [51] において自己反映計算を考慮した並行自己反映計算は，信頼性，効率性が高く，変更が容易な並行・分散システムを構築するための有用な枠組として認識されている [46]．その理由として，並行自己反映計算は，計算資源管理機構をその計算資源上で稼働するアプリケーションに応じたモジュールとして扱うことが可能であることが挙げられる．

並行・分散システムでは，一般にシステムの分散の度合が大きくなると，システム全体に関する自分自身の構造や計算過程をモデル化した表現を一ヶ所にまとめた表現として構築することは不可能に近くなる．そこで，システムを機能単位に分割し，それらを組み合わせることによってシステム全体を表現するという構築方法を考えられる．このような構築方法においては，メモリやハードディスクといった個々の計算資源やその上で動作させるアプリケーションなどが機能単位に分割されたモジュールとみなすことができる．計算資源管理はアプリケーションの計算過程の制御を行なうので，アプリケーションの立場からすれば，自分自身の計算を制御するメタレベルの計算を行なっていることになる．これらのモジュールを有効に活用するために，並行・分散システムに自己反映計算を導入することが考えられた．

並行・分散システムに自己反映計算を応用する場合，システムの部分的なメタレベル表現が分散して存在させる形式が 2 種類提案されている．それらは，個々の並行オブジェクトがメタレベル表現の単位となっている形式 [44]，および，複数の並行オブジェクトから構成されるグループがメタレベル表現の単位となっている形式 [45] である．後者はグループワイドアーキテクチャ(Group-Wide Architecture, GWA) と呼ばれ，前者では表現が困難

な複数の並行オブジェクトに関連する大域的な情報に関わる自己反映計算が可能となる．

以上に述べたようなシステムを，より信頼性高く設計，構築，保守，拡張する手段のひとつとして，対象システムの正確な仕様記述を与えることが考えられる．高品質な仕様記述が与えられれば，その記述からシステムをより信頼性高く実現できるだけでなく，システム自身の性質が解析できるようになると考えられる．

本研究では，並行自己反映計算の例として，グループワイドアーキテクチャおよび有機的プログラミング言語 GAEA をとりあげ，それらの操作的意味を記述し，その記述に基づいてシステムが持つ性質についての考察を行なっている．ここでいう操作的意味とは，プログラムの文面およびデータを入力して，プログラムの実行をシミュレートする抽象機械を定義し，その抽象機械の動作の系列を定義することによってプログラムの意味を与える方法のことをいう．

本研究の主要部分では，まず始めにグループワイドアーキテクチャに関連する研究について述べている [17][18][19][20] [21][22]．先にも述べたように，グループワイドアーキテクチャは並行・分散システムに自己反映計算を提供する枠組として知られてる．自己反映計算が可能な計算システムを実現する手段のひとつに，リフレクティブタワーと呼ばれる計算システムの階層構造を構築するものがある．このようは構造においては常に階層間の関係に留意して仕様を記述する必要がある．ある時点においては，計算を行なう主体であったシステムが，次の瞬間にはデータとして扱われるため，計算が実行される階層が動的に変化する．したがって，リフレクティブタワーの仕様を記述するのは困難であると考えられる．そこで，我々はベースレベルシステム (またはオブジェクトレベルシステムともいう) の一段だけ上位レベルの計算システムである，メタレベルシステムに着目した．メタレベルシステムにおいては，ベースレベルシステムはメタレベルシステムでのデータとして扱われる．本研究では

- ベースレベルシステムのメタレベル表現
- メタレベルシステムの構成
- メタレベルシステムの操作的意味

についての記述を与えた．また，システムの性質の解析という観点では，与えた記述に基づいてメタレベルシステムの正当性を証明した．

次に，GAEA に関する研究について述べている [23][24][25]．GAEA[1] は，有機的プログラミング [36][37] と呼ばれる，柔軟に協調的動作を行なうプログラムを構築する方法論に基づいた言語である．この方法論は，動作環境の状況推論，セルと呼ばれるプログラム記述単位の操作，および自己反映計算といった概念を合わせ持っている．また，GAEA はマルチスレッド処理が可能である．したがって GAEA は並行自己反映計算の一例と位置付けることが可能である．GAEA における自己反映計算は前出のような自己反映計算ではなく，文脈的自己反映計算と呼ばれるものである．GAEA は論理型プログラミング言語 Prolog[3][40] を基礎としているので，節定義の集合がプログラムを構成する．GAEA における自己反映計算は，プログラムを構成する節定義そのものを変更するものではなく，節定義の順番を状況に応じて動的に変化させるものである．

有機的プログラミング言語 GAEA においては，GAEA 自身の操作的意味，いわゆるベースレベルシステムの操作的意味を与えている．これにより，自己反映計算を実現している組み込み述語やセル変数を操作する述語の操作的意味がより明確になっている．

上記 2 題の研究では，書き換え論理 [29] [30] [31] [32] と呼ばれる，並行システムの動的な振舞いを表現することが可能な論理を用いて操作的意味を与えている．

また，仕様記述には関数型言語および処理系を持つ仕様記述言語を使用している．したがって，単に並行自己反映計算システムの仕様を記述するだけでなく，その記述に基づいた例題を実際に処理系で動かすことにより，記述の妥当性を確認している．本研究では，第 3章においては関数型言語 Gofer[11] を使用した．仕様記述言語として CafeOBJ[35] および Maude[32][5] を使用した．ただし，第 4章や付録においては，混乱を避ける目的で Maude の記述のみを用いて操作的意味の説明をおこなっている．

本論文の構成は以下のようである．第 2章では，本研究で用いた諸概念について概説している．第 3章では，並行自己反映計算を構築する枠組のひとつとして提案されている GWA の宣言的記述に関して述べている．前半は，アクターモデルに基づいた GWA のモデル化について，後半は，書き換え論理に基づいた GWA のモデル化について言及している．第 4章では，並行自己反映言語とみなすことが可能な，有機的プログラミング言語 GAEA に操作的意味を与えている．第 5章では関連研究について簡単に触れ，第 6章では，本研究の結論と今後の課題について述べている．また，付録として第 4章で導入した GAEA の操作的意味の記述，および，その記述に基づいた自律ロボットの例題の実行結果を与えている．

第 2 章

準備

本章では，本研究で必要となる諸概念である，並行自己反映計算，書き換え論理，仕様記述言語について概説する．有機的プログラミング言語 GAEA および仕様記述言語 Maude の解説は第 4章で行なっている．

2.1 並行自己反映計算

この節では自己反映計算と，並行・分散システムにおける自己反映計算の基本的な 2 つの枠組である

- 個別の並行オブジェクトについてのメタレベルアーキテクチャ (Individual-Based Architecture(IBA))，および
- 複数の並行オブジェクトから構成されるグループに関するメタレベルアーキテクチャ (Group-Wide Architecture(GWA))

についての一般的な解説を行なう．GWA については加えて第 3章において詳細を述べることにする．

2.1.1 自己反映計算

計算システムは，計算システムの領域と呼ばれる世界のある部分について判断，作用するものである．ここでいう計算システムとはその領域を表現するデータ，データの処理手

順をあるプログラミング言語で記述しておくプログラム，および処理を実行する評価器から構成されると考える．計算システムはその領域について新しい情報を伝達したり，その領域に作用するという結果を返す.

計算システムとその領域において，もしそれら二つのうちの一方が変化するとき，もう一方もそれに応じて適切に変化するという関係があるとき，それらは因果的結合 (cousal connection) があるという.

ある計算システムがその問題領域と因果的結合をもつとは,

- 計算システムのデータが変化するとき，これらのデータで表現された実体は影響を受ける.
- 領域の実体が変化するとき，計算システム内のこれらを表現しているデータが影響を受ける.

ときをいう.

メタシステムとは，その領域としてオブジェクトシステムと呼ばれる他の計算システムを持つ計算システムのことをいう．したがって、メタシステムは他の計算システムについて判断し，それに作用するシステムであるということができる．メタシステムはそのデータの中にオブジェクトシステムの表現を持つ．そのプログラムはオブジェクトシステムについてのメタ計算を特定するので，それはメタプログラムと呼ばれる．メタ計算とは，メタシステムが行なう計算のことであり，オブジェクトシステムについての新しい情報を返したり、実際にオブジェクトシステムに作用するものである.

オブジェクトシステムやそのメタシステムは一般には様々な言語によって記述されるが，ここでは共に同一言語で記述されていると考えることにする．このとき，メタシステムは，その言語のメタインタプリタと呼ばれる．この場合，オブジェクトシステムにおける計算をメタシステムで実行することが可能である．したがって，オブジェクトシステムからそのメタレベル表現を操作，参照することが可能であると考えられる．また，オブジェクトシステムのメタレベル表現への操作は，オブジェクトシステム自身に反映される．すなわち，オブジェクトシステムと因果的に結合したそのメタレベル表現を得ることができる.

このように計算システムが，自分自身の構造や計算過程をモデル化した表現を内部に持ち，その表現を操作することにより，自己の構造や計算過程の進行を動的に操作することを自己反映計算と呼ぶ [28][46]．このとき，その計算システムと自分自身の構造や計算過程をモデル化した表現との間には因果的結合があるという (図 2.1(a)).

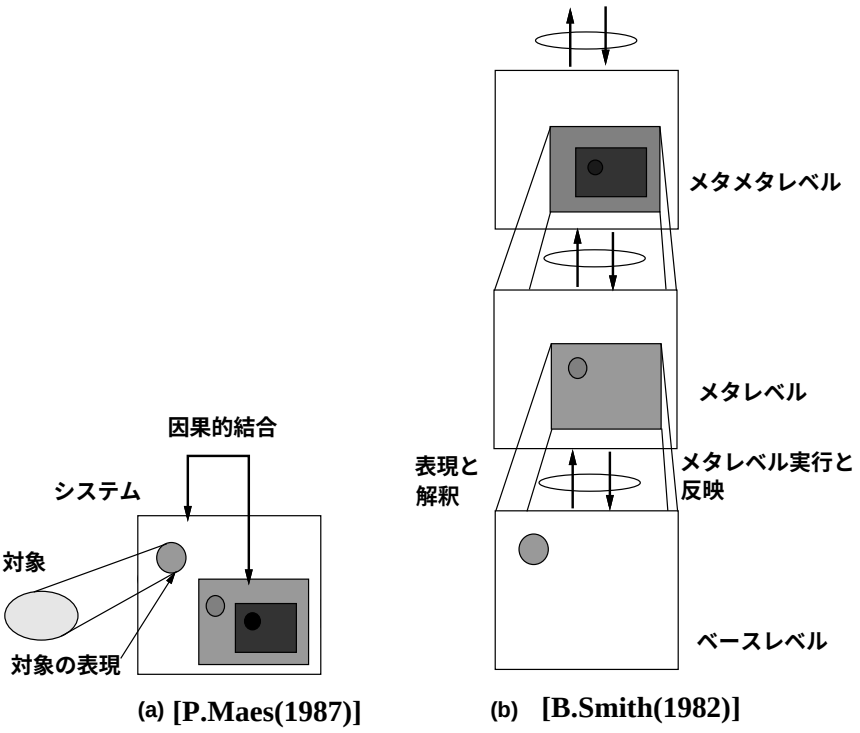

図 2.1: リフレクティブなシステム: (a) 概念図 [28], (b) リフレクティブタワー

自己反映計算を実現化する手法の代表的なものにメタインタプリタの無限階層 (リフレクティブタワー (図 2.1(b))) を利用したものがある [39][46]．これにより，システム本来の計算を記述しているプログラムは同一の言語で記述されたインタプリタ (メタレベル) で解釈実行される．メタレベルはさらに同様に定義された別のインタプリタ (メタメタレベル) で解釈実行される．これによって因果的結合が自然に成立し，自己反映計算が可能となる．このように解釈実行の手続きとしてメタレベルを構成し，自己反映計算を実現する方法が一般的に用いられる．モデルを考える上ではリフレクティブタワーの階層は無限であるが，実システムを構築する場合は，リフレクティブタワーを有限の計算資源によって実現する必要がある．

2.1.2 並行自己反映計算の枠組

並行・分散システムでは，一般にシステムの分散の度合が大きくなると，全体に関する自分自身の構造や計算過程をモデル化した表現を一ヶ所にまとめた表現として構築することは不可能に近い．よって並行・分散システムでは，システムの部分的なメタレベル表現

が分散して存在する形式が考えられる．その形式は大別して 2 通り提案されている．それらについて概要を述べる．

IBA

個々の並行オブジェクトがメタレベル表現の単位となっている形式で，Individual-Based Architecture(IBA)[44] と呼ばれている (図 2.2(a))．この形式では，各並行オブジェクトについて必ずメタオブジェクトが存在している．あるオブジェクトに対するメタレベル操作は，そのメタオブジェクトにメッセージを送信することによって実行される．自分のメタオブジェクトにメッセージを送信し，そのオブジェクト単位での自己反映計算が実行される．この形式に基づくシステムには，並列オブジェクト指向言語 ABCL/R[44] がある．

GWA

複数の並行オブジェクトから構成されるグループがメタレベル表現の単位となっている形式で，Group-Wide Architecture(GWA)[45] と呼ばれている (図 2.2(b))．IBA では表現が困難な，複数の並行オブジェクトから構成されたグループに関する大域的な情報に関わる自己反映計算を行なうことが可能である．この形式に基づくシステムには，アクターモデル [2] に基づいた並行オブジェクト指向言語 ACT/R[45] がある．

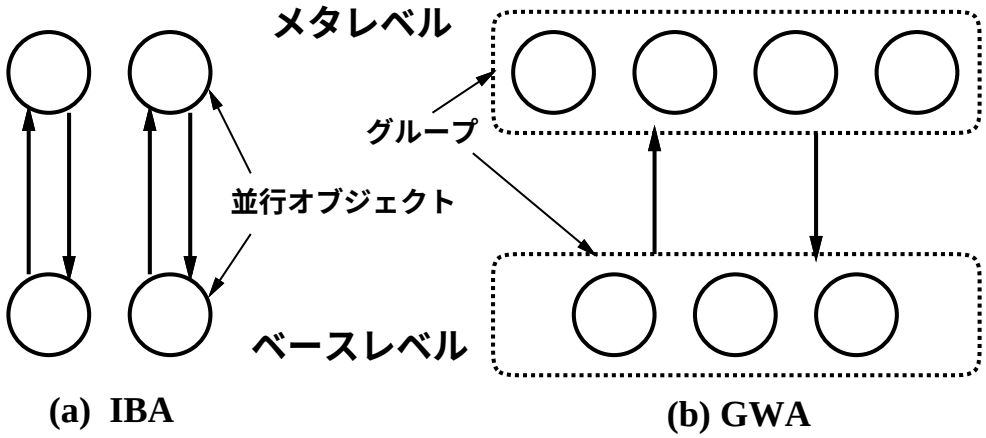

図 2.2: 並行自己反映計算: (a)IBA,(b)GWA

2.2 書き換え論理

書き換え論理 [29][30] [31][32] はλ-計算，ペトリネット，CCS など他の計算モデルを包含する一般的な論理である．書き換え論理は計算システムの動的な変化が表現可能な論理で，計算システムの状態は命題に対応し，計算システムの遷移は演繹に対応する．そして書き換え論理の意味は，状態を持つ並行計算システムに対応する．以下に書き換え論理の定義を与える．

2.2.1 書き換え論理の定義

ランク付けされた関数記号の集合を $\Sigma = \{\Sigma_n \mid n \in \mathbf{N}\}$ とし，変数の集合を $X = \{x_1, \ldots, x_n, \ldots\}$ としたとき，全ての項からなる集合 $T_\Sigma(X)$ は以下のように定義される．

1. $a \in \Sigma_0 \Rightarrow a \in T_\Sigma(X)$
2. $x \in X \Rightarrow x \in T_\Sigma(X)$
3. $t_1, \ldots, t_n \in T_\Sigma(X) \wedge f \in \Sigma_n \Rightarrow f(t_1, \ldots, t_n) \in T_\Sigma(X)$

また項の間の同値関係を定める等式の集合を E とするとき， E に関する項の同値類からなる集合を $T_{\Sigma,E}(X)$ で表し，$[t]_E$ あるいは $[t]$ で E に関する t の同値類を表すものとする．以降の議論ではこの項の同値類を対象とし，同値類に対して推論規則が定められているものとする．つまり E によって $t_1 = {t'}_1, t_2 = {t'}_2$ であるならば，$t_1 \to t_2$ が出現している場所を ${t'}_1 \to {t'}_2$ で置き換えることを許している．例えば，並行オブジェクト指向計算の計算システムの状態はマルチセットとして表現されるので，項の並びに依存することなく書き換えを行なうことが可能となる．

定義 1 書き換え定理 $\mathcal{R}$ は (Σ, E, L, R) という四つ組である．ただし，Σ はランク付けされた関数記号の集合，E は $T_\Sigma(X)$ の 2 つの要素の間の等式の集合， L は書き換え規則に付けることが可能なラベルの集合[1]， R は $(L \times (T_{\Sigma,E}(X)^2))$ の部分集合で R の要素は書き換え規則と呼ばれる．書き換え規則 $(r : ([t], [t']))$ は $r : [t] \to [t']$ と表記する．また t, t' に出現する変数 $\{x_1, \ldots, x_n\}$ を表記するために $r : [t(x_1, \ldots, x_n)] \to [t'(x_1, \ldots, x_n)]$ あるいは $r : [t(\bar{x})] \to [t'(\bar{x})]$ と書くことにする．項 t の中の変数 x に項 t' を代入した項を $t(t'/x)$ で表現し，同時代入は $t(\bar{t'}/\bar{x})$ で表すことにする． ■

[1] 本研究ではラベルは使用していない．

定義 2　書き換え定理 $\mathcal{R}$ に対し，以下の演繹規則から $[t] \to [t']$ が導かれるとき，$\mathcal{R} \vdash [t] \to [t']$ と書くことにする．

1.Reflexivity. 任意の項 t に対して

$$\frac{}{[t] \to [t]}$$

2.Congruence. n 個の引数をとる関数記号 f に対して

$$\frac{[t_1] \to [t'_1] \quad \cdots \quad [t_n] \to [t'_n]}{[f(t_1, \ldots, t_n)] \quad \to \quad [f(t'_1, \ldots, t'_n)]}$$

3.Replacement.

$r : [t(x_1, \ldots, x_n)] \to [t'(x_1, \ldots, x_n)] \in R$ のとき

$$\frac{[\omega_1] \to [\omega'_1] \quad \cdots \quad [\omega_n] \to [\omega'_n]}{[t(\bar{\omega}/\bar{x})] \quad \to \quad [t'(\bar{\omega}'/\bar{x})]}$$

4.Transitivity.

$$\frac{[t_1] \to [t_2] \qquad [t_2] \to [t_3]}{[t_1] \quad \to \quad [t_3]}$$

■

この定義に以下の規則を追加すると等式論理を得ることができる．

5.Symmetry.

$$\frac{[t_1] \to [t_2]}{[t_2] \to [t_1]}$$

このことから書き換え論理は等式論理を含む一般的な論理体系であるといえる．

2.2.2 書き換え論理による並行計算システムの表現例

書き換え論理による並行計算システムの記述として銀行口座の例 [32] を紹介する．この場合，銀行口座と口座を操作するための合図 (メッセージ) を項によって表現する．さらに，これらの集合によってある時点における銀行口座の状態を表す．口座は口座番号によって一意に定められ，口座を操作するための合図も識別子を持つことによって一意に定めることが可能なので，同一の情報が複数存在することはない．

銀行口座の状態は口座を操作する合図に応じて変化する．状態の合図に応じた変化を状態遷移規則によって規定する．具体的には，状態遷移規則の左辺は口座と口座を操作する合図の組，右辺は変更後の口座によって定める．

簡単のため以下のような項や規則によって銀行口座の例を考える．個人口座は口座の名前で識別され，残高のみをその情報として持っているものとする．個人口座を操作するメッセージは貸し方 (credit) と借り方 (debit) の 2 種類とする．

- 個人口座

  < 個人名 : Accnt | bal : 金額 >

- 個人口座を操作する合図 (メッセージ)

  credit(個人名 , 金額)
  debit(個人名 , 金額)

- 口座の操作 (状態遷移規則)

  貸し方

  ```
  < Name : Accnt | bal : M > credit( Name , N )
    => < Name : Accnt : bal : M + N >
  ```

  借り方

  ```
  < Name : Accnt | bal : M > debit( Name , N )
    => < Name : Accnt : bal : M - N >
  ```

書き換え論理では並行書き換えを基礎としているので，書き換え規則は計算状態を表現している共通部分を持たない複数の部分項に同時に適用されることを想定している．した

がって，例えば図2.3の左の状態においては3つ部分項に対して状態遷移規則が同時に適用可能であり，適用の結果，右の状態に遷移する．

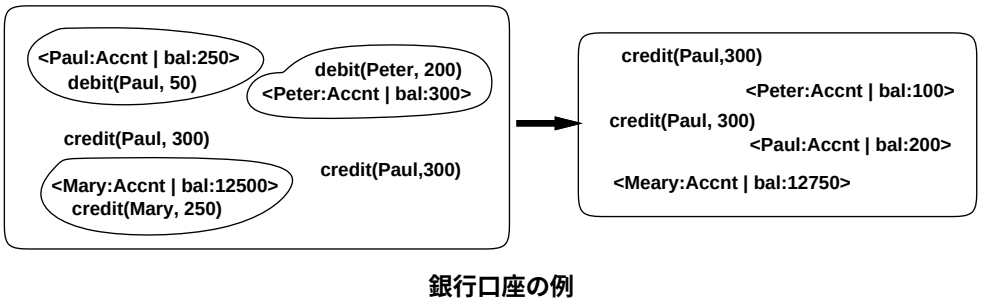

図 2.3: 銀行口座の例

2.3 仕様記述

システムの意味を数学的に厳密に定義したり，検証をおこなうための基礎を提供する方法として，代数仕様記述法 [13][14][15][16] [12][7] が提案されている．代数仕様は抽象代数をモデルとする形式仕様記述のひとつであり，一般的には

- 抽象データ型の概念に厳密なモデルの提供
- 等式論理に基づく意味論
- 項書換えシステム [8][12] による操作的な意味の提供

といった特徴を備えている．本研究では，等式論理の部分はその論理体系を包含可能な書き換え論理に置き換わっている．

特徴の第一にある抽象代数とは台集合とその上の演算とから構成され，演算の性質，振舞いによって規定される構造を持つ．これは抽象データ型の「表現形態とは独立に，演算の定義だけを通してデータ構造を定義する」考え方と非常に馴染みやすい．ただし，従来の抽象代数は主に一つの台集合上の演算に着目したのに対し，抽象データ型は複数のデータ型の存在を自明とみる．

群のように台集合が一つであるような代数を単一ソート代数と呼ぶのに対し，複数の台集合とその間の演算から構成される代数は多ソート代数と呼ばれる．代数仕様で使われるモデルはこの多ソート代数である．

ある代数を定めるためには，台集合と演算，またその振舞いを与える必要がある．したがって，記述要素には少なくとも次のものが必要である．

- 台集合の種類

  代数仕様では，台集合はソートと呼ばれる．仕様の記述ではソートを表す識別子を宣言する必要がある．

- 演算の種類とその型

  多ソート代数では，単に引数の数だけでなく，それぞれの引数がどのソートに属しているかを明示しなければならない．したがって，記述には，演算の識別子，引数ソート名の並び，値のソート名が含まれる必要がある．

- 演算の定義

  各演算は引数ソートに属する要素とその値の組で定義される．代数仕様では，各ソートの要素は関数合成によって得られる項で表されるから，この定義は引数を表す項 t_i に演算 f を適用した結果を表す合成項 $f(t_1, \ldots, t_n)$ が他のどのような項に等しいかを表す等式による．書き換え論理においては，項によって表現された計算システムの状態を遷移させる規則を定義する．

仕様記述言語はいくつか提案されているが，実行可能な処理系を持つ仕様記述言語を利用すれば，単に仕様を与えるだけでなく，その記述と処理系を通して記述の検証、解析を自動的に行なうことが可能となる．

本研究では，代数仕様言語として CafeOBJ[35] および Maude(第 4章を参照) を採用している．代数仕様言語 CafeOBJ は OBJ3[10] の流れを汲む言語であり，多ソート論理 [13][7]，順序ソート論理 [7]，Hidden Algebra[9]，書き換え論理などの，複数の論理を組み合わせることにより，より適切な仕様を記述することが可能である．また，実行可能な処理系を持っているので，仕様記述対象を CafeOBJ で記述することによって，単に宣言的記述を与えるだけでなく，その記述を実行することができる．これにより，与えた記述の正確さや仕様記述対象が持つ性質の解析についての考察が可能となる．

第 3 章

グループワイドアーキテクチャの宣言的記述

本章では，アクターモデルに基づくグループワイドアーキテクチャのモデル化について解説し，そのメタレベルシステムに対して操作的意味を与える．さらに，その操作的意味に基づいて，オブジェクト移送の例を考察したので，その概要を説明する．最後にメタレベルシステムの正当性を証明している．

3.1 アクターモデル

本節では，アクターモデルの概説と，アクターモデルと書き換え論理との関連について述べる．

3.1.1 アクターモデル

アクターモデル [2][41] は並行計算モデルのひとつであり，その計算システムは，アクターと呼ばれる能動的な計算主体の集まりでモデル化されている．アクターはそれぞれ独立した計算能力を持ち，互いにメッセージ通信を行ないながら計算をすすめていく．メッセージを受理したアクターはそれを自分で解釈し，そのメッセージに対応する計算を実行する．図 3.1は逐次オブジェクト指向モデルとアクターモデル (並行オブジェクト指向モデル) のメッセージ送信の差異を示すものである．図中のオブジェクト内の縦軸は時間の流れ，破

線はオブジェクトが動作していない状態，オブジェクト間の矢印は制御の流れを表現している．

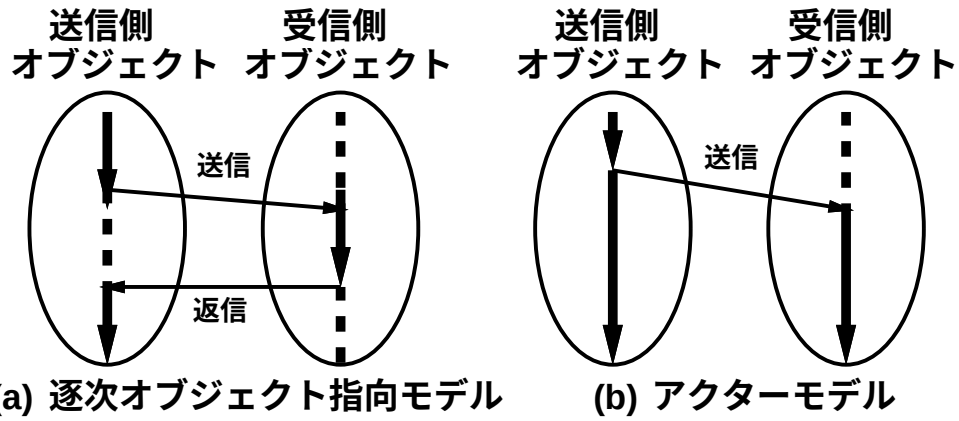

図 3.1: メッセージ送信の差異: (a) 逐次オブジェクト指向モデル,(b) アクターモデル

アクター間で共有されるものはそれぞれのメールアドレス[1]のみである．アクターがメッセージを送信するときは常に宛先のメールアドレスを明示的に指定する．メッセージ通信は一対一，非同期で行なわれる．送信されたメッセージの有限時間内での到着は保証されているが，送信順序に対する到着順序の保存は保証されていない．アクターモデルにおける非決定性はここに見られる．

一つのメッセージに対するアクター内の計算は必ず有限時間内で終了することが仮定されている．従って，`while` 文に相当する非有界の繰り返し処理は自分に対するメッセージ送信として表現される．メッセージを受信したアクター内では

- 自分自身の状態の変更
- 有限個のアクターの生成
- 有限個のメッセージ送信

といった計算が行なわれる．

アクターモデルにおいて，計算システムはアクターとメッセージの集合で表現される．計算システムのある時点におけるスナップショットを計算状態 (コンフィギュレーション) という．アクターモデルのような計算システムでは，計算はコンフィギュレーションの遷移

[1] 各アクタに一意に割り当てられており，アクターの識別子となる．

とみなすことができ，計算の意味はコンフィギュレーションの遷移列で表現される．ただし，公平性に関する条件があり，永久に処理されないようなメッセージが存在するコンフィギュレーションの列は計算の意味から除外される．

3.1.2 書き換え論理のアクターモデルとの対応

本節では，並行オブジェクト指向計算の書き換え論理によるモデル化の手法と，アクターモデルとオブジェクト指向プログラミングとの間の用語の対応について簡単に述べる [32]．一般的な並行オブジェクト指向計算の場合，書き換え規則[2]を

$$\begin{aligned} M_1 \ldots M_n O_1 \ldots O_m \quad \rightarrow \quad & O_{i_1} \ldots O_{i_k} \\ & Q_1 \ldots Q_p \\ & {M_1}' \ldots {M_q}' \\ & if\ C \end{aligned}$$

と定義する．この規則は

- 条件 C を充足しているならば，
- メッセージ $M_1 \ldots M_n$ が消滅し，
- オブジェクト $O_{i_1} \ldots O_{i_k}$ が変化し，
- 新しいオブジェクト $Q_1 \ldots Q_p$ が生成され，
- 新たにメッセージ ${M_1}' \ldots {M_q}'$ が送信される

という動作を表現したものである．アクターモデルの場合はひとつのアクターはメッセージをひとつづつ処理するので，書き換え規則は

$$\begin{aligned} M\ O \quad \rightarrow \quad & O' \\ & Q_1 \ldots Q_p \\ & {M_1}' \ldots {M_q}' \end{aligned}$$

[2]計算状態 (コンフィグレーション) を遷移させるので，状態遷移規則と呼ぶこともある．

というように，書き換え規則の左辺は，1 個のアクターと 1 個のメッセージの対によって規定される．またアクターモデルとオブジェクト指向モデルとの間には表 3.1のような対応関係がある．

アクターモデル	オブジェクト指向モデル
スクリプト (Script)	クラス宣言 (Class declaration)
アクター (Actor)	オブジェクト (Object)
タスク (Task)	メッセージ (Message)
自分自身が知っている アクターの名前 (Acquaintances)	属性値として持っている オブジェクトの名前 (Attributes)

表 3.1: アクターモデルとオブジェクト指向モデルとの間の用語の対応

3.2 アクターによる GWA の書き換え論理に基づく記述

本節ではアクターモデルに基づく Group-Wide Architecture(GWA)[45] のモデル化についての概説を行なう．そのモデルを書き換え論理によって再度モデル化したので，その解説を行なう．本研究では，書き換え論理に基づく記述を用いてオブジェクト移送の例題を記述，実行した．またメタレベルシステムの性質について考察した．よってこれらについても言及する．

3.2.1 GWA

並行自己反映計算の枠組の一つである Group-WideArchitecture(GWA) は，複数のオブジェクトから構成されたグループがメタレベル表現の単位となっている．これにより，メッセージ通信の意味の動的変更や資源管理などの，複数の並行オブジェクトから構成されたグループに関する大域的な情報に関する自己反映計算が可能となる．

GWA の構成のためには，複数の並行オブジェクトから構成されるグループの実現には依存しないモデル化や，グループ間の計算の意味を正確に捉える必要がある．その理由により，[45] では基礎となる計算モデルとしてアクターモデルを採用している．

3.2.2 メタレベルアーキテクチャ

複数のアクターで構成されるグループ S に対し，これをモデル化するメタレベルグループ $\uparrow S$ を導入する．$\uparrow S$ も複数のアクターから構成されており，S を遷移システムとして見た場合のインタプリタとなっている．そして S内のアクターと$\uparrow S$ 内のアクターがメッセージ通信を行なうための機構を設け，Sにおける自己反映計算を実現している．

計算状態 (以降，コンフィギュレーションと記述する) はベースレベルのある時点におけるスナップショットであり，アクターおよび未処理の (アクターに受理されていない) メッセージの集合として構成される．

メタ計算状態 (以降，メタコンフィギュレーションと記述する) はメタレベルのある時点におけるスナップショットであり，コンフィギュレーションと同様にアクターおよびメッセージの集合によって構成される．ある時点のコンフィギュレーションの情報は，ある時点のメタコンフィギュレーションに完全に包含されている．逆は明らかに成立しないことに注意されたい．

メタレベルを構成するアクターは図3.2のようになっている．ここで，四角で囲まれた図形はアクター，矢印はメッセージが送信される方向，矢印の脇の記述は送信される (相手のアクターが受理可能な) メッセージの種類，破線は間接的な関連および呼出しを表現している．

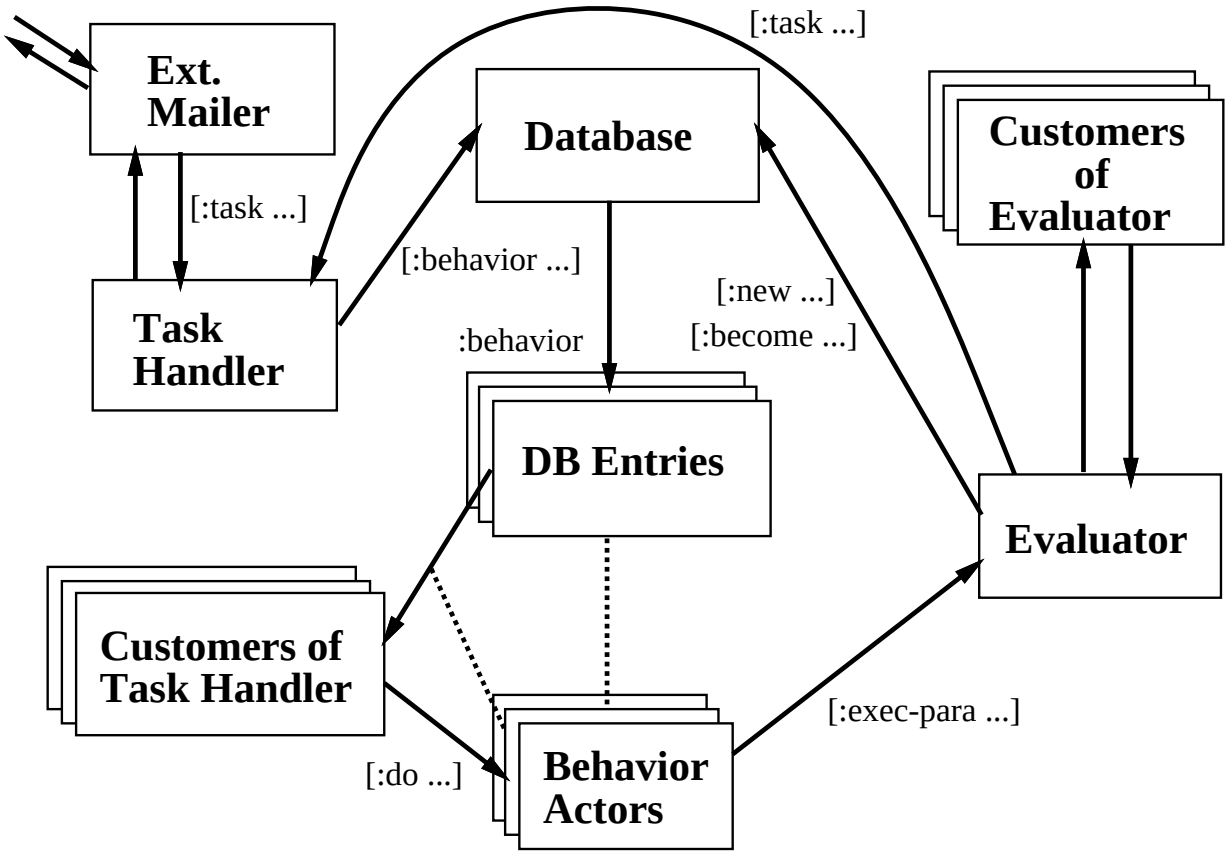

図 3.2: メタレベルのアクターとメッセージの流れ [45]

図中に現れるそれぞれのアクターは以下のような役割を持つ.

External Mailer 自分自身が属するグループおよび, 他のグループからのメッセージを受け付けるアクター. 受理したメッセージの引数にあるメールアドレスが, 自身が属しているグループ内のアドレスか否かを判断する. 同じグループ内のアドレスであればそのアドレスをもつアクターにメッセージを送信する. そうでなければ, そのメールアドレスを持っている別のグループの External Mailer(図中では Ext. Mailer と略記している) アクターにメッセージを送信する.

Task Handler 受理したメッセージの引数にあるメールアドレスが, 自身が属しているグループ内のアドレスか否かを判断する. 同じグループ内のアドレスであれば, Database アクターにメッセージを送信する. そのとき, Database アクターからの検索結果を含んだメッセージを受理するための Customer アクターを生成する. これによって,

Task Handler アクター自身は Database アクターからの検索結果を待つことなく，新たなメッセージを受理することが可能となる．一方，同じグループ内のアドレスでないと判断されたら，受理したメッセージをそのまま External Mailer アクターに送信する．

Customers of Task Handler ベーレベルの情報を持つ Database アクターからの返答メッセージを受理するためのアクター．受理したメッセージの内容に従って，Evaluator アクターもしくは External Mailer アクターにメッセージを送信し，自分自身は消滅する．

Database メタレベルでのアドレスとベースレベルの各情報との関連テーブルを持つアクター．メッセージを受理したら，検索対象の実体を検索するために，DB Entries アクターにメッセージを送信する．

DB Entries ベースレベルおよびメタレベルの情報を持っている．検索結果を Task Handler アクターが生成した Customer アクターに送信する．DB Entries アクターの各アクタはメッセージを受理した時の振舞いの記述をメタレベル表現した Behavior アクターとして持っている．

Behavior Actors ベースレベルおよびメタレベルにおける各アクターの振舞いを記述したもの．Evaluator アクターではこの記述を解釈し実行している．

Evaluator 受理したメッセージが持っている振舞いの記述 (スクリプト) を実行するインタプリタ．実行結果を受け取るための Customer アクターを生成する．これにより，実行結果を待つことなく，新たなメッセージを受理し，実行することが可能となる．

Customers of Evaluator 実行結果を含んだメッセージを受理し，処理を継続するためメッセージを適切な相手に送信し，自分自身は消滅する．

3.2.3 自己反映計算の実現

まず，ベースレベルシステムからメタレベルシステムへの通信について述べる．ベースレベルシステムからみれば，メタレベルシステムはベースレベルシステムの外部のアクターシステムである．これはベースレベルシステム内のアクターからメタレベルシステム内の

アクターに送信されたメッセージでメタレベルシステムの External Mailer アクターによって処理されることになる．

図 3.3 は External Mailer アクターがある特定のメッセージを受理したときに，いかに振舞うかを定義したものである．External Mailer アクターは

```
[:task _TagHandle _AddrHandle _Message]
```

というメッセージを受理したとき，まずメッセージの引数にある_AddrHandle を解析する．これは，メタレベル表現が _Message であるようなメッセージの，送信先のアドレスである．_AddrHandle を解析した結果，そのアドレスがこの External Mailer アクターが属するグループ内のアドレスをメタレベル表現したものである場合には，メタレベル表現される以前のメッセージ ↓_Message を該当するアドレスに向けて送信する．そうでない場合は他の Mailer アクターに向けて受理したメッセージを送信する．

```
(defBehavior anExtMailer (DB OtherMailers)
  (=> [:task _TagHandle _AddrHandle _Message]
      (become-ready)
      (case (parse-handle _AddrHandle)
        (is [:meta _Addr]
            [_Addr <= ↓_Message])
        (is [_Mailer _LocalHandle]
            [_Mailer <= [:task _TagHandle _LocalHandle _Message]]))))
```

図 3.3: ベースレベルシステムからメタレベルシステムへのメッセージ送信

逆に，メタレベルシステムからベースレベルシステムへの通信を考える．メールアドレスが m であるアクターにメッセージ k を送信することはメタタスクを生成することにより実現される．即ち，処理すべきタグ，メールアドレス，メッセージを各々メタレベル表現 (↑) し，Task Handler アクター θ に向けて送信すればよい．

```
[ θ  <= [:task ↑t ↑m ↑k]]
```

図 3.4: メタレベルシステムからベースレベルシステムへのメッセージ送信

3.3 書き換え論理によるモデル化

次に，前述のモデルを書き換え論理に基づき，関数型言語 Gofer によって実装したので，その実装に関する解説を行なう．書き換え論理は項書換えシステムに基づいているので，計算システムの状態を項で，状態の遷移を書き換え規則で表現する必要がある．したがって，計算システムを構成する要素を項で，計算状態はそれらから構築される項で，計算規則は計算状態の部分項に適用可能な書き換え規則で表現する．その方針に従って以下に示すようなデータ設計を行なった．ここでは，3.2節のアクターをオブジェクトして扱う．オブジェクト，メッセージを項[3]で表現し，状態遷移を行なう関数 `rewrite` を，想定されるコンフィギュレーションおよびメタコンフィギュレーション全てに対して定義している．

3.3.1 項の定義

本モデルで使用する項は，

- オブジェクト
  - ベースレベルシステムのオブジェクト
  - メタレベルシステムのオブジェクト
- メッセージ
  - ベースレベルシステムのメッセージ
  - メタレベルシステムのメッセージ
- コンフィグレーション
- メタコンフィグレーション

である．

まず，基本的なデータタイプの宣言をする．

[3] オブジェクトやメッセージなど，コンフィギュレーションを構成する要素はその部分項と考えられるが，それらも単に項と呼ぶことにする．

```
type Name       = String          -- 名前
type Value      = [String]        -- 値

type Attribute  = (Name,Value)    -- 属性
type Attributes = [Attribute]     -- 属性名と属性値の組のリスト

type Argument   = (Name,Value)    -- 引数値
type Arguments  = [Argument]      -- 引数名と引数値の組のリスト

type Ctr        = Int             -- カウンター(Customer オブジェクト生成用)
type Level      = Int             -- レベル
type LocalDB    = Name            -- データベース名
type NodeSet    = [Name]          -- グループ名のリスト

type MId        = String          -- メッセージ識別子
type OId        = String          -- オブジェクト識別子
type Des        = String          -- メッセージの行き先
type Ret_des    = String          -- メッセージの返り先
type MHname     = String          -- Message Handler 名
type THname     = String          -- Task Handler 名
type DBname     = String          -- Database 名
type EVname     = String          -- Evaluator 名
type Other      = String          -- 他のグループ名
type Others     = [Other]         -- 他のグループ名の一覧
```

図 3.5: 基本的なデータタイプの宣言

これらを使用して，ベース/メタレベルシステムのオブジェクト，メッセージを以下のように定義する．各々構成子 (Constructor) を利用している．それらは，メタオブジェクトの場合はオブジェクトの種類を，メッセージについてはメッセージ名を意味している．この規則をベースレベルシステムのオブジェクトについても踏襲すべきであったが，今回は OBJ という構成子で統一した．また，メッセージについては構成子をメッセージ名として

利用することにした.

```
-- オブジェクトの定義
data Obj = MH  OId THname Others                -- メッセージハンドラ
         | TH  OId MHname DBname EVname Ctr     -- タスクハンドラ
         | DB  OId [(Des,MId)] [(OId,Obj)]      -- データベース
         | EV  OId MHname DBname                -- エバリュエータ
         | CT  OId MHname DBname EVname         -- タスクハンドラのカスタマ
         | CV  OId MHname DBname                -- エバリュエータのカスタマ
         | OBJ OId Attributes                   -- ベースレベルのオブジェクト
         | MI  OId LocalDB NodeSet              -- オブジェクト移送用
-- メタレベルメッセージの定義
data Msg = TASK     Des Level Msg
         | FIND     Des Level Ret_des Msg
         | FOUND    Des Level Msg Obj
         | NOTFOUND Des Level Msg
         | EVAL     Des Level Msg Obj
         | MIGRATE  Des Arguments
         | DBDEL    Des Obj
         | DBADD    Des Obj
         | META BMsg
-- ベースレベルメッセージの定義
data BMsg = ADD Des Arguments
          | DEL Des Arguments
          | MEM Des Argument
          | ANS Des Argument
```

図 3.6: オブジェクトとメッセージの定義

コンフィギュレーションを図3.7のように定義している．基本的には，メタ/ベースレベルシステムを表現する構成子とメッセージのリストおよびオブジェクトのリストの並びによって構成される.

```
-- コンフィギュレーションの定義
data Con = MCON  [Msg] [Obj]              -- メタレベル
         | BCON [BMsg] [Obj]              -- ベースレベル
```

図 3.7: コンフィギュレーションの定義

3.3.2 書き換え規則の例

コンフィギュレーションを表現している項の部分項を書き換える関数

```
rewrite ::  Con -> Con
```

を定義した．想定可能なすべてのコンフィギュレーションの部分項に対してこの関数を定義している．これをコンフィギュレーションの書き換え規則と呼ぶことにする．書き換え規則の例として，Task Handler オブジェクトに関する規則について説明する．

```
rewrite (MCON ((TASK oid lvl bmsg):msgs)
              ((TH oid' mhname dbname evname ctr):objs))


  | oid == oid'
  = rewrite (MCON (msgs ++ [FIND dbname lvl ret_des bmsg])
                  ((TH oid mhname dbname evname (ctr+1)):
                   (CT ret_des mhname dbname evname):objs))
  where ret_des = "cu-t" ++ [ chr (ctr+48) ]
```

図 3.8: Task Handler オブジェクトの書き換え規則

上記の例のように，本実装で定義した書き換え規則は，

> メッセージが持つメールアドレス (この場合はオブジェクト識別子)`oid` とあるオブジェクトの識別子 `oid'` が一致したら，コンフィギュレーションを左辺から右辺に遷移させる．

という具合に定義している．図3.8 の場合，メッセージ（TASK oid ...）が持つメールアドレス oid が Task Handler オブジェクトの識別子 oid’ と等しいとき，即ち，Task Handler オブジェクトがメッセージ（TASK oid ...）を受理したら，Task Handler オブジェクトは Database オブジェクトに対してメッセージ（FIND dbname ...）を送信する．さらに，このメッセージの返答メッセージを受理するための Customer オブジェクト（CT ret_des ...）を生成する．

次に，Task Handler オブジェクトの Customer オブジェクトに関する書き換え規則について説明する (図 3.9)．

```
rewrite (MCON ((FOUND oid lvl bmsg obj):msgs)
              ((CT oid' mhname dbname evname):objs))
  | oid == oid'
  = rewrite (MCON (msgs ++ [EVAL evname lvl bmsg obj]) objs)
```

図 3.9: Task Handler オブジェクトの書き換え規則 (Customer オブジェクトの場合)

Task Handler オブジェクトの Customer オブジェクトが Database オブジェクトから送信されたメッセージ（FOUND ...）を受理したら，受理したメッセージが保持している Database オブジェクトからの情報をメッセージ（EVAL evname ...）という形で Evaluator オブジェクトに送信する．同時に，返答メッセージを受理するための目的で生成された自分自身を消去する．これにより，Customer オブジェクトがメッセージを受理し，Customer オブジェクトを生成するまでに実行してきた処理の続きを実行することが可能となる．よって Customer オブジェクトは継続 (Continuation) を表現していることになる．

3.3.3 自己反映計算の実現について

本実装ではベースレベルシステムの書き換え規則とメタレベルシステムの書き換え規則をおのおの定義した．想定される全てのコンフィギュレーションに対して書き換え規則を定義することにより，レベルの変化に伴うデータ変換については省略した．すなわち，ベースレベルシステムで行なわれる書き換えはメタレベルシステムでシミュレート可能であることを意味している．

よってベースレベルシステムで実行される規則とメタレベルシステムで実行される規則では，明確なベース，メタレベルシステムの境界は存在しない．あえてするならば，メッセージ名とそのメッセージを受理するオブジェクトの種類によって，現在行なっている計算のレベルを判断することになる．本実装では，メタレベルシステムの操作的意味を与えているが，これはメタレベルシステムのインタプリタを定義したことと等価である．したがって，本実装で行なわれている計算はメタレベルシステムの書き換え規則によるものである．

3.4 例題 — オブジェクト移送 —

GWA の例題としてオブジェクト移送を取り扱う．まず，問題の設定を解説する．次に，アクターモデルの場合のグループ間のアクターの移送に関する機構を記述する．最後に本研究で実装したオブジェクト移送に関連する書き換え規則を示す.

3.4.1 問題の設定

グループをネットワークによって他のノードと結合された単独プロセッサのノードを抽象化したものとみなす．N_1 を N_2 はそれぞれノードであるとする．ノード N_1 のアクター α は自分自身をノード N_2 に移送したいという要求があるものとする．

3.4.2 アクターモデルの場合

ノードを定義するアクターを記述する．そのノードが意味するものは，複数のアクターによって構成されるグループで，それ自身アクターとして表現される．ノードアクターは初期設定として，Evaluator アクター，Task Handler アクター，Database アクター，および External Mailer アクターをメタレベルのとして持ち，アクターの移送を実行する Migrator アクターをオプションとして持つ．External Mailer アクターと Migrator アクターは移送対象のアクターのグループへの受け入れ，放出の管理を行なっている．

```
(defGroup aNode (aMailer aMigrator)
  :meta ((evaluator (new anEvaluator))
         (database (new aNodeDB (empty-table)))
```

```
            (taskhandler (new aTaskHandler database aMailer evaluator)))
      :taskhandler taskhandler
      :export ((migrator aMigrator))
    )
```

図 3.10: グループ (ノードアクター) の定義

次に，オブジェクト移送を実行する Migrator アクターの振舞いを記述する．

```
(defBehavior aMigrator (LocalDB NodeSet)
    ;; アクター移送の要求
  (=> [:migrate _AddrHandler _DestinationNodeAddr]
      ;; 移送先のノードの migrator のアドレスを獲得する
    (case (find-destination-migrator _DestinationNodeAddr NodeSet)
      (is _Migrator where (non-nil _Migrator)
          ;; 移送先のデータベースのアドレスを確認する
        [_Migrator <= :database @
          [customer DestinationDB
              ;; 移送先のノードでの新しいアドレスを獲得する
            [DestinationDB <= :new-immigrant-addr @
              [customer NewLocalAddr
                  ;; 移送を実行する
                [LocalDB <= [:migrate-to _AddrHandle DestinationDB
                                         NewLocalAddr]]]]]]))))
)
```

図 3.11: Migrator アクターの振舞いの定義

Migrator アクターの振舞いに加え， Migrator アクターを持つノード内の Database アクターのアドレスを管理している LocalDB と，その他のノードの集合である NodeSet を Evaluator アクターに渡せば，その内容を解釈し，図 3.11のコメントにあるような処理を実行し，アクターの移送は完了する．

3.4.3 本実装の場合

問題を以下に示すように簡略化し，その条件のもとで例題の実行を試みる．図 3.12では，ノードは 2 個，移送先は必ず移送させたいオブジェクトが属するノードと異なるものが指定されるようになっている．つまり必ずオブジェクトの移送が確実に実行されることになる．また，実行手順は図 3.11中のコメントに従っている．図 3.11のプログラムでは各手順とも Migrator オブジェクトの属するグループ (ノード) の Evaluator オブジェクトによってメタレベル実行されている．本実装では，メタレベル実行されたという仮定のもとに必要と思われる関数を定義し，例題を実行した．

```
--移送先が別のグループ
rewrite (MCON ((MIGRATE oid target des):msgs)
              ((MI oid' localdb nodeset):objs))
  | oid == oid' && mig /= []
      -- 移送元の Database オブジェクトからオブジェクトを削除する
  = rewrite (MCON ((DBDEL localdb target):msgs)
                  ((MI oid' localdb nodeset):objs))
    where (mig,(MCON ms os))
             -- 移送先の Migrator オブジェクトのアドレスを獲得する
          = find_migrator des nodeset
            where
              find_migrator des nodeset
                | [des] == nodeset
                = ([],MCON [] [])
                | des == "mh1"
                = (find_migrator_sub conf2, conf2)
                | des == "mh2"
                = (find_migrator_sub conf1, conf1)
                   where find_migrator_sub (MCON m ((MI oid _ _):_))
                           = oid
                         find_migrator_sub (MCON m (_:r))
                           = find_migrator_sub (MCON m r)
```

```
    -- 移送先の Datbase オブジェクトのアドレスを獲得する
  db = find_destination_db os
       where find_destination_db ((DB db _ _):_)
               = db
             find_destination_db (_:rest)
               = find_destination_db rest
    -- 移送を実行する
  result = rewrite (MCON ((DBADD db target):ms) os)
```

図 3.12: オブジェクト移送に関する書き換え規則

```
--
-- 例題用のデータ定義
--

-- ノード N1 内のオブジェクト
mh1  = MH "mh1" "th1" []
th1  = TH "th1" "mh1" "db1" "ev1" 1
db1   = DB "db1" acceptable1 objlist1
ev1  = EV "ev1" "mh1" "db1" 1
mi1   = MI "mi1" "db1" ["N2"]
conf11 = MCON [msg11] [mi1,mh1, th1, db1, ev1]
conf12 = MCON [msg12] [mi1,mh1, th1, db1, ev1]
msg11 = MIGRATE "mi1" oa1 "mh2"
msg12 = MIGRATE "mi1" oa1 "mh1"
acceptable1 = [("A1","mem"),("A1","add"),("A1","del"),
               ("B1","mem"),("B1","add"),("B1","del"),
               ("C1","mem"),("C1","add"),("C1","del")]
objlist1 = [("A1",oa1),("B1",ob1),("C1",oc1)]
```

```
-- ノード N2 内のオブジェクト
mh2  = MH "mh2" "th2" []
th2  = TH "th2" "mh2" "db2" "ev2" 1
db2   = DB "db2" acceptable2 objlist2
ev2  = EV "ev2" "mh2" "db2" 1
mi2   = MI "mi2" "db" ["mh1"]
conf2 = MCON [] [mh2, th2, db2, ev2, mi2]

acceptable2 = [("A2","mem"),("A2","add"),("A2","del"),
               ("B2","mem"),("B2","add"),("B2","del"),
               ("C2","mem"),("C2","add"),("C2","del")]
objlist2 = [("A2",oa2),("B2",ob2),("C2",oc2)]

-- ノード N1,N2 のベースレベルのオブジェクトの定義
oa1    = OBJ "A1" [("member",["a11","a12","a13"])]
ob1    = OBJ "B1" [("member",["b11","b12","b13"])]
oc1    = OBJ "C1" [("member",["c11","c12","c13"])]
oa2    = OBJ "A2" [("member",["a21","a22","a23"])]
ob2    = OBJ "B2" [("member",["b21","b22","b23"])]
oc2    = OBJ "C2" [("member",["c21","c22","c23"])]
```

図 3.13: 例題のデータ

```
-- 例題の実行
-- ノード N1 のオブジェクト A1 を ノード N2 へ移送する

-- 実行前のノード N1
N1 = MCON [] [MH "mh1" "th1" [],
              TH "th1" "mh1" "db1""ev1" 1,
              DB "db1" [("A1","mem"), ("A1","add"), ("A1","del"),
                        ("B1","mem"), ("B1","add"), ("B1","del"),
```

```
                        ("C1","mem"), ("C1","add"), ("C1","del")]
                       [("A1",OBJ "A1" [("member",["a11","a12","a13"])]),
                        ("B1",OBJ "B1" [("member",["b11","b12","b13"])]),
                        ("C1",OBJ "C1" [("member",["c11","c12","c13"])])],
              EV "ev1" "mh1" "db1" 1,
              MI "mi1" "db1" ["N2"]]

-- 実行前のノード N2
N2 = MCON [] [MH "mh2" "th2" [],
              TH "th2" "mh2" "db2" "ev2" 1,
              DB "db2" [("A2","mem"), ("A2","add"), ("A2","del"),
                        ("B2","mem"), ("B2","add"), ("B2","del"),
                        ("C2","mem"), ("C2","add"), ("C2","del")]
                       [("A2",OBJ "A2" [("member",["a21","a22","a23"])]),
                        ("B2",OBJ "B2" [("member",["b21","b22","b23"])]),
                        ("C2",OBJ "C2" [("member",["c21","c22","c23"])])]
              EV "ev2" "mh2" "db2" 1,
              MI "mi2" "db2" ["N1"]]


---実行後のノード N1
N1 = MCON [] [DB "db1" [("B1","mem"), ("B1","add"), ("B1","del"),
                        ("C1","mem"), ("C1","add"), ("C1","del")]
                       [("B1",OBJ "B1" [("member",["b11","b12","b13"])]),
                        ("C1",OBJ "C1" [("member",["c11","c12","c13"])])],
              EV "ev1" "mh1" "db1" 1,
              MI "mi1" "db1" ["N2"],
              MH "mh1" "th1" [],
              TH "th1" "mh1" "db1""ev1" 1]
```

```
---実行後のノード N2
N2 = MCON [] [DB "db2" [("A2","mem"), ("A2","add"), ("A2","del"),
                        ("B2","mem"), ("B2","add"), ("B2","del"),
                        ("C2","mem"), ("C2","add"), ("C2","del"),
                        ("A1","mem"), ("A1","add"), ("A1","del")]
                       [("A2",OBJ "A2" [("member",["a21","a22","a23"])]),
                        ("B2",OBJ "B2" [("member",["b21","b22","b23"])]),
                        ("C2",OBJ "C2" [("member",["c21","c22","c23"])]),
                        ("A1",OBJ "A1" [("member",["a11","a12","a13"])])],
              EV "ev2" "mh2" "db2" 1,
              MI "mi2" "db2" ["N1"],
              MH "mh2" "th2" [],
              TH "th2" "mh2" "db2" "ev2" 1]
```

図 3.14: 例題の実行例

詳細な実行手順は省略したが，図 3.14中の実行前後の各ノードのデータベースの中身を比較すると，ノード N1 のオブジェクト `OBJ "A1" ["member",["a11","a12","a13"]]` がノード N2 のデータベースに移動していることがわかる．

3.4.4 メタレベルシステムの正当性

書き換え論理に基づいた宣言的記述を利用してそのシステムの様々な性質を解析することが可能と予測される．本研究ではそのうちの一つと考えられる，メタレベルシステムの正当性の証明を行った．

ここで，メタレベルシステムが正当であるとは，

- ベースレベルシステムの一回の遷移 (書き換え) に対し，それをシミュレートするようなメタレベルシステムの (複数回の) 遷移が存在し，
- ベースレベルシステムのメタレベル表現が変化するようなメタレベルシステムでの遷移に対し，それに対応するベースレベルシステムでの一回の遷移が存在する

ことを意味する [45]．本研究では，オブジェクト移送の例題の記述を基礎とし，メタコンフィギュレーションに関する構造帰納法 [49] を用いて，メタレベルシステムが上記の意味で正当であることを証明した．具体的には以下のように証明している．

記法の定義

以下のように記法の定義を行う．

記法	その意味
S	項からなる (ベースレベル) システム
C	S のコンフィギュレーションで，$\langle M, O \rangle$ と書く
M	C のメッセージの集合
O	C のオブジェクトの集合
R	S の書き換え規則の集合
Γ_S	S のすべてのコンフィギュレーションの集合
$\Gamma_{\uparrow S}$	S のメタレベル表現 $\uparrow S$ に関するすべてのコンフィギュレーション (メタコンフィギュレーションとよぶ) の集合
$\mathcal{R}$	$\uparrow S$ の書き換え規則の集合
$\langle\, \mathtt{n} \mid atts \,\rangle$	名前 n，属性 $atts$ をもつオブジェクト
$[\mathtt{t}, \mathtt{v}]$	ターゲット t に値 v を送信するメッセージ
$(\mathtt{t1} \to \mathtt{t2})$	以下に示す書き換え規則の略記 $[\mathtt{t1}, \mathtt{v}] \langle\, \mathtt{t1} \mid atts \,\rangle \to [\mathtt{t2}, \mathtt{v}] \langle\, \mathtt{t1} \mid atts \,\rangle$ または $[\mathtt{t1}, \mathtt{v}] \langle\, \mathtt{t1} \mid atts \,\rangle \to [\mathtt{t2}, \mathtt{v}] \langle\, \mathtt{t1} \mid atts \,\rangle \langle\, \mathtt{t3} \mid atts' \,\rangle$
$\uparrow C$	C のメタレベル表現 (メタコンフィギュレーション)
$\uparrow M$	$\uparrow C$ のメッセージの集合
$\mathcal{O}(O, R)$	$\uparrow C$ のオブジェクトの集合 この集合に属しているデータベースオブジェクトは O と R のメタレベル表現をデータとして持っている

表 3.2: 記法の定義

表中において，S をシステム，$\uparrow S$ をシステム S のメタレベル表現とする．ここで，システム S は各々項で定義されたオブジェクトのグループ，メッセージ，および書き換え規則から構成されているものとする．

本研究においては，$\uparrow S$ は 5 種類の特徴づけられたオブジェクトから構成されている．それらは各々メッセージハンドラ (Message Handler, `mh` と略記，以下同様), タスクハンドラ (Task Handler, `th`)，データベース (Database `db`)，エバリュエータ (Evaluator, `ev`)，および `th` に対応するカスタマ (Customer, `cus`) である．文献 [45] で定義されているこれらのオブジェクトは本研究でもほぼ同様な役割を果たしている．詳細は以下の通りである．

- メッセージハンドラ (Message Handler, `mh`):
  メッセージハンドラは $\uparrow S$ もしくは $\uparrow S$ 以外のシステムからのメッセージを受理し，その送信先を確認して $\uparrow S$ 内のタスクハンドラもしくは $\uparrow S$ 以外のメッセージハンドラに再送信する．

- タスクハンドラ (Task Handler, `th`):
  タスクハンドラは自分自身が属しているシステムのすべての (メタレベル) オブジェクトを把握している．このオブジェクトはベースレベルで処理されるメッセージのメタレベル表現に関して，メタレベル計算の準備を行う．タスクハンドラがメッセージハンドラによって送信されたメタメッセージを受理すると，続いて実行すべき計算を続けるためのオブジェクト (後述) を生成する．

- データベース (Database, `db`):
  データベースは S のオブジェクトおよび書き換え規則のメタレベル表現をデータとして保持している．

- エバリュエータ (Evaluator, `ev`):
  エバリュエータはメッセージ，オブジェクト，書き換え規則のメタレベル表現を受け取り，規則で規定された遷移を実行する．

m を S 内のあるメッセージを表現するものと仮定する．そのとき，m のメタレベル表現は $\uparrow m$ と表現する．さらに，$\uparrow m$ に対応するメタメッセージは $[\texttt{mh}, \uparrow m]$ と表現する．S のすべての書き換え規則は$\uparrow S$ を構成するオブジェクト `db` のデータとしてメタレベル表現される．

表 3.2 の記法を用いて，コンフィギュレーション C のメッセージの集合 M のメタレベル表現，メタコンフィギュレーション $\uparrow C$，および$\uparrow S$ の書き換え規則の集合 $\mathcal{R}$ を以下のように定義する．

$$
\begin{aligned}
\uparrow M &= \{[\mathtt{mh}, \uparrow m] \mid m \in M\} \\
\uparrow C &= \langle \uparrow M, \mathcal{O}(O, R) \rangle \\
\mathcal{R} &= \{(\mathtt{mh} \to \mathtt{th}), (\mathtt{mh} \to \mathtt{omh}), (\mathtt{th} \to \mathtt{db}), (\mathtt{db} \to \mathtt{cus}), \\
&\qquad (\mathtt{cus} \to \mathtt{ev}), (\mathtt{cus} \to \mathtt{mh}), (\mathtt{ev} \to \mathtt{db})\}
\end{aligned}
$$

ここで，omh は $\uparrow S$ 以外のメッセージハンドラの名前である．

さらに，遷移関係とコンフィギュレーション間の遷移列は表 3.3 にあるように定義される．

記法	その意味
$C_1 \longrightarrow C_2$	Γ_S に属する C_1 に $\mathcal{R}$ のある書き換え規則を 1 回適用した結果 Γ_S に属する C_2 に遷移する． もし C_1 に適用した書き換え規則 r を明示したい場合は $C_1 \xrightarrow{r} C_2$ と表現する．
$K_1 \xrightarrow{(n)} K_2$	$\Gamma_{\uparrow S}$ に属する K_1 に $\mathcal{R}$ の適当な書き換え規則を n 回適用すると $\Gamma_{\uparrow S}$ に属する K_2 に遷移する． ただし，この書き換えによって S のオブジェクトや書き換え 規則のメタレベル表現は変化していないものとする．

表 3.3: 遷移関係の定義

証明の概要

証明を始める前に次のことに注意しておく．それは，

> $\Gamma_{\uparrow S}$ に属する任意のメタコンフィギュレーション K に対し，それに対応する Γ_S に属するコンフィギュレーション C が必ず存在する

ということ，すなわち，K は C のメタレベル表現のひとつであるということである．そこで表 3.3 で与えた遷移列の定義を利用し集合

$$\overline{\uparrow C} = \{K \in \Gamma_{\uparrow S} \mid K \xrightarrow{(*)} \uparrow C, \text{ or } \uparrow C \xrightarrow{(*)} K\}$$

を定義する．ここで，K が $\Gamma_{\uparrow S}$ の要素であるとき，遷移列の定義から，$\overline{\uparrow C}$ の要素ならば S におけるオブジェクトと書き換え規則のメタレベル表現は $\uparrow C$ と一致していることに注意する．また，$\Gamma_{\uparrow S}/\overset{()}{\rightarrow}$ における擬似遷移関係$\overset{\sim}{\Longrightarrow}$ を以下のように定義する．

定義: もし，$\overline{\uparrow C_1}$の任意の元K_1に対して

$$K_1 \xrightarrow{(*)} \xrightarrow{\sigma} \xrightarrow{(*)} K_2$$

を満たす$\overline{\uparrow C_2}$の要素K_2が存在するとき，$\overline{\uparrow C_1}$と$\overline{\uparrow C_2}$は擬似遷移関係にあるといい，$\overline{\uparrow C_1} \overset{\sim}{\Longrightarrow} \overline{\uparrow C_2}$と書く．ただし，$\xrightarrow{(*)}$ はデータベースの内容を書き換えない任意回数の書き換え列，$\xrightarrow{\sigma}$ はデータベースの内容を書き換える $\mathcal{R}$ の書き換え規則 σ の適用を意味する．■

以上の準備によって，$\uparrow S$ が正当であるということは次の定理にまとめることができる．

定理: C_1, C_2 は Γ_S の要素であると仮定する．このとき，$\overline{\uparrow C_1} \overset{\sim}{\Longrightarrow} \overline{\uparrow C_2}$ であることの必要十分条件は $C_1 \xrightarrow{r} C_2$ を満たす R の書き換え規則 r が存在することである．ここで，$\overset{\sim}{\Longrightarrow}$ は$\Gamma_{\uparrow S}/\overset{()}{\rightarrow}$ における擬似遷移関係である． ■

この定理の証明の概要は次のようになる．

必要性: 必要性を示すためには $\uparrow C_1 \xrightarrow{*} \uparrow C_2$ が成り立つことを確認すれば良い．m をベースレベルの書き換え規則 r を 1 回適用して C_1 を C_2 に遷移させるベースレベルのメッセージであるとする．このとき，m のメタレベル表現は $[\mathtt{mh}, \uparrow m]$ という形のメッセージが持っている値として出現する．この値 $\uparrow m$ に留意し以下のような実際に生じる遷移列を生成することが可能である．$\xrightarrow[n]{}$ は以下の遷移列における n 回目の書き換えであることを示している．

$$\begin{aligned}
&\uparrow C_1 = \langle \uparrow M_1, \mathcal{O}(O_1, R)\rangle \\
&= \langle \{[\mathtt{mh}, \uparrow m]\} \cup \uparrow M_1', \mathcal{O}(O_1, R)\rangle &\xrightarrow[1]{}\ & \langle \{[\mathtt{th}, \uparrow m]\} \cup \uparrow M_1', \mathcal{O}(O_1, R)\rangle \\
&&\xrightarrow[2]{}\ & \langle \{[\mathtt{db}, \uparrow m]\} \cup \uparrow M_1', \mathcal{O}(O_1, R) \cup \{cus(\uparrow m)\}\rangle \\
&&\xrightarrow[3]{}\ & \langle \{[\mathtt{cus}, \uparrow m]\} \cup \uparrow M_1', \mathcal{O}(O_1, R) \cup \{cus(\uparrow m)\}\rangle \\
&&\xrightarrow[4]{}\ & \langle \{[\mathtt{ev}, \uparrow m]\} \cup \uparrow M_1', \mathcal{O}(O_1, R)\rangle \\
&&\xrightarrow[5]{}\ & \langle \uparrow M_1' \cup \uparrow M_1'', \mathcal{O}(mod(O_1), R)\rangle \\
&&=\ & \langle \uparrow M_2, \mathcal{O}(O_2, R)\rangle = \uparrow C_2
\end{aligned}$$

ここで， $M_1' = M_1 - \{m\}$ である．さらに，$M_1{}''$ は書き換え規則 r の右辺に出現するメッセージの集合を，$mod(O_1)$ は書き換え規則 r によって O_1 を変更したオブジェクトもしくは，新たに生成されたオブジェクトの集合を表現している．

各々の書き換えステップの説明は以下の通りである．

1 : $\uparrow C$ の書き換え規則 ($\mathtt{mh} \rightarrow \mathtt{th}$) は mh が $[\mathtt{mh}, \uparrow m]$ という形をした $\uparrow C_1$ のメタメッセージを受理した場合に適用される．

2 : 書き換え規則 ($\mathtt{th} \rightarrow \mathtt{db}$) は th がメタメッセージ $[\mathtt{th}, \uparrow m]$ を受理した場合に適用される．このとき，新たなオブジェクト (タスクハンドラのカスタマ (customer)) $cus(\uparrow m)$ がその規則によって生成される．

3 : 書き換え規則 ($\mathtt{db} \rightarrow \mathtt{cus}$) は db がメタメッセージ $[\mathtt{db}, \uparrow m]$ を受理した場合に適用される．

4 : 書き換え規則 ($\mathtt{cus} \rightarrow \mathtt{ev}$) は $cus(\uparrow m)$ がメタメッセージ $[\mathtt{cus}, \uparrow m]$ を受理した場合に適用される．このとき，2 で生成されたオブジェクト $cus(\uparrow m)$ はこの書き換え規則を適用することによって消去される．

5 : 書き換え規則 ($\mathtt{ev} \rightarrow \mathtt{db}$) は db がメタメッセージ $[\mathtt{ev}, \uparrow m]$ を受理した場合に適用される．このとき，この書き換え規則によって，メッセージ m に関するメタレベル実行が引き起こされる．

5 番目の書き換えステップではメタレベル実行が引き起こされる．それは書き換え規則 ($\mathtt{ev} \rightarrow \mathtt{db}$) はベースレベルのオブジェクトおよび書き換え規則のメタレベル表現を変更することを意味している．さらに，その書き換え規則によって幾つかの新たなメッセージが送信されている可能性がある．それらのメッセージの各々は書き換え規則 r によって送信されたメッセージのメタレベル表現を値として保持している．

このようにして$\uparrow M_1' \cup \uparrow M''_1 = M_2$ かつ $mod(O_1) = O_2$ を得ることができた．よって必要性は証明された．

十分性: 十分性を示すために，まず次の補題を証明しておく．

補題: $\Gamma_{\uparrow S}$ の要素である任意の K に対して，$K \xrightarrow{(n)} K'$ かつ
$\mathcal{M}(K') = \{[\mathtt{ev}, \uparrow m] \mid m \in M\}$ が成立するような$\Gamma_{\uparrow S}$ の要素である K' と自然数 n が存在

する．ここで，$\mathcal{M}(K')$ は K' におけるメッセージの集合，M は K が $\overline{\uparrow C}$ の要素となるような C のメッセージの集合を意味する．

補題の証明:　α を $\mathcal{M}(K)$ の要素と仮定すると，$\alpha = [\mathtt{t}, \uparrow m]$ である．ここで，$\mathtt{t}$ は $\{\mathtt{mh}, \mathtt{th}, \mathtt{db}, \mathtt{cus}, \mathtt{ev}\}$ のいずれかであり，m はベースレベルのメッセージである．$(\mathtt{ev} \rightarrow \mathtt{db})$ 以外のすべての書き換え規則，すなわち，ベースレベルのオブジェクトや書き換え規則のメタレベル表現を変更しないような書き換え規則を K に適用することにより，送信先が $\mathtt{ev}$ でないメッセージを $\mathtt{ev}$ であるメッセージに変更することが可能である．■

先の補題により，K_1, K_1' がともに $\overline{\uparrow C_1}$ の要素である遷移列 $K_1 \xrightarrow{(*)} K_1'$ を得ることができる．そこで，σ をベースレベルのオブジェクトや書き換え規則のメタレベル表現を変更するような $\mathcal{R}$ の書き換え規則で，$\Gamma_{\uparrow C_2}$ の要素 K_2 で $K_1' \xrightarrow{\sigma} K_2$ を満足するものとする．このとき，遷移列 $K_1 \xrightarrow{(*)} K_1' \xrightarrow{\sigma} K_2$ を得る．K_1' に書き換え規則 σ を適用するためには $\mathtt{ev}$ に送信されたメッセージが必要である．それを $[\mathtt{ev}, \uparrow m]$ と書くことにする．このような m の存在は メッセージ m を受理すると $C_1 \xrightarrow{r} C_2$ なる遷移を引き起こす書き換え規則 r が存在することを意味している．したがって十分性も証明された．■

以上のことからこの定理が成立することが証明された．

第 4 章

有機的プログラミング言語 GAEA の操作的意味

本章では，自己反映計算が可能な言語の操作的意味について考察している．操作的意味，すなわちインタプリタの定義は書き換え論理に基づき，仕様記述言語 CafeOBJ[35] および Maude[32][5] を用いて記述した．ここでは Maude による記述のみを用いて説明をおこなっている．以降，有機的プログラミング言語 GAEA の概要，操作的意味の定義，その記述に基づいた例題，および考察の順に述べる．

4.1 有機的プログラミング言語 GAEA

ここでは，有機的プログラミング言語 GAEA[1] の概要を述べる．その際，本言語が並行自己反映計算の一例と考えられる理由を述べる．また，GAEA で記述されたプログラムを用い，自己反映計算が行なわれる様子を観察する．

本研究において操作的意味を与える対象とした言語は，有機的プログラミング言語 GAEA と呼ばれるものである．本言語は有機的プログラミング [36][37] という，新しいソフトウェア構成方法に基づいた協調システム記述言語である．この言語は電子技術総合研究所で現在も開発中である．

有機的プログラミングとは，

- 状況推論

- モジュール構成の操作
- 自己反映計算

といったアイディアに基づき，協調計算システムを容器に記述可能とするために開発されたソフトウェア構築方法論である．

GAEA の構文や動作は論理型プログラミング言語 Prolog[40] に類似しているが，Prolog にはない機能が豊富に組み込まれている．具体的には，

- セルと呼ばれるプログラム記述単位を基礎とし，それらを組み合わせることにより完全なプログラムを構成する
- プロセスと呼ばれる処理単位がある．プロセスは，環境と呼ばれるセルの名前のスタックを持つ
- マルチプロセス処理が可能である
- セルはプロセス間通信や同期処理を行なうために，セル変数を設定することが可能である
- “@”-infon と呼ばれる，部分的に特定可能な項が使用可能である
- 分散，マルチプロセス環境のためのネットワークインターフェイスを備えている

といった機能である．本研究では，これらの機能の内，第一から第四までの機能に着目して操作的意味を定義している．

次に，GAEA の重要概念であるセルとプロセスについて説明する．

セル

セルは有機的プログラミングにおいて最も重要な概念のひとつである．セルはプログラム記述単位であり，全体のプログラムの一部分や情報を含んでいる．プログラムの一部分は節定義の集合として定義され，情報はセル中に定義可能なセル変数に格納される．セルはプログラム記述単位であったので，複数のセルを組み合わせることによって，目的の処理を行なうためのプログラム (データベース) を構成することができる．

通常の Prolog においては，質問を処理するためのデータベース，すなわち，節定義や事実の集合は唯一つ存在する．ところが，GAEA では処理単位であるプロセス毎に状況に応じたプログラム (データベース) を動的に生成，変更することが可能である．

プロセスの状況に応じたプログラム (データベース) は，そのプロセスが持つ環境と呼ばれる情報によって構築される．GAEA においては，プロセスの環境とはセルの名前のスタックのことを指す.

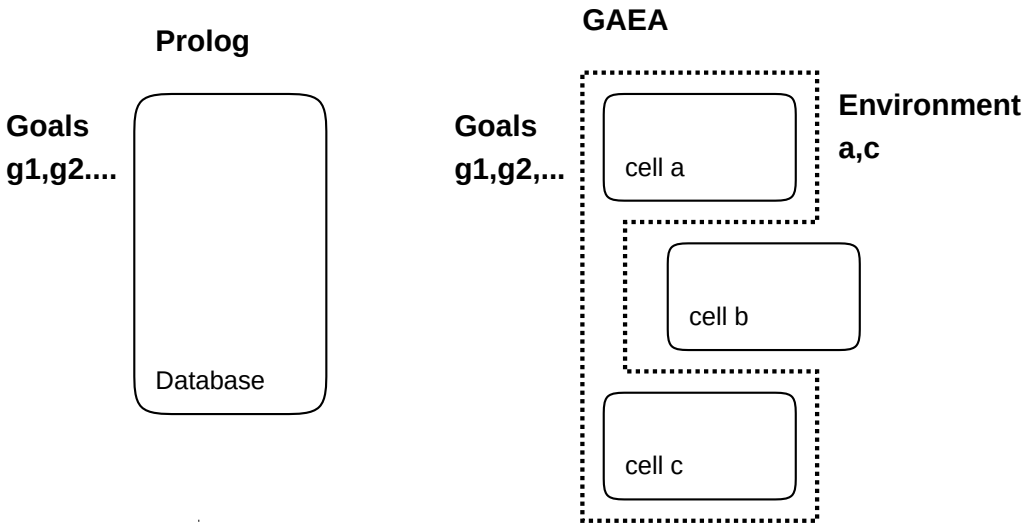

図 4.1: Prolog と GAEA のデータベース

プロセスは環境やセル変数の値を組込み述語によって参照することができる．その結果，自分自身の置かれた状況を判断することが可能である．ここで，状況とは，プロセスの環境や，セル変数の値のことである．その状況に応じて，プロセスは組み込み述語を使って自分自身の環境，すなわちセルのスタックを動的に変更することが可能である．このことから GAEA は自己反映計算が可能な言語であるということができる．ただし，ここでいう自己反映計算では，第 2章で述べたようなリフレクティブタワーは構成されない．具体的には次に述べるように実現されている．

図 4.2において，四角はセル，破線は環境を表現しているものとする．述語`subst_cell/2` は GAEA が提供している組み込み述語で，環境 (セルのモジュール構造) を操作するものである．セル `a` およびセル `a'` にはヘッド部のパターンが等しく (例えば`head(X)`)，ボディ部が異なる節が定義されていると仮定する．このとき図中の左側の環境で，あるゴール `head(X)` を実行した場合と，左側の環境をセルのモジュール構造を操作する組み込み述語

subst_cell/2 によって変更した環境で先程と同じ形のゴールhead(X) を実行した場合とでは，実行結果が異なっている可能性がある．

このように，状況に応じて環境 (セルのモジュール構造) 操作述語によって環境を変更すると，その変更が次の実行に反映されていることが確認できる．同一の形をしたゴールの処理を試みてはいるが，そのゴールと照合した節定義のヘッド部に対応するボディ部があたかも別のゴール列に書き換えられたかのように振舞っている．このような自己反映計算を文脈的自己反映計算と呼ぶ.

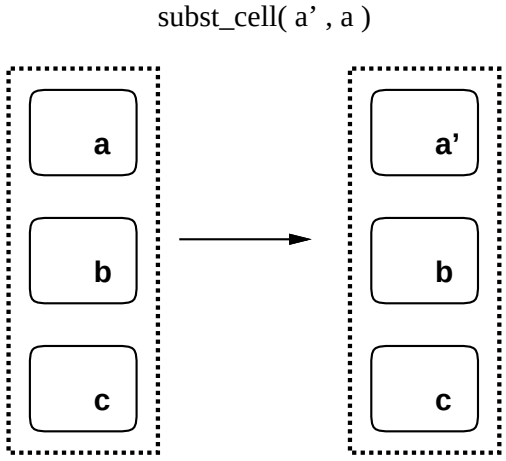

図 4.2: 環境の操作

ここでセルについて以下の注意を与えておく．

- セルはそれ自身が持つ名前によって識別可能である．
- GAEA システムはプログラムを読み込むと main，stdlib および system_cell と名付けられたセルをデフォルトのセルとして生成する．これらはセル変数を含んでいない．ただし，利用者がそれらにセル変数を設定することは可能である．
- セル main は明示的に特定のセルに定義されていない節や事実を含むことがある．逆に，明示的にセルを生成し，その中に節定義を記述すれば，それらの節定義はセル main に格納されることはない．
- セル stdlib は once/1，succeed/1，not/1，or/2，repeat/1 といった標準的な述語の定義を含んでいる．ただし，pred/n は述語名がpred，引数 n の述語を意味しているものとする．

- セル system_cell は GAEA 実行時にシステム内に存在するセルの名前およびプロセスの名前を管理するデータベースとして使用される．

プロセス

プロセスはある質問を処理するための単位であり，その質問を処理するための情報を含んでいる．その情報とは環境，発火点位置，ゴール列，バックトラック用の情報である．したがって，ひとつのプロセスは Prolog 処理系 (もしくはそれを拡張したもの) に対応する．GAEA はマルチプロセス処理が可能であるから，複数の Prolog 処理系 (もしくはそれを拡張したもの) が同時に動作可能であることに対応し，したがって並行計算を行なうことが可能となっている．複数のプロセスが独立に動作している間は問題無いのであるが，共通のデータが操作可能である並行計算を行なうためにはプロセス間の同期処理をおこなう機構が必要である．

GAEA ではセル変数とそれを操作する組み込み述語を利用して同期処理機構を実現している．セル変数とは，セル中に定義可能な，情報格納用スロットである．セル変数を生成するとシステムはデフォルト値undef を割り当てる．もちろん，所望の初期値を割り当てることも可能である．セル変数を操作する組み込み述語はいくつか用意されているが，同期処理機構に関連するものは以下のものである．

組込み述語	値ありの場合	値無し (undef) の場合
cv_write/3	wait	succeed
cv_give_wait/3	wait	succeed
cv_read/3	succeed	wait
cv_xread/3	unassign	wait
cv_ref_wait/3	succeed	wait
cv_take_wait/3	unassign	wait

表 4.1: 同期処理機構に関連するセル変数操作述語

述語cv_write(Cell,CV,V) の場合で説明する．この述語は指定されたセルCell 中に定義されているセル変数CV へ値V を書き込む．ただし，セル変数に値が割り当てられていない場合，すなわちデフォルト値undef の場合は成功するが，そうでない場合はセル変数の

値がデフォルト値になるまで待ち状態となる．このようにセル変数を利用して同期処理機構を実現している．

また，セル変数はプロセス間通信にも利用される．異なる 2 つのプロセスを各々P_1，P_2と仮定する．プロセス P_1はプロセス P_2で使用されている論理変数の値を操作することはできない．逆も同様である．しかし P_1における計算結果をプロセス P_2に渡したい場合も考えられる．そこで，適当なセルに設定されたセル変数を介し，各プロセスからその値を読み書きすることによって，プロセス間の情報をやりとりを実現している．

このように，セルやプロセスの特徴を活用することによって，GAEA が自己反映計算可能や並行計算が可能である論理型言語であることがわかった．

例題による GAEA の動作の説明

次に，具体例を用いて GAEA によるプログラムと自己反映計算がどのように行なわれているかを説明する．例題は自律ロボットの挙動をシミュレートするものである[1]．この例題は，複数の自律ロボットが道路上のある地点から交差点に向かって進み，同時に 2 台以上侵入することなく交差点を通過し，ある地点で自ら消滅する動作をシミュレートするものである．自律ロボットの動作は

- 道路を進む (`proceed` モード)
- 交差点内への侵入を要求する (`want` モード)
- 交差点内へ侵入し，進み，交差点から出る (`enter` モード)
- 交差点内への侵入を待つ (`stop` モード)

の 4 種類に分類されているものとする．以下の GAEA プログラムでは，上記の各モードをセルとして定義している．セル中には各モードで行なうべき動作を述語として定義している．

セル`com` は同期処理およびプロセス間通信を実現するために設けられている．このセルはセル変数`message` を持つ．初期値は`undef` である．

```
%-----------------------------------------------------------------------
% 汎用的に用いられる述語の定義
% セル main に定義される
%-----------------------------------------------------------------------
```

[1] GAEA ホームページに掲載されている．`http://cape.etl.go.jp/gaea/sample/intersection.g`.

```
% 自律ロボットの動作をシミュレートするプロセスを生成する
test() := fork(agent(a,7)),fork(agent(b,7)),fork(agent(c,7)).

% ロボットを生成する
agent(Name,Loc) := new_cell(Name, loc, iloc),
  push(Name),
  (name(Name);assert(name(Name))),
  cv_write(Name,loc,Loc),
  push(proceed),
  output(initializing(Name,Loc)),
  repeat(e()).       %ヘッド部が e() である節定義を繰り返し呼び出す

% セル変数の値の検索
find(M) := cv_ref(com,message,[Name,M]),
  name(Me),
  not(Name = Me).

% メッセージをセル変数と標準出力に書き出す
message(Mes) := name(Me),
  cv_write(com,message,[Me,Mes]),
  output([Me,Mes]).

% セル変数の値の検索
loc(L) := name(Me),cv_read(Me,loc,L).

% セル変数の値の検索
iloc(L) := name(Me),cv_read(Me,iloc,L).

% セル変数の値の操作
decr(V) : name(Me),cv_take(Me,V,L) := L1 is L-1, cv_set(Me,V,L1).

% セル変数の値の操作
inc(V) : name(Me),cv_take(Me,V,L) := L1 is L+1, cv_set(Me,V,L1).

% 標準出力への書き出し
output(M) := write(M),nl.

%-----------------------------------------------------------------------
% セル com の定義
%      セル変数 message のみを持つ．節定義はなし
%-----------------------------------------------------------------------
?- new_cell(com,message).

%-----------------------------------------------------------------------
% セル proceed の定義
%       セル変数はなく，節定義を持つ
%-----------------------------------------------------------------------
?- new_cell(proceed),push(proceed).

% 交差点の入口に到着したら，proceed モードから want モードに切り替わる
e() : loc(0) := name(M),output(want_enter(M)),subst_cell(want,proceed).
```

```
% 道路の終点に到着したら消滅する
e() : loc(-3) : =name(M),output(finished(M)),
  cv_take(M,loc,_),     % cleaning garbabe info
  cv_take(M,iloc,_),    % for reincarnation
  die().

% 上記以外の場合は道路を進む
e() := decr(loc),loc(N),name(M),output(at(M,N)).

?-pop(_).

%-------------------------------------------------------------------------
% セル want の定義
%      セル変数はなく，節定義を持つ
%-------------------------------------------------------------------------
?- new_cell(want),push(want).

% 交差点内に誰か存在するか否かを確認する.
% 誰か存在するならば want モードから stop モードに切り替わる
e() : find(entered) := output(someone_there),subst_cell(stop,want).

% 交差点内に誰も存在しなければ交差点内に侵入する
% すなわち，want モードから enter モードに切り替わる
e() := enter(),subst_cell(enter,want).

%補助述語
enter() := name(Me),output(entering(Me)),
  message(entered), % waits till writable
  cv_set(Me,iloc,5).

?- pop(_).

%-------------------------------------------------------------------------
% セル enter の定義
%      セル変数はなく，節定義を持つ
%-------------------------------------------------------------------------
?- new_cell(enter),push(enter).

% 交差点の出口に到着したら外に出る
% すなわち，enter モードから proceed モードに切り替わる
e() : iloc(0) := name(M),output(exiting(M)),
  succeed(cv_take(com,message,_)),
  decr(loc),subst_cell(proceed,enter).

% 交差点内を進む
e() := decr(iloc),iloc(N),name(M),output(inside(M,N)).

?- pop(_).

%-------------------------------------------------------------------------
% セル stop の定義
%      セル変数はなく，節定義を持つ
%-------------------------------------------------------------------------
```

```
?- new_cell(stop),push(stop).

% 交差点内に誰も存在しなければ，stop モードから proceed モードに切り替わる
e() : not(find(_)) := output(resume),subst_cell(proceed,stop).

% 交差点内に誰か存在するならば待つ
e() := name(M),output(waiting(M)).

?- pop(_).
```

図 4.3: GAEA によるプログラム例

GAEA における節定義は

```
Fact .
Head_Part : Body_Part .
Fact := .
Head_Part := Body_Part .
Head_Part : Condition := Body_Part .
```

図 4.4: GAEA での節定義

で与えられるが，通常の Prolog においてはそれぞれ

```
Fact .
Head_Part :- Body_Part .
Fact :- ! .
Head_Part :- ! , Body_Part .
Head_Part :- Conditon , ! , Body_Part .
```

図 4.5: 通常の Prolog で書き直した場合

と定義されるものと同等である．

図 4.3でみられるように，セルは述語 `new_cell/1` によって生成される．セル変数を持つセルを生成したい場合は，`new_cell/`n を用いる．この述語の第一引数にはセルの名前を，第二引数目から第 n 引数目はセル変数名を指定する．節定義を持つセルを定義したい場合には，

```
?- new_cell( セル名 ) , push( セル名 ) .
  節定義
  ......
?- pop( _ ) .
```

図 4.6: 節定義を含むセルの定義方法

とすれば良い．図4.3中のはじめの 9 個の節定義は図 4.6 のような記法に基づいていないので，このプログラムを GAEA システムが読み込んだ際に自動的に生成するセル main 内に定義される．

GAEA システムが図 4.3のプログラムを読み込んだ際，自動的に生成するセルは，セル main の他に，セル stdlib，セルsystem_cell があることは前述した．ここで，トップレベルで質問 test() を行なうと，プロセス main が生成される．このプロセスの環境，つまりセル名のスタックは上から順に main，stdlib，system_cell で与えられる．質問 test() と照合可能な節定義はセル main 中に定義されている．照合が成功すると，その節定義のボディ部

```
!,fork(agent(a,7)),fork(agent(b,7)),fork(agent(c,7))
```

図 4.7: 未処理のゴール列

がプロセスmain の未処理のゴール列となる．

述語 fork/1 は引数のゴール列を処理するための新しいプロセスを生成する GAEA システムの組み込み述語である．このとき，新しいプロセスの環境は特に指定しない限り，そのプロセスを生成したプロセスの環境が継承される．プロセスの名前は GAEA システムによって自動生成される．述語 fork/1 は新たなプロセスを生成して成功する．上記の場合，プロセス main は新たなプロセスを 3 つ生成して成功する．ロボットの動作は新たに生成された各プロセスにおいてシミュレートされることになる．

各プロセスは並行に実行されるが，簡単のため，ゴールagent(a,7) を処理するプロセスに注目する．agent(a,7) はセル main 中に定義されており，ゴール列

```
!,new_cell(a, loc, iloc),push(a),(name(a);assert(name(a))),
```

```
cv_write(a,loc,7),push(proceed),output(initializing(a,7)),
repeat(e())
```

図 4.8: agent/2 のボディ部

を処理することになる．カットオペレータ (!) に続く 2 つの述語new_cell(a，loc，iloc) およびpush(a) は，セル名 a，2 つのセル変数loc および\verbiloc+を持つセルを生成し，その名前を環境の先頭に追加する．そこで，(name(a);assert(name(a))) を処理すると，後者が実行され，セルa の節として事実name(a) が登録される．

セル変数を操作する組み込み述語cv_write(a,loc,7) によってセルa のセル変数loc に値7 が割り当てられ，push(proceed) によって，環境の先頭にproceed が追加される．この時点における環境は，

```
proceed,a,main,stdlib,system_cell
```

図 4.9: 環境 (セル名のスタック，左が先頭)

となっている．

このような環境でrepeat(e()) を実行する．ヘッド部がe() である節定義は，現在の環境内にはセルproceed 中にのみ 3 つ存在している．環境が変化しなければプログラムの集合 (データベース) は固定なので，通常の Prolog と同様の処理を行なう．

ロボットが交差点の入口に到着した時，セルproceed 中の節

```
e() : loc(0) := name(M),output(want_enter(M)),subst_cell(want,proceed).
```

図 4.10: セル proceed の一番目の節定義

の条件部分loc(0) が成功し，その節定義のボディ部がプロセスの未処理のゴール列として処理される．その際，組み込み述語subst_cell(proceed,want) が呼び出されるが，この述語は，環境内の第二引数に一致する箇所に第一引数を代入するというものであり，結果として環境はスタックの上から順に，

```
proceed,a,main,stdlib,system_cell
```

から

```
want,a,main,stdlib,system_cell
```

図 4.11: 環境の変化

へと変化する．上記のように環境は変化するが，repeat(e()) によってe() が繰り返し呼ばれているので，変化後の環境内で，ヘッド部がe() である節定義を検索すると，セルwant 内に定義されている節

```
e() : find(entered) := output(someone_there),subst_cell(stop,want).
e() := enter(),subst_cell(enter,want).
```

図 4.12: セル want の節定義

が見つかる．

このように，同じ名前のゴールe() を繰り返し処理しているが，環境に応じて処理内容が変化していることがわかる．

以上のように，

- 動的にプロセスの生成，消去を行ない，
- 状況を判断 (セル変数に割り当てられた値によって条件分岐する) し，
- その状況に応じてプログラム (環境，すなわち節定義集合) を変更する

ことによって状況に適応する処理，動作をおこなっていることがわかる．言語システムのこのような複雑な動作を推測することは極めて困難であると考えられる．

そこで言語システムの操作的意味を定義するために以下に述べるモデル化を提案する．このモデル化では GAEA の重要概念であるセルとプロセスに着目し，GAEA の計算状態を表現している．操作的意味を定義することにより，言語システムを実際に使用しなくても，言語の動作を推測することが可能になる．したがって，言語を理解する上での有効な手段のひとつになると考えられる．

4.2 有機的プログラミング言語 GAEA の操作的意味

GAEA の操作的意味を記述するために仕様記述言語 CafeOBJ および Maude を利用した．双方とも記述した仕様を固有の処理系によって実行可能である．ここでは Maude による記述例を用いて説明している．

4.2.1 Maude

Maude[32][5] は書き換え論理に基づいた仕様記述言語であり，等式プログラミングおよびオブジェクト指向プログラミングを統一的に扱うことが可能である．Maude は実行可能な処理系を持っており，高速な書き換えが可能である [5]．

Maude の構文を簡単に説明する．Maude において基本的な仕様記述単位は**モジュール**である．モジュールには輸入宣言，ソート，部分ソート宣言，オペレータ宣言，変数宣言，等式宣言，書き換え規則宣言などを記述する．

純粋に等式からなる仕様を記述する際には，キーワード `fmod` および `endfm` を用いてモジュールを定義する．このようなモジュールは**関数的モジュール**と呼ばれる．このとき，このモジュール内では書き換え規則は定義できない．書き換え論理においては，書き換え規則とは計算状態の遷移を意味しており等式では表現できない．これに対し，キーワード `mod` および `endm` を用いてモジュールを定義するとき，そのモジュールは**システムモジュール**と呼ばれ，書き換え規則を定義することができる．我々は計算状態の遷移で操作的意味を与えるので，後者のキーワードを使用する．我々の仕様の最初の部分は以下のように与えられる．

```
mod GAEA is
--- term and termlist ---
  protecting QID .

  sorts Term NeTermList TermList .
  subsort Qid < Term .
  subsort Term < NeTermList < TermList .

  op nil : -> TermList .
  op termlist : TermList TermList -> TermList [ assoc ] .
  vars TL TL' : TermList .
```

```
  vars T B : Term .

  eq termlist(nil,TL) = TL .
  eq termlist(TL,nil) = TL .
   ......
endm
```

図 4.13: GAEA の仕様 (始めの部分)

輸入宣言は他のモジュールを輸入するために用いる．図 4.13では，Maude の組み込みモジュール QID(クォート付きの識別子) が輸入されている．

ソートは分類された要素の集合を宣言するものである．例えば，

```
sorts Term NeTermList TermList .
```

図 4.14: ソート宣言の例

は 3 種類のソート Term，NeTermList および TermList を定義している．さらに，

```
subsort Qid < Term .
subsort Term < NeTermList < TermList .
```

図 4.15: サブソート宣言の例

という具合に，ソート間の半順序を定義することも可能である．

オペレータ宣言ではアリティおよびコアリティ付きの関数記号を定義する．構文規則は

op *operator-symbol* : *list-of-sort-names* -> *sort-name* [*attributes*]

図 4.16: オペレータ宣言

で与えられる．ここで，

- *list-of-sort-names* は関数記号のアリティを与える，ソート名を空白で区切ったリストで，そのリストが空の場合は，関数記号は定数とみなす．

- *sort-name* は関数記号のコアリティを与える．

さらに [*attributes*] は関数記号の属性を規定している．書き換え論理においては，等式の集合 E を法として項を類別し，類別された項に対して書き換え規則が適応される．規定できる属性は，

- 結合的 `[ assoc ]`
- 結合的かつ交換的 `[ assoc comm ]`
- 結合的かつ交換的かつ単位元を持つ`[ assoc comm id:` *term* `]`

の 3 種類である．

等式宣言は抽象データ型の意味を定義するものである．等式宣言や書き換え規則宣言で使用される変数の宣言は以下のように記述する．

`var` *variable-name* `:` *sort-name* `.`
`vars` *list-of-variable-name* `:` *sort-name* `.`

図 4.17: 変数宣言

等式宣言には 3 種類の宣言方法がある．

`eq` *term* `=` *term* `.`
`eq` *term* `=` `if` *boolean_expression* `then` *term* `else` *term* `fi` `.`
`cq` *term* `=` *term* `if` *boolean_expression* `.`

図 4.18: 等式宣言

同様に，書き換え規則宣言にも 3 種類の宣言方法がある．

`rl` *term* `=>` *term* `.`
`rl` *term* `=>` `if` *boolean* `then` *term* `else` *term* `fi` `.`
`crl` *term* `=>` *term* `if` *boolean* `.`

図 4.19: 書き換え規則宣言

4.2.2 有機的プログラミング言語 GAEA の操作的意味

GAEA の操作的意味を与えるための方針について述べる [24]．GAEA の計算状態を項によって表現するために，計算状態を構成する要素を項で，計算状態を遷移させる書き換え (状態遷移) 規則を計算状態を表現する項の部分項に適用できる形で定義する必要がある．はじめに計算状態を構成する要素について言及する．先に述べたように，GAEA の重要な計算主体であるセルとプロセスに着目する．セルはプログラム記述単位であったので,

- 自分自身の名前,
- セル変数とそれらに対応する値の組のリスト,
- 節の定義や事実

という情報を含んでいる．プロセスは処理単位であり，並行計算が可能な GAEA では Prolog を拡張したひとつの処理系と考えられるので,

- 自分自身の名前,
- 自分自身の環境 (セルの名前のスタック),
- 発火点情報,
- 未処理のゴール列,
- バックトラック情報

を含んでいると考えられる．

GAEA の計算状態遷移は図 4.20のようなイメージである．計算状態は先にあげたセルおよびプロセスの集合によって構築されている．

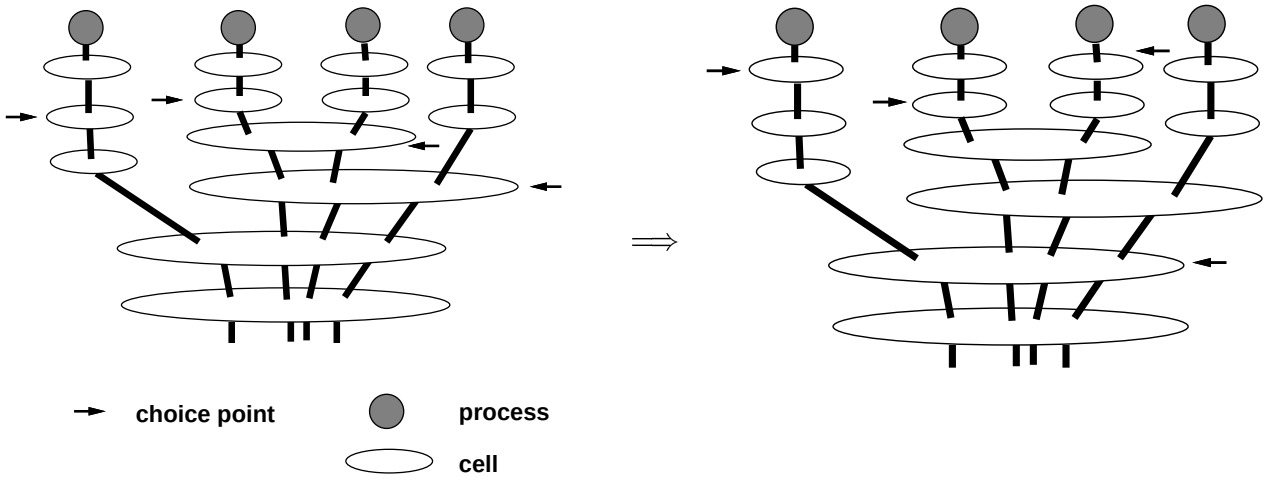

図 4.20: 計算状態の遷移

本研究では，セルおよびプロセスをおのおの `cell` および `process` といった構成子を用いた項で表現する．また，GAEA の計算状態を `state` という結合的かつ交換的な属性を持つ構成子によって構成された項で表現する．このような属性を持つ構成子によって組み上げられた項は，部分項の並び順は任意である．結合的かつ交換的な属性を持つ構成子によって構成された項では多重集合は表現できるが集合は不可能である．しかし，セルやプロセスは各々名前を持っており，それらは重複を許していないので，セルやプロセスの存在を一意に決定することが可能であることを仮定している．この仮定と，計算状態を構成する構成子の性質をあわせることにより，計算状態をセルやプロセスの集合として表現できる．

操作的意味は状態間に定義された書き換え (状態遷移) 規則で表現する．書き換え規則の左辺は計算状態全体ではなく，計算状態の部分状態 (部分項) に適用可能な形で定義する．並行計算は，計算状態の共通部分を持たないいくつかの部分項に対して書き換え規則を同時に適用することによって実現される．本研究で提案している設計方針にしたがえば，書き換え規則は大別すると

- カレントゴールを処理するための節定義を検索する規則
- GAEA の組み込み述語に対応する規則

の 2 種類が規定される．

4.2.3節以降は項および書き換え規則の定義を与えている．4.2.3節で与えた操作的意味を GAEA 開発メンバーに説明したところ，4.2.4節で述べるような有益なコメントをいただいくことができた．4.2.5節の記述はそれらのコメントを反映させた 4.2.3節の改良版である．

4.2.3 操作的意味その 1

セルの項表現は

```
op cell : Qid CVPairList ClauseList -> Object .
```

図 4.21: セルの定義

と定義された構成子によって構築する．ここで，Qid はセルの名前を表現するためのソート，CVPairList はセル変数とそれに対応する値の組のリストを表現するソート，ClauseList は事実や節のリストを表現するソートである．セルやプロセスはObject というソートに含まれることにしている．

プロセスは 3 種類の構成子

```
op process : Qid EnvList Loc TermList ControlList -> Object .
op cprocess : Qid EnvList Loc Loc TermList ControlList -> Object .
op eprocess : Qid EnvList Loc TermList ControlList -> Object .
```

図 4.22: プロセスの定義その 1

を用いて表現する．ここで，Qid はプロセスの名前を表現するためのソート，EnvList は環境を表現するソート，Loc は発火点位置を表現するソートで loc(C,M,N) なる構造の項を要素とする．この項は EnvList 中の先頭から M 番目のセル C 中の N 番目に定義されている節の位置を表現している．EnvList は同一セル名を複数含んでいる可能性があるため，発火点位置情報は環境内のセルの位置が必要となる．ただし，プロセスcprocess の第 4 引数Loc は NoLoc もしくは Meta のいずれかである．詳細については後述する．TermList は未処理のゴール列を表現するソート，ControlList はバックトラック処理のための情報を表現するソートで，ctrl(TermList,Loc) なる構造の項を要素とする．

GAEA では組み込み述語と同名，同引数の述語がユーザによって定義可能である．その場合，ある述語が組み込み述語であるか否かを判断する必要があると考えた．そのため，プロセス process に含まれる未処理のゴール列の先頭の述語名をチェックし，その結果をプロセス cprocess の第 4 引数Loc に反映させている．もしカレントゴールの述語名とある組み込み述語名が一致したら Meta，そうでなければ NoLoc とした．この情報によって，環

境内の節の定義をすべて検索した後，さらに組み込み述語に対応する書き換え規則を適応するか，直ちにバックトラック処理を行なうかといった判断が可能となる．eprocess は実行中にエラーが発生したプロセスを表現するために用いる．

GAEA システムの状態を構成する構成子は

```
sorts Object State .
subsort Object < State .
op state : State State -> State [ assoc comm ] .
```

図 4.23: システムの状態

と定義する．セルやプロセスの存在は一意的，すなわち，同一のセルやプロセスは存在しないという仮定と，上記の構成子の性質によって，計算状態はセルとプロセスの集合として表現することができる．

次に GAEA の操作的意味を定義するために重要だと考えられるいくつかの書き換え規則について述べる．

現段階での Maude の仕様では，単位元を持ちかつ結合的であるようなオペレータを宣言することができない．したがって，書き換え規則の左辺に出現するプロセスを表現する項中の未処理のゴール列を表現する項 termlist(T,TL) は単位項 T とマッチすることができない．我々が実際に与えた書き換え規則は，ゴール列 (項の列) 用および単項用の 2 種類がある．しかし本節ではゴール列用の書き換え規則を用いて説明する．

まず，プロセス内のカレントゴールによる分類をおこなうための書き換え規則を与える．以下のプログラム中で，--- で始まる行はプログラムコメントである．

```
--- classification rule
--- goal part is termlist
rl
process(P,EL,L,termlist(T,TL),Ctrls)
=>
if belongsto(T,termlist(!,fail,'write,'nl,'add,'eq,
                        'fork,'new-cell,'push,'assert,
                        'cv-write,'cv-take,'cv-set,
                        'cv-ref,'subst-cell,'cv-read,'die))
then cprocess(P,EL,L,Meta,termlist(T,TL),Ctrls)
```

```
else cprocess(P,EL,L,NoLoc,termlist(T,TL),Ctrls)
fi .
```

図 4.24: 分類規則

ここで，P はプロセスの名前，EL は環境，L は発火点位置，T はカレントゴール，TL は未処理のゴール列，Ctrls はバックトラック処理情報のスタックである．ここで，process，termlist やbelongsto といった項の構成子や書き換え規則の右辺の if 文の条件部に出現する述語名には引用符がなく，GAEA の述語名には引用符がつけられていることに注意されたい．

書き換え規則の右辺の条件部分では，カレントゴール T の述語名が組み込み述語名と一致するかどうかを確認している．述語 belongsto は

```
op belongsto : Term TermList -> Bool .
```

と定義されている．図 4.24の書き換え規則で分類されたプロセスに対し，次に説明する 2 つの主要規則によってプロセス内の未処理ゴール列を処理するための計算がおこなわれる．

第一の書き換え規則は，cprocess の第 4 引数がNoLoc の場合に適用される．

```
--- main rule No.1
--- goal part is termlist
rl
state(cell(E,CVS,CLS) ,
      cprocess(P,EL,loc(E,M,N),NoLoc,termlist(T,TL),Ctrls))
=>
if in(E,EL)
--- セルが環境内に存在し，
then
  --- N 番目の節が存在し
  if nthcl(N,CLS) =/= ClError
  then
    if unify(eqn(head-part(nthcl(N,CLS)),T)) =/= failure
    --- その節とユニフィケーションが成功したら処理を進める
    then state(cell(E,CVS,CLS),
               process(P,EL,loc(nthe(1,EL),1,1) ,
```

```
                    subst(unify(eqn(head-part(nthcl(N,CLS)),T)),
                          termlist(body-part(nthcl(N,CLS)),TL)),
                    ctrllist(ctrl(termlist(T,TL),loc(E,M,N+1)),
                    Ctrls)))
    --- ユニフィケーションが失敗したら次の候補節を探す
    else state(cell(E,CVS,CLS),
               cprocess(P,EL,loc(E,M,N+1),BL,termlist(T,TL),Ctrls))
    fi
  else
    if nthe(M+1,EL) =/= EnvError
    --- M+1 番目のセルが環境内に存在するなら発火点情報を更新する
    then state(cell(E,CVS,CLS),
               cprocess(P,EL,loc(nthe(M+1,EL),M+1,1),
                        BL,termlist(T,TL),Ctrls))
    else
    --- 次のセルが環境に存在せず,
      if Ctrls =/= Empty
      --- バックトラック情報が存在すればバックトラック処理をおこなう
      then state(cell(E,CVS,CLS),
                 process(P,EL,snd(head(Ctrls)),
                         fst(head(Ctrls)),tail(Ctrls)))
      --- バックトラック情報が存在しなければエラー
      else state(cell(E,CVS,CLS),
                 eprocess(P,EL,BL,termlist(T,TL),Ctrls))
      fi
    fi
  fi
else
  if Ctrls =/= Empty
  --- バックトラック情報が存在すればバックトラック処理をおこなう
  then state(cell(E,CVS,CLS),
             process(P,EL,snd(head(Ctrls)),
                     fst(head(Ctrls)),tail(Ctrls)))
  --- バックトラック情報が存在しなければエラー
  else state(cell(E,CVS,CLS),
```

```
            eprocess(P,EL,BL,termlist(T,TL),Ctrls))
  fi
fi .
```

図 4.25: 主要規則その 1

ここで E はセルの名前，CVS はセル変数とそれに対応する値の組のリスト，CLS は節のリストである．さらに，in，nthcl，unify，subst，head-part，body-part，nthe などは関数として定義されている．

第二の書き換え規則は，cprocess の第 4 引数がMeta の場合に使用される．

```
--- main rule No.2
--- goal part is termlist
rl
state(cell(E,CVS,CLS) ,
      cprocess(P,EL,loc(E,M,N),Meta,termlist(T,TL),Ctrls))
=>
if in(E,EL)
--- セルが環境内に存在し，
then
  if nthcl(N,CLS) =/= ClError
  --- N 番目の節が存在し
  then
    if unify(eqn(head-part(nthcl(N,CLS)),T)) =/= failure
    --- その節とユニフィケーションが成功したら処理を進める
    then state(cell(E,CVS,CLS),
               process(P,EL,loc(nthe(1,EL),1,1),
                       subst(unify(eqn(head-part(nthcl(N,CLS)),T)),
                             termlist(body-part(nthcl(N,CLS)),TL)),
                       ctrllist(ctrl(termlist(T,TL),loc(E,M,N+1)),
                       Ctrls)))
    --- ユニフィケーションが失敗したら次の候補節を探す
    else state(cell(E,CVS,CLS),
               cprocess(P,EL,loc(E,M,N+1),BL,
                        termlist(T,TL),Ctrls))
```

```
    fi
  else
    if nthe(M+1,EL) =/= EnvError
    --- M+1 番目のセルが環境内に存在するなら発火点情報を更新する
    then state(cell(E,CVS,CLS),
               cprocess(P,EL,loc(nthe(M+1,EL),M+1,1),
                        BL,termlist(T,TL),Ctrls))
    --- 次のセルが環境に存在しなければ組み込み述語の適用を試みる
    else state(cell(E,CVS,CLS),
               cprocess(P,EL,Meta,Meta,termlist(T,TL),Ctrls))
    fi
  fi
else
  if Ctrls =/= Empty
  --- バックトラック情報が存在すればバックトラック処理をおこなう
  then state(cell(E,CVS,CLS),
             process(P,EL,snd(head(Ctrls)),
                     fst(head(Ctrls)),tail(Ctrls)))
  --- バックトラック情報が存在しなければエラー
  else state(cell(E,CVS,CLS),
             eprocess(P,EL,BL,termlist(T,TL),Ctrls))
  fi
fi .
```

図 4.26: 主要規則その 2

上記 2 つの書き換え規則の差異は，カレントゴールを処理するために環境内の節をすべて検索し終った後の処理にある．前者の場合は，カレントゴールの述語名が GAEA の組み込み述語と一致していなかったため，バックトラック処理を，可能ならば行なっている．後者の場合は，規則の左辺を，組み込み述語に対応する書き換え規則が適用可能な形，具体的には，プロセス`cprocess` の第 4 引数が`Meta` となるように定義している．

次に GAEA の組み込み述語のうち，環境 (セルのモジュール構造) を操作する述語である `'push`，`'subst-cell` およびセル変数を操作する述語`'cv-write` に対応する書き換え規則の定義を与える．これら以外の組み込み述語に対応する書き換え規則も与えているがここ

では触れない.

述語 push(Cell) はこの述語を呼び出したプロセスの環境の先頭にセルの名前 Cell を追加する．もし，セルの名前 Cell が名前管理用のセル 'system-cell 内に登録されていなければ，そのプロセスはセルの名前 Cell がセル 'system-cell に登録されるまで待ち状態になる．我々の表記では，述語push(Cell) を項pred('push,Cell) と表現している．述語 push(Cell) に対応する書き換え規則は次のように与えることができる.

```
--- push
crl
state(cell('system-cell,CVS,CLS),
      cprocess(P,EL,Meta,Meta,termlist(pred('push,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,env(Cell,EL),loc(Cell,1,1),GL,Ctrls))
if cell-exists(Cell,CLS) .
```

図 4.27: 述語 push(Cell) に対応する書き換え規則

セルの名前が名前管理用セルに登録されていたら図 4.27 の書き換え規則が適用される．そうでなければ，述語 push(Cell) を処理する書き換えは生じない．よって，待ち状態を表現していることになる.

述語 subst-cell(Old,New) は，この述語を呼び出したプロセスの環境内のセル名 Old を既に名前管理用セル内に存在するセル名 New に置き換える．もし，環境内にセル名 Old が存在しなければ失敗し，セル名 New が名前管理用セルに存在しなければエラーとなる．述語 subst_cell(Old,New) に対応する書き換え規則は以下のように与えることができる.

```
--- subst_cell
--- Old が EL 中に，New が 名前管理用セルに存在する場合
crl
state(cell('system-cell,CVS,CLS),
      cprocess(P,EL,Meta,Meta,
               termlist(pred('subst-cell,termlist(Old,New)),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
```

```
      process(P,subst-cell(New,Old,EL),
              loc(nthe(1,subst-cell(New,Old,Env)),1,1),GL,Ctrls))
if and(in(Old,EL),cell-exists(New,CLS)) .

--- Old が EL 中存在せず, New が 名前管理用セルに存在する場合
--- バックトラック情報が空の場合
crl
state(cell('system-cell,CVS,CLS),
      cprocess(P,EL,Meta,Meta,
               termlist(pred('subst-cell,termlist(Old,New)),GL),Empty))
=>
state(cell('system-cell,CVS,CLS),
      eprocess(P,EL,Meta,
               termlist(pred('subst-cell,termlist(Old,New)),GL),Empty))
if and(in(Old,EL)==false,cell-exists(New,CLS)) .

--- Old が EL 中存在せず, New が 名前管理用セルに存在する場合
--- バックトラック情報が空でない場合
crl
state(cell('system-cell,CVS,CLS),
      cprocess(P,EL,Meta,Meta,
               termlist(pred('subst-cell,termlist(Old,New)),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,snd(head(Ctrls)),fst(head(Ctrls)),tail(Ctrls)))
if and(in(Old,EL) == false,cell-exists(New,CLS),Ctrls =/= Empty) .

--- New が 名前管理用セルに存在しない場合
--- エラー
crl
state(cell('system-cell,CVS,CLS),
      cprocess(P,EL,Meta,Meta,
               termlist(pred('subst-cell,termlist(Old,New)),GL),Ctrls))
=>
state(cell(stdlib,CVS,CLS),
```

```
      eprocess(P,EL,Meta,
               termlist(pred('subst-cell,termlist(Old,New)),GL),Ctrls))
if cell-exists(New,CLS) == false .
```

図 4.28: 述語 subst_cell(Old,New) に対応する書き換え規則

述語 cv_write(Cell,CV,V) は，セル Cell 中のセル変数CV に値 V を書き込む．セル変数にデフォルト値 undef 以外の値が格納されていたら，その値がデフォルト値に変更されるまで待ち状態になる．指定されたセル変数が存在しない場合は，それが生成されるまで待ち状態になる．述語 cv_write(Cell,CV,V) に対応する書き換え規則は次のように与えることができる．

```
--- cv_write
--- セル変数が存在し，値が割り当てられていない場合
crl
state(cell(Cell,CVS,CLS),
      cprocess(P,EL,Meta,Meta,
               termlist(pred('cv-write,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(nthe(1,EL),1,1),TL,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .
```

図 4.29: 述語 cv_write(Cell,CV,V) に対応する書き換え規則

セル変数が存在し，値が割り当てられている場合や，セル変数が設定されていない場合に対応する書き換え規則が定義されていないということは，状態の書き換えが生じないことを意味するので，この述語が処理されず待ち状態となっていることを表現している．

4.2.4 操作的意味その 1 の反省

4.2.3節で与えた操作的意味，特に，図 4.24，図 4.25 および 図 4.26 の書き換え規則は，左辺がどのように書き換えられるかを，その右辺で if 文を再帰的に使用することによって規定している．

実行可能な記述を与えたという意味においては特に問題はないと考えられる．しかしながら，この記述を用いてシステムの性質を解析を試みる場合には少々問題が生じる．書き換え規則によって操作的意味を定義した場合，この記述を利用した解析手法として考えられるのは状態を表現している項に関する構造帰納法である．項に関する構造帰納法では，状態を項の構造で分類し，おのおのの項に適用可能な書き換え規則によって，それらの項を書き換え，書き換えられた項を再度構造に応じて分類し，適用可能な書き換え規則によって書き換えるといった操作を繰り返し行なう．この際，ある項に対して適応可能な書き換え規則はできる限り少ない方が，可能ならば高々1 つであることが望ましい．なぜなら，適応可能な書き換え規則が存在しなければその項に関する計算は停止したことを意味し，唯一つしか書き換え規則が存在しなければ，書き換えられる構造は一意に決定されるからである．

4.2.3で与えたように，書き換え規則の右辺が `if` 文を再帰的に使用した表現である場合，その書き換え規則の左辺にマッチした項が，どの項に書き換えられるかを瞬時に判断するのは困難である．したがって，これらの記述はシステムの解析には不向きであると考えられる．

また，4.2.3節で与えた操作的意味を GAEA 開発メンバーに説明したところ，

- GAEA の実装の方では、カレントゴールの述語名が組み込み述語と一致するか否かという前処理は行なっていない．
- `loc(Cell,M,N)` という発火点情報には，セル名に関する情報が`Cell` および `M` の 2 種類があり冗長である．
- セルやプロセスの名前を表現するソートの名前が `Qid` では，仕様の可読性は低い．

といった指摘を受けた．

以上のような理由から，4.2.3節で与えた図 4.24，図 4.25 および図 4.26 の書き換え規則を上記の問題点を解消するよう改良を試みる．

4.2.5 操作的意味その 2

セルの定義は 4.2.3節の定義中のセルの名前を表現するソートを`Qid` から`CName` へと変更する．これにより，各引数が何を意味しているのかを容易に知ることができる．

プロセスの定義を以下のように変更する．

```
op process : PName Environment Location TermList ControlList -> Object .
op process-e : PName Environment Location TermList ControlList
               CName -> Object .
op process-e-c : PName Environment Location TermList ControlList
                 CName Clause -> Object .
```

図 4.30: プロセスの定義その 2

セルやプロセスに含まれる情報の項による表現を簡単にまとめておく.

表現すべき情報	項による表現
環境	`environment(CName1,CName2,...)`
発火点位置	`loc(M,N), Meta, NoLoc`
セル変数とその値の組	`cvpair(CellVariable,Value)`
そのリスト	`cvs(cvpair(CV,V),...)`
節定義	`cldef(HeadPart,ConditionPart,BodyPart)`
そのリスト	`clauselist(cldef(H,C,B),...)`
未処理ゴール列	`termlist(pred(PredicateName,Arguments),...)`
バックトラック情報	`ctrl(loc(M,N),termlist(pred(P,A),...))`
そのリスト	`ctrllist(ctrl(Loc,Goals),...)`

図 4.31: セルやプロセスに含まれる項の定義

4.2.3節および本節で与えたプロセス定義 (図 4.22 と図 4.30) の差異は,

- プロセスの名前を表現するソートを `PName` と変更,
- プロセス構成子と引数のソートを変更,
- 発火点位置を `loc(M,N)` で表現,
- 発火点位置が示すセル名 `CName` や 節定義 `Clause` を表出させる

といった点である. このような変更により, 定義の可読性が向上し, さらにプロセス内部での計算, すなわち, 発火点位置情報から節定義を検索し, カレントゴールとのユニフィ

ケーション (プロセスの内部での計算) を実行する場合と，組み込み述語によってカレントゴールを処理する場合とが明確に差別化可能となった．プロセスの内部での計算を表現する項はその構成子が`process-e` や`process-e-c` であるために，他プロセスからの作用を受けないような書き換え規則を容易に定義可能となった．この件に関する詳細は考察において言及する．

プロセスを再定義したことによって，計算状態中のプロセス`process` は以下のように書き換えられる．

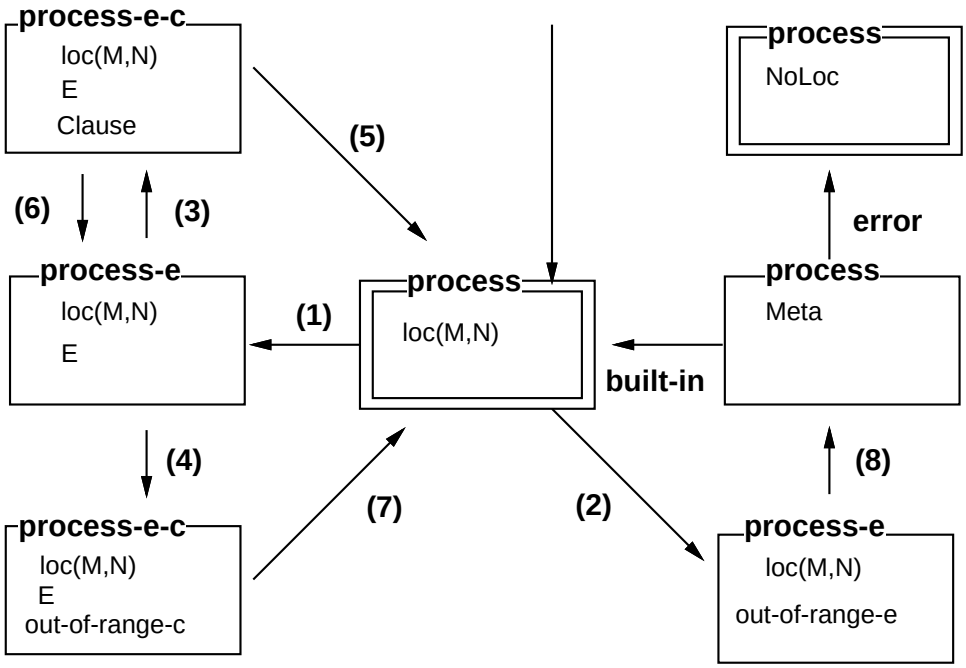

図 4.32: プロセスの遷移

図中で，二重枠線の図形は初期状態および終了状態にあるプロセス，単線枠の図形は初期状態のプロセス内の未処理のゴール列を処理する中間過程を表現するプロセスを意味する．枠内の情報は，発火点位置，発火点位置情報に基づいて抽出したセル名や節定義を意味している．プロセスは矢印の方向に遷移する．矢印の脇の`built-in` および `error` は組み込み述語に対応する書き換え規則による遷移を意味する．以下に各々のプロセスが表現している意味とその表現に適用可能な書き換え規則を定義する．

初期状態のプロセスは，発火点位置情報に基づいたカレントゴールを処理するための初期状態を表現している．この状態にに適用可能な書き換え規則を次のように定義する．

```
--- 環境から M 番目の要素を取り出す.
  rl
  process(P,EL,loc(M,N),termlist(T,TL),Ctrls)
  =>
  process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,nthe(M,EL)) .
```

図 4.33: セル名の取り出し

ただし，nthe(M,EL) は EL の長さが

- M 以上であれば，先頭からM 番目の要素
- M 未満であれば，out-of-range-e

を返す関数とする．この書き換え規則によって，プロセス

```
process(P,EL,loc(M,N),termlist(T,TL),Ctrls)
```

図 4.34: プロセスの初期状態

は図 4.32中の遷移 (1) によって，発火点位置が指し示すセル名が自分自身の環境から獲得できた状態

```
process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,E)
```

図 4.35: セル名が獲得できた状態

もしくは遷移 (2) によって，発火点位置が自分自身の環境から外れてしまった状態，すなわち，自分自身の環境に定義されている節定義をすべて見尽くしてしまった状態

```
process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,out-of-range-e)
```

図 4.36: セル名が獲得できなった状態

へと変化する．

発火点位置が指し示すセル名が自分自身の環境から獲得した項(図 4.35) は，そのセル名を持つセルを表現する項と組み合わせて，図 4.37の書き換え規則によって発火点位置が指し示す節定義を獲得した状態へと書き換えられる．

```
--- セル名が獲得できたら，発火点位置が示す節定義を取り出す
  rl
  state(cell(E,CVS,CLS),
        process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,E))
  =>
  state(cell(E,CVS,CLS),
        process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,E,nthcl(N,CLS))) .
```

図 4.37: 節定義の獲得

ただし，`nthcl(N,CLS)` は `CLS` の長さが

- `N` 以上であれば，先頭から`N` 番目の要素
- `N` 未満であれば，`out-of-range-c`

を返す関数とする．この書き換え規則によって，計算状態の部分項

```
state(cell(E,CVS,CLS),
      process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,E))
```

図 4.38: 節定義を得るための状態

は図 4.32中の遷移 (3) をたどり，発火点位置が指し示す節定義を獲得した状態

```
state(cell(E,CVS,CLS),
      process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,
                  E,cldef(Head,Cond,Body))
```

図 4.39: 節定義が得られた状態

もしくは遷移 (4) をたどり，発火点位置が指し示すセル中の節定義を見尽くしてしまった状態

```
state(cell(E,CVS,CLS),
      process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,
                  E,out-of-range-c))
```

図 4.40: 節定義が得られなかった状態

へと書き換えられる．ここで，cldef(Head,Cond,Body) は

```
op cldef : Term TermList TermList -> Clause .
```

図 4.41: 節定義のための構成子

と定義された節定義構成子cldef から構成される項である．

次に遷移 (3) で生成されたプロセスを含む状態を表現する項に適用可能な書き換え規則を与える．発火点位置が示す節定義のヘッド部がカレントゴールとユニフィケーションが成功した場合 (遷移 (5)) と失敗した場合 (遷移 (6)) の書き換え規則である．ユニフィケーションが成功した場合は，プロセス構成子がprocess-e-c からprocess に変化している．変更後のプロセスは，発火点位置は環境の先頭に，未処理ゴール列は，ユニフィケーションの結果を，節定義の条件部，ボディ部および未処理のゴール列に反映させた項，バックトラック情報には，変更前の発火点位置の節定義を指す位置を更新したものと，未処理ゴール列の組が格納される．

```
--- 節定義のヘッド部とカレントゴールとのユニフィケーションが成功した場合
  crl
  process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,
              E,cldef(Head,Cond,Body))
  =>
  process(P,EL,loc(1,1),
          subst(unify(eqn(Head,T)),termlist(Cond,Body,TL)),
          ctrllist(ctrl(loc(M,N + 1),termlist(T,TL)),Ctrls))
  if unify(eqn(Head,T)) =/= failure .
```

図 4.42: ユニフィケーション成功

ユニフィケーションが失敗した場合は新たな節定義を検索するために発火点位置の節定義を指し示す位置を更新し，プロセス構成子をprocess-e に戻している．

```
--- 節定義のヘッド部とカレントゴールとのユニフィケーションが失敗した場合
  crl
  process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,
              E,cldef(Head,Cond,Body))
  =>
  process-e(P,EL,loc(M,N + 1),termlist(T,TL),Ctrls,E)
  if unify(eqn(Head,T)) == failure .
```

図 4.43: ユニフィケーション失敗

発火点位置が指し示すセル中の節定義を見尽くしてしまった状態を表現するプロセスは，発火点位置を自身の環境内の次のセルを指し示すように以下の書き換え規則によって書き換えられる (遷移 (7))．

```
--- 発火点位置を環境内の次のセルへ移す
  rl
  process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,E,out-of-range-c)
  =>
  process(P,EL,loc(M + 1,1),termlist(T,TL),Ctrls) .
```

図 4.44: 次のセル名に移る

発火点位置が自分自身の環境から外れてしまった状態，すなわち，自分自身の環境に定義されている節定義をすべて見尽くしてしまった状態は，以下の書き換え規則によって，プロセスをカレントゴールに組み込み述語を適用できる状態に書き換える．

```
--- セル名が獲得できていなかったら組み込み述語に対応する
--- 書き換え規則の適用を考慮する．
  rl
  process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,out-of-range-e)
  =>
  process(P,EL,Meta,termlist(T,TL),Ctrls) .
```

図 4.45: 組み込み述語適用の準備

プロセスprocess(P,EL,NoLoc,GL,Ctrls) は,

- 未処理ゴール列GL が空列nil ならば，そのプロセスの計算は正常終了
- 未処理ゴール列GL が空列でなければ，そのプロセスは未処理ゴール列の先頭ゴールの処理に失敗し，エラーが発生

したことを表現している．

組み込み述語に対応する書き換え規則は，4.2.3節で導入した書き換え規則に出現するプロセスの表現を本節で導入した表現に変更すればよい．本節で与えた表現方法に基づく操作的意味は Appendix A に掲載する．

4.3 自律ロボットの例題

4.2.3，4.2.5節の操作的意味記述は代数仕様言語 Maude で与えているので，いずれもその処理系で実行可能である．この記述に基づき，GAEA の動作を説明するために導入した自律ロボットの例題を記述し実行した．4.2.5節で定義した改良された操作的意味に基づいた例題の記述や実行結果を Appendix B で与えている．

4.4 考察

本章では，並行自己反映計算の例として有機的プログラミング言語 GAEA をとりあげ，書き換え論理に基づいて操作的意味を定義した．したがって，計算状態を項で，計算を状態遷移列で，状態遷移は状態の部分項に適用可能な書き換え規則によって定義している．これらの記述に基づいていくつかの考察を行なう．

4.4.1 待ち状態の表現

GAEA には，push(Cell) のように,

> 指定されたセル名Cell が既に存在すればそれを自身の環境の先頭に追加し，そうでなければその名前を持つセルが生成されるまで待つ

といった処理を行なう述語が組み込まれている．セル名を環境の先頭に追加する処理は図4.46のように状態遷移規則を明示的に記述することによって定義できる．

```
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('push,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(Cell,EL),loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) .
```

図 4.46: 組み込み述語push/1 に対応する書換え規則

一方，カレントゴールが組み込み述語で，それに対応する状態遷移規則が存在しても，適用するための条件が成立しなければ状態は遷移しない．

セル名がシステム内に存在しない場合のpush(Cell) の動作に対応する状態遷移規則を定義しなければ，その条件を満たす状態遷移は生じないので，待ち状態を表現していることがわかる．

4.4.2 同期処理

GAEA ではcv_write(Cell,CV,V) のようなセル変数を操作する組み込み述語によって同期処理機構を実現している．4.2.5節で再定義したプロセスの表現方法を用いた，述語 cv_write(Cell,CV,V) に対応する書き換え規則は以下のように定義される．

```
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-write,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),TL,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .
```

図 4.47: 述語 cv_write(Cell,CV,V) に対応する書き換え規則

自律ロボットの例題においては，交差点には同時に 2 台以上の自律ロボットが存在しないという制約条件があったが，その条件は必ず守られることが以下のように確認できる．

簡単のため 2 台のロボットで考えることにする．ロボット名およびプロセス名をおのおの'r1，'r2 および'p1，'p2 で表現する．ロボット'r1 および'r2 が交差点の入口に到着したとき，そのロボットに対応するプロセスは

```
process('p1,environment('want,'r1,'main,'stdlib,'system-cell),
        loc(1,1),termlist(pred('e,nil),GL1),Ctrls1)
```

図 4.48: プロセス'p1 の項表現

および

```
process('p2,environment('want,'r2,'main,'stdlib,'system-cell),
        loc(1,1),termlist(pred('e,nil),GL2),Ctrls2)
```

図 4.49: プロセス'p2 の項表現

という項で表現されている．

交差点にロボットが 1 台存在している場合と，1 台も存在していない場合を考える．

交差点にロボットが 1 台存在している場合:

ロボット'r1 が交差点内に存在していると仮定する．そのような計算状態を表現している項はその部分項として

```
cell('com,cvpair('message,pred('pair('r1,'entered))),NoClause)
```

図 4.50: ロボット'r1 が交差点にいる場合の状態表現

という形の，プロセス間通信のため (もちろん同期処理のためでもある) に設定されたセル'com を含んでいる．このとき，ロボット'r2 に対応するプロセス (図 4.49) からどのように遷移するかを考える．

発火点情報とカレントゴールにしたがえば，まずセル'want に定義されている一番目の節定義

```
cldef(pred('e,nil),pred('find,'entered),
      termlist(!,pred('output,'someone-there),
               pred('subst-cell,termlist('stop,'want))))
```

図 4.51: セル'want に格納されている一番目の節定義

とのユニフィケーションを試みることになる．この場合，カレントゴールと節定義のヘッド部とのユニフィケーションは成功するので次に条件部pred('find,'entered) のテストを行なう．述語pred('find,'entered) は利用者定義の述語で，セル'main 中で

```
cldef(pred('find,'Mode),nil,
      termlist(!,pred('cv-ref,termlist('com,'message,
                                pred('pair,termlist('Name,'Mode)))),
               pred('name,'M),
               pred('not,pred('eq,termlist('Name,'M)))))
```

図 4.52: 述語find/1 の定義

のように定義されている．指定されたセル変数の値を参照する組み込み述語

```
pred(cv-ref,termlist('com,'message,
                     pred('pair,termlist('Name,'Mode))))
```

図 4.53: 組み込み述語cv-rev/3 の呼出し

によって，値pred('pair,termlist('r1,'entered)) が得られ，pred('name,'M) をプロセス'p2 の環境で処理すればpred('name,'r2) を得る．従って， ボディ部の最後のゴール pred('not,pred('eq,termlist('r1,'r2))) は成功する．

これにより，図 4.51の節定義の条件部は成功し，ロボット'r2 に対応するプロセスの持つ環境の先頭が'want から'stop に変わる (図 4.54)．

```
process('p2,environment('stop,'r2,'main,'stdlib,'system-cell),
        loc(1,1),termlist(pred('e,nil),GL2'),Ctrls2')
```

図 4.54: ロボット'r2 のモードが変化

上記のプロセスはセル'stop に定義されている節によって，

- 交差点にロボットが存在しないことが判明したら，自らのプロセスの環境の先頭を 'stop から'proceed に変更するか，
- 交差点にロボットが存在している間は，待ち状態のままである

かのいずれかの処理をおこなう．これらの場合，セル'com のセル変数'message の値を参照するだけであり，その値を変更することはない．したがって，ロボット'r1 が交差点に存在する間はロボット'r2 は交差点には入ることができない．

交差点に 1 台もロボットが存在していない場合:

交差点に 1 台もロボットが存在していない場合，そのような計算状態を表現している項は，以下のような形の部分項を持っている (図 4.55)．

```
cell('com,cvpair('message,undef),NoClause)
```

図 4.55: 交差点に誰も存在しない状態

これはプロセス間通信のため (もちろん同期処理のためでもある) に設定されたセルである．

ロボット'r1 を表現しているプロセス (図 4.48) は，発火点情報とカレントゴールにしたがえば，まず，セル'want に定義されている一番目の節定義 (図 4.51) とのユニフィケーションを試みる．この場合，カレントゴールと節定義のヘッド部とのユニフィケーションは成功するが，条件部pred('find,'entered) では，セル'com のセル変数'message に割り当てられている値とpred('pair,termlist('Name,'Mode)) とのパターンマッチに失敗する．従って，一番目の節定義とのユニフィケーションは失敗する．次に，セル'want に定義されている二番目の節定義

```
cldef(pred('e,nil),nil,
      termlist(!,pred('enter,nil),
               pred('subst-cell,termlist('enter,'want))))
```

図 4.56: セル'want に格納されている二番目の節定義

とのユニフィケーションを試みる．一番目の節定義のユニフィケーションに失敗したことは，交差点内にはロボットが存在しないことを意味している．この節定義中のボディ部の

部分ゴールpred('enter,nil) によって，セル'com のセル変数'message にロボットr1 が存在していることを意味する値pred('pair,termlist('r1,'entered)) が書き込まれ，ロボットr1 は交差点に侵入したことになる．そこで，プロセス'p1 がセル'com のセル変数'message に値を書き込む直前の状態に遷移したと仮定する．そのときのプロセス'p1 は

```
process('p1,environment('want,'r1,'main,'stdlib,'system-cell),
        Meta,
        termlist(pred('cv-write,
                      termlist('com,'message,
                          pred('pair,termlist('r1,'entered)))),GL1'),
        Ctrls1')
```

図 4.57: プロセス'p1 の変化

と表現されている．同様にプロセス'p2 も

```
process('p2,environment('want,'r2,'main,'stdlib,'system-cell),
        Meta,
        termlist(pred('cv-write,
                      termlist('com,'message,
                          pred('pair,termlist('r2,'entered)))),GL2'),
        Ctrls2')
```

図 4.58: プロセス'p2 の変化

へ遷移してきたとする．このときの計算状態はロボット'r1 および'r2 の双方とも交差点へ入ろうとしている状況を表現している．そこで次の状態遷移が重要となる．プロセス'p1(図 4.57) もしくは'p2(図 4.58) のいずれかとセル'com(図 4.55) の組に対してセル変数を操作する組み込み述語に対応する状態遷移規則

```
crl
state(cell(E,CVS,CLS),
```

```
      process(P,EL,Meta,
              termlist(pred('cv-write,termlist(E,CV,V)),GL),Ctrls))
=>
state(cell(E,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .
```

図 4.59: 組み込み述語cv-write/3 に対応する状態遷移規則

が適用される．並行項書換え系を仮定しているので，この規則が上記の 2 つのプロセスに対して同時に適用されることはない．ここではプロセス'p1(図 4.57) とセル'com(図 4.55) の組に対して適用されたとすると，計算状態を表現する項の部分項

```
state(
  cell('com,cvpair('message,undef),NoClause),
  process(p1,environment('want,'r1,'main,'stdlib,'system-cell),
          Meta,
          termlist(pred('cv-write,termlist('com,'message,
                        pred('pair,termlist('r1,'entered)))),GL1'),
          Ctrls1'))
```

図 4.60: 状態遷移規則適用前の状態

は以下に示す状態へと遷移する．

```
state(
  cell('com,cvpair('message,pred('pair,termlist('r1,'entered))),
       NoClause),
  process(p1,environment('want,'r1,'main,'stdlib,'system-cell),
          loc(1,1),GL1',Ctrls1'))
```

図 4.61: 状態遷移規則適用後の状態

セル'com のセル変数'message にundef 以外の値が割り当てられたので，プロセス'p2 自身 (図 4.58) やプロセス'p2(図 4.58) とセル'com(図 4.55) の組に適用可能な状態遷移規則は存在しない．したがって，セル'com のセル変数'message に新たな値を割り当てることが不可能である．以上により交差点には同時に 2 台以上のロボットが存在できないことが確認できた．

4.4.3 他プロセスへの作用

GAEA には自分自身のみならず，自分自身以外のプロセスにも影響を与える述語が提供されている．例えば，述語 push(Cell) は自分自身のプロセスの先頭にセル名 Cell を追加するものであった．これに類する組み込み述語 push(Cell,P2) は，指定されたプロセス名 P2 を持つプロセスの環境の先頭にセル名 Cell を追加するものである．もし指定されたプロセス名が名前管理用セルに登録されていなければエラーである．

プロセスを表現する項の構成子は 4.2.5節で見たように，

- process
- process-e
- process-e-c

の 3 種類であった．これらの構成子の内，process-e およびprocess-e-c で構成された項によって表現されるプロセスは，そのプロセス内部の計算をおこなっている状態を表現している．したがって，例にあげたような，他プロセスの環境を操作する述語push(Cell,P2) に対応する状態遷移規則を以下のように定義することによって，プロセス内部の計算への干渉を回避することが可能である．

```
--- 述語が正常に処理される場合
rl
state(process(P1,Environment1,Meta,
              termlist(pred('push,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,Environment2,Location,GL2,Ctrls2))
=>
state(process(P1,Environment1,loc(1,1),GL1,Ctrls1),
      process(P2,environment(Cell,Environment2),Location,GL2,Ctrls2)) .
```

```
--- エラーが発生する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,Environment1,Meta,
              termlist(pred('push,termlist(Cell,P2)),GL1),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,Environment1,NoLoc,
              termlist(pred('push,termlist(Cell,P2)),GL1),Ctrls1))
if process-exists(P2,CLS) .
```

図 4.62: 述語 push(Cell,P2) に対応する書き換え規則

4.4.4 言語仕様の考察

書き換え論理に基づいて GAEA を表現するための項設計や操作的意味に基づいて言語仕様，具体的には環境変化後のバックトラックについての考察をおこなった．

本研究ではバックトラック情報を発火点位置情報と未処理のゴール列の組のスタックで表現している．以下のような仮定をする．

> プロセスP のカレントゴールCurrentGoal が環境の先頭からM 番目のセルC 中に格納されているN 番目の節定義とユニフィケーションが成功する．このときバックトラック情報として
>
> ```
> ctrl(loc(M,N+1),termlist(CurrentGoal,Goals))
> ```
>
> がバックトラック情報格納スタックの先頭に追加される．ここで，Goals は未処理ゴール列を表現しているものとする．その後，プロセスP が未処理ゴール列の処理を続け，環境の先頭からM 番目のセルC がセルC' に変更される (図 4.63)．環境変更後バックトラックが発生し，先のバックトラック情報が巻戻され，リトライする場面になる．

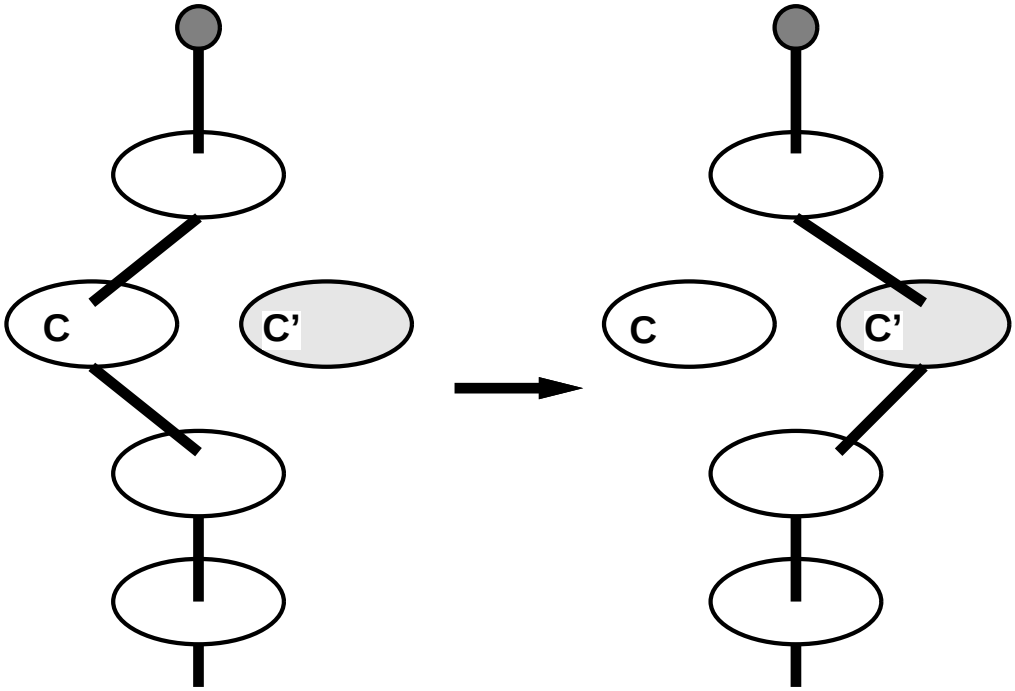

図 4.63: 本研究の項設計に基づく環境変化後のバックトラック

プロセスP の環境が変化しているので，発火点位置情報loc(M,N+1) が指し示すセル名はC ではなくC' である．このとき，未処理ゴール列termlist(CurrentGoal,Goals) のカレントゴールCurrentGoal は先に処理されたときとは全く異なる環境で処理されることになる (図 4.63)．

ところがGAEA システムにおいては，同じ前提であってもリトライ時にはセルC' 中ではなくセルC に格納されているN+1 番目の節定義とカレントゴールCurrentGoal のユニフィケーションを試みる．環境が変化しても変更対象のセルC が環境から完全には分離されていないからである (図 4.64)．バックトラック時の節定義検索が環境の先頭から行なわれる場合はC 中に格納された節定義は検索対象とはならない．しかしセルC の節定義とユニフィケーションが成功したゴールのリトライ時には，C が既に環境外にあってもそれが格納している節定義は検索対象となっている．セルC に格納されているすべての節定義とカレントゴールとのユニフィケーションが失敗した場合は，セルC より下位 (すなわち，セルC' より下位) にあるセル中の節定義を検索することになる．

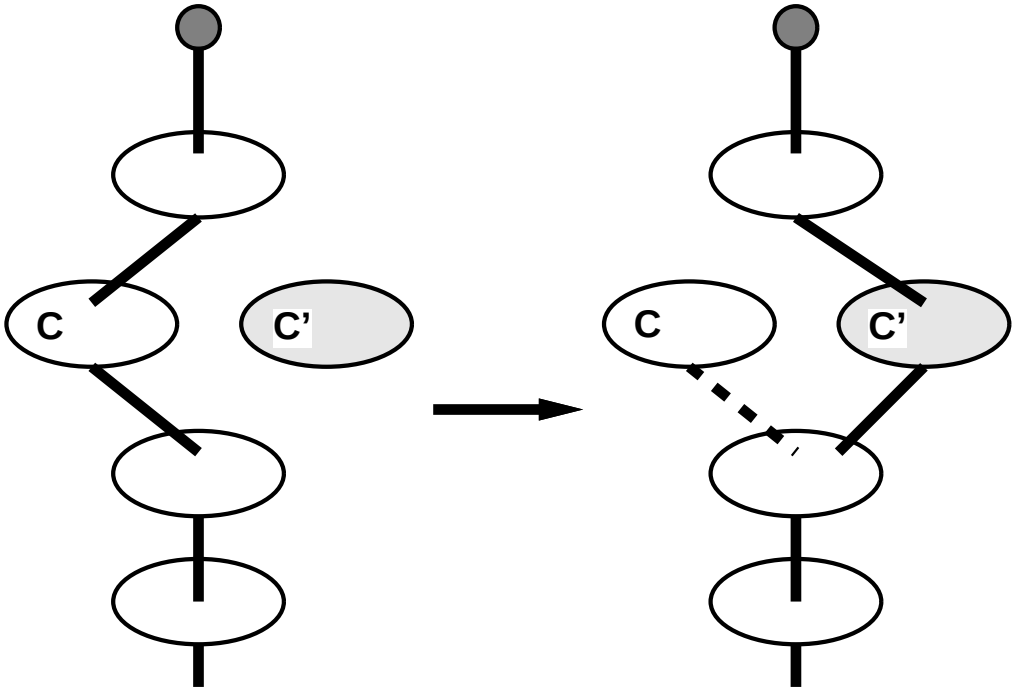

図 4.64: GAEA システムでの環境変化後のバックトラック

前者はバックトラック時のリトライが環境変化を反映しているのに対し，後者では環境変化が反映されておらず，やや不自然だと考えられる．

第 5 章

関連研究

本研究では，並行計算，自己反映計算，書き換え論理といったテーマを含んでいる．これらを組み合わせた内容に関連する研究がいくつか報告されている．

5.1 自己反映計算の操作的意味

山岡らは，一般的な計算システムのモデルである抽象書換え系を用いてリフレクティブタワーをモデル化した [50]．自己反映計算を実現する枠組として，手続き的リフレクション [39] が提案されている．手続き的リフレクションにおいて，自己反映的なシステムは，インタプリタの無限階層としてモデル化される．このインタプリタの階層をリフレクティブタワーと呼ぶ．リフレクティブタワーを構成している各インタプリタは，一段上のインタプリタによって解釈，実行されるプログラムであり，ユーザレベルのプログラムはリフレクティブタワーの最下層のインタプリタによって解釈，実行されていると見なせる．この最下層のインタプリタをベースレベルシステム，ベースレベルシステムを解釈，実行しているインタプリタをメタレベルシステムと呼ぶ．ここでは，最下層から n 段目のインタプリタを n レベルシステムと呼ぶことにする．

タワーの各レベルはおのおの抽象書換え系によってモデル化されている．そこで抽象書換え系の包含関係を定義し，$n+1$ レベルシステムに n レベルシステムの厳密なモデルを持たせることは現実的に不可能であることを示し，それを踏まえて，n レベルシステムの変更部分のモデルのみを $n+1$ レベルシステムに持たせるといったアイディアに基づいてリフレクティブタワーを構築している．

プログラミング言語としての項書換えシステムに自己反映計算を導入した言語的枠組 REPS (Reflective Equational Programming System) [38][27] が提案されている．この枠組においては，リデックスの引数の各項，リデックスのまわりの文脈，および現在の書き換え規則の集合をメタオブジェクトとみなし，構成子項 (構成子のみから作られる項) をベースレベルオブジェクトとみなしている．この研究では，言語 L のインタプリタを言語 L 自身で記述し，インタプリタの無限階層を実現することによってリフレクティブタワーを構築し，そのタワーを上下に移動して計算を実行するというタイプの自己反映計算を採用していない．プログラムやその実行環境などのメタオブジェクトからユーザプログラムが参照・操作することができるベースレベルオブジェクトへの変換およびその逆変換をもって自己反映計算を実現している．

本論文の第3章では，計算対象を表現している項やそれらに関する書き換え規則の集合をベースレベルとみなし，それらをデータとして保持するような項やそれらに関する書き換え規則の集合をメタレベルとみなしている．よって REPS とはアプローチが異なっている．

5.2 並行計算と自己反映計算

永藤は並行計算モデルとして CCS[33] を考え，自己反映的並行計算の形式的意味について考察している [34]．CCS のより本質的な部分を抽出し，それを CCS のメタ言語として定義する．このようにして CCS の拡張を考えるとき，付加すべき機能は状態遷移の推論規則である．したがって，メタ言語には推論規則を定義できる機構が必要となる．この言語にレベルシフト演算子を追加することによって自己反映計算が実現される．

並行計算と自己反映計算に関連する研究としては，CCS に基づいた仕様記述言語 LOTOS に自己反映計算の概念を導入した RLOTOS が提案されている [42][43]．RLOTOS での自己反映計算の導入は，オブジェクトレベル (通常の計算を行なうレベル) とメタレベルに情報を分離して記述することにより，可読性の高い記述を行なうことを目的としている．

LOTOS による仕様は，動作に関する記述をおこなう部分と，データに関する記述をおこなう部分から構成される．前者はプロセス代数，後者は多ソート代数によって意味づけされている．

RLOTOS は LOTOS の動作に関する記述をおこなう部分に自己反映計算機能を導入している．メタレベル上には LOTOS のプロセスとして記述されたインタプリタが存在し，オブジェクトレベルの LOTOS 記述を解釈・実行する．メタレベル上のインタプリタは，オ

ブジェクトレベルの記述を LOTOS で扱うことのできる抽象データ型言語 ACTONE の項表現に変換したものを入力としている．オブジェクトレベルとメタレベルの動作は互いに対応付けられており，RLOTOS ではオブジェクトレベルの動作はメタレベル上のインタプリタの出力ゲートを通して出力される．

RLOTOS では，リフレクティブプロセスとリフレクティブゲートと呼ばれる特別なプロセスとゲートを用いてメタレベルへのアクセスを行ない，自己反映計算を実現している．リフレクティブプロセスからのインタプリタへのアクセスは，LOTOS の同期の概念を用いたプロセス間の通信によって行なうことができる．

以上の研究は，並行計算モデルを形式的に記述可能であるという点においては第 3章の研究に類似すると考えられる．しかしながらグループワイドアーキテクチャをモデル化している訳ではない．

5.3 書き換え論理と仕様記述

書き換え論理を仕様記述に応用したフィールド指向言語 Flage[26] が提案されている．Flage は協調型ソフトウェア，アーキテクチャを記述するための仕様記述言語である．協調型ソフトウェア，アーキテクチャは複数のエージェントから構成されているので，そのようなシステムの仕様は，Flage を用いてエージェントおよびエージェント間通信の仕様を規定したものとして提供される．ここでエージェントとは，オブジェクトに自律性を付加したものと考える．Flage では，各エージェントはメタ階層を持っており，これにより柔軟な処理の記述が可能となる．

Flage は複数の協調の範囲を表す場 (フィールド) の概念を有している．本言語ではシステムをエージェントの多重集合として捉え，エージェントの挙動を多重集合上の並行書き換えとして定式化している．

エージェント・モデルの記述の観点からいえば，Flage においてエージェントの自律性は，場への出入りによるメッセージの主体的な選択および，メッセージの受渡しをメタレベルで制御することによるメッセージの主体的な取捨選択によって実現されている．他のエージェントとの協調は場およびメタレベル間の通信によって実現できている．また，自分自身のモデルの記述とその操作はメタ構造によって可能となっている．

仕様変更が生じた時には，各エージェントにおいて自分自身のモデルとして自己形成プロセスを記述することにより，お互いの自己形成プロセスを融合，再利用することにより

仕様変化を満たすプログラムを合成する動作を記述することが可能になっている．

Flage は並行システム記述言語であり，同じく書き換え論理に基づいた仕様記述言語 Maude とは，メタ構造や場といった，協調システムの記述に必要と思われる概念を提供しているといった点が異なっている．

5.4 書き換え論理と自己反映計算

書き換え論理は自己反映的であることを示した研究がある [4][5][6]．

Tを書き換え定理，t, t'をTに含まれる項であるとし，

$$T \vdash t \longrightarrow t'$$

によって，

書き換え定理Tにおいて，$t \longrightarrow t'$が導出可能である

ことを表現しているものとする．さらに，$sen(T)$ によって，$t \longrightarrow t'$と表現されるシーケントの集合を表現する．

定理のクラス$\mathcal{C}$において我々の興味のあるものは，有限な要素から構成される表現可能な書き換え定理のクラスである．そこで$\mathcal{C}$-普遍的 $(\mathcal{C} - universal)$ であるような有限な要素から構成される表現可能な書き換え定理Uを構成する．具体的には，$T \in \mathcal{C}, \varphi \in sen(T)$ に対し

$$T \vdash \varphi \Leftrightarrow U \vdash \overline{T \vdash \varphi}$$

が成立するような表現関数

$$\overline{(_ \vdash _)} : \bigcup_{T \in \mathcal{C}} \{T\} \times sen(T) \rightarrow sen(U)$$

を定義する．このような表現関数を用いて構築した定理Uは$\mathcal{C}$-普遍的 $(\mathcal{C} - universal)$ であるという．さらに，$U \in \mathcal{C}$のとき，Uは$\mathcal{C}$-自己反映的 $(\mathcal{C} - reflective)$ であるという．

先に定義した同値な関係

$$T \vdash \varphi \Leftrightarrow U \vdash \overline{T \vdash \varphi}$$

は次のように拡張することができる．項 φ を $t \longrightarrow t'$ とする．このとき，U の項 $\overline{T \vdash \varphi}$ に $\overline{T}@\,\overline{t} \longrightarrow \overline{T}@\,\overline{t'}$ を対応させる．ここで “@” は U の二項演算子とする．したがって，先の

同値な関係は

$$T \vdash t \longrightarrow t' \Leftrightarrow U \vdash \overline{T}@\overline{t} \longrightarrow \overline{T}@\overline{t'}$$

へと拡張することが可能である．

このことにより，通常の計算 (オブジェクトレベル) を書き換え論理に基づいてモデル化し，上記にあげた条件をみたすような表現関数を構成し，メタレベルシステムに相当する書き換え定理を構築すれば，自己反映的なモデルを構築することが可能である．さらに，メタレベルにおいて，オブジェクトレベルのメタレベル表現を操作することが可能であるので，オブジェクトレベルの書き換え規則の適用順序を操作するようなメタレベルの書き換え規則を提供することが可能となる．

第 6 章

結論および今後の課題

本章では，第 3，4章の研究成果に関する，結論および今後の課題について言及する．

6.1 結論

6.1.1 グループワイド アーキテクチャ関連

第 3章では，アクターモデルによるグループワイドアーキテクチャのモデル化を基礎とし，それを書き換え論理に基づいてモデル化を行なった．グループワイドアーキテクチャのモデル化は様々な方法が考えられるが，本研究ではアクターモデルに基づくモデル化を参考にしている．

第 1章でも言及したように，リフレクティブタワーの仕様を記述するのは困難であるので，本研究ではメタレベルシステムの操作的意味を与えている．メタレベルシステムにはベースレベルの情報がメタレベルシステムのデータとして表現されており，メタレベルシステムの操作的意味の定義はベースレベルのメタレベル実行を実現することになる．

書き換え論理に基づくモデル化では，メタレベルシステムはオブジェクトとメッセージの集合で構成されている．メタレベルシステムの操作的意味は，想定しうるメタレベルシステムのコンフィギュレーション (メタコンフィギュレーション) すべてに対して定義した書き換え規則によって与えられている．また，オブジェクト移送を行なうための書き換え規則を定義することによって，メタレベルシステム間のデータの移動をシミュレートすることができた．さらに，我々が与えた操作的意味記述に基づいてメタレベルシステムの正

当性を証明した．ここで，メタレベルシステムが正当であるとは，

- ベースレベルシステムの一回の遷移 (書き換え) に対し，それをシミュレートするようなメタレベルシステムの (複数回の) 遷移が存在し，
- ベースレベルシステムのメタレベル表現が変化するようなメタレベルシステムでの遷移に対し，それに対応するベースレベルシステムでの一回の遷移が存在する

ことを意味する．このように，操作的意味を与えることによって，システムが持つ性質の解析が可能になることがわかった．

6.1.2 GAEA 関連

第 4章では，有機的プログラミング言語 GAEA の概要を自律ロボットの例題を用いて説明し，言語の操作的意味を実行可能な処理系を持つ仕様記述言語 CafeOBJ および Maude を用いて書き換え論理に基づいて記述した．ただし，本論文では Maude による記述を用いて説明を行っている．その記述を基に，待ち状態，同期処理機構，他のプロセスへの作用，およびプロセスの環境変化後のバックトラック処理に関する考察を行った．

待ち状態や同期処理機構については，それぞれシステム内にセルが存在していない，セル変数にデフォルト値以外の値が割り当てられている，といった条件が満たされている場合に計算状態を遷移させない機構が必要である．これは，前述の条件を満たす場合に適用可能な状態遷移規則を定義しないことによって実現できた．

他のプロセスの環境を操作する組み込み述語に対応する状態遷移規則を定義できた最大の要因は計算状態を表現するための項の設計，特にプロセスを表現する項の設計であった．プロセス内部の計算を表現する項とそれ以外の項を区別することによって，前者の項を書き換えてしまう危険を回避している．

プロセスの環境変化後のバックトラック処理についても，項の設計とバックトラック情報を明示したことによって 4.4.4節で言及したような考察が可能となった．

以上のことから，書き換え論理に基づいた操作的意味記述によって有機的プログラミング言語 GAEA の

- 複雑な動作が理解しやすくなった
- データ構造に基づく言語仕様についての議論が可能になった

といえる．これらの理由によって，我々のアプローチに基づく操作的意味記述は，システムの機能拡張，仕様変更，類似システムのプロトタイプ作成，実行可能な処理系を利用した実験を行うための支援的ツールになる．また，システムを実際に使用することなくシステムを理解するための手段を提供している．

6.2 今後の課題

6.2.1 グループワイドアーキテクチャ関連

第3章においては，

- グループワイドアーキテクチャの新たなモデル化
- リフレクティブタワーの操作的意味記述
- 新たな例題
- システムの性質の解析

などが今後の課題として挙げられる．本研究はアクターモデルに基づくグループワイドアーキテクチャのモデル化を基礎としているが，他の体系に基づくモデル化を試みることにより，グループワイドアーキテクチャの普遍的な性質が抽出できると予想される．

本研究ではリフレクティブタワーはベースレベル，メタレベルの2階層であると仮定し，そのうちのメタレベルシステムの操作的意味についてのみ記述している．自己反映計算の操作的意味を与えるという観点からすると，メタレベルシステムのみの意味定義では不十分であると考えられる．

オブジェクト移送以外の例題を記述し実験をおこない，我々のモデル化手法の妥当性を確認する必要がある．多くの例題を記述することにより，我々が与えた操作的意味の長所，短所を発見することが可能となる．

並行自己反映計算の操作的意味を与えることの第一の目的は，その計算メカニズムの性質を解析することにある．本研究では，メタレベルシステムの正当性についてのみ考察したが，さらなる性質の解析が期待される．

6.2.2 GAEA 関連

第 4章では有機的プログラミング言語 GAEA の一部の機能についてのみ操作的意味を与えた．これは GAEA のインタプリタの一部分を定義したことに相当する．本研究では GAEA の特徴のひとつである “@”-infon と呼ばれる，部分的に特定可能な項の扱いに関しては考慮していない．また，操作的意味記述を利用したシステムの性質については，2，3 の例について考察しているが．それでは充分とは言えない．したがって GAEA の操作的意味を与えるという観点からは，

- 言語全体の操作的意味
- システムの性質の解析

といった点が課題としてあげられる．言語全体の操作的意味を与え，それを GAEA システムマニュアルに付録として添付することも考慮している．言語の正確な意味定義を提供することには

- 言語のより深い理解
- 保守・拡張の利便性の向上
- 類似の言語のプロトタイプ作成のための有用な情報

といった利点があると考えられるからである．

6.2.1節も含めた，一般的な並行自己反映計算の操作的意味を記述するという観点からは，

- モデル化手法の妥当性の確認
- その他の並行自己反映計算システムの操作的意味の記述
- システムの持つ性質の考察，解析

などが課題としてあげられる．

モデル化手法の妥当性の確認に関しては，データ構造や状態遷移規則の改良が必要になると考えられる．例えば，同期処理機構をより正確に表現するためには時間の概念が必要となる．したがって，時間の概念を表現するデータ構造が必要になる．状態遷移規則については，条件付き状態遷移規則をできる限りなくすことが望まれる．状態遷移規則の左辺

は，計算システムが取り得る状態の一部分を規定したものである．条件付き状態遷移規則の条件を左辺に埋め込んでしまうことにより，状態遷移規則の集合は計算システムが取り得る状態の集合とみなすことが可能となる．状態遷移規則によって計算システムの取り得る状態がすべて表現されているので，システムを実行することなくその挙動を推測することが可能となる．条件付き状態遷移規則の状態を取り除く例として GAEA の組み込み述語 cv-write(Cell,CV,V) の場合を考える．この述語に対応する状態遷移規則は

```
crl
state(cell(Cell,CVS,CLS),
      process(P,Env,Meta,
              termlist(pred('cv-write,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,Env,loc(1,1),GL,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .
```

図 6.1: 組み込み述語cv-write/3 に対応する条件付き状態遷移規則

と与えている．ここで，セル変数とその値の組を構成する構成子cvpair を図 6.2に示すように拡張し，セル変数に値が割り当てられているかどうかを明示するよう定義すると，図 6.1でみられた条件部分が状態遷移規則の左辺に埋め込むことが可能となる．

```
--- 新たなソートの宣言
sort Flag .
--- フラグ付きのセル変数とその値の組
op cvpair : Qid Term Flag -> CVPair .
--- フラグ付きのセル変数とその値の組のリスト
op cvs : CVPairList CVPairList -> CVPairList [ assoc comm ] .
--- セル変数に値が割り当てられている場合のフラグ
op assigned : -> Flag .
--- セル変数に値が割り当てられていない (デフォルト値の) 場合のフラグ
op unassigned : -> Flag .
```

```
rl
state(cell(Cell,cvs(cvpair(CV,undef,unassigned),CVS),CLS),
      process(P,Env,Meta,
              termlist(pred('cv-write,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,cvs(cvpair(CV,V,assigned),CVS),CLS),
      process(P,Env,loc(1,1),GL,Ctrls)) .
```

図 6.2: 組み込み述語cv-write/3 に対応する条件のない状態遷移規則

このようにデータ構造や状態遷移規則の改良を行うことによって，計算システムの状態がより理解しやすくなり，構造帰納法を利用した計算システムの解析が容易となると期待される．また，実行可能な処理系を用いて，システムが持つ性質の検証の自動化も容易に行なえると考えられる．

その他の並行自己反映計算システムの操作的意味の記述を行うことで，並行自己反映計算システムの一般的な性質を抽出することができると予想される．

操作的意味記述に基づいてシステムの持つ性質の考察，解析を行うことにより，システムの仕様変更，拡張，議論，より深い理解のための手段の提供が可能となる．

以上に述べた課題を解決することにより，一般的な並行自己反映計算の操作的意味の提供，システムの持つ性質の解析，より信頼性の高い仕様を提供することが可能となる．

謝辞

本研究をおこなうにあたり，終始ご指導していただいた情報科学研究科情報システム学専攻言語設計学講座 二木 厚吉博士，同講座助教授渡部 卓雄博士に深く感謝いたします．

本論文の審査委員である，情報科学研究科情報システム学専攻ソフトウエア基礎講座教授 片山 卓也博士，情報科学研究科情報処理学専攻計算機言語学講座教授 外山 芳人博士，電子技術総合研究所協調アーキテクチャ計画室室長 中島 秀之博士からは多くのご指導，ご助言をいただきました．ここに感謝の意を表します．

有機的プログラミング言語 GAEA の操作的意味の記述に関して，GAEA 開発メンバーである中島博士および電子技術総合研究所半田 剣一博士，野田 五十樹博士から有益なご助言をいただきました．ありがとうございました．

日頃，議論につきあってくださった言語設計学講座助手 緒方 和博博士および言語設計学講座の諸氏に感謝致します．

1997 年 8 月 8 日から 9 月 28 日まで，SRI(Stanford Research Institute) International において International Fellow として滞在，研究をおこなうという貴重な体験をすることができました．そのような機会を与えてくださった情報処理振興事業協会，SRI International において，御指導くださった José Meseguer 博士および議論につきあってくださった Manuel Clavel 氏に感謝致します．

最後に，この 5 年間終始励ましてくれた友人達，親類，両親に心から感謝致します．

参考文献

[1] GAEA *Version 1.1 Manual Revision 0*, Cooperative Architecture Project Team, ETL, 1996.

[2] Gul Agha. *Actors : A Model of Concurrent Computation in Distributed Systems*. The MIT Press, 1987.

[3] 有川 節夫, 原口 誠述語論理と論理プログラミング. 知識工学講座 *4*, オーム社, 1988.

[4] Manuel Clavel and José Meseguer. Axiomatizing Reflective Logics and Languages. In *Reflection 96*, 1996.

[5] Manuel Clavel, Steven Eker, Patric Lincoln, and José Meseguer. Princeple of Maude. In José Meseguer, editor, *Proceedings of the First International Workshop on Rewriting Logic*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pp. 65–89. Elsevier, September 1996.

[6] Manuel Clavel and José Meseguer. Reflection and Strategies in Rewriting Logic. In *Electronic Notes in Theoretical Computer Science, volume 4*, 1996.

[7] 二木 厚吉. チュートリアル 代数モデルの基礎. コンピュータソフトウェア, Vol.13, No.1, pages 4–22, 1996.

[8] 二木 厚吉, 外山 芳人. 項書き換え型計算モデルとその応用. 情報処理, Vol.24, No.2, pages 133–146, 1983.

[9] Joseph Goguen and Grant Malcolm. Hidden Agenda. UCSD Technical Report CS97-538, April, 1997. `http://www.cse.ucsd.edu/users/goguen/ps/ha.ps.gz`.

[10] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsuki, and Jean-Pierre Jouannaud. Introducing OBJ. Technical Report, SRI–CSL–92–03, 1992.

[11] Paul Hudack, Philip Wadler, et al. Report on the programming language Haskell, a non-strict purely functional language (Version 1.2). *Technical report*, Yale University / Glasgow University, 1992.

[12] 井田 哲雄. 計算モデルの基礎理論. 岩波講座 ソフトウェア科学 *12*, 岩波書店, 1991.

[13] 稲垣 康善, 坂部 俊樹.— 抽象データタイプの代数的仕様記述法の基礎 (1) — In 情報処理, Vol.25, No.1, page 47–53, 情報処理学会, 1984.

[14] 稲垣 康善, 坂部 俊樹.— 抽象データタイプの代数的仕様記述法の基礎 (2) — In 情報処理, Vol.25, No.5, page 491–501, 情報処理学会, 1984.

[15] 稲垣 康善, 坂部 俊樹.— 抽象データタイプの代数的仕様記述法の基礎 (3) — In 情報処理, Vol.25, No.7, page 708–716, 情報処理学会, 1984.

[16] 稲垣 康善, 坂部 俊樹.— 抽象データタイプの代数的仕様記述法の基礎 (4) — In 情報処理, Vol.25, No.9, page 971–986, 情報処理学会, 1984.

[17] 石川 洋, 二木 厚吉, 渡部 卓雄. 書き換え論理に基づく並行自己反映計算のモデル化. 日本ソフトウェア科学会第 *11* 回大会論文集, pages 313–316. 日本ソフトウェア科学会, 1994.

[18] 石川 洋. 書き換え論理に基づく並行自己反映計算のモデル化. 修士論文, 北陸先端科学技術大学院大学, 1995.

[19] Hiroshi Ishikawa, Kokichi Futatsugi, and Takuo Watanabe. Concurrent Reflective Computations in Rewriting Logic. *RIMS Workshop on Theory of Rewriting Systems and its Applications, Research Institute of Mathematical Science*, In RIMS Kokyuroku 918, pp. 292–298, Kyoto, July,1995.

[20] 石川 洋, 二木 厚吉, 渡部 卓雄. 書き換え論理に基づく並行自己反映計算について. 日本ソフトウェア科学会第 *12* 回大会論文集, pages 49–52. 日本ソフトウェア科学会, 1995.

[21] Hiroshi Ishikawa, Kokichi Futatsugi, and Takuo Watanabe. An Example for Concurrent Reflective Computations in Rewriting Logic. In Proceedings of 1st IFIP International workshop on Formal Methods for Open Object-based Distributed Systems, pp. 173–187, 1996.

[22] 石川 洋, 二木 厚吉, 渡部 卓雄. 書き換え論理に基づく並行自己反映計算のモデル化の一例. 第 *12* 回オブジェクト指向計算ワークショップ *WOOC'96*, 1996.

[23] 石川 洋, 二木 厚吉, 渡部 卓雄. 並行自己反映計算の宣言的記述. 日本ソフトウェア科学会第 *14* 回大会論文集, pages 205–208. 日本ソフトウェア科学会, 1997.

[24] Hiroshi Ishikawa, José Meseguer, Takuo Watanabe, Kokichi Futatsugi, and Hideyuki Nakashima. On the semantics of GAEA — An Object-Oriented Specification of a Concurrent Reflective Language in Rewriting Logic —, In *Proceedings of the International Symposium on New Models for Software Architecutre '97*, pp. 70–109, 1997.

[25] Hiroshi Ishikawa, Takuo Watanabe, Kokichi Futatsugi, José Meseguer, and Hideyuki Nakashima. On the semantics of GAEA, To Appear in *Proceedings of the Third Fuji International Symposium on Functional and Logic Programming*, April, 1998.

[26] 粂野 文洋，田原 康之，大須賀 昭彦，本位田 真一. フィールド指向言語 Flage. In 第 *1* 回ソフトウェア工学の基礎ワークショップ *(FOSE'94)*, 1994, pp.25–32.

[27] 栗原 正仁，佐藤 崇昭，大内 東. 項書換えシステムにおける自己反映計算. In コンピュータソフトウェア, Vol.12,No.4, 1995, pp.3–14.

[28] Pattie Maes. Computational reflection. Ph. D. Thesis 87-2, Artificial Intelligence Laboratory, Vrije Universiteit Brussel, 1987.

[29] José Meseguer. Rewriting as a unified model of concurrency. In J.C.M.Baeten and J.W.Klop eds., *Proc. CONCUR'90*, LNCS 458, Springer-Verlag, pp.384–400,1990.

[30] José Meseguer. Conditional rewriting logic as a unified model of concurrency. Technical Report SRI-CSL-92-08, Computer Science Laboratory, SRI International, 1992.

[31] José Meseguer, Kokichi Futatsugi, and Timothy Winkler. Using Rewriting Logic to Specify, Program, Integrate, and Reuse Open Concurrent Systems of Cooperating Agents. Technical Report SRI-CSL-92-11, Computer Science Laboratory, SRI International, 1992.

[32] José Meseguer. A logical theory of concurrent objects and its realization in the maude language. In Peter Wegner Gul Agha and Akinori Yonezawa, editors, *Research Directions in Concurrent Object Oriented Programming*. The MIT Press, 1993.

[33] Robin Milner. Communication and Concurrency. Prentice-Hall, 1989.

[34] 永藤 直行. 自己反映的並行計算の操作的意味について. 修士論文, 北陸先端科学技術大学院大学, 1996.

[35] Ataru T. Nakagawa, Toshimi Sawada, and Kokichi Futatsugi. CafeOBJ *User's Manual — ver.1.3 —*, 1997.

[36] 中島 秀之, 野田 五十樹, 半田 剣一. 有機的プログラミング言語 GAEA の設計と実装. 日本ソフトウェア科学会第 *13* 回大会論文集, pages 225–228. 日本ソフトウェア科学会, 1996.

[37] Hideyuki Nakashima. Organic Programming for Cooperative Computation, In *Proceedings of the World-Wide Computing and its Applications (WWCA'97)*,B-1-3-1–8, 1997.

[38] 佐藤 崇昭, 栗原 正仁, 大内 東. 自己反映計算機能をもつ等式プログラム処理系の実現. 電子情報通信学会技術研究報告 コンピュテーション *COMP92-85*〜*92*, pages 69–76. 電子情報通信学会, 1993. (信学技報 COMP92-92).

[39] Brian Cantwell Smith. Reflection and semantics in Lisp. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*, pages 23–35, 1984.

[40] Leon Sterling and Ehud Shapiro. *The Art of Prolog, second edition*, The MIT Press, 1994.

[41] 所 真理雄, 松岡 聡, 垂水浩幸編. オブジェクト指向コンピューティング. 岩波コンピュータサイエンス. 岩波書店, 1993.

[42] 鵜飼 孝典, 広井 武, 佐伯 元司. LOTOS におけるリフレクティブ計算について. In 日本ソフトウェア科学会第 *8* 回大会論文集, 1991, pp.537–540.

[43] 鵜飼 孝典, 広井 武, 佐伯 元司. 仕様記述言語 LOTOS におけるリフレクション : RLOTOS. In 情報処理学会研究報告会, Vol.92, No.20,1992, pp.1–10.

[44] Takuo Watanabe, Akinori Yonezawa. Reflection in an Object-Oriented Concurrent Language. In *ABCL:An Object-Oriented Concurrent System*, pages 45–70. The MIT Press, 1990.

[45] Takuo Watanabe, Akinori Yonezawa. An Actor-Based Metalevel Architecture for Group-Wide Reflection. In *Proceedings of REX School/Workshop on Foundations of Object-Oriented Languages(REX/FOOL)*, Lecture Notes in Computer Science 489, pages 405–425, New York, 1991. Springer-Verlag.

[46] 渡部 卓雄. リフレクション. コンピュータソフトウェア, 11(3):pages 5–14, May 1994.

[47] 渡部 卓雄, 増原 英彦, 松岡 聡. 自己反映並列オブジェクト指向言語 ABCL/R2 の設計と実現. コンピュータソフトウェア, 11(3):pages 15–32, May 1994.

[48] Takuo Watanabe, Hiroshi Ishikawa, and Kokichi Futatsugi. Towards Declarative Description of Computational Reflection. In *Proceedings of the International Symposium on New Models for Software Architecutre '96*, pp. 113–128, 1996.

[49] Glynn Winskel. *The Formal Semantics for Programming Languages*. The MIT Press, 1993.

[50] 山岡 順一, 渡部 卓雄. 自己反映的な値呼びλ-計算の操作的意味論. In 情報処理学会研究報告 *(*プログラミング – 言語・基礎・実践– 研究会*)*, 95-PRG-21,page 49–56. 1995.

[51] 米澤明憲, 柴山悦哉. モデルと表現. 岩波書店, 1992.

本研究に関する発表論文

[1] 石川 洋，二木 厚吉，渡部 卓雄: “書換え論理に基づく並行自己反映計算のモデル化”，日本ソフトウェア科学会第 11 回大会論文集，pp.313–316, 1994 年 11 月.

[2] H, Ishikawa, K. Futatsugi, T, Watanabe: “Concurrent Reflective Computations in Rewriting Logic”, Theory of Rewriting System and its Applications, RIMS Kokyuroku 918, pp.292–298, July, 1995.

[3] 石川 洋，二木 厚吉，渡部 卓雄: “書換え論理に基づく並行自己反映計算について”，日本ソフトウェア科学会第 12 回大会論文集，pp.49–52，1995 年 9 月.

[4] H, Ishikawa, K. Futatsugi, T, Watanabe: “An Example for Concurrent Reflective Computations in Rewriting Logic”, Formal Methods for Open Object-based Distributed Systems, pp.178–185, March, 1996.

[5] 石川 洋，二木 厚吉，渡部 卓雄: “書換え論理に基づく並行自己反映計算の一例”，第 12 回オブジェクト指向計算ワークショップ，1996 年 3 月.

[6] T, Watanabe, H, Ishikawa, K. Futatsugi: “Towards Declarative Description of Computational Reflection”, Proceedings of the International Symposium on New Models for Software Architecture’96, pp.113–128, December, 1996.

[7] 石川 洋，二木 厚吉，渡部 卓雄: “並行自己反映計算の宣言的記述”，日本ソフトウェア科学会第 14 回大会論文集，pp.205–208，1997 年 10 月.

[8] H. Ishikawa, J. Meseguer, T. Watanabe, K. Futatugi, H. Nakashima: “On the semantics of GAEA — An Object-Oriented Specification of a Concurrent Reflective

Language in Rewriting Logic —", Proceedings of the International Symposium on New Models for Software Architecture'97, pp.70–109, October, 1997.

[9] H. Ishikawa, T. Watanabe, K. Futatugi, J. Meseguer, H. Nakashima: "On the semantics of GAEA", To appear in Proceedings of The Third Fuji International Symposium on Functional and Logic Programming, April, 1998.

Appendix A: GAEA の操作的意味記述

4.2.5節で述べたGAEAの操作的意味記述を掲載する．仕様記述言語Maudeは SRI(Stanford Research Institute) International で開発中の言語である．本研究で使用した版は alpha 27 であり，現状では，結合的および単位元を持つオペレータを宣言することができない．そのため，結合的および単位元を持つオペレータを定義する場合は，結合的オペレータ宣言に加え，単位元を引数としてとる場合の処理を等式として与えることで対応している．Maudeの文法については，4.2.1節を参照されたい．

```
--- ---------------------------------------------------------------------
--- Maude による GAEA の操作的意味定義
---
---   使用した Muade のバージョン
---     Maude term rewriting engine: alpha 27
---     This executable built on Sep  3 1997 at 19:29:07
--- ---------------------------------------------------------------------

--- モジュール定義
--- 現在の Maude では 1 モジュール内にすべての定義を記述する．
mod GAEA is
  --- 組み込みモジュールの輸入
  protecting QID .
  protecting MACHINE-INT .

  --- 項の定義
  sorts Term NeTermList TermList .
  subsort Qid < Term .
  subsort MachineInt < Term .
  subsort Term < NeTermList < TermList .

  op nil       : -> TermList .
  op True      : -> Term .
  op !         : -> Term .
  op fail      : -> Term .
  op undef     : -> Term .
  op termlist  : TermList TermList -> TermList [ assoc ] .
```

```
op termlist  : NeTermList TermList -> NeTermList .
op length    : TermList -> MachineInt .
op pred      : Qid TermList -> Term .
op belongsto : Term TermList -> Bool .
op and : Bool Bool -> Bool [ assoc ] .

vars T B : Term .
vars TL TL' : TermList .

--- 項のリストが結合的かつ単位元を持つように定義
eq termlist(nil,TL) = TL .
eq termlist(TL,nil) = TL .

--- ユーテリティの定義
eq length(nil) = 0 .
eq length(T) = 1 .
eq length(termlist(T,TL)) = 1 + length(TL) .

eq belongsto(!,TL) = true .
eq belongsto(fail,TL) = true .
eq belongsto(pred(T,TL),nil) = false .
eq belongsto(pred(T,TL),B)
   = if T == B then true else false fi .
eq belongsto(pred(T,TL),termlist(B,TL'))
   = if T == B then true else belongsto(pred(T,TL),TL') fi .

eq and(true,false) = false .
eq and(false,true) = false .
eq and(false,false) = false .
eq and(true,true) = true .

--- ユニフィケーション (ここではパターンマッチ) のための
--- ソートおよびシグネチャの定義
sorts Eqn NeSystem System .
subsort Eqn < NeSystem < System .
op null : -> System .
op && : System System -> System [ assoc ] .
op eqn : Term Term -> Eqn .
op sys : System -> System .
op eqn : TermList TermList -> System .
op syslen : System -> MachineInt .

vars U V : Term .
var UL : NeTermList .
var S : System .
var Sn : NeSystem .
var Eq : Eqn .
vars M N : MachineInt .
```

```
--- オペレータ && が結合的かつ単位元を持つように定義
eq &&(null,S) = S .
eq &&(S,null) = S .

--- オペレータ && および eqn に関する等式の定義
eq eqn(termlist(T,TL),termlist(U,UL))
   = &&(eqn(T,U),eqn(TL,UL)) .

eq syslen(Eq) = 1 .
eq syslen(&&(Eq,S)) = 1 + syslen(S) .

--- オペレータ let-be-in-* の定義
op let-be-in-term     : Qid Term Term     -> Term .
op let-be-in-termlist : Qid Term TermList -> TermList .
op let-be-in-eqn      : Qid Term Eqn      -> Eqn .
op let-be-in-system   : Qid Term System   -> System .

vars F X Y : Qid .

eq let-be-in-termlist(X,T,nil) = nil .

eq let-be-in-term(X,T,Y) = if X == Y then T else Y fi .
eq let-be-in-term(X,T,B) = if X == B then T else B fi .
eq let-be-in-term(X,T,pred(F,TL))
   = if length(TL) == 1
     then pred(F,let-be-in-term(X,T,TL))
     else pred(F,let-be-in-termlist(X,T,TL))
     fi .

eq let-be-in-termlist(X,T,termlist(U,UL))
   = if length(UL) == 1
     then termlist(let-be-in-term(X,T,U),let-be-in-term(X,T,UL ))
     else termlist(let-be-in-term(X,T,U),let-be-in-termlist(X,T,UL ))
     fi .

eq let-be-in-eqn(X,T,eqn(U,V))
   = eqn(let-be-in-term(X,T,U),let-be-in-term(X,T,V)) .

eq let-be-in-system(X,T,null) = null .
eq let-be-in-system(X,T,&&(eqn(U,V),Sn))
   = if syslen(Sn) == 1
     then &&(let-be-in-eqn(X,T,eqn(U,V)),let-be-in-eqn(X,T,Sn))
     else &&(let-be-in-eqn(X,T,eqn(U,V)),let-be-in-system(X,T,Sn))
     fi .

--- ユニフィケーションの定義
op unify : System -> System .
op failure : -> Eqn .
```

```
vars Tn Un : TermList .
vars S' S'' : System .
vars G G' : Qid .

eq unify(S) = sys(sys(S)) .

eq &&(S,eqn(T,T)) = S .
eq &&(eqn(T,T),S) = S .

eq &&(S,failure) = failure .
eq &&(failure,S) = failure .

eq let-be-in-system(X,T,failure) = failure .
eq sys(eqn(T,T)) = null .
eq sys(null) = null .
eq sys(failure) = failure .
eq eqn(T,!) = failure .
eq eqn(T,fail) = failure .

--- patch for pred('name,'M) -------------------------------
eq sys(eqn(pred('name,T),pred('name,B)))
   = sys(eqn(B,T)) .

eq eqn(M,N) = if M == N then null else failure fi .
--- patch for pred('name,'M) -------------------------------

eq sys(eqn(pred(F,Tn),pred(G,Un)))
   = if F == G
     then if length(Tn) == length(Un)
          then sys(eqn(Tn,Un))
          else failure fi
     else failure fi .

eq sys(&&(S,eqn(pred(F,Tn),pred(G,Un))))
   = if F == G
     then if length(Tn) == length(Un)
          then sys(&&(S,eqn(Tn,Un)))
          else failure fi
     else failure fi .

eq sys(&&(eqn(pred(F,Tn),pred(G,Un)),S'))
   = if F == G
     then if length(Tn) == length(Un)
          then sys(&&(eqn(Tn,Un),S'))
          else failure fi
     else failure fi .

--- sys の引数による場合分けを行い，個々について交換性の定義を行う
--- commutative for eqn 1
```

```
eq sys(sys(eqn(X,T)))
   = if X == T
     then sys(null)
     else sys(eqn(X,T))
     fi .
--- commutative for eqn 2
eq sys(sys(eqn(T,X)))
   = if X == T
     then sys(null)
     else sys(eqn(X,T))
     fi .

--- commutative for eqn 1
eq sys(sys(&&(S,eqn(X,T))))
   = if X == T
     then sys(S)
     else if syslen(S) == 1
          then sys(&&(eqn(X,T),sys(let-be-in-eqn(X,T,S))))
          else sys(&&(eqn(X,T),sys(let-be-in-system(X,T,S))))
          fi
     fi .
--- commutative for eqn 2
eq sys(sys(&&(S,eqn(T,X))))
   = if X == T
     then sys(S)
     else if syslen(S) == 1
          then sys(&&(eqn(X,T),sys(let-be-in-eqn(X,T,S))))
          else sys(&&(eqn(X,T),sys(let-be-in-system(X,T,S))))
          fi
     fi .

--- commutative for eqn 1
eq sys(sys(&&(eqn(X,T),S)))
   = if X == T
    then sys(S)
     else if syslen(S) == 1
          then sys(&&(eqn(X,T),sys(let-be-in-eqn(X,T,S))))
          else sys(&&(eqn(X,T),sys(let-be-in-system(X,T,S))))
          fi
     fi .
--- commutative for eqn 2
eq sys(sys(&&(eqn(T,X),S)))
   = if X == T
     then sys(S)
     else if syslen(S) == 1
          then sys(&&(eqn(X,T),sys(let-be-in-eqn(X,T,S))))
          else sys(&&(eqn(X,T),sys(let-be-in-system(X,T,S))))
          fi
     fi .
```

```
--- commutative for eqn 1
eq sys(&&(S,sys(eqn(X,T))))
   = if X == T
     then sys(S)
     else if syslen(S) == 1
          then sys(&&(eqn(X,T),let-be-in-eqn(X,T,S)))
          else sys(&&(eqn(X,T),let-be-in-system(X,T,S)))
          fi
     fi .
--- commutative for eqn 2
eq sys(&&(S,sys(eqn(T,X))))
   = if X == T
     then sys(S)
     else if syslen(S) == 1
          then sys(&&(eqn(X,T),let-be-in-eqn(X,T,S)))
          else sys(&&(eqn(X,T),let-be-in-system(X,T,S)))
          fi
     fi .

--- commutative for eqn 1
eq sys(&&(S,sys(&&(S',eqn(X,T)))))
   = if X == T
     then sys(&&(S,S'))
     else if syslen(S) == 1
          then if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys(let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys(let-be-in-system(X,T,S'))))
               fi
          else if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys(let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys( let-be-in-system(X,T,S'))))
               fi
          fi
     fi .
--- commutative for eqn 2
eq sys(&&(S,sys(&&(S',eqn(T,X)))))
   = if X == T
     then sys(&&(S,S'))
     else if syslen(S) == 1
          then if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys(let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys(let-be-in-system(X,T,S'))))
```

```
               fi
          else if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys(let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys(let-be-in-system(X,T,S'))))
               fi
          fi
     fi .

--- commutative for eqn 1
eq sys(&&(S,sys(&&(eqn(X,T),S'))))
   = if X == T
     then sys(&&(S,S'))
     else if syslen(S) == 1
          then if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys( let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys(let-be-in-system(X,T,S'))))
               fi
          else if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys(let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys(let-be-in-system(X,T,S'))))
               fi
          fi
     fi .
--- commutative for eqn 2
eq sys(&&(S,sys(&&(eqn(T,X),S'))))
   = if X == T
     then sys(&&(S,S'))
     else if syslen(S) == 1
          then if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys(let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-eqn(X,T,S),
                           sys(let-be-in-system(X,T,S'))))
               fi
          else if syslen(S') == 1
               then sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys(let-be-in-eqn(X,T,S'))))
               else sys(&&(eqn(X,T),let-be-in-system(X,T,S),
                           sys( let-be-in-system(X,T,S'))))
              fi
          fi
     fi .
--- end unify -------------------------------------------------------
```

```
--- subst 変数に束縛値を代入する関数の定義
op subst : System TermList -> TermList .
op substmain : Eqn TermList -> TermList .

eq subst(null,TL) = TL .
eq subst(eqn(X,T),TL) = substmain(eqn(X,T),TL) .
eq subst(sys(eqn( X,T)),TL) = substmain(eqn(X,T),TL) .
eq subst(sys(&&(eqn(X,T),S)),TL) = subst(S,substmain(eqn(X,T),TL)) .

eq substmain(eqn(X,T),nil) = nil .
eq substmain(eqn(X,T),pred(F,TL)) = pred(F,substmain(eqn(X,T),TL)) .
eq substmain(eqn(X,T),termlist(pred(F,TL),TL'))
   = termlist(pred(F,substmain(eqn(X,T),TL)),
              substmain(eqn(X,T),TL')) .

eq substmain(eqn(X,T),U) = if X == U then T else U fi .
eq substmain(eqn(X,T),termlist(U,TL))
   = if X == U
     then termlist(T,substmain(eqn(X,T),TL))
     else termlist(U,substmain(eqn(X,T),TL))
     fi .

--- 節定義および節定義リストの定義
sorts Clause ClauseList .
subsort Clause < ClauseList .

op NoClause : -> ClauseList .
op cldef : Term TermList TermList -> Clause .
op clauselist : ClauseList ClauseList -> ClauseList [ assoc ] .
  --- GAEA では節定義方法は 3 種類ある.
  --- _:_ is represented by means of cldef(Head,nil,Body) .
  --- _:=_ is represented by means of cldef(Head,nil,termlist(!,Body)) .
  --- _:_:=_ is represented by means of
  ---         cldef(Head,Cond,termlist(!,Body)) .

op head-part : Clause -> Term .
op body-part : Clause -> TermList .

op nthcl : MachineInt ClauseList -> Clause .

op out-of-range-c : -> Clause .

op cell-exists : Term ClauseList -> Bool .
op process-exists : Term ClauseList -> Bool .
op process-delete : Term ClauseList -> ClauseList .

var Cl : Clause .
var CLS : ClauseList .
```

```
var T' : Term .

--- clauselist を結合的かつ単位元を持つように定義
eq clauselist(NoClause,CLS) = CLS .
eq clauselist(CLS,NoClause) = CLS .

eq head-part(cldef(T,TL,TL')) = T .

eq body-part(cldef(T,nil,TL)) = TL .
eq body-part(cldef(T,TL,TL')) = termlist(TL,TL') .

cq nthcl(M,CLS) =  out-of-range-c if M < 1 .
eq nthcl(M,NoClause) = out-of-range-c .
eq nthcl(M,Cl) = if M == 1 then Cl else out-of-range-c fi .
eq nthcl(M,clauselist(Cl,CLS))
   = if M == 1 then Cl else nthcl(M - 1,CLS) fi .

--- セル system_cell 中にあるセル名の存在確認
eq cell-exists(T,NoClause) = false .
eq cell-exists(T,cldef(pred(X,T'),TL,TL'))
   = if X == 'cell
     then if T == T'
          then true else false fi
     else false
     fi .
eq cell-exists(T,clauselist(cldef(pred(X,T'),TL,TL'),CLS))
   = if X == 'cell
     then if T == T'
          then true
          else cell-exists(T,CLS)
          fi
     else cell-exists(T,CLS)
     fi .

--- セル system_cell 中にあるプロセス名の存在確認
eq process-exists(T,NoClause) = false .
eq process-exists(T,cldef(pred(X,T'),TL,TL'))
   = if X == 'process
     then if T == T'
          then true else false fi
     else false
     fi .
eq process-exists(T,clauselist(cldef(pred(X,T'),TL,TL'),CLS))
   = if X == 'process
     then if T == T'
          then true else process-exists(T,CLS)
          fi
     else process-exists(T,CLS)
     fi .
```

```
--- セル system_cell 中にあるセル名の消去
eq process-delete(T,NoClause) = NoClause .
eq process-delete(T,cldef(pred(X,T'),nil,!))
   = if X == 'process
     then if T == T'
          then NoClause
          else cldef(pred(X,T'),nil,!)
          fi
     else cldef(pred(X,T'),nil,!)
     fi .
eq process-delete(T,clauselist(cldef(pred(X,T'),nil,!),CLS))
   = if X == 'process
     then if T == T'
          then CLS
          else clauselist(cldef(pred(X,T'),nil,!),process-delete(T,CLS))
          fi
     else clauselist(cldef(pred(X,T'),nil,!),process-delete(T,CLS))
     fi .

--- 発火点位置情報
sorts CName Location .
subsort Qid < CName .

op loc : MachineInt MachineInt -> Location .
op Meta : -> Location .
op NoLoc : -> Location .
op change-location : Location -> Location .

eq change-location(NoLoc) = NoLoc .
eq change-location(Meta) = Meta .
eq change-location(loc(M,N)) = loc(M + 1,N) .

--- セル変数と束縛値の組
sorts CVPair CVPairList .
subsort CVPair < CVPairList .

op NoCVPair : -> CVPairList .
op cvpair : Qid Term -> CVPair .
op cvs : CVPairList CVPairList -> CVPairList [ assoc ] .

op cv-exists : Term CVPairList -> Bool .
op write-value : Term Term CVPairList -> CVPairList .
op get-value : Term CVPairList -> Term .
op reset-value : Term CVPairList -> CVPairList .
op create-pid : Term CVPairList -> Qid .
op create-cvpairs : TermList -> CVPairList .
op add-cvs : TermList CVPairList -> CVPairList .
```

```
var CVS : CVPairList .

--- cvs が結合的および単位元を持つように定義
eq cvs(NoCVPair,CVS) = CVS .
eq cvs(CVS,NoCVPair) = CVS .

--- 指定されたセル変数の存在確認
eq cv-exists(T,NoCVPair) = false .
eq cv-exists(T,cvpair(T',V)) = if T == T' then true else false fi .
eq cv-exists(T,cvs(cvpair(T',V),CVS))
   = if T == T' then true else cv-exists(T,CVS) fi .

--- 指定されたセル変数の値を獲得
eq get-value(T,cvpair(T,V)) = V .
eq get-value(T,cvs(cvpair(T',V),CVS))
   = if T == T' then V else get-value(T,CVS) fi .

--- 指定されたセル変数に指定された値を束縛する
eq write-value(T,V,cvpair(T,U)) = cvpair(T,V) .
eq write-value(T,V,cvs(cvpair(T',U),CVS))
   = if T == T'
     then cvs(cvpair(T,V),CVS)
     else cvs(cvpair(T',U),write-value(T,V,CVS))
     fi .

--- 指定されたセル変数の値を初期化
eq reset-value(T,cvpair(T,V)) = cvpair(T,undef) .
eq reset-value(T,cvs(cvpair(T',V),CVS))
   = if T == T'
     then cvs(cvpair(T,undef),CVS)
     else cvs(cvpair(T',V),reset-value(T,CVS))
     fi .

--- 例題のために作成した関数
--- プロセス番号の生成
--- 現状では，予め与えられた値を逐次獲得している
eq create-pid(pred('agent,termlist(T,N)),cvpair(T,X)) = X .
eq create-pid(pred('agent,termlist(T,N)),cvs(cvpair(T',X),CVS))
   = if T == T'
     then X
     else create-pid(pred('agent,termlist(T,N)),CVS)
     fi .

--- 指定されたセル変数に初期値を割り当てる
eq create-cvpairs(nil) = NoCVPair .
eq create-cvpairs(T) = cvpair(T,undef) .
eq create-cvpairs(termlist(T,TL))
   = cvs(cvpair(T,undef),create-cvpairs(TL)) .
```

```
--- セル変数の追加
eq add-cvs(nil,CVS) = CVS .
eq add-cvs(T,CVS)
   = if cv-exists(T,CVS) then CVS else cvs(cvpair(T,undef),CVS) fi .
eq add-cvs(termlist(T,TL),CVS)
   = if cv-exists(T,CVS)
     then add-cvs(TL,CVS)
     else add-cvs(TL,cvs(cvpair(T,undef),CVS))
     fi .

--- バックトラック情報の定義
sorts Control ControlList .
subsorts Control < ControlList .


op Empty : -> ControlList .
op ctrl : Location TermList -> Control .
op ctrllist : ControlList ControlList -> ControlList [ assoc ] .

op fst : Control -> Location .
op snd : Control -> TermList .
op head : ControlList -> Control .
op tail : ControlList -> ControlList .
op push-cell : ControlList -> ControlList .

var L : Location .
var Ctrl : Control .
vars Ctrls Ctrls1 Ctrls2 : ControlList .

--- ctrllist が結合的かつ単位元を持つように定義
eq ctrllist(Empty,Ctrls) = Ctrls .
eq ctrllist(Ctrls,Empty) = Ctrls .

--- バックトラック情報リストを操作する関数の定義
eq fst(ctrl(L,TL)) = L .
eq snd(ctrl(L,TL)) = TL .

eq head(Ctrl  = Ctrl .
eq head(ctrllist(Ctrl,Ctrls)) = Ctrl .
eq tail(Ctrl) = Empty .
eq tail( ctrllist( Ctrl , Ctrls ) ) = Ctrls .

eq push-cell(Empty) = Empty .
eq push-cell(ctrl(loc(M,N),T)) = ctrl(loc(M + 1,N),T) .
eq push-cell(ctrl(loc(M,N),TL)) = ctrl(loc(M + 1,N),TL) .
eq push-cell(ctrllist(ctrl(loc(M + 1,N),T),Ctrls))
   = ctrllist(ctrl(loc(M + 1,N),T),push-cell(Ctrls)) .
eq push-cell(ctrllist(ctrl(loc(M + 1,N),TL),Ctrls))
    = ctrllist(ctrl(loc(M + 1,N),TL),push-cell(Ctrls)) .
```

```
--- 環境 (セル名のスタック) の定義
sorts Env Environment .
subsort CName < Environment .

op NoEnv : -> CName .
op environment : Environment Environment -> Environment [ assoc ] .
op nthe : MachineInt Environment -> CName .
op out-of-range-e : -> CName .
op subst-cell : CName CName Environment -> Environment .
op in : CName Environment -> Bool .
op cell-delete : CName Environment -> Environment .

vars E E' E'' : CName .
var EL : Environment .
vars Old New : CName .

--- environment が結合的かつ単位元を持つように定義
eq environment( NoEnv , EL ) = EL .
eq environment( EL , NoEnv ) = EL .

--- 環境を操作する関数の定義
cq nthe(M,EL) = out-of-range-e if M < 1 .
eq nthe(M,E) = if M == 1 then E else out-of-range-e fi .
eq nthe(M,environment(E,EL))
   = if M == 1 then E else nthe(M - 1,EL) fi .

eq subst-cell(Old,New,NoEnv) = NoEnv .
eq subst-cell(Old,New,Old) = New .
eq subst-cell(Old,New,environment(E,EL))
   = if Old == E
     then environment(New,EL)
     else environment(E,subst-cell(Old,New,EL))
     fi .

eq in(E,NoEnv) = false .
eq in(E,E') = if E == E' then true else false fi .
eq in(E,environment(E',EL)) = if E == E' then true else in(E,EL) fi .

eq cell-delete(E,NoEnv) = NoEnv .
eq cell-delete(E,E') = if E == E' then NoEnv else E' fi .
eq cell-delete(E,environment(E',EL))
   = if E == E' then EL else environment(E',cell-delete(E,EL)) fi .

--- 計算状態の定義
sorts Object Message State .
subsort Object < State .

sort PName .
```

```
  subsort Qid < PName .

  op cell : CName CVPairList ClauseList -> Object .
  op process : PName Environment Location TermList ControlList -> Object .
  op process-e : PName Environment Location TermList ControlList
                 CName -> Object .
  op process-e-c : PName Environment Location TermList ControlList
                   CName Clause -> Object .

  op state : State State -> State [ assoc comm ] .

  var CV : Qid .
  vars P P' P1 P2 : PName .
  var CVS' : CVPairList .
  var CLS' : ClauseList .
  vars EL1 EL2 : Environment .
  var Loc : Location .
  vars W A1 A2 Head : Term .
  vars GL WL GL1 GL2 Cond Body : TermList .
  var Cell : CName .

--- ------------------------------------------------------------------------
--- 発火点位置情報に基づき，セル名を獲得
--- ------------------------------------------------------------------------
  --- ゴール列が単項の場合
  rl
  process(P,EL,loc(M,N),T,Ctrls)
  =>
  process-e(P,EL,loc(M,N),T,Ctrls,nthe(M,EL)) .

  --- ゴール列が単項でない場合
  rl
  process(P,EL,loc(M,N),termlist(T,TL),Ctrls)
  =>
  process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,nthe(M,EL)) .

--- ------------------------------------------------------------------------
--- 発火点位置情報に基づき，述語を獲得
--- ------------------------------------------------------------------------
  --- ゴール列が単項で，セル名が得られなかった場合
  rl
  process-e(P,EL,loc(M,N),T,Ctrls,out-of-range-e)
  =>
  process(P,EL,Meta,T,Ctrls) .

  --- ゴール列が単項で，セル名が得られた場合
  rl
  state(cell(E,CVS,CLS),
        process-e(P,EL,loc(M,N),T,Ctrls,E))
```

```
  =>
  state(cell(E,CVS,CLS),
        process-e-c(P,EL,loc(M,N),T,Ctrls,E,nthcl(N,CLS))) .

  --- ゴール列が単項でなく，セル名が得られなかった場合
  rl
  process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,out-of-range-e)
  =>
  process(P,EL,Meta,termlist(T,TL),Ctrls) .

  --- ゴール列が単項でなく，セル名が得られた場合
  rl
  state(cell(E,CVS,CLS),
        process-e(P,EL,loc(M,N),termlist(T,TL),Ctrls,E))
  =>
  state(cell(E,CVS,CLS),
        process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,E,nthcl(N,CLS))) .

--- ------------------------------------------------------------------------
--- 次の節定義の獲得
--- ------------------------------------------------------------------------
  --- ゴール列が単項の場合
  rl
  process-e-c(P,EL,loc(M,N),T,Ctrls,E,out-of-range-c)
  =>
  process(P,EL,loc(M + 1,1),T,Ctrls) .

  --- ゴール列が単項でない場合
  rl
  process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,E,out-of-range-c)
  =>
  process(P,EL,loc(M + 1,1),termlist(T,TL),Ctrls) .

--- ------------------------------------------------------------------------
--- ユニフィケーション
--- ------------------------------------------------------------------------
  --- ゴール列が単項で，ユニフィケーション成功
  crl
  process-e-c(P,EL,loc(M,N),T,Ctrls,E,cldef(Head,Cond,Body))
  =>
  process(P,EL,loc(1,1),subst(unify(eqn(Head,T)),termlist(Cond,Body)),
          ctrllist(ctrl(loc(M,N + 1),T),Ctrls))
  if unify(eqn(Head,T)) =/= failure .

  --- ゴール列が単項で，ユニフィケーション失敗
  crl
  process-e-c(P,EL,loc(M,N),T,Ctrls,E,cldef(Head,Cond,Body))
  =>
  process-e(P,EL,loc(M,N + 1),T,Ctrls,E)
```

```
  if unify(eqn(Head,T)) == failure .

  --- ゴール列が単項でなく，ユニフィケーション成功
  crl
  process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,
              E,cldef(Head,Cond,Body))
  =>
  process(P,EL,loc(1,1),
          subst(unify(eqn(Head,T)),termlist(Cond,Body,TL)),
          ctrllist(ctrl(loc(M,N + 1),termlist(T,TL)),Ctrls))
  if unify(eqn(Head,T)) =/= failure .

  --- ゴール列が単項でなく，ユニフィケーション失敗
  crl
  process-e-c(P,EL,loc(M,N),termlist(T,TL),Ctrls,
              E,cldef(Head,Cond,Body))
  =>
  process-e(P,EL,loc(M,N + 1),termlist(T,TL),Ctrls,E)
  if unify(eqn(Head,T)) == failure .

--- ------------------------------------------------------------------------
--- 組み込み述語適用の準備
--- ------------------------------------------------------------------------
  --- ゴール列が単項で，バックトラック情報が空の場合
  crl
  process(P,EL,Meta,T,Empty)
  =>
  process(P,EL,NoLoc,T,Empty)
  if belongsto(T,termlist(!,fail,'write,'nl,'add,'eq,
                          'fork,'new-cell,'push,'assert,
                          'cv-write,'cv-take,'cv-set,
                          'cv-ref,'subst-cell,'cv-read,'die))
     == false .

  --- ゴール列が単項で，バックトラック情報が空でない場合
  crl
  process(P,EL,Meta,T,Ctrls)
  =>
  process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls))
  if and(belongsto(T,termlist(!,fail,'write,'nl,'add,'eq,
                              'fork,'new-cell,'push,'assert,
                              'cv-write,'cv-take,'cv-set,
                              'cv-ref,'subst-cell,'cv-read,'die))
     == false,Ctrls =/= Empty ) .

  --- ゴール列が単項でなく，バックトラック情報が空の場合
  crl
  process(P,EL,Meta,termlist(T,TL),Empty)
  =>
```

```
  process(P,EL,NoLoc,termlist(T,TL),Empty)
  if belongsto(T,termlist(!,fail,'write,'nl,'add,'eq,
                          'fork,'new-cell,'push,'assert,
                          'cv-write,'cv-take,'cv-set,
                          'cv-ref,'subst-cell,'cv-read,'die))
     == false .

  --- ゴール列が単項でなく，バックトラック情報が空でない場合
  crl
  process(P,EL,Meta,termlist(T,TL),Ctrls)
  =>
  process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls))
  if and(belongsto(T,termlist(!,fail,'write,'nl,'add,'eq,
                              'fork,'new-cell,'push,'assert,
                              'cv-write,'cv-take,'cv-set,
                              'cv-ref,'subst-cell,'cv-read,'die))
     == false,Ctrls =/= Empty ) .

--- ------------------------------------------------------------------------
--- 組み込み述語に対応する状態遷移規則の定義
--- ------------------------------------------------------------------------
  --- ------------------------------------------------------------------------
  --- cut operator
  --- ------------------------------------------------------------------------
  --- ゴール列が単項で，バックトラック情報が空の場合
  rl
  process(P,EL,Meta,!,Empty)
  =>
  process(P,EL,loc(1,1),nil,Empty) .

  --- ゴール列が単項で，バックトラック情報がひとつの場合
  rl
  process(P,EL,Meta,!,Ctrl)
  =>
  process(P,EL,loc(1,1),nil,Empty) .

  --- ゴール列が単項で，バックトラック情報が 2 つ以上の場合
  rl
  process(P,EL,Meta,!,ctrllist(Ctrl,Ctrls))
  =>
  process(P,EL,loc(1,1),nil,Ctrls) .

  --- ゴール列が単項でなく，バックトラック情報が空の場合
  rl
  process(P,EL,Meta,termlist(!,TL),Empty)
  =>
  process(P,EL,loc(1,1),TL,Empty) .

  --- ゴール列が単項でなく，バックトラック情報がひとつの場合
```

```
rl
process(P,EL,Meta,termlist(!,TL),Ctrl)
=>
process(P,EL,loc(1,1),TL,Empty) .

--- ゴール列が単項でなく，バックトラック情報が 2 つ以上の場合
rl
process(P,EL,Meta,termlist(!,TL),ctrllist(Ctrl,Ctrls))
=>
process(P,EL,loc(1,1),TL,Ctrls) .

--- ------------------------------------------------------------------
--- fail
--- ------------------------------------------------------------------
--- ゴール列が単項で，バックトラック情報が空の場合
rl
process(P,EL,Meta,fail,Empty)
=>
process(P,EL,NoLoc,fail,Empty) .

--- ゴール列が単項で，バックトラック情報がひとつの場合
rl
process(P,EL,Meta,fail,ctrl(loc(M,N),TL))
=>
process(P,EL,loc(M,N),TL,Empty) .

--- ゴール列が単項で，バックトラック情報が 2 つ以上の場合
rl
process(P,EL,Meta,fail,ctrllist(ctrl(loc(M,N),TL),Ctrls))
=>
process(P,EL,loc(M,N),TL,Ctrls) .

--- ゴール列が単項でなく，バックトラック情報が空の場合
rl
process(P,EL,Meta,termlist(fail,TL),Empty)
=>
process(P,EL,NoLoc,termlist(fail,TL),Empty) .

--- ゴール列が単項でなく，バックトラック情報がひとつの場合
rl
process(P,EL,Meta,termlist(fail,TL),ctrl(loc(M,N),TL'))
=>
process(P,EL,loc(M,N),TL',Empty) .

--- ゴール列が単項でなく，バックトラック情報が 2 つ以上の場合
rl
process(P,EL,Meta,termlist(fail,TL),
        ctrllist(ctrl(loc(M,N),TL'),Ctrls))
=>
```

```
process(P,EL,loc(M,N),TL',Ctrls) .

--- ------------------------------------------------------------------
--- write
--- 標準出力の替わりにセル 'display 内に書き込む
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
state(cell('display,CVS,CLS),
      process(P,EL,Meta,pred('wreite,T),Ctrls))
=>
state(cell('display,CVS,clauselist(cldef(T,nil,nil),CLS)),
      process(P,EL,NoLoc,nil,Ctrls)) .

--- ゴール列が単項でない場合
rl
state(cell('display,CVS,CLS),
      process(P,EL,Meta,termlist(pred('write,T),TL),Ctrls))
=>
state(cell('display,CVS,clauselist(cldef(T,nil,nil),CLS)),
      process(P,EL,loc(1,1),TL,Ctrls)) .

--- ------------------------------------------------------------------
--- nl
--- 標準出力の替わりにセル 'display 内に書き込む
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
process(P,EL,Meta,pred('nl,nil),Ctrls)
=>
process(P,EL,NoLoc,nil,Ctrls) .

--- ゴール列が単項でない場合
rl
process(P,EL,Meta,termlist(pred('nl,nil),TL),Ctrls)
=>
process(P,EL,loc(1,1),TL,Ctrls) .

--- ------------------------------------------------------------------
--- add
--- 足し算を行う
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
process(P,EL,Meta,pred('add,termlist(M,N,T)),Ctrls)
=>
process(P,EL,loc(1,1),nil,Ctrls) .

--- ゴール列が単項でない場合
```

```
rl
process(P,EL,Meta,termlist(pred('add,termlist(M,N,T)),TL),Ctrls)
=>
process(P,EL,loc(1,1),subst(sys(eqn(T,M + N)),TL),Ctrls) .

--- ------------------------------------------------------------------
--- eq
--- 二つの項のユニフィケーションを行う
--- ------------------------------------------------------------------
--- ゴール列が単項，バックトラック情報が空で，
--- 二つの項がユニファイした場合
crl
process(P,EL,Meta,pred('eq,termlist(TL,UL)),Empty)
=>
process(P,EL,loc(1,1),subst(unify(eqn(TL,UL)),nil),Empty)
if unify(eqn(TL,UL)) =/= failure .

--- ゴール列が単項，バックトラック情報が空で，
--- 二つの項がユニファイしない場合
crl
process(P,EL,Meta,pred('eq,termlist(TL,UL)),Empty)
=>
process(P,EL,NoLoc,pred('eq,termlist(TL,UL)),Empty)
if unify(eqn(TL,UL)) == failure .

--- ゴール列が単項，バックトラック情報が空でなく，
--- 二つの項がユニファイした場合
crl
process(P,EL,Meta,pred('eq,termlist(TL,UL)),Ctrls)
=>
process(P,EL,loc(1,1),subst(unify(eqn(TL,UL)),nil),Ctrls)
if and(unify(eqn(TL,UL)) =/= failure,Ctrls =/= Empty) .

--- ゴール列が単項，バックトラック情報が空でなく，
--- 二つの項がユニファイしない場合
crl
process(P,EL,Meta,pred('eq,termlist(TL,UL)),Ctrls)
=>
process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls))
if and(unify(eqn(TL,UL)) == failure,Ctrls =/= Empty) .

--- ゴール列が単項でなく，バックトラック情報が空で，
--- 二つの項がユニファイした場合
crl
process(P,EL,Meta,termlist(pred('eq,termlist(TL,UL)),WL),Empty)
=>
process(P,EL,loc(1,1),subst(unify(eqn(TL,UL)),WL),Empty)
if unify(eqn(TL,UL)) =/= failure .
```

```
--- ゴール列が単項でなく，バックトラック情報が空で，
--- 二つの項がユニファイしない場合
crl
process(P,EL,Meta,termlist(pred('eq,termlist(TL,UL)),WL),Empty)
=>
process(P,EL,NoLoc,termlist(pred('eq,termlist(TL,UL)),WL),Empty)
if unify(eqn(TL,UL)) == failure .

--- ゴール列が単項でなく，バックトラック情報が空でなく，
--- 二つの項がユニファイした場合
crl
process(P,EL,Meta,termlist(pred('eq,termlist(TL,UL)),WL),Ctrls)
=>
process(P,EL,loc(1,1),subst(unify(eqn(TL,UL)),WL),Ctrls)
if and(unify(eqn(TL,UL)) =/= failure,Ctrls =/= Empty) .

--- ゴール列が単項でなく，バックトラック情報が空でなく，
--- 二つの項がユニファイしない場合
crl
process(P,EL,Meta,termlist(pred('eq,termlist(TL,UL)),WL),Ctrls)
=>
process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls))
if and(unify(eqn(TL,UL)) == failure,Ctrls =/= Empty) .

--- ------------------------------------------------------------------
--- fork
--- プロセスを生成する
--- ------------------------------------------------------------------
--- ゴール列が単項，fork の引数が単項の場合
rl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('fork,T),Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('process,create-pid(T,CVS)),nil,!),
                      CLS)),
       process(create-pid(T,CVS),EL,loc(1,1),T,Empty),
       process(P,EL,NoLoc,nil,Ctrls)) .

--- ゴール列が単項，fork の引数が単項でない場合
rl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('fork,TL),Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('process,create-pid(TL,CVS)),nil,!),
                      CLS)),
       process(create-pid(TL,CVS),EL,loc(1,1),TL,Empty),
       process(P,EL,NoLoc,nil,Ctrls)) .
```

```
--- ゴール列が単項でなく，fork の引数が単項の場合
rl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('fork,T),GL),Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('process,create-pid(T,CVS)),nil,!),
                      CLS)),
       process(create-pid(T,CVS),EL,loc(1,1),T,Empty),
       process(P,EL,loc(1,1),GL,Ctrls)) .

--- ゴール列が単項でなく，fork の引数が単項でない場合
rl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('fork,TL),GL),Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('process,create-pid(TL,CVS)),nil,!),
                      CLS)),
       process(create-pid(TL,CVS),EL,loc(1,1),TL,Empty),
       process(P,EL,loc(1,1),GL,Ctrls)) .

--- ----------------------------------------------------------------------
--- assert
--- 節定義の追加を行う
--- ----------------------------------------------------------------------
--- ゴール列が単項の場合
rl
state(cell(E,CVS,CLS),
      process(P,environment(E,EL),Meta,pred('assert,T),Ctrls))
=>
state(cell(E,CVS,clauselist(cldef(T,nil,!),CLS)),
      process(P,environment(E,EL),loc(1,1),nil,Ctrls)) .

--- ゴール列が単項でない場合
rl
state(cell(E,CVS,CLS),
      process(P,environment(E,EL),Meta,
              termlist(pred('assert,T),GL),Ctrls))
=>
state(cell(E,CVS,clauselist(cldef(T,nil,!),CLS)),
       process(P,environment(E,EL),loc(1,1),GL,Ctrls)) .

--- ----------------------------------------------------------------------
--- die
--- 自分自身のプロセスを消去する
--- ----------------------------------------------------------------------
--- ゴール列が単項の場合
```

```
rl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('die,nil),Ctrls))
=>
cell('system-cell,CVS,process-delete(P,CLS)) .

--- ゴール列が単項でないの場合
rl
state(cell('system-cell,CVS,CLS) ,
      process(P,EL,Meta,termlist(pred('die,nil),GL),Ctrls))
=>
cell('system-cell,CVS,process-delete(P,CLS)) .

--- ------------------------------------------------------------------
--- kill-process
--- 指定されたプロセス名のプロセスを消去する
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたプロセスがシステム内に存在する場合
rl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('kill-process,P2),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,process-delete(P2,CLS)),
      process(P1,EL1,loc(1,1),nil,Ctrls1)) .

--- ゴール列が単項で，指定されたプロセスがシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('kill-process,P2),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,NoLoc,pred('kill-process,P2),Ctrls))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，指定されたプロセスがシステム内に存在する場合
rl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('kill-process,P2),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,process-delete(P2,CLS)),
      process(P1,EL1,loc(1,1),GL1,Ctrls1)) .

--- ゴール列が単項でなく，指定されたプロセスがシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,
```

```
              termlist(pred('kill-process,P2),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('kill-process,P2),GL),Ctrls))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- push/1
--- 指定されたセル名を環境の先頭に追加する
--- 指定されたセル名がシステム内に存在しない場合は,
--- それが生成されるまで待ち状態になる.
--- ------------------------------------------------------------------
--- ゴール列が単項で, 指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('push,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(Cell,EL),loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項なくで, 指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('push,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(Cell,EL),loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) .

--- ------------------------------------------------------------------
--- push/2
--- 指定されたセル名を指定されたプロセスの環境の先頭に追加する
--- 指定されたセル名がシステム内に存在しない場合は,
--- それが生成されるまで待ち状態になる.
--- 指定されたプロセス名がシステム内に存在しない場合は,
--- エラーとなる.
--- ------------------------------------------------------------------
--- ゴール列が単項で, 指定されたセル名, プロセス名が
--- システム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('push,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,environment(Cell,EL2),change-location(Loc),GL2,Ctrls2))
```

```
if cell-exists(Cell,CLS) .

--- ゴール列が単項で，指定されたセル名はシステム内に存在するが，
--- プロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('push,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('push,termlist(Cell,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名，プロセス名が
--- システム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('push,termlist(Cell,P2)),GL),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL,Ctrls1),
      process(P2,environment(Cell,EL2),change-location(Loc),GL2,Ctrls2))
if cell-exists(Cell,CLS) .

--- ゴール列が単項なく，指定されたセル名はシステム内に存在するが，
--- プロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('push,termlist(Cell,P2)),GL),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('push,termlist(Cell,P2)),GL),Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- push-second/1
--- 指定されたセル名を環境の 2 番目に追加する
--- 指定されたセル名がシステム内に存在しない場合は，
--- それが生成されるまで待ち状態になる.
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,environment(E,EL),Meta,pred('push-second,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
```

```
      process(P,environment(E,Cell,EL),loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,environment(E,EL),Meta,
              termlist(pred('push-second,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(E,Cell,EL),loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) .

--- ------------------------------------------------------------------
--- push-second/2
--- 指定されたセル名を指定されたプロセスの環境の 2 番目に追加する
--- 指定されたセル名がシステム内に存在しない場合は,
--- それが生成されるまで待ち状態になる.
--- 指定されたプロセス名がシステム内に存在しない場合は,
--- エラーとなる.
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('push-second,termlist(Cell,P2)),Ctrls1),
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,environment(E,Cell,EL2),Loc,GL2,Ctrls2))
if cell-exists(Cell,CLS) .

--- ゴール列が単項で，指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('push-second,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('push-second,termlist(Cell,P2)),
              Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('push-second,termlist(Cell,P2)),GL1),
              Ctrls1),
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2))
```

```
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,environment(E,Cell,EL2),Loc,GL2,Ctrls2))
if cell-exists(Cell,CLS) .

--- ゴール列が単項なくで,
--- 指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('push-second,termlist(Cell,P2)),GL),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('push-second,termlist(Cell,P2)),GL),Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- fpush/1
--- 指定されたセル名を環境の先頭に追加する.
--- もしセル名がシステム内に存在していない場合は,
--- そのセル名を持つセルを生成し，セル名を環境の先頭に追加する.
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('fpush,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(Cell,EL),loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項で，指定されたセル名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('fpush,Cell),Ctrls))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P,environment(Cell,EL),loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項なくで，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('fpush,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(Cell,EL),loc(1,1),GL,Ctrls))
```

```
if cell-exists(Cell,CLS) .

--- ゴール列が単項なくで，指定されたセル名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('fpush,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P,environment(Cell,EL),loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) == false .

--- ---------------------------------------------------------------------
--- fpush/2
--- 指定されたセル名を指定されたプロセスの環境の先頭に追加する.
--- もしセル名がシステム内に存在していない場合は,
--- そのセル名を持つセルを生成し，セル名を環境の先頭に追加する.
--- もし指定されたプロセスがシステム内に存在しない場合は,
--- エラーとなる.
--- ---------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名がシステム内に存在し,
--- 指定されたプロセス名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('fpush,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,environment(Cell,EL2),change-ocation(Loc),GL2,Ctrls2))
if cell-exists(Cell,CLS) .

--- ゴール列が単項で，指定されたセル名がシステム内に存在せず,
--- 指定されたプロセス名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('fpush,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,environment(Cell,EL2),change-location(Loc),GL2,Ctrls2))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項で，指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('fpush,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
```

```
      process(P1,EL1,NoLoc,pred('fpush,termlist(Cell,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在し，
--- 指定されたプロセス名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('fpush,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,environment(Cell,EL2),change-location(Loc),GL2,Ctrls2))
if cell-exists(Cell,CLS) .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在せず，
--- 指定されたプロセス名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('fpush,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,environment(Cell,EL2),change-location(Loc),GL2,Ctrls2))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項でなく，
--- 指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('fpush,termlist(Cell,P2)),GL1),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('fpush,termlist(Cell,P2)),GL1),Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- pop/1
--- 現在の環境の先頭を削除する
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
process(P,environment(E,EL),Meta,pred('pop,Cell),Ctrls)
=>
```

```
process(P,EL,loc(1,1),subst(eqn(Cell,E),nil),Ctrls) .

--- ゴール列が単項でない場合
rl
process(P,environment(E,EL),Meta,termlist(pred('pop,Cell),GL),Ctrls)
=>
process(P,EL,loc(1,1),subst(eqn(Cell,E),GL),Ctrls) .

--- ------------------------------------------------------------------
--- pop/2
--- 指定されたプロセスの環境の先頭を削除する
--- 指定されたプロセスがシステム内に存在しない場合はエラーとなる
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたプロセスがシステム内に存在する場合
rl
state(process(P1,EL1,Meta,pred('pop,termlist(Cell,P2)),Ctrls1),
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2))
=>
state(process(P1,EL1,loc(1,1),subst(eqn(Cell,E),nil),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2)) .

--- ゴール列が単項で，指定されたプロセスがシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('pop,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('pop,termlist(Cell,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，指定されたプロセスがシステム内に存在する場合
rl
state(process(P1,EL1,Meta,
              termlist(pred('pop,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2))
=>
state(process(P1,EL1,loc(1,1),subst(eqn(Cell,E),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2)) .

--- ゴール列が単項でなく，指定されたプロセスがシステム内に存在しない場合
rl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('pop,termlist(Cell,P2)),GL1),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('pop,termlist(Cell,P2)),GL1),Ctrls1))
if process-exists(P2,CLS) == false .
```

```
--- ----------------------------------------------------------------------
--- needs/1
--- 指定されたセル名がシステム内に存在すればなにもしない.
--- 指定されたセル名がシステム内に存在しなければ，セルを生成し，
--- 環境の先頭に追加する.
--- ----------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('needs,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項で，指定されたセル名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('needs,Cell),Ctrls))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P,environment(Cell,EL),loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('needs,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('needs,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P,environment(Cell,EL),loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) == false .

--- ----------------------------------------------------------------------
--- needs/2
--- 指定されたセル名がシステム内に存在すればなにもしない.
--- 指定されたセル名がシステム内に存在しなければ，セルを生成し，
--- 指定されたプロセスの環境の先頭に追加する.
--- 指定されたプロセス名がシステム内に存在しなければエラーとなる.
```

```
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('needs,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
if cell-exists(Cell,CLS) .

--- ゴール列が単項で，指定されたセル名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('needs,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,environment(Cell,EL2),Loc,GL2,Ctrls2))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項で，指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('needs,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('needs,termlist(Cell,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('needs,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
if cell-exists(Cell,CLS) .

--- ゴール列が単項でなく，指定されたセル名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('needs,termlist(Cell,P2)),GL1),Ctrls1),
```

```
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,clauselist(cldef(Cell,nil,!),CLS)),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,environment(Cell,EL2),Loc,GL2,Ctrls2))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項でなく,
--- 指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('needs,termlist(Cell,P2)),GL1),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('needs,termlist(Cell,P2)),GL1),Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- repush/1
--- 指定されたセル名が環境内に存在すれば，それを先頭に移動する.
--- 指定されたセル名が環境内に存在せず，システム内に存在すれば,
--- 環境の先頭に追加する.
--- 指定されたセル名が環境内に存在せず，システム内に存在しなければ,
--- それが生成されるまで待ち状態になる.
--- ------------------------------------------------------------------
--- ゴール列が単項で，セル名が環境内に存在する場合
crl
process(P,EL,Meta,pred('repush,Cell),Ctrls)
=>
process(P,environment(Cell,cell-delete(Cell,EL)),loc(1,1),nil,Ctrls)
if in(Cell,EL) .

--- ゴール列が単項で，セル名が環境内に存在せず，システム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
     process(P,EL,Meta,pred('repush,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(Cell,EL),loc(1,1),nil,Ctrls))
if and(in(Cell,EL) == false,cell-exists(Cell,CLS)) .

--- ゴール列が単項でなく，セル名が環境内に存在する場合
crl
process(P,EL,Meta,termlist(pred('repush,Cell),GL),Ctrls)
=>
process(P,environment(Cell,cell-delete(Cell,EL)),loc(1,1),GL,Ctrls)
if in(Cell,EL) .
```

```
--- ゴール列が単項でなく，セル名が環境内に存在せず,
--- システム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
     process(P,EL,Meta,termlist(pred('repush,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,environment(Cell,EL),loc(1,1),GL,Ctrls))
if and(in(Cell,EL) == false,cell-exists(Cell,CLS)) .

--- ---------------------------------------------------------------------
--- repush/2
--- 指定されたセル名が指定されたプロセスの環境内に存在すれば,
--- それを先頭に移動する.
--- 指定されたセル名が指定されたプロセスの環境内に存在せず,
--- システム内に存在すれば，その環境の先頭に追加する.
--- 指定されたセル名が指定されたプロセスの環境内に存在せず,
--- システム内に存在しなければ，それが生成されるまで待ち状態になる.
--- ---------------------------------------------------------------------
--- ゴール列が単項で，セル名が指定されたプロセスの環境内に存在する場合
crl
state(process(P1,EL1,Meta,pred('repush,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,environment(Cell,cell-delete(Cell,EL2)),Loc,GL2,
              Ctrls2))
if in(Cell,EL2) .

--- ゴール列が単項で，セル名が指定されたプロセスの環境内に存在せず,
--- システム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('repush,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,environment(Cell,EL2),Loc,GL2,Ctrls2))
if and(in(Cell,EL2) == false,cell-exists(Cell,CLS)) .

--- ゴール列が単項で，指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('repush,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('repush,termlist(Cell,P2)),Ctrls1))
```

```
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく,
--- セル名が指定されたプロセスの環境内に存在する場合
crl
state(process(P1,EL1,Meta,
              termlist(pred('repush,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,environment(Cell,cell-delete(Cell,EL2)),Loc,GL2,
              Ctrls2))
if in(Cell,EL2) .

--- ゴール列が単項でなく，セル名が指定されたプロセスの環境内に存在せず,
--- システム内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('repush,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,environment(Cell,EL2),Loc,GL2,Ctrls2))
if and(in(Cell,EL2) == false,cell-exists(Cell,CLS)) .

--- ゴール列が単項でなく,
--- 指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('repush,termlist(Cell,P2)),GL1),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('repush,termlist(Cell,P2)),GL1),Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- remove-cell/1
--- 指定されたセル名を環境内から削除する
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名が環境内に存在する場合
crl
process(P,EL,Meta,pred('remove-cell,Cell),Ctrls)
=>
process(P,cell-delete(Cell,EL),loc(1,1),nil,Ctrls)
if in(Cell,EL) .
```

```
--- ゴール列が単項で，指定されたセル名が環境内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('remove-cell,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if in(Cell,EL) == false .

--- ゴール列が単項でなく，指定されたセル名が環境内に存在する場合
crl
process(P,EL,Meta,termlist(pred('remove-cell,Cell),GL),Ctrls)
=>
process(P,cell-delete(Cell,EL),loc(1,1),GL,Ctrls)
if in(Cell,EL) .

--- ゴール列が単項でなく，指定されたセル名が環境内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('remove-cell,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if in(Cell,EL) == false .

--- ----------------------------------------------------------------------
--- remove-cell/2
--- 指定されたセル名を指定されたプロセスの環境内から削除する
--- ----------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名が指定されたプロセスの
--- 環境内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('remove-cell,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,cell-delete(Cell,EL2),Loc,GL2,Ctrls2))
if in(Cell,EL2) .

--- ゴール列が単項で，指定されたセル名が指定されたプロセスの
--- 環境内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('remove-cell,termlist(Cell,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
```

```
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
if in(Cell,EL2) == false .

--- ゴール列が単項で，指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('remove-cell,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('remove-cell,termlist(Cell,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名が指定されたプロセスの
--- 環境内に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('remove-cell,termlist(Cell,P2)),GL1),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,cell-delete(Cell,EL2),Loc,GL2,Ctrls2))
if in(Cell,EL2) .

--- ゴール列が単項でなく，指定されたセル名が指定されたプロセスの
--- 環境内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('remove-cell,termlist(Cell,P2)),GL1),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
if in(Cell,EL2) == false .

--- ゴール列が単項でなく，
--- 指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('remove-cell,termlist(Cell,P2)),GL1),
              Ctrls1))
```

```
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('remove-cell,termlist(Cell,P2)),GL1),
              Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- subst-cell/2
--- 第一引数に指定されたセル名を第二引数に指定されたセル名と交換する
--- 第一引数に指定されたセル名がシステム内に存在しない場合はエラー
--- 第二引数に指定されたセル名が環境内に存在しない場合は失敗
--- ------------------------------------------------------------------
--- ゴール列が単項で，指定されたセル名が所定の位置に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('subst-cell,termlist(New,Old)),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,subst-cell(Old,New,EL),loc(1,1),nil,Ctrls))
if and(cell-exists(New,CLS),in(Old,EL)) .

--- ゴール列が単項で,
--- 第一引数に指定されたセル名はシステム内に存在するが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('subst-cell,termlist(New,Old)),Empty))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,NoLoc,pred('subst-cell,termlist(New,Old)),Empty))
if and(cell-exists(New,CLS),in(Old,EL) == false) .

--- ゴール列が単項で,
--- 第一引数に指定されたセル名はシステム内に存在するが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('subst-cell,termlist(New,Old)),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cell-exists(New,CLS),in(Old,EL) == false,Ctrls =/= Empty) .

--- ゴール列が単項で,
--- 第一引数に指定されたセル名はシステム内に存在しない場合
crl
```

```
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('subst-cell,termlist(New,Old)),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,NoLoc,pred('subst-cell,termlist(New,Old)),Ctrls))
if cell-exists(New,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名が所定の位置に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('subst-cell,termlist(New,Old)),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,subst-cell(Old,New,EL),loc(1,1),GL,Ctrls))
if and(cell-exists(New,CLS),in(Old,EL)) .

--- ゴール列が単項でなく,
--- 第一引数に指定されたセル名はシステム内に存在するが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('subst-cell,termlist(New,Old)),GL),Empty))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('subst-cell,termlist(New,Old)),GL),Empty))
if and(cell-exists(New,CLS),in(Old,EL) == false) .

--- ゴール列が単項でなく,
--- 第一引数に指定されたセル名はシステム内に存在するが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('subst-cell,termlist(New,Old)),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cell-exists(New,CLS),in(Old,EL) == false,Ctrls =/= Empty) .

--- ゴール列が単項でなく,
--- 第一引数に指定されたセル名はシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,
```

```
              termlist(pred('subst-cell,termlist(New,Old)),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('subst-cell,termlist(New,Old)),GL),Ctrls))
if cell-exists(New,CLS) == false .

--- ---------------------------------------------------------------------
--- subst-cell/3
--- 第一引数に指定されたセル名を，第三引数に指定されたプロセスの環境中の
--- 第二引数に指定されたセル名と交換する
--- 第一引数に指定されたセル名がシステム内に存在しない場合はエラー
--- 第三引数に指定されたプロセス名がシステム内に存在しない場合はエラー
--- 第二引数に指定されたセル名が環境内に存在しない場合は失敗
--- ---------------------------------------------------------------------
--- ゴール列が単項で，条件を満足している場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('subst-cell,termlist(New,Old,P2)),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),nil,Ctrls1),
      process(P2,subst-cell(Old,New,EL2),Loc,GL2,Ctrls2))
if and(cell-exists(New,CLS),in(Old,EL2)) .

--- ゴール列が単項で，第一，第三引数は条件を満たすが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('subst-cell,termlist(New,Old,P2)),Empty),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('subst-cell,termlist(New,Old,P2)),
              Empty),
      process(P2,EL2,Loc,GL2,Ctrls2))
if and(cell-exists(New,CLS),in(Old,EL2) == false) .

--- ゴール列が単項で，第一，第三引数は条件を満たすが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('subst-cell,termlist(New,Old,P2)),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
```

```
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,fst(head(Ctrls1)),snd(head(Ctrls1)),tail(Ctrls1)),
      process(P2,EL2,Loc,GL2,Ctrls2))
if and(cell-exists(New,CLS),in(Old,EL2) == false,Ctrls1 =/= Empty) .

--- ゴール列が単項で,
--- 第一引数に指定されたセル名がシステム中に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('subst-cell,termlist(New,Old,P2)),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('subst-cell,termlist(New,Old,P2)),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
if cell-exists(New,CLS) == false .

--- ゴール列が単項で,
--- 第三引数に指定されたプロセス名がシステム中に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              pred('subst-cell,termlist(New,Old,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              pred('subst-cell,termlist(New,Old,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でなく，条件を満足している場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),GL1,Ctrls1),
      process(P2,subst-cell(Old,New,EL2),Loc,GL2,Ctrls2))
if and(cell-exists(New,CLS),in(Old,EL2)) .

--- ゴール列が単項でなく，第一，第三引数は条件を満たすが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空の場合
crl
```

```
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Empty),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Empty),
      process(P2,EL2,Loc,GL2,Ctrls2))
if and(cell-exists(New,CLS),in(Old,EL2) == false) .

--- ゴール列が単項でなく，第一，第三引数は条件を満たすが,
--- 第二引数に指定されたセル名が環境内に存在せず,
--- バックトラック情報が空でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,fst(head(Ctrls1)),snd(head(Ctrls1)),tail(Ctrls1)),
      process(P2,EL2,Loc,GL2,Ctrls2))
if and(cell-exists(New,CLS),in(Old,EL2) == false,Ctrls1 =/= Empty) .

--- ゴール列が単項でなく,
--- 第一引数に指定されたセル名がシステム中に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
if cell-exists(New,CLS) == false .

--- ゴール列が単項でなく,
--- 第三引数に指定されたプロセス名がシステム中に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
```

```
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('subst-cell,termlist(New,Old,P2)),GL1),
              Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- environment/1
--- 現在の環境を獲得する
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
process(P,EL,Meta,pred('environment,EL1),Ctrls)
=>
process(P,EL,loc(1,1),subst(eqn(EL1,EL),nil),Ctrls) .

--- ゴール列が単項でない場合
rl
process(P,EL,Meta,termlist(pred('environment,EL1),GL),Ctrls)
=>
process(P,EL,loc(1,1),subst(eqn(EL1,EL),GL),Ctrls) .

--- ------------------------------------------------------------------
--- environment/2
--- 指定されたプロセスの環境を獲得する
--- 指定されたプロセス名がシステム内に存在しない場合はエラー
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
state(process(P1,EL1,Meta,pred('environment,termlist(EL,P2)),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(process(P1,EL1,loc(1,1),subst(eqn(EL,EL2),nil),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2)) .

--- ゴール列が単項で，指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('environment,termlist(EL,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('environment,termlist(EL,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でない場合
rl
```

```
state(process(P1,EL1,Meta,
              termlist(pred('environment,termlist(EL,P2)),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2))
=>
state(process(P1,EL1,loc(1,1),subst(eqn(EL,EL2),GL1),Ctrls1),
      process(P2,EL2,Loc,GL2,Ctrls2)) .

--- ゴール列が単項でなく,
--- 指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('environment,termlist(EL,P2)),GL1),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('environment,termlist(EL,P2)),GL1),Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- top_cell/1
--- 現在の環境の先頭のセル名を獲得する
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
process(P,environment(E,EL),Meta,pred('top-cell,Cell),Ctrls)
=>
process(P,environment(E,EL),loc(1,1),subst(eqn(Cell,E),nil),Ctrls) .

--- ゴール列が単項でない場合
rl
process(P,environment(E,EL),Meta,
        termlist(pred('top-cell,Cell),GL),Ctrls)
=>
process(P,environment(E,EL),loc(1,1),subst(eqn(Cell,E),GL),Ctrls) .

--- ------------------------------------------------------------------
--- top-cell/2
--- 指定されたプロセスの環境の先頭のセル名を獲得する
--- 指定されたプロセス名がシステム内に存在しない場合はエラー
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
rl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('top-cell,termlist(Cell,P2)),Ctrls1),
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),subst(eqn(Cell,E),nil),Ctrls1),
```

```
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2)) .

--- ゴール列が単項で，指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,pred('top-cell,termlist(Cell,P2)),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,pred('top-cell,termlist(Cell,P2)),Ctrls1))
if process-exists(P2,CLS) == false .

--- ゴール列が単項でない場合
rl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('top-cell,termlist(Cell,P2)),GL1),Ctrls1),
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,loc(1,1),subst(eqn(Cell,E),GL1),Ctrls1),
      process(P2,environment(E,EL2),Loc,GL2,Ctrls2)) .

--- ゴール列が単項でなく,
--- 指定されたプロセス名がシステム内に存在しない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,Meta,
              termlist(pred('top-cell,termlist(Cell,P2)),GL1),Ctrls1))
=>
state(cell('system-cell,CVS,CLS),
      process(P1,EL1,NoLoc,
              termlist(pred('top-cell,termlist(Cell,P2)),GL1),Ctrls1))
if process-exists(P2,CLS) == false .

--- ------------------------------------------------------------------
--- new-cell/1
--- 指定されたセル名のセルを生成する
--- 指定されたセル名がシステム内に存在する場合はなにもしない
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('new-cell,Cell),Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('cell,Cell),nil,!),CLS)),
      cell(Cell,NoCVPair,NoClause),
      process(P,EL,loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) == false .
```

```
--- ゴール列が単項で，指定されたセル名が既に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('new-cell,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('new-cell,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('cell,Cell),nil,!),CLS)),
      cell(Cell,NoCVPair,NoClause),
      process(P,EL,loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項でなく，指定されたセル名が既に存在する場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('new-cell,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) .

--- ------------------------------------------------------------------
--- new-cell/n, n >= 1
--- 指定されたセル名で，指定されたセル変数を持つセルを生成する
--- 指定されたセル名がシステム内に存在する場合はなにもしない
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('new-cell,termlist(Cell,T,U)),Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('cell,Cell),nil,!),CLS)),
      process(P,EL,loc(1,1),nil,Ctrls),
      cell(Cell,add-cvs(termlist(T,U),NoCVPair),NoClause))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項で，指定されたセルが既に存在する場合
rl
state(cell('system-cell,CVS,CLS),
```

```
      cell(Cell,CVS',CLS'),
      process(P,EL,Meta,pred('new-cell,termlist(Cell,T,U)),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      cell(Cell,add-cvs(termlist(T,U),CVS'),CLS'),
      process(P,EL,loc(1,1),nil,Ctrls)) .

--- ゴール列が単項でない場合
crl state(cell('system-cell,CVS,CLS),
          process(P,EL,Meta,
                  termlist(pred('new-cell,termlist(Cell,T,U)),GL),
                  Ctrls))
=>
state(cell('system-cell,CVS,
           clauselist(cldef(pred('cell,Cell),nil,!),CLS)),
      process(P,EL,loc(1,1),GL,Ctrls),
      cell(Cell,add-cvs(termlist(T,U),NoCVPair),NoClause))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項でなく，指定されたセルが既に存在する場合
rl
state(cell('system-cell,CVS,CLS),
      cell(Cell,CVS',CLS'),
      process(P,EL,Meta,
              termlist(pred('new-cell,termlist(Cell,T,U)),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      cell(Cell,add-cvs(termlist(T,U),CVS'),CLS'),
      process(P,EL,loc(1,1),GL,Ctrls)) .

--- ------------------------------------------------------------------
--- cell-exist/1
--- 指定されたセル名がシステム内に存在するかどうかを確認する
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('cell-exist,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項で，バックトラック情報が空の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('cell-exist,Cell),Empty))
=>
state(cell('system-cell,CVS,CLS),
```

```
      process(P,EL,NoLoc,pred('cell-exist,Cell),Empty))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項で，バックトラック情報が空でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,pred('cell-exist,Cell),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cell-exists(Cell,CLS) == false,Ctrls =/= Empty) .

--- ゴール列が単項でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('cell-exist,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if cell-exists(Cell,CLS) .

--- ゴール列が単項でなく，バックトラック情報が空の場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('cell-exist,Cell),GL),Empty))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,NoLoc,termlist(pred('cell-exist,Cell),GL),Empty))
if cell-exists(Cell,CLS) == false .

--- ゴール列が単項でなく，バックトラック情報が空でない場合
crl
state(cell('system-cell,CVS,CLS),
      process(P,EL,Meta,termlist(pred('cell-exist,Cell),GL),Ctrls))
=>
state(cell('system-cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cell-exists(Cell,CLS) == false,Ctrls =/= Empty) .

--- ------------------------------------------------------------------
--- add-cell-var/2
--- 指定されたセルに，指定されたセル変数を追加する
--- 指定されたセル変数が既に存在する場合は失敗
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('add-cell-var,termlist(Cell,CV)),Ctrls))
=>
```

```
state(cell(Cell,cvs(cvpair(CV,undef),CVS),CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if cv-exists(CV,CVS) == false .

--- ゴール列が単項で，指定されたセル変数が既に存在し，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('add-cell-var,termlist(Cell,CV)),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,pred('add-cell-var,termlist(Cell,CV)),Empty))
if cv-exists(CV,CVS) .

--- ゴール列が単項で，指定されたセル変数が既に存在し，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('add-cell-var,termlist(Cell,CV)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),Ctrls =/= Empty) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('add-cell-var,termlist(Cell,CV)),GL),Ctrls))
=>
state(cell(Cell,cvs(cvpair(CV,undef),CVS),CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if cv-exists(CV,CVS) == false .

--- ゴール列が単項でなく，指定されたセル変数が既に存在し，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('add-cell-var,termlist(Cell,CV)),GL),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('add-cell-var,termlist(Cell,CV)),GL),Empty))
if cv-exists(CV,CVS) .

--- ゴール列が単項でなく，指定されたセル変数が既に存在し，
--- バックトラック情報が空でない場合
crl
```

```
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('add-cell-var,termlist(Cell,CV)),GL),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),Ctrls =/= Empty) .

--- ------------------------------------------------------------------
--- check-cell-var/2
--- 指定されたセルに，指定されたセル変数が存在するかどうかを確認する
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('check-cell-var,termlist(Cell,CV)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if cv-exists(CV,CVS) .

--- ゴール列が単項で，指定されたセル変数が存在せず，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('check-cell-var,termlist(Cell,CV)),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,pred('check-cell-var,termlist(Cell,CV)),Empty))
if cv-exists(CV,CVS) == false .

--- ゴール列が単項で，指定されたセル変数が存在せず，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('check-cell-var,termlist(Cell,CV)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS) == false,Ctrls =/= Empty) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('check-cell-var,termlist(Cell,CV)),GL),
              Ctrls))
=>
state(cell(Cell,CVS,CLS),
```

```
      process(P,EL,loc(1,1),GL,Ctrls))
if cv-exists(CV,CVS) .

--- ゴール列が単項でなく，指定されたセル変数が存在せず，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('check-cell-var,termlist(Cell,CV)),GL),
              Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('check-cell-var,termlist(Cell,CV)),GL),
              Empty))
if cv-exists(CV,CVS) == false .

--- ゴール列が単項でなく，指定されたセル変数が存在せず，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('check-cell-var,termlist(Cell,CV)),GL),
              Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS) == false,Ctrls =/= Empty) .

--- ------------------------------------------------------------------
--- cv-read/3
--- 指定されたセル中のセル変数の値を獲得する.
--- セル変数に値が割り当てられていない場合は割り当てられるまで待つ
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-read,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,loc(1,1),
              subst(eqn(V,get-value(CV,CVS)),nil),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) =/= failure) .

--- ゴール列が単項で，第三引数が変数でなく，
--- バックトラック情報が空の場合
crl
```

```
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-read,termlist(Cell,CV,V)),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,pred('cv-read,termlist(Cell,CV,V)),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure) .

--- ゴール列が単項で，第三引数が変数でなく，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-read,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure,Ctrls =/= Empty) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-read,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,loc(1,1),
              subst(eqn(V,get-value(CV,CVS)),GL),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) =/= failure) .

--- ゴール列が単項でなく，第三引数が変数でなく，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-read,termlist(Cell,CV,V)),GL),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('cv-read,termlist(Cell,CV,V)),GL),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure) .

--- ゴール列が単項でなく，第三引数が変数でなく，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
```

```
              termlist(pred('cv-read,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure,Ctrls =/= Empty) .

--- ---------------------------------------------------------------------
--- cv-ref-wait/3
--- 指定されたセル中のセル変数の値を獲得する.
--- セル変数に値が割り当てられていない場合は割り当てられるまで待つ
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ---------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-ref-wait,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,loc(1,1),
              subst(eqn(V,get-value(CV,CVS)),nil),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) =/= failure) .

--- ゴール列が単項で，第三引数が変数でなく，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-ref-wait,termlist(Cell,CV,V)),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,pred('cv-ref-wait,termlist(Cell,CV,V)),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure) .

--- ゴール列が単項で，第三引数が変数でなく，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-ref-wait,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure,Ctrls =/= Empty) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
```

```
      process(P,EL,Meta,
              termlist(pred('cv-ref-wait,termlist(Cell,CV,V)),GL),
              Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,loc(1,1),
              subst(eqn(V,get-value(CV,CVS)),GL),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) =/= failure) .

--- ゴール列が単項でなく，第三引数が変数でなく，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-ref-wait,termlist(Cell,CV,V)),GL),
              Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('cv-ref-wait,termlist(Cell,CV,V)),GL),
              Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure) .

--- ゴール列が単項でなく，第三引数が変数でなく，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-ref-wait,termlist(Cell,CV,V)),GL),
              Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,
       unify(eqn(V,get-value(CV,CVS))) == failure,Ctrls =/= Empty) .

--- ------------------------------------------------------------------
--- cv-xread/3
--- 指定されたセル中のセル変数の値を獲得し，値を初期化する
--- セル変数に値が割り当てられていない場合は割り当てられるまで待つ
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-xread,termlist(Cell,CV,V)),Ctrls))
=>
```

```
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),nil),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-xread,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),GL),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ------------------------------------------------------------------
--- cv-ref/3
--- 指定されたセル中のセル変数の値を獲得する.
--- セル変数に値が割り当てられていない場合は失敗する
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-ref,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),nil),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ゴール列が単項で，セル変数に値が割り当てられておらず,
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-ref,termlist(Cell,CV,V)),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,pred('cv-ref,termlist(Cell,CV,V)),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ゴール列が単項で，セル変数に値が割り当てられておらず,
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-ref,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef,Ctrls =/= Empty) .
```

```
--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-ref,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),GL),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ゴール列が単項でなく，セル変数に値が割り当てられておらず,
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-ref,termlist(Cell,CV,V)),GL),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,
              termlist(pred('cv-ref,termlist(Cell,CV,V)),GL),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ゴール列が単項でなく，セル変数に値が割り当てられておらず,
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-ref,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef,Ctrls =/= Empty) .

--- ----------------------------------------------------------------------
--- cv-take-wait/3
--- 指定されたセル中のセル変数の値を獲得し，値を初期化する
--- セル変数に値が割り当てられていない場合は待つ
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ----------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-take-wait,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),nil),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ゴール列が単項でない場合
```

```
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-take-wait,termlist(Cell,CV,V)),GL),
              Ctrls))
=>
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),GL),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ------------------------------------------------------------------
--- cv-take/3
--- 指定されたセル中のセル変数の値を獲得し，値を初期化する
--- セル変数に値が割り当てられていない場合は失敗する
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-take,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),nil),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ゴール列が単項で，セル変数に値が割り当てられておらず,
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-take,termlist(Cell,CV,V)),Empty))
=>
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,NoLoc,pred('cv-take,termlist(Cell,CV,V)),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ゴール列が単項で，セル変数に値が割り当てられておらず,
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-take,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef,Ctrls =/= Empty) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
```

```
              termlist(pred('cv-take,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,loc(1,1),subst(eqn(V,get-value(CV,CVS)),GL),Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ゴール列が単項でなく，セル変数に値が割り当てられておらず,
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-take,termlist(Cell,CV,V)),GL),Empty))
=>
state(cell(Cell,reset-value(CV,CVS),CLS),
      process(P,EL,NoLoc,
              termlist(pred('cv-take,termlist(Cell,CV,V)),GL),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ゴール列が単項でなく，セル変数に値が割り当てられておらず,
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-take,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef,Ctrls =/= Empty) .

--- ------------------------------------------------------------------
--- cv-set/3
--- 指定されたセル中のセル変数の値を更新する
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-set,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if cv-exists(CV,CVS) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-set,termlist(Cell,CV,V)),GL),Ctrls))
=>
```

```
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if cv-exists(CV,CVS) .

--- ------------------------------------------------------------------
--- cv-write/3
--- 指定されたセル中のセル変数の値を更新する
--- 指定されたセル変数に値が割り当てられていたら，初期化されるまで待つ
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-write,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-write,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ------------------------------------------------------------------
--- cv-give-write/3
--- 指定されたセル中のセル変数の値を更新する
--- 指定されたセル変数に値が割り当てられていたら，初期化されるまで待つ
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-give-write,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-give-write,termlist(Cell,CV,V)),GL),
```

```
              Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ------------------------------------------------------------------
--- cv-give/3
--- 指定されたセル中のセル変数の値を更新する
--- 指定されたセル変数に値が割り当てられていたら失敗する
--- 指定されたセル変数が存在しない場合，それが生成されるまで待つ
--- ------------------------------------------------------------------
--- ゴール列が単項の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-give,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),nil,Ctrls))
if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

--- ゴール列が単項で，セル変数に値が割り当てられており，
--- バックトラック情報が空の場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-give,termlist(Cell,CV,V)),Empty))
=>
state(cell(Cell,CVS,CLS),
      process(P,EL,NoLoc,pred('cv-give,termlist(Cell,CV,V)),Empty))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

--- ゴール列が単項で，セル変数に値が割り当てられており，
--- バックトラック情報が空でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,pred('cv-give,termlist(Cell,CV,V)),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,Ctrls =/= Empty) .

--- ゴール列が単項でない場合
crl
state(cell(Cell,CVS,CLS),
      process(P,EL,Meta,
              termlist(pred('cv-give,termlist(Cell,CV,V)),GL),Ctrls))
=>
state(cell(Cell,write-value(CV,V,CVS),CLS),
      process(P,EL,loc(1,1),GL,Ctrls))
```

```
  if and(cv-exists(CV,CVS),get-value(CV,CVS) == undef) .

  --- ゴール列が単項でなく，セル変数に値が割り当てられており，
  --- バックトラック情報が空の場合
  crl
  state(cell(Cell,CVS,CLS),
        process(P,EL,Meta,
                termlist(pred('cv-give,termlist(Cell,CV,V)),GL),Empty))
  =>
  state(cell(Cell,CVS,CLS),
        process(P,EL,NoLoc,
                termlist(pred('cv-give,termlist(Cell,CV,V)),GL),Empty))
  if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef) .

  --- ゴール列が単項でなく，セル変数に値が割り当てられており，
  --- バックトラック情報が空でない場合
  crl
  state(cell(Cell,CVS,CLS),
        process(P,EL,Meta,
                termlist(pred('cv-give,termlist(Cell,CV,V)),GL),Ctrls))
  =>
  state(cell(Cell,write-value(CV,V,CVS),CLS),
        process(P,EL,fst(head(Ctrls)),snd(head(Ctrls)),tail(Ctrls)))
  if and(cv-exists(CV,CVS),get-value(CV,CVS) =/= undef,Ctrls =/= Empty) .

endm
--- ------------------------------------------------------------------------
--- End of Definition
--- ------------------------------------------------------------------------
```

Appendix B: 例題 —自律ロボット—

本研究で与えた操作的記述に基づいた，自律ロボットの例題の設定と実行例について述べる．

自律ロボットは，自らの状況を判断し，モード変更によって自分の取るべき行動を変えていく．ロボットが取り得るモードは，

- proceed モード：道路を一歩ずつ進む
- want モード：交差点の入口で止まり，交差点内の状況を確認する．交差点に誰もいなかったら enter モードになる．誰かいたら stop モードになる．
- enter モード：交差点に入り，一歩ずつ進む．交差点の出口に到着したら交差点から出る．
- stop モード：交差点の入口で停止する．交差点内の状況を確認し，誰もいなければ want モードになる．

の 4 種類である．それぞれのモード時において取るべき動作は述語として定義されている．それらの述語のヘッド部はすべて同一のパターン e() で定義されている．そのため，ロボットが proceed モードにある場合に e() を実行した場合と，wait モードにある場合に e() を実行した場合とでは，対応するボディ部が異なっているので，モード固有の動作が観測される．

この例題では，モードはセルとして表現されている．モードに対応する動作は，セル内に述語として定義されている．また，ロボットはプロセスで表現されている．この場合，ロボットのモードはプロセスの環境，特に環境の先頭部分に対応している．

以下は自律ロボットの例題を実行するための項の初期値である．

```
state(
--- proceed モード
```

```
cell('proceed,NoCVPair,
     clauselist(cldef(pred('e,nil),
                      pred('loc,5),
                      termlist(!,pred('name,'M),
                      pred('output,pred('want-enter,'M)),
                      pred('subst-cell,termlist('want,'proceed)))),
                cldef(pred('e,nil),
                      pred('loc,8),
                      termlist(!,pred('name,'M),
                               pred('output,pred('finished,'M)),
                               pred('cv-take,termlist('M,'loc,'None1)),
                               pred('cv-take,termlist('M,'iloc,'None2)),
                               pred('die,nil))),
                cldef(pred('e,nil),
                      nil,
                      termlist(!,pred('inc,'loc),pred('loc,'L),
                      pred('name,'M),
                      pred('output,pred('at,termlist('M,'L)))))))),
--- want モード
cell('want,NoCVPair,
     clauselist(cldef(pred('e,nil),pred('find,'entered),
                      termlist(!,pred('output,'someone-there),
                      pred('subst-cell,termlist('stop,'want)))),
                cldef(pred('e,nil),nil,
                      termlist(!,pred('enter,nil),
                      pred('subst-cell,termlist('enter,'want)))),
                cldef(pred('enter,nil),nil,
                      termlist(!,pred('name,'M),
                      pred('output,pred('entering,'M)),
                      pred('message,'entered),
                      pred('cv-set,termlist('M,'iloc,1)))))),
--- enter モード
cell('enter,NoCVPair,
     clauselist(cldef(pred('e,nil),pred('iloc,3),
                      termlist(!,pred('name,'M),
                      pred('output,pred('exiting,'M)),
                      pred('succeed,
                           pred('cv-take,termlist('com,'message,'None))),
                      pred('inc,'loc),
                      pred('subst-cell,termlist('proceed,'enter)))),
                cldef(pred('e,nil),nil,
                      termlist(!,pred('inc,'iloc),pred('iloc,'LL),
                      pred('name,'M),
                      pred('output,pred('inside,termlist('M,'LL)))))) ,
--- stop モード
cell('stop,NoCVPair,
     clauselist(cldef(pred('e,nil),pred('not,pred('find,'None3)),
                      termlist(!,pred('output,'resume),
                      pred('subst-cell,termlist('proceed,'stop)))),
                cldef(pred('e,nil),nil,
                      termlist(!,pred('name,'M),
                               pred('output,pred('waiting,'M))))) ,
--- 述語の定義 (ユーティリティ)
```

```
cell('main,NoCVPair,
     clauselist(cldef(pred('test,nil),nil,
                 termlist(!,pred('fork,pred('agent,termlist('a,1))),
                             pred('fork,pred('agent,termlist('b,1))))),
           cldef(pred('agent,termlist('M,'Loc)),nil,
                 termlist(!,pred('new-cell,termlist('M,'loc,'iloc)),
                 pred('push,'M),
                 pred('or ,termlist(pred('goals ,pred('name,'M)),
                 pred('goals,pred('assert,pred('name,'M))))),
                 pred('cv-write,termlist('M,'loc,'Loc)),
                 pred('push,'proceed),
                 pred('output,pred('initializing,termlist('M,'Loc))),
                 pred('repeat,pred('e,nil)))) ,
           cldef(pred('find,'Mode),nil,
                 termlist(!,pred('cv-ref,termlist('com,'message,
                                 pred('pair,termlist('Name,'Mode)))),
                 pred('name,'M),
                 pred('not,pred('eq,termlist('Name,'M))))) ,
           cldef(pred('message,'Mes),nil,
                 termlist(!,pred('name,'M),
                 pred('cv-write,termlist('com,'message,
                                        pred('pair,termlist('M,'Mes)))),
                      pred('output,pred('pair,termlist('M,'Mes)))) ,
           cldef(pred('loc,'L),nil,
                 termlist(!,pred('name,'M),
                 pred('cv-read,termlist('M,'loc,'L)))) ,
           cldef(pred('iloc,'LL),nil,
                 termlist(!,pred('name,'M),
                  pred('cv-read , termlist('M,'iloc,'LL)))) ,
           cldef(pred('inc,'V),termlist(pred('name,'M),
                 pred('cv-take,termlist('M,'V,'LV))) ,
                 termlist(!,pred('add,termlist('LV,1,'L1)) ,
                 pred('cv-set,termlist('M,'V,'L1)))) ,
           cldef(pred('output,'M),nil,termlist(!,pred('write,'M)))) ,
--- 述語の定義 (システム提供のユーティリティ)
cell('stdlib,NoCVPair,
     clauselist(cldef(pred('once,'Goal),nil,termlist('Goal,!)),
           cldef(pred('succeed,'Goal),nil,termlist('Goal,!)),
           cldef(pred('succeed,'None4),nil,nil),
           cldef(pred('not,'Goal),pred('once,'Goal),termlist(!,fail)),
           cldef(pred('not,'None5),nil,nil),
           cldef(pred('or,termlist(pred('goals,'X),
                                   pred('goals,'Y))),nil,'X),
           cldef(pred('or,termlist(pred('goals,'X),
                                   pred('goals,'Y))),nil,'Y),
           cldef(pred('repeat,'Goal),nil,
                      termlist(pred('repeat-try,'Goal),fail)),
           cldef(pred('repeat-try,'Goal),pred('not,'Goal),fail),
           cldef(pred('repeat-try,'Goal),nil,
                 termlist(!,pred('repeat-try,'Goal)))),
--- セル名, プロセス名のデータベース
cell('system-cell,cvs(cvpair('a,'newa),cvpair('b,'newb)),
     clauselist(cldef(pred('cell,'proceed),nil,!),
```

```
            cldef(pred('cell,'want),nil,!),
            cldef(pred('cell,'stop),nil,!),
            cldef(pred('cell,'enter),nil,!),
            cldef(pred('cell,'main),nil,!),
            cldef(pred('cell,'stdlib),nil,!) ,
            cldef(pred('cell,'system-cell),nil,!),
            cldef(pred('cell,'display),nil,!) ,
            cldef(pred('process,'main),nil,!))) ,
--- 述語 write/1 の出力用
cell('display,NoCVPair,NoClause),
--- スレッド間通信用セル
cell('com,cvpair('message,undef),NoClause),
--- プロセス
process('main,environment('main,'stdlib,'system-cell),loc(1,1),
        pred('test,nil),Empty)) .
```

ここで，セル 'display は引数を標準出力する述語 write/1 の出力情報を述語形式に変換して，節定義部分に保存するために設けた構造である．

先に記述した項は，モードを表現するセル，自律ロボットを生成するプロセスを，結合的および交換的属性を持つ構成子state によって組みあげられており，GAEA システムの計算状態を表現している．構成子 state によって組みあげられた項を，本研究で与えた状態遷移規則によって遷移させること，具体的には，計算状態を表現している項の

- プロセス，特に，ゴール列
- セル，特に，セル変数の値

を書き換えることによって計算が実行される．

初期項においては，プロセスを表現する項は

```
process('main,environment('main,'stdlib,'system-cell),loc(1,1),
        pred('test,nil),Empty)
```

のみであるが，セル 'main の述語 pred('test,nil) を実行することにより，プロセスを生成する GAEA の組み込み述語pred('fork,GL) が 2 度呼ばれる．この述語に対応する状態遷移規則を適用すると 2 つのプロセス

```
process('newa,environment('main,'stdlib,'system-cell),loc(1,1),
        pred('agent,termlist('a,1)),Empty)
process('newb,environment('main,'stdlib,'system-cell),loc(1,1),
        pred('agent,termlist('b,1)),Empty)
```

が生成される．これらのプロセスはおのおの自律ロボットを生成し，自身の環境を動的に変化させながら各モードに定義されている述語 pred('e,nil) を実行する．

以下に状態遷移が停止したときの項の一部分を表示する．モードを表現しているセルは初期項から変化していない．プロセス 'main は

```
process('main,environment('main,'stdlib,'system-cell),NoLoc,nil,Empty)
```

と変化する．動的に生成された自律ロボットを表現するプロセスは，消去されており存在しない．

セル 'display の節定義部分には，自律ロボットが一歩ずつ道路を移動し，交差点に侵入し，その中を一歩ずつ進み，交差点から出て，再び道路を一歩ずつ移動し，所定の位置に到着した時点でロボットが消滅するといった様子が保存されている．移動情報は，下から上へ向かう順で読む．

```
cell('display, NoCVPair,
clauselist(cldef(pred('finished, 'b), nil, nil),
       cldef(pred('at, termlist('b, 8)), nil, nil),
       cldef(pred('at, termlist('b, 7)), nil, nil),
       cldef(pred('exiting, 'b), nil, nil),
       cldef(pred('inside, termlist('b, 3)), nil, nil),
       cldef(pred('inside, termlist('b, 2)), nil, nil),
       cldef(pred('pair, termlist('b, 'entered)), nil, nil),
       cldef(pred('entering, 'b), nil, nil),
       cldef(pred('want-enter, 'b), nil, nil),
       cldef(pred('at, termlist('b, 5)), nil, nil),
       cldef(pred('at, termlist('b, 4)), nil, nil),
       cldef(pred('at, termlist('b, 3)), nil, nil),
       cldef(pred('at, termlist('b, 2)), nil, nil),
       cldef(pred('initializing, termlist('b, 1)), nil, nil),
       cldef(pred('finished, 'a), nil, nil),
       cldef(pred('at, termlist('a, 8)), nil, nil),
       cldef(pred('at, termlist('a, 7)), nil, nil),
       cldef(pred('exiting, 'a), nil, nil),
       cldef(pred('inside, termlist('a, 3)), nil, nil),
       cldef(pred('inside, termlist('a, 2)), nil, nil),
       cldef(pred('pair, termlist('a, 'entered)), nil, nil),
       cldef(pred('entering, 'a), nil, nil),
       cldef(pred('want-enter, 'a), nil, nil),
       cldef(pred('at, termlist('a, 5)), nil, nil),
       cldef(pred('at, termlist('a, 4)), nil, nil),
       cldef(pred('at, termlist('a, 3)), nil, nil),
       cldef(pred('at, termlist('a, 2)), nil, nil),
       cldef(pred('initializing, termlist('a, 1)), nil, nil)))
cell('a, cvs(cvpair('loc, undef), cvpair('iloc, undef)),
     cldef(pred('name, 'a), nil, !))
cell('b, cvs(cvpair('loc, undef), cvpair('iloc, undef)),
     cldef(pred('name, 'b), nil, !))
```

セル 'a および 'b は各自律ロボットの位置情報を保存しておくために動的に生成されたセルである．セル変数 'loc には，道路に置ける位置情報，セル変数 'iloc には，交差点内における位置情報が保存されていたが，自律ロボットを消去する際に，位置情報を保存するセルは消去せずに，それらの値は初期化するようにプログラムされているので，上記のような項が残っている．