日本ソフトウェア科学会第 29 回大会 (2012 年度) 講演論文集

複数の計算モデルをサポートするモデル検査器の実現

目黒 学　谷口 直輝　上田 和紀

LMNtal は階層グラフの書き換えによって計算をすすめる言語であり，実行時処理系は並列 LTL モデル検査器を備えている．これまでに λ 計算をはじめ，Petri Net，Ambient 計算，CHR，MSR，最適化問題やリアルタイムシステムなどさまざまなモデルをエンコードし検証してきた．近年では新たなデータ構造の導入による階層ハイパーグラフ書き換え言語への拡張を行いさらに表現力が向上した．本研究ではより統合的に計算モデルを扱うため，階層ハイパーグラフの状態空間構築処理の設計・実装を行うとともに，計算モデルを統合した meta-calculus である Bigraph をエンコードする手法について検討する．本論文では Bigraph は LMNtal によって構造の大半が表現可能で，拡張によりその機能全般をエンコード可能であることを論じる．そして拡張された LMNtal によって複数の計算モデルをサポートする枠組みについて解説する．

1 はじめに

社会に IT（情報技術）システムの役割が大きくなる一方で，個々のシステムの役割はブラックボックス化され複雑化する一方である．そのような環境でシステムをモデル化し性質を検証することはモデルの理解や安全性・信頼性を証明するうえで非常に重要である．一方計算機科学では多くの計算プロセスを扱うモデルが考案され，これらの複雑なシステムを理解する上で有用である．しかしこれらの計算モデルは数多く存在し，それらを統一的に扱うモデルは少なく個々に検証が必要である．Bigraph [11] はこれらの計算モデルを統一的に扱う meta-calculus として考案された．代数学的記法だけでなく可視化できるグラフの記法も形式化され，多くの並行プロセスが表現可能である．理論的には確立しているもののその理論が実装されている例はあまり多くない．その中でも BPLTool [13] や DBtk [14] は Bigraph の構造の表現やマッチングにおいて有用なツールであるが，性質の検証は行えない．Bigraph のモデル検査器として開発されている BigMC [15] では Bigraph の構造を忠実に再現し，可視化機能も備えているが一部の構造が表現できなかったり性質の表現が限定されている．一方 LMNtal では様々は計算モデルを階層グラフとして記述し可視化・検証可能な統合開発環境を備えた言語である．本研究ではこの Bigraph をエンコードし検証可能となるように LMNtal を拡張することで，複数の計算モデルをサポートするモデル検査器を構築することを目的とする．

LMNtal [1] [2] は階層グラフ書き換えに基づく並行言語モデルであり，さまざまな計算モデルを統合的に扱うことを目指している．近年 LMNtal は新たなデータ構造の導入によってハイパーグラフをサポートする HyperLMNtal へ拡張された [9] [10]．従来の膜，アトム，リンクという構造にハイパーリンクを加えた階層ハイパーグラフ構造によって，より表現力が向上した．LMNtal は実行時処理系として並列 LTL モデル検査器 SLIM を備えている [8]．LTL モデル検査とは線形時相論理で記述したモデルの性質を状態を網羅的に探索し検証する形式的手法であり，SLIM は共有メモリ環境において動的負荷分散を行い，マル

Model checker for multiple models of computation

Manabu Meguro, Yaguchi Naoki, 早稲田大学理工学部情報学科, Graduate School of Fundamental Science and Engineering, Waseda University.

Kazunori Ueda, 早稲田大学理工学術院情報理工学科, Faculty of Science and Engineering, Waseda University.

チスレッドによる実行に対応している．さらにこれらの機能を包含した統合開発環境 LaViT [7] では通常実行・非決定実行ともに可視化環境が提供され，WEB 上で公開されている．このように類似した言語はあるものの，階層グラフや状態空間の可視化，並列モデル検査の機能が統合された言語は他にない強みである．

過去に LMNtal によってさまざまなモデルがエンコードされ，λ 計算 [5]，π 計算，Ambient 計算 [4]，CHR，Petri Net，リアルタイムシステム [6] など多岐に渡る．しかし Bigraph についてはハイパーグラフ構造を扱う上で記述や状態空間が冗長になり実現していなかった．そこで本研究では Bigraph の構造を，新たに導入されたハイパーリンクでエンコードし検証を行う．そのためにハイパーリンクを管理できる状態空間構築処理の設計・実装を行い階層ハイパーグラフの書き換えをモデル検査に対応させた．そしてハイパーリンクを利用した Bigraph の構造と反応ルールをエンコードする手法の検討を行い実現した．さらに全ての制御構造と反応規則をエンコードするにあたって必要な拡張について検討した．本論文では非決定実行の詳細と LMNtal による Bigraph のエンコード手法および LMNtal の拡張機能について解説を行う．さらに LMNtal によって複数の計算モデルをエンコードする手法について考察する．

2 LMNtal

2.1 並行言語 LMNtal

LMNtal は階層グラフ書き換えに基づく並行言語モデルである．階層グラフは**アトム**，**リンク**，**膜**で構成され，このような構造を**プロセス**と呼ぶ．**ルール**によってプロセスを書き換えることで処理を進める．ルールは

$$Head \ :\text{-} \ Guard \mid Body$$

のように記述できる．書き換え対象のプロセスが *Head*，条件を *Guard*，書き換え後が *Body* である．ルールにおいて，書き換え対象のプロセスを探索する処理を**マッチング**という．ルール適用は書き換え可能なプロセスがなくなるまで行われ，どのルールを適用するか，どのプロセスを書き換えるかは非決定的である．膜にはプロセスやルールのワイルドカードを記述することが可能でそれぞれ**プロセス文脈**及び**ルール文脈**という．これらの記述によって膜はプロセスやルールを局所化しひとまとめにして扱うことが可能である．例えば

```
open(X), {in(X), $p} :- $p.
```

は `open` と膜の中の `in` がリンク `X` によって接続されているプロセスを探索し，膜の中の `in` 以外のプロセスを膜外に出すルールである．さらに局所化によって膜内のルールは膜外には適用されない．プロセス文脈の拡張として，**型付きプロセス文脈** [3] がある．これは膜の中以外にも記述することができ，*Guard* 部で型検査を行うことができる．

```
cnt(X), $p[X] :- int($p), $p2=$p-1 | cnt($p2).
```

は `cnt` に接続するアトムが整数型であることを *Guard* 部で検査している．さらにこの記述は

```
cnt($p) :- int($p) | cnt($p-1).
```

のように表記することができる．主要な型を表 1 に示す． ground 型は unary 型を，unary 型は int 型と float 型を含んでいるので，int 型と float 型も一価のアトムであり ground 型は最も制約が弱い．プロセス文脈と同様に型付きプロセス文脈も，ルール上に明示的に記述されない構造の操作を行うための文法として LMNtal の表現力の向上のために導入された．

表 1 主要なプロセス型

int	整数型のアトム
float	浮動小数点数型のアトム
unary	価数が 1 のアトム
ground	膜を含まないリンクによって接続されたアトムの集合

予約アトム名 `=` は**コネクタ**と呼ばれ，`X = Y` はリンク `X` の一端と `Y` の一端を接続し，`X=a(Y)` は `a(X,Y)` と同義である．また略記法として，*Head* 部に記述された `\` 以前のプロセスはコンパイル時に *Body* 部にもコピーされる．

LMNtal 処理系では通常実行とは別に全ての状態，実行経路を探索する非決定実行を行うことができる．非決定実行によって全ての状態が状態空間として保存

(生成)	*Head* `:-` `new($x)` \| *Atom*(`$x`)
(型制約)	*Atom*(`$x`) `:-` `hlink($x)` | *Body*
(併合)	*Atom*(`$x`), *Atom*(`$y`) `:-` `$x` | `>< $y`
(要素数取得)	*Atom*(`$x`) `:-` `$n=num ($x)` | *Body*
(探索)	*Atom*(`$x`), *Atom*(`$x`) `:-` | *Body*

図 1　ハイパーリンク関連の記法

され，この状態空間を網羅的に探索し LTL（線形時相論理）で記述した性質の検証を行うモデル検査器を備えている．

3　HyperLMNtal への拡張

LMNtal のリンクは一対一の接続に特化されているが，モデルを記述する際に一対一ではない接続を表現したいときがある．例えば 3 つのノード `a, b, c` に接続関係を持たせるとき，従来の LMNtal では

```
a(X),b(Y),c(Z),{+X,+Y,+Z}
```

のように膜を介した接続や

```
a(link1),b(link1),c(link1)
```

のように同じ名前のアトムを持たせて疑似的に接続を表現する方法でモデル化していた．このような記述はあまり自然ではなく，さらに前者では記述が複雑化し，後者は接続を持たないのでルール適用の際マッチングの効率が悪い．

このため，LMNtal のリンクを拡張した**ハイパーリンク**を導入し階層グラフから階層ハイパーグラフへの拡張を行った [9] [10]．ハイパーグラフとはグラフを一般化したものでエッジが任意個数のノードを接続可能なグラフである．ハイパーリンクの設計に関しては並行制約プログラミング言語である CHR の論理変数 [16] を参考にしている．

3.1　HyperLMNtal の実装

HyperLMNtal の実装は

(1) ハイパーリンク型・演算の導入

(2) 状態空間構築処理への対応

に分けられる．ハイパーリンクの概念は従来のリンクを拡張したものだが，実装ではプロセス型のひとつである unary 型のアトムを拡張したハイパーリンク

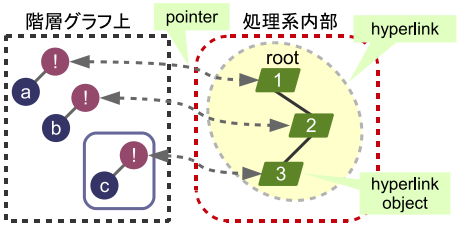

図 2　ハイパーリンクの概要図

アトムによって実現された．図 1 に示すようにハイパーリンク型の制約，演算は既存の型を拡張することで導入した．ハイパーリンクの構造の概要図を図 2 に示す．図のようにハイパーリンクの併合関係は処理系の内部ではハイパーリンクオブジェクトの木構造として扱われている．設計概念や CHR との関連，実装の詳細は文献 [9] 及び [10] に委ねる．さらに簡単のため，これ以降ハイパーリンク型へと接続するリンクを $!X$ として表記する．

ハイパーリンクが導入されたことによって Bigraph のハイパーグラフ構造を表現することが可能になった．しかしエンコードしたモデルをモデル検査するためには状態空間の構築が必要である．そこで本研究ではまず階層ハイパーグラフを状態空間で管理する手法の設計・実装を行った．LMNtal 実行時処理系 [8] は，状態空間構築において状態をバイト列に圧縮し保存することでメモリ使用量の大幅な削減を行っている．さらに状態登録の際には新たに登録する階層グラフに対してハッシュ値を生成し，既に保存されている階層グラフのハッシュ値と衝突した場合にはグラフ同型性判定を行う．ハイパーリンクを状態空間構築処理に導入するために行った大きな変更として，

(a) ハッシュ値の生成

(b) 階層ハイパーグラフからバイト列への圧縮

(c) ハイパーリンクを含んだグラフ同型性判定

が挙げられる．まず (a) においては，ハッシュ値はハイパーリンクに接続されたアトムの個数を利用している．(b) については，ハイパーリンクの接続は代表となるハイパーリンクに対してその参照となる番号をタグとともに保存することでバイト列への変換を行っている．(c) では状態空間に保存されたバイト列と階

層グラフとの同型性判定を行うため，バイト列上に保存されたハイパーリンクの集合と同型のものが階層グラフ上にも存在するか判定する．これはハイパーリンクの出現位置とそれが接続する代表のハイパーリンク，全てのハイパーリンクアトムの個数の一致を確認することで実現している．(a) についてはハイパーリンクの個数ではなく接続しているアトムの構造を反映したハッシュ値を生成することが今後の課題である．

3.2 性能評価

ハイパーリンクによる接続を利用したマッチング処理により，表現力だけでなく既存の LMNtal で表現したモデルと比べ性能も向上している．通常実行における性能の向上については文献[9] 及び[10] に委ねる．

非決定実行の性能測定について表 2 から表 6 に示す．実験環境のハードウェアは Intel(R) Core(TM)2 Quad 2.60GHz, RAM 4GB, OS は ubuntu 10.04 である．表 2 は既存の λ 計算をエンコードした例題[4] において，データ構造の管理をハイパーリンクで行うように修正したものと既存版との比較である．ハイパーリンク版は状態数よりも生成されたハッシュ値がわずかに少ないのでハッシュ値の衝突がおきていると考えられるが，比率としてはほとんど衝突していないといえる．β 簡約の際に膜で行っていた構造の管理をハイパーリンクで行うことで実行時間を減らし構造が単純化したことでメモリ使用量を減らすことができた．表 3 は素集合の高速な管理手法である Union-Find algorithm を LMNtal で実装した例題である．表中の項目の膜版というのは膜を介することでハイパーグラフを表現した場合について行った性能測定の結果で，アトム版はアトムを持たせることで疑似的に接続を表現した場合の実行性能である．ハイパーリンク版では木構造の要素をハイパーリンクで表現し，各命令で共有することで状態数を大きく削減し，非常に少ない実行時間となっている．表 4 は与えられたグラフ上から 3 頂点からなる閉路を探索し消去する問題である．アトム版では接続は数値アトムで疑似的に行っているので，状態数はアトムの数だけ増えメモリ使用量が増加する．膜版の場合には状態数は増えないが膜の構造は生成や削除のコストが高いので，実行時間が遅くなる．ハイパーリンク版は状態数とメモリ使用量は少ないがハッシュ値の衝突によって実行時間はやや遅くなっている．表 5 は 1...N の値を持つ A, B の制約の集合から同じ値を持つ制約を探索する例題である．アトム版は A, B の接続はないが，膜版およびハイパーリンク版は A, B が値を共有し接続構造を持っているためマッチングの効率が良い．この例題ではアトム版はマッチングの効率が悪く非常に時間がかかるので測定していない．膜版とハイパーリンク版は状態数は一致するが，膜の処理のオーバヘッドで遅くなっている．表 5 はアドレスを持つメモリ領域に格納されているデータを，命令とプログラムカウンタの値によって操作する例題である．膜版は構造が複雑化によって記述が困難と考え実装を行っていない．この例題ではハイパーリンク版がデータをハイパーリンクで共有することによって状態数を減らしているが，ハッシュ値の衝突が多く実行時間はアトム版よりも遅い．

全体としてみると，ハイパーリンク版はハイパーリンクの接続を利用することで実行時間の面でも状態数やメモリ使用量の面でも優れている．しかしハッシュ値の衝突が多く実行時間で遅くなっている例題も多い．3.1 節で触れたように，ハッシュ値の改良は今後の課題である．

4 Bigraph

Bigraph[11][12] は Milner らによってさまざまな計算モデルを統一的に扱うことができる meta-calculus として提唱された．ユビキタスコンピューティングにおける複雑なシステムを理解する手段となることを目的としてる．Bigraph の構造を図 3 に示す．

Bigraph は複数の木構造を持つ **place graph** およびハイパーグラフ構造を持つ **link graph** によって構成されている．木構造では局所性と包含性を表し，ハイパーグラフでは接続や関係を表現している．Place graph はノード同士がネストすることによって構成され，link graph はノード同士が **edge** によって接続されることで構成される．Place graph と link graph は外側や内側へのインターフェースを持ち，それぞれを **outer face**, **inner face** という．Place graph の場

表 2　λ 計算 3^2 の実験結果

	状態数	異なるハッシュ値の個数	実行時間 (s)	メモリ使用量 (MB)
ハイパーリンク版	63998	63562	18.46	32.86
既存版	63998	63940	22.78	37.91

表 3　Union-Find Algorithm の実験結果

	状態数	異なるハッシュ値の個数	実行時間 (s)	メモリ使用量 (MB)
ハイパーリンク版	393	393	0.04	0.11
膜版	1211350	1211350	146.6	256.6
アトム版	1211351	1211351	46.77	222.9

表 4　閉路検出問題の実験結果

	状態数	異なるハッシュ値の個数	実行時間 (s)	メモリ使用量 (MB)
ハイパーリンク版	2140	1028	57.13	0.84
膜版	2141	2141	299.7	0.93
アトム版	32802	32802	50.16	10.68

表 5　制約集合探索の実験結果

	状態数	異なるハッシュ値の個数	実行時間 (s)	メモリ使用量 (MB)
ハイパーリンク版	1792	1792	8.07	0.87
膜版	1792	1792	15.55	0.83

合は outer face は木構造の親とのインターフェースである **region**，inner face はノードの子に対するインターフェースである **site** である．Link graph では outer face は **outer name** でこれは region の外側へとつながるリンクで，inner face はノードの内側の site へとつながる **inner name** である．各ノードは arity，識別名を持っていて，識別名のことを **control** という．**arity** はノード同士が edge によって結合できる **port** の個数を表す．ノードは control によって **active** もしくは **passive** であるかが決まる．あるノードが acitve であるか passive であるかによって place graph においてそのノードの子孫であるノードに reaction rule が適用可能かどうかが決まる．

Bigraph は $R \to R'$ という形式の **reaction rule** によって動的に書き換えられる．R は **redex** で active な文脈の中の構造に対してマッチする，R' は **reactum** でマッチした構造を置き換えるものである．Reaction rule 上では region は木構造の任意の場所にマッチさせることができ，site は任意の子・兄弟にマッチさせることができる．

Bigraph には抽象構文が存在するが，具体的な構文は確立していない．そこで Bigraph のモデル検査器として開発されている BigMC[15] の構文に一部手を加えたものを扱う．その構文を図 4 に示す． 構文は control の宣言，Bigraph の記述，reaction rule の三つが並んだものである．N はノードを表し，ノード名である control は c で表される．control の宣言では passive, active の属性と arity を定義する．“ . ”によってノードのネストを表現し，“`|` ”はノードが同じ region に併置されていることを意味する．例として `A. B` は `B` をネストする `A` を表現し，`C.(A | B)` は `A` と `B` が `C` の中にネストされている構造を表現する．`A.nil` は `A` には何も含まれていないことを表す．Edge および inner name や outer name は構文では *names* に対応している．例えばノード同士が *names* によって接続していればその部分は edge と考えることができ，

表 6 RAM simulator の実験結果

	状態数	異なるハッシュ値の個数	実行時間 (s)	メモリ使用量 (MB)
ハイパーリンク版	7217	6025	20.43	2.01
アトム版	25259	25259	2.81	14.60

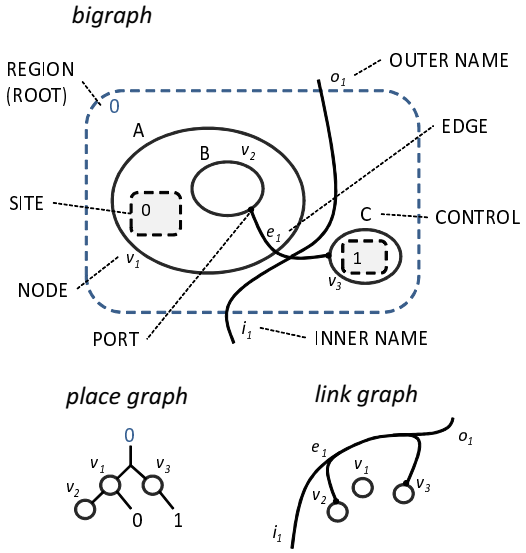

図 3 bigraph の構造

$M ::= S;M \mid S;$
$S ::= \texttt{\%passive}\ c{:}arity \mid \texttt{\%active}\ c{:}arity$
$\quad \mid B \mid B\texttt{->}B$
$B ::= N.B \mid B\texttt{|}B \mid B\texttt{||}B \mid \$p \mid N \mid \texttt{nil}$
$\quad \mid names\texttt{/}names$
$N ::= c\texttt{[}names\texttt{]} \mid c$
$p ::= n\texttt{[}names\texttt{]} \mid n$
$names ::= n\texttt{,}\ names \mid n$
$c, n ::= [\texttt{a}-\texttt{zA}-\texttt{Za}][\texttt{a}-\texttt{zA}-\texttt{Z0}-\texttt{9}]*$

図 4 Bigraph の構文

site とつながっていればその接続部分は inner name である．三つの引数を持つノード A は `A[x,y,z]` のようにあらわされ，link graph はハイパーグラフなので，同一の *names* に複数の port を接続することができる．`x,y,z` は `A` 以外のノードに接続されている場合も含まれているので，edge と outer name の両方を表現していることになる．異なる *names* の接続は *names*/*names* で表す．“ $n ” は site を表す．2 つ以上の region をもつ bigraph は reaction rule の中のみで許可され，そのルールは **wide** と呼ばれる．異なる region を区切るときには “ || ” で表現する．一つの region のみを扱うルールを **prime** と呼ぶ．

5 Bigraph のエンコード

LMNtal によって Bigraph を表現するためには，

(a) Bigraph の構造
  (a-1) place graph: node のネスト
  (a-2) link graph: node の接続
(b) reaction rule
  (b-1) prime: 単一の region における反応
  (b-2) wide: 複数の region における反応
  (b-3) passive, active: 反応の制御構造

を表現することが必要である．LMNtal による Bigraph の表現方法は複数通り考えられるがそのうち最も表現力の高かった二通りの方法について次節で紹介する．

5.1 方法 1: 膜による place graph の表現

最も自然な方法は膜によってノードを表現する方法である．各ノードを膜で表現する方法を方法 1 とし，図 5 に方法 1 を用いた場合の構造を，表 7 に Bigraph の構造と LMNtal による表現の対応を示す．

方法 1 では各ノードは膜で表現され，ノードの識別名である control はアトムで表現される．この方法ではノードのネストを膜のネストで記述でき，site も膜のワイルドカードであるプロセス文脈で表現できる．Link graph はハイパーグラフなので新たに導入されたハイパーリンクを使って表現する．接続点である port は control アトムの引数とする．Link graph の構築，すなわち *names* はハイパーリンクで表現する．

表 7　方法 1 の Bigraph との対応

Bigraph の構造	図 4 との対応	LMNtal による表現
node	N	膜，アトム
control	c	アトム名
outer name inner name edge	$names$	ハイパーリンク
site	$\$n$	プロセス文脈

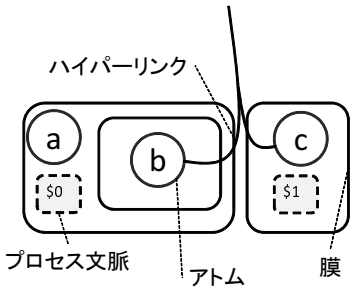

図 5　方法 1 を用いた構造

5.1.1　wide と prime

Bigraph の reaction rule はどの region にあるノードに対しても有効であるが，LMNtal では膜はルールの局所化の役割を持っているので，ルールの存在する階層によって制限が与えられる．そこで reaction rule を通常のルールではなくシステムルールセットのユーザ定義によって記述した．システムルールセットは処理系によってすべての膜内に配置される組み込みのルールセットで，算術演算などがあらかじめ組み込まれている．`system_ruleset` という名前のアトムが存在する膜内にルールを記述することでユーザによって定義することができる．システムルールセットとして reaction rule を記述すると，例として

```
Room.(Alice | Bob) -> Room.Bob | Alice;
```

のように単一の region のみ出現するルール (prime) の場合，

```
{room,{alice},{bob}} :- {room,{bob}},{alice}.
```

と表現することができる．一方複数の region が出現するルール（wide）の場合，LMNtal では複数の階層

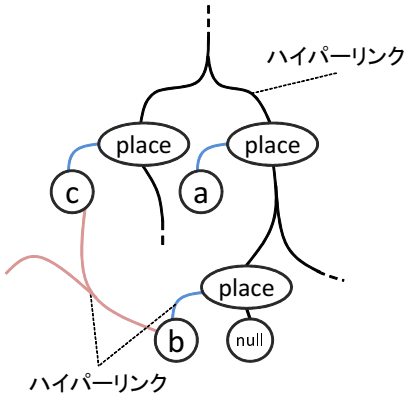

図 6　方法 2 を用いた構造

の異なるプロセスが存在することになる．前述したようにルールは膜によって局所化されるので，複数の階層をまたがる書き換えは行うことができない．方法 1 では Bigraph の構造を簡潔に表現できるが wide なルールは表現することができない．

5.2　方法 2:　ハイパーリンクを用いた place graph の表現

もうひとつの方法として，ハイパーリンクを用いた place graph の表現方法を考案した．Bigraph の構造と LMNtal の要素の対応を表 6 に示す．　この構

表 8　方法 2 の Bigrpah との対応

Bigraph の構造	図 4 との対応	LMNtal による表現
node	N	ハイパーリンクで接続されたアトムの集合
control	c	アトム名
outer name inner name edge	$names$	ハイパーリンク
site	$\$n$	ハイパーリンク

造ではノードを同一のハイパーリンクで接続されたアトムの集合とする．アトムの集合は place アトムや control を表すアトム，さらに passive や active を

表現するアトム (5.5 節) を含む．図 6 のように place graph は place アトム同士をハイパーリンクで接続することによって表現している．place アトムは第一引数は親，第二引数は子，第三引数はノードを表す集合へとつながるハイパーリンクを持った，arity が 3 のアトムである．Link graph は方法 1 と同様にハイパーリンクを使って表現する．図 6 において，`c` と `b` を直接つなぐハイパーリンクはハイパーグラフ構造，つまり *names* を表現している．

5.2.1 wide と prime

この方法を用いることによって，place graph は LMNtal において同じ階層に展開されるので，どの region 同士でもルール適用を行うことができる．例えば 5.1.1 節で扱った prime なルールは

```
!R=room,!R=place(!P1,!P2),
!B=bob,!B=place(!P2,null),\
!A=alice,!A=place(!P2,null)
:- num(!P2)=3 |
!A=alice,!A=place(!P1,null).
```

と記述する．`!P1`, `!P2` はツリー構造，`!R`, `!A`, `!B` はノードを表現するハイパーリンクである．`room` が保持する place アトムに `bob`, `alice` の place アトムが接続されているので，親子関係，兄弟関係が *Head* 部でチェックできている．*Guard* 部では `P2` に接続するアトムの個数，つまり `room` の子の個数と `room` 自身を合わせた個数である．ここでは `room` に `alice` と `bob` 以外の子がいないことを確認している．`alice` を `room` の外に出すことは `room` の親につながるハイパーリンクと `alice` の place アトムの第一引数を接続することで実現できる．二つのノードが別の region に存在することはそれぞれの place アトムの第一引数が異なることを示せばよいので，wide なルールについても表現することができる．具体的には，

```
A || B -> ;
```

のように複数の region がルール中に出現する wide ルールの場合，

```
!A=a,!A=place(!P1, null),
!B=b,!B=place(!P2, null) :-
!P1 \== !P2 | .
```

と表現できる．これは `A` と `B` の place アトムの第一引数が異なるハイパーリンクに接続されていることで `A` と `B` が異なる region に接続されていることを表している．

5.3 Site への reaction

Site への reaction の適用はルール上で明示的に出現しない構造の書き換えであり，具体的に構造が出現する場合に比べて表現が難しい．この書き換えはグラフ構造の移動，複製，もしくは削除を行うものである．まず方法 1 の場合 Bigraph の site を LMNtal のプロセス文脈によって表現している．2.1 節で述べたようにプロセス文脈は明示的に示されていないプロセスを表現し，制限があるものの移動や削除，複製が可能である．つまり方法 1 ではこのプロセス文脈という文法を LMNtal がサポートしていることで site への reaction を表現することができる．

一方，方法 2 の場合，place アトムの第一引数を親，第二引数を子として place graph を表現しているので，place アトムの第二引数の接続先のプロセスの移動・複製・削除を行うことになる．つまりルール中に明示的に示されていないプロセスで，かつハイパーリンクの接続を含むものを扱わなければならない．明示的に示されていないプロセスについては前述したとおりプロセス文脈，その拡張である型付きプロセス文脈によってサポートされているが，ハイパーリンクの接続を含むようなプロセス型は定義されていない．よって方法 2 の場合には site の削除や複製といった書き換えはエンコードできない．今回 Bigraph をエンコードした方法 2 のように多くのデータ構造をハイパーリンクを使って表現していくためには，他の構造と同様にハイパーリンクの接続をプロセス文脈でサポートすることが必須である．ハイパーリンクの接続を含んだプロセス型については 6.1 節で扱う．

5.4 *Names* の表現

names を表現するためには region の外へのインターフェースである outer name と site の中へのインターフェースの inner name, さらにノード同士が接続する edge を表現できなければならない．方法 1

でも方法 2 でも *names* の表現にはハイパーリンクを使っているが，outer name と edge の表現は簡単である．edge はハイパーリンクの接続で表現できるが，ルール中のハイパーリンクはルールの外のプロセスにつながっている場合も含んでいるので outer name も同時に表現している．edge の生成・削除・併合はハイパーリンクの同様の操作でそのまま表現できる．

その一方で，Inner name の場合，例として

```
send[x,y] | get[x,z].$0[z] ->
y/w | $0[w] | get[x,z].$0[z]
```

は非同期 π 計算の通信部分の一部を Bigraph によってエンコードした例である．このルールではある region 内で `send` につながった `get` 内の site をコピーして外に出す処理が行われている．Outer name が `x,y` で `get` 内の site およびその複製したものにつながる inner name は `z,w` である．これを方法 1 で表現すると

```
{send(!X,!Y)},{get(!X,!Z), $0[!Z]} :-
!Y><!W,$0[!W],{get(!X,!Z), $0[!Z]}.
```

と表現することが期待される．ここで，ハイパーリンクは arity が 1 である unary 型の拡張であり，unary 型は型付きプロセス文脈であることから上記のルールは

```
{send($x,$y)},{get($x,$z),$0[L],$z[L]} :-
hlink($x),hlink($y),hlink($z),new($w)|
$y><$w,$0[L2],$w[L2],{get($x,$z),$0[L],$z[L]}.
```

の省略記法であることがわかる．ルール中の `$0, $z, $w` はプロセス文脈であり，`$0` と`$z` がリンク L によって，`$0` と`$z` がリンク `L2` によって接続されている．しかし LMNtal ではプロセス文脈同士の接続を許可していないので上記のルールを実行することはできない．例えば `{$p[X]}, $q[X]` のようにマッチングにおいて探索の起点となるアトムがない場合に，マッチングのコストが高くなってしまうことがプロセス文脈同士の接続を許可しない理由である．

方法 2 の場合も同様に，「ハイパーリンクとプロセス文脈を接続できない」という理由で inner name を実現できない．以上の点の解決策について，5.3 節で述べたことも踏まえプロセス文脈の拡張について 6.1 節で検討する．

```
!A=place(!A1,!A2),!A=a(1),!A=act,
!B=place(!A2,!B2),\ !B=a(0) :- !B=a(1).
!A=place(!A1,!A2),!A=pas,
!B=place(!A2, !B2),\ !B=a(1) :- !B=a(0).
!A=place(!A1,!A2),!A=a(0),
!B=place(!A2,!B2),\ !B=a(1) :- !B=a(0).
```

図 7　active, passive の表現

5.5　active, passive の表現

LMNtal には active や passive に該当する機能は組み込まれていないため，これらを実現するためには LMNtal のルールを用いる必要がある．しかし passive および active は木構造においてその子孫全てに影響を与えるので，木構造が変化した場合その変化を子孫に伝搬していかなければならない．これを多重集合の書き換えで 1 ステップで行うのは難しい．

そこで，LMNtal ではこの子孫への伝搬を行う制御ルールを独立して定義した．図 7 に定義したルールを記載する． `t` はツリー構造を表すアトム，`act` と `pas` はノードが active か passive か，`a(1)`, `a(0)` はノードが active な文脈内に存在する（reaction rule が適用可能である）か否かを表す．つまり 5.2 節で扱った構造に active, passive の属性を表現するアトム (`act` と `pa`) と active な文脈内にノードが存在するか否かを表現するアトム (`a(1)`, `a(0)`) を加えることになる．方法 1 の場合も同様の構造を膜内に追加すればよい．制御ルールでは，

(1) 親が active かつ reaction rule が適用可能で子は適用不可の場合，子を適用可能にする
(2) 親が passive で子が reaction rule が適用可能ならば子を適用不可にする
(3) 親が reaction rule が適用不可で子が適用可能ならば子を適用不可にする

の三つの書き換えを行っており，reaction rule を適用するたびにこの書き換えを可能な限り行う．これによって passive, active を考慮してルール適用を制御することが可能になるが，状態が制御ルール適用のたびに遷移し増加してしまう．LMNtal 統合開発環境である LaViT [7] の状態空間の抽象化機能を利用するこ

とによって制御ルールによる状態遷移の抽象化を行うことができる．ただし，これは状態空間構築後の抽象化であり，実際には制御ルールが 0 ステップで適用されたわけではない．

このように計算モデルのデータ構造を表現する場合，そのデータ構造の管理を通常の処理とは別にして考えたいことがしばしばある．そこで新たな機能として，ルールの抽象化，つまり状態遷移することなくルールを適用できるような制御構造について 6.2 節で述べる．

6 HyperLMNtal の拡張機能の設計

6.1 プロセス文脈の拡張

5.3, 5.4 節で述べたようにハイパーリンクを含むプロセス文脈を定義する必要がある．まずハイパーリンクは数学的概念としてリンクの拡張であるので，プロセス文脈の引数は通常のリンクと同様にハイパーリンクを利用できるべきである．そこでまず通常リンクに加えハイパーリンクもプロセス文脈の引数にできるように拡張を行う．

次にハイパーリンクは言語設計の上では unary 型の拡張として扱っているので，プロセス文脈には接続することが許可されていない問題が挙げられる．そこで新たに ground 型を拡張しハイパーリンクの接続を含めた hlground 型を設計する．hlground 型を**ground 型に加えてハイパーリンクの接続を含む**構造とする．あるアトムに接続するハイパーリンクに“ハイパーリンクによる接続”と“ハイパーリンクアトム”という二つの性質を持たせることで hlground 型としても，unary 型としても扱うことが可能となる．ただし条件として，

(1) 明示的に示したプロセスへのハイパーリンクの接続は除く
(2) (1) の接続が存在してもマッチング失敗にならない

ものとする．(1) の条件は ground と同じだが，(2) については hlground 独自の定義である．例として

```
a($x),b($hl) :- hlink($x),hlground($hl) |
a($x),b.
```

(生成)	$Head$ `:-` `new($x,$y)`| A(`$x`)
(型制約)	$Atom$(`$x`) `:-` `hlink($x,$y)`| $Body$

図 8 ハイパーリンク記法の拡張

というルールのもとで `a(!X),b(!X),b(!X),b(!X)` という構造が存在した場合，hlground は `b(!X),b(!X)` にマッチする．

hlground 型の導入と同様に hlground 型の削除や複製についてもサポートする．しかし，(1) の性質により hlground 型によってマッチングではすべてのハイパーリンクの接続を探索するので，マッチした構造に含まれるすべてのハイパーリンクの先のプロセスの削除などを行ってしまう．これは Bigraph のエンコードの方法 2 のように，ハイパーリンクを複数の目的で扱う場合には不都合である．そのため，hlground 型において異なる種類のハイパーリンクで接続された集合を区別できるようになるとより汎用性が向上するはずである．方法のひとつとしてそれぞれのハイパーリンクにその属性を表すアトムを接続しておくことが考えられるが，そのアトムをすべてのハイパーリンクで探索するコストやルール上の記述の複雑化を考えると，ハイパーリンクに値を持たせてしまうのがもっとも簡潔である．そこで，ハイパーリンクのデータ構造を拡張し unary 型のアトムを持たせる．unary 型を持たせる理由は int 型や float 型を含み，ハイパーリンクを識別するうえで十分な情報かつ拡張性を持っているからである．実装では処理系内部のハイパーリンクオブジェクトの構造体に unary 型のアトムへのポインタを持たせることで表現する．このアトムは $Head$ 部や $Body$ 部には出現しないが，$Guard$ 部で記述することができる．3 節の図 1 を拡張した記法を図 8 に示す．

ハイパーリンクの記法のうち二つの記法の拡張を行う．それぞれの記法において引数をひとつ追加し，既存の記法についても同様に利用できる．生成については第二引数にもたせたアトムを値としてハイパーリンクを生成する．型制約では第一引数のアトムがハイパーリンクアトムであることに加え，さらに第二引数のアトムと等しいことの検査を行う．hlground を `hlground`($p, $X_1, \ldots, X_m$) と定義する．`$p` はルール

左辺に出現した型付きプロセス文脈，$X_1, \ldots, X_m$ はハイパーリンクの値であり，hlground を可変長引数とする．このとき$p は $X_1, \ldots, X_m$ のいずれかの値を持つハイパーリンクの接続を含む ground 型とする．例として，

```
Room.$0 -> Room;
```

という Room のノード内の site を削除するような reaction rule を方法 2 で表現する場合を考える．木構造を表現するハイパーリンクに 1，ハイパーグラフを表現するハイパーリンクに値として 2 の int 型のアトムを持たせるとして，

```
!R=room,!R=place(!P1,!P2) :- hlground(!P2,1)
 | !R=room,!R=place(!P1,!P3).
```

と記述できる．これによって Room の子のみ，つまり木構造でつながるノードのみ削除することができる．hlground の第二引数を持たせない場合，全てのハイパーリンクの接続を含むため木構造とハイパーグラフ構造の両方を削除する．複製についても同様に記述できる．これらの拡張によってハイパーリンクを含むプロセス文脈の記述・操作を，他のリンクやアトムなどの構造と同レベルで行うことができるようになる．さらに値を持ったハイパーリンクは複数の構造を区別して扱うことができる点で優れているといえる．

6.2 システムルールセットの拡張

5.5 節 で述べたように，データ構造の管理を行うルールなど計算モデルの本質的な操作ではないルールは抽象化すると記述や状態空間の構築において都合が良い．そこで，ルールの抽象化を行うためシステムルールセットの拡張を行う．抽象化されたルールは，システムルールセットの定義に加え

(1) ルール適用によって状態遷移しない
(2) 最優先で適用される
(3) 抽象化されたルール同士の優先度は定義順

とする．記述の方式として，システムルールセットのユーザ定義にしたがい

```
{
 abstract_system_ruleset.
 A :- B
 ...
}
```

のように表現する．`abstract_system_ruleset` アトムがある膜内のルールはすべて抽象化され，ルール適用を行っても状態遷移をしない．ただし状態空間が非決定的に構築されないようにするためには抽象化されたルールは最優先で適用する．さらに抽象化されたルール内でも書き換え結果の一意性を保証するために，上から優先的に実行する．局所的なルールの抽象化を行わず，システムルールセットに限定する理由は抽象化されたルールの適用に非決定性が生じるのを防ぐためである．

このような抽象化された書き換えを行う類似機能として，LMNtal にすでに組み込まれているコネクタが挙げられる．コネクタは拡張文法であるが，ルール中に=が出現したとき，右辺のアトムと左辺のアトムの最終引数が接続されるように 0 ステップで決定的な書き換えを行うルールとも考えられる．今回定義したルールの抽象化はコネクタのような処理系に組み込まれた書き換えの一般化に近いものである．

7 複数の計算モデルに対応したモデル検査器

6 節で設計した拡張機能も含め，LMNtal によって計算モデルをエンコードし検証する枠組みについて述べる．計算モデルをエンコードするにはそのデータ構造と操作を表現できればよい．データ構造の表現は

(1) **アトム** データ構造の基本要素を構成
(2) **リンク** 要素の一対一の接続を構成
(3) **膜** 要素の多重集合を局所化をする
(4) **ハイパーリンク** 複数の要素間の接続を構成

によって行われる．膜はアリティに制限のないアトムとして利用することもできる．これまで要素のネスト構造は膜によって表現していたが，局所化がエンコードの障害となることがあった．しかし HyperLMNtal への拡張によってネスト構造の表現はハイパーリンクによって行うことが可能になった．さらに Bigraph のエンコードにおける木構造とハイパーグラフ構造の両方をハイパーリンクで表現するような，複数の構造をハイパーリンクで扱う際に必要となるプロセス

型を定義することができた．この設計ではネスト構造をハイパーリンクで，局所化は膜によって行うことでより，より柔軟な表現が可能である．

計算モデルの各操作はその操作の入力と出力に対して，LMNtal の *Head* 部と *Body* 部に対応させたエンコードを行うことでエンコードを行う．しかし，高度な計算モデルでは LMNtal による表現は複雑になり，構造の管理が必要となる．つまり一度の操作によって，構造の書き換えを行うルールが複数本必要となったり，複数回のルール適用が必要になったりするということである．そのために構造の管理を行うルールを抽象化する制御構造を 6.2 節で示した．構造の管理をシステムルールに任せることで操作と LMNtal のルールは一対一に対応させることができる．

計算モデルの入力と出力が LMNtal のルールに一対一に対応することで，計算モデルの動作と等しい状態空間を構築することができる．すなわち，元の計算モデルから LMNtal への変換を行って，そのモデルに忠実な仕様の下で検証を行うことができる．つまり独自の構文で記述可能な，専用のモデル検査器が短期間の工数で実現可能となる．

8 まとめと今後の課題

本論文では，さまざまな計算モデルを統合した Bigraph を LMNtal によってエンコードする手法を提案した．LMNtal の処理系の状態空間構築機能を新たにハイパーグラフに対応させることで，階層ハイパーグラフを利用したモデルのモデル検査を実現した．階層ハイパーグラフによって bigraph の構造の大半は表現可能であり，特にハイパーリンクを利用することで wide ルールが簡単に表現可能であった．hlground 型などプロセス型の拡張はハイパーリンクを実用的に扱う上で必須となる機能である．ルール抽象化機能はデータ構造の管理において一般的に必要な機能である．この機能は既にエンコード方法の確立した λ 計算の β 簡約で用いたルールの抽象化など多くのケースに利用できる．これらの機能を加えた HyperLMNtal は，Bigraph をはじめとしたさまざまな計算モデルについて，モデルの動作を忠実に再現した状態空間を構築可能であると考えられる．

今後の課題は設計した機能を処理系に実装することである．実装した処理系によって bigraph のモデル検査を行い構築された状態空間の確認を行う．Petri Net や MSR など他の計算モデルにおいても，拡張された LMNtal によって同様に確認を行う必要がある．また，Bigraph と LMNtal の変換を行うパーザを作成し，Bigraph 独自の構文で記述し検証できる環境を構築することが挙げられる．さらに統合開発環境として，現在の LMNtal 可視化環境の複数モデル対応という課題もある．**謝辞** 本研究の対象である LMNtal の研究開発に携わってきた早稲田大学上田研究室の皆様に感謝します．本研究の一部は，科学研究費補助金 (基盤研究 (B) 23300011) の補助を得て行った．

参 考 文 献

[1] Ueda, K.: LMNtal as a Hierarchical Logic Programming Language, Theoretical Computer Science, Vol. 410, No. 46, pp. 4784-4800, 2009.

[2] Ueda, K. and Kato, N.: LMNtal: A Language Model with Links and Membranes, in Proc. Fifth Int. Workshop on Membrane Computing (WMC 2004), Vol. 3365 of LNCS, Springer, pp. 110-125, 2005.

[3] 乾敦行, 工藤晋太郎, 原耕司, 水野謙, 加藤紀夫, 上田和紀: 階層グラフ書換えモデルに基づく統合プログラミング言語 LMNtal, 日本ソフトウェア科学会第 23 回大会 PPL 推薦論文, 日本ソフトウェア科学会, 2006.

[4] Ueda, K.: Encoding Distributed Process Calculi into LMNtal, Electronic Notes in Theoretical Computer Science, Vol. 209, pp. 187-200, 2008.

[5] Ueda, K: Encoding the Pure Lambda Calculus into Hierarchical Graph Rewriting, in Proc. 19th International Conference on Rewriting Techniques and Applications (RTA 2008), Vol. 5117 of LNCS, pp. 392-408, Springer, 2008.

[6] 清水涼子, 川端聡基, 上田和紀: Explicit-time method によるモデル検査器 SLIM におけるリアルタイムモデル検査, 日本ソフトウェア科学会第 28 回大会, 日本ソフトウェア科学会, 2010.

[7] Ueda, K., Ayano, T., Hori, T., Iwasawa, H. and Ogawa, S.: Hierarchical Graph Rewriting as a Unifying Tool for Analyzing and Understanding Nondeterministic Systems, in Proc. Sixth International Colloquium on Theoretical Aspects of Computing (ICTAC 2009), Vol. 5684 of LNCS, pp. 349-355, Springer, 2009.

[8] 後町将人, 堀泰祐, 上田和紀: LMNtal 実行時処理系の並列モデル検査器への展開, 日本ソフトウェア科学会第 27 回大会論文集, 3B-1, 2010.

[9] Kazunori Ueda and Seiji Ogawa.: HyperLMNtal: An Extension of a Hierarchical Graph Rewrit-

ing Model, Künstliche Intelligenz, Vol.26, No.1 (2012), pp.27-36
[10] 小川誠司, 目黒学, 上田和紀, 階層グラフ書換えモデルを拡張した HyperLMNtal の実現, 第 25 回人工知能学会全国大会, 2I2-3, 2011.
[11] R. Milner. The space and motion of communicating agents. Cambridge University Press, 2009.
[12] Ole Høgh Jensen , Robin Milner.: Bigraphs and Mobile Processes (revised), TechnicalReport UCAM-CL-TR-580, University of Cambridge, Computer Laboratory, Feb 2004.
[13] A.J. Glenstrup, T.C. Damgaard, L. Birkedal, and E. Højsgaard. An implementation of bigraph matching. 2007.
[14] G. Bacci, D. Grohmann, and M. Miculan. DBtk: a toolkit for directed bigraphs. Algebra and Coalgebra in Computer Science, 2009.
[15] Gian Perrone, Søren Debois and Thomas T. Hildebrandt.: A model checker for Bigraphs, ACM-SAC, 2012.
[16] Frühwirth, T.: Constraint Handling Rules, Cambridge University Press, The Ebinburgh Building, Cambrige CB2 8RU, UK, 2009.