LMNtalを用いた状態空間探索

State space search using LMNtal

小川誠司*1 綾野貴之*1 上田和紀*2
Seiji OGAWA Takayuki AYANO Kazunori UEDA

*1早稲田大学大学院基幹理工学研究科
Graduate School of Fundamental Science and Engineering, Waseda University

*2早稲田大学理工学術院
Faculty of Science and Engineering, Waseda University

LMNtal (pronounced "elemental") is a concurrent language model based on hierarchical graph rewriting. We built a model checker which uses LMNtal as a modeling language, and have been exploring its applications to state space search. The *uniq* constraint, newly introduced for history management, enabled concise description of a class of problems and achieved reduction of the number of states generated. By means of several examples, this paper introduces state space search using LMNtal.

1. はじめに

LMNtal[1] は階層グラフ書換えに基づく並行計算モデルであり，接続構造や階層構造を動的に変化させることによって複雑なデータ構造の操作を伴うプログラムを簡潔に記述することができる [3]．我々は LMNtal で表されたシステムが取り得る状態を網羅的に探索し，そのシステムに要求される性質を満たすか判定するモデル検査器を構築することで，LMNtal の応用分野をシステム検証の分野にまで拡大させた [5][6]．

LMNtal の書換え規則は適用可能な限り繰り返し適用されるため，同型のグラフ構造に対する書換え規則の適用回数を制限するようなモデリングが困難である．またモデリングできたとしても，冗長な記述がモデルの取り得る状態数を増大させてしまい探索結果の把握を困難にしている場合が多々あった．そこで書換え対象のグラフ構造の履歴を管理し，同型グラフ構造に対して高々一回だけ適用が許される `uniq` 制約をモデル検査器に新たに導入した．これにより書換え規則の適用箇所を効率よく制限することができるようになった．

本稿では LMNtal を用いて状態空間探索が行えること，およびモデル検査器への `uniq` 制約の導入によって，これまで表現が困難であった LMNtal モデルが簡潔に記述でき，モデルが取り得る状態数も削減できたことを例題を通して示す．

2. 言語モデル LMNtal

LMNtal の基本データ構造である階層グラフは，アトムを基本構成要素として，それを膜とリンクの二つの手段で構造化したものである．膜はアトムの多重集合を構成し，多重集合は入れ子にできる．またリンクはアトム同士を一対一で接続する．どちらの構造も動的再構成が可能である．

2.1 基本構文と省略構文

LMNtal の構文は図 1 のように定義される．ただし本論文で使用しない構文要素は省いている．X_i は**リンク名**，p は**アトム名**であり，P は**プロセス**である．具象構文ではリンクは大文字から始まる識別子，アトム名は小文字から始まる識別子で表現する．T はプロセスの書換え規則の表現に用いる**プロセステンプレート**であり，局所文脈 (特定のセルの内部での文脈) を扱う機能をもつ．

連絡先: 小川誠司，早稲田大学大学院基幹理工学研究科情報理工学専攻，〒 169-8555 新宿区大久保 3-4-1 63 号館 5 階 02 号，03-5286-3340，seiji(at)ueda.info.waseda.ac.jp

0 は中身のないプロセス，$p(X_1, \ldots, X_m)$ は m 価のアトム，P, P はプロセスの並列合成である．`{`P`}` は**セル**と呼び，プロセスを**膜** `{ }` によってグループ化したものである．T `:-` T はプロセスの書換え規則で，膜が形成する階層構造の同じ場所に置かれたプロセスを適用対象とする．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 の一端とを接続する機能をもつ．**ルール文脈**は膜の中のすべてのルールの多重集合とマッチし，**プロセス文脈**は膜の中のルール以外のプロセスのうち，明示的に指定されていないもの全体とマッチする．

木構造のデータを他言語と同様に記述するための**項記法**が用意されており，アトム a の第 k 引数として，(リンク名のかわりに) 最終引数を省略したアトム b を書くと，a の第 k 引数と b の最終引数とがリンク接続されているものとみなす．

2.2 拡張構文

LMNtal における数値は，`8(X)` のような 1 価アトムで表現する．引数はその数値を参照するプロセスにつながる．

アトムが整数型などの基本型に属するかどうかの検査や基本型に対する演算などを指定するために，**型付きプロセス文脈**および**ガード付きルール**という拡張構文を導入している．

通常のプロセス文脈のマッチする相手が膜 (階層構造) によって決まるのに対し，型付きプロセス文脈のマッチする相手はグラフ構造 (接続構造) とグラフ中のアトム名によって決まる．たとえばガード付きルール

```
p(X),$n[X] :- int($n),$n>0 | p(Y),$n[Y],p(Z),$n[Z]
```

は，1 価アトム `p` が正整数アトムにつながっている場合，その構造の複製を作る．ガード条件 `int($n)` は， `$n[X]` が整数アトムを表す型付きプロセス文脈であることを求め， `$n>0` はその整数値が正であることを求めている．条件 `$n>0` は条件 `int($n)` を含意するので後者は省くことができ，さらに型付きプロセス文脈をリンクと同じ記法で書くことも認めているので，上記のルールは `p(N) :- N>0 | p(N),p(N)` と書いてもよい．現処理系では `int` 以外に `unary` や `ground` 型などをガード条件として指定でき，それぞれ 1 価アトムおよび自由リンクを 1 本だけ持つ無階層グラフを表す．

ルール左辺の膜の後に "`/`" を付加すると，それ以上簡約不能なセルにしかマッチしなくなる．これは子孫膜の計算終了の検出に利用する．

ルールの前に *name*`@@` と書くことでルールに *name* という名前をつけることができる．また膜にも名前 (膜名) をつけることができる．名前 m をもつ膜は $m\{\ldots\}$ と表記する．膜名

$$\text{(プロセス)}\quad P ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid P,P \mid \{P\} \mid T \texttt{:-} T$$
$$\text{(プロセステンプレート)}\quad T ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid T,T \mid \{T\} \mid T \texttt{:-} T \mid \texttt{@}p \mid \texttt{\$}p$$

図 1: LMNtal の構文

はルール名と異なり単なる注釈ではなく，ルール左辺の膜は同一の名前を持つ膜とだけマッチする．

3. LMNtal モデル検査

LMNtal モデル検査器は通常の実行時処理系で動作するほとんどすべての LMNtal プログラムを書き直すことなくモデル検査の対象にすることができる．

3.1 モデル検査の実行

モデル検査を行うに際は，検証したい性質の記述に SPIN[2] で用いられている Never Claim を用いる．LTL を使用する場合は LTL から等価な Never Claim への変換を行う LTL2BA[8] を使用する．

LTL 式や Never Claim 中の命題記号の定義には，階層グラフ構造に対する制約として LMNtal のルール左辺（ガードを含む）の構文と意味をそのまま用いる．たとえば，命題記号 `f` の意味を “アトム `a` の第一引数が正整数アトムである” と定義するには `f = p($x):-$x>0 |` と記述する．

モデル検査は対象の LMNtal プログラムに加えて，Never Claim の記述と命題記号の定義の記述を与えることで実行する．

またモデル検査の関連機能として，LMNtal プログラムがとりうる全状態を探索する**非決定実行機能**が存在する．性質の記述が必要なく全実行経路を出力するため最終結果や全状態数などを容易に確認することができる．通常の実行では停止しないプログラムであっても，全状態数が有限である限り LTL によるモデル検査と非決定実行機能は必ず停止する．

3.2 モデル検査の実装

LMNtal の処理系には Java によって実装されたコンパイラとランタイム [4] に加え，C 言語で実装されたランタイムがある．LMNtal モデル検査器は，高速軽量化のため C 言語で実装されたランタイムに組み込む形で実装された．

モデル検査器は，検証対象の LMNtal プログラムのルール適用と，Never Claim で記述された性質オートマトンの状態遷移とを交互に行うことで，実行を進めつつ状態遷移グラフを生成する．探索は SPIN でも用いられている nested DFS[2] で行い，探索中に受理サイクルが見つかった場合はそこまでの経路を反例として出力する．状態展開は効率よく行えるように，新たに状態を展開する際は階層グラフ構造のハッシュ値を用いて既に展開済みかどうかの判定を行う．ハッシュ値が同じ階層グラフ構造に対してはグラフの同型性判定を行い，正確に等しい状態か確認をする．

3.3 モデル検査可視化環境

モデル検査器のユーザーインターフェースとして Never Claim の記述や命題記号の定義の記述，LTL2BA の操作などを容易にするグラフィカルな環境 [6] が存在する．この環境はモデル検査器とは別に Java で実装されており，モデル検査器の操作性を向上させるだけでなく，モデル検査器が行った状態の探索結果を状態遷移グラフとして可視化しモデルの性質や挙動を視覚的に示す機能も含んでいる．

4. uniq 制約の設計と実装

多重集合書換えに基づく言語モデルである CHR (Constraint Handling Rules)[7] は構文を含めた多くの点で LMNtal と類似している言語であり，CHR モデルを LMNtal でエンコードする試みは兼ねてより行われてきた．しかし CHR は LMNtal には無い概念である propagation rule を持っており，これを LMNtal 上で実現することは困難であったため，同型グラフ構造に対して高々一回だけ適用が許されるルールを記述するための **uniq 制約**が Java 版コンパイラ，ランタイムに導入された [4]．`uniq` 制約はルールのガードに記述する制約であり，マッチしたグラフ構造の履歴を管理することでルールの適用を制限している．これにより CHR モデルを LMNtal 上にエンコードすることが可能となった．

LMNtal は CHR と異なり非決定実行や検証機能を持つため，これに propagation rule の概念を導入することは LMNtal によるモデル検査の応用分野拡大につながることが予想されていた．そこで今回我々はこの `uniq` 制約を C 版ランタイムに導入し，さらにモデル検査器への対応を行った．

4.1 構文・意味

`uniq` 制約を含むルールの構文は以下の通りである．

$$Head :- \text{uniq}(L_1,\ldots,L_n) \mid Body.\ (n \geq 0)$$

`uniq` 制約は LMNtal ルールのガードに記述する．$L_1,\ldots,L_n$ は `ground` 型の構造であるグラフに接続されたリンクの集合であり，これらのリンクに束縛された同型なグラフ構造に対してルールが適用されるのは高々一回である．また $n = 0$ の場合はリンク集合を囲む ( ) を省略して `uniq` のみを記述でき，`uniq` をガードに持つルールが適用されるのは高々一回である．`uniq` 制約はガード制約の一つと位置づけているが，`int` 型や `ground` 型のような型に関する制約ではない．

たとえば，プログラム

```
a(1), a(1), a(2), a(2), a(b(c)), a(b(c)).
a(X) :- uniq(X) | ok(X).
```

を実行すると結果は下記のようになる．

```
ok(1), a(1), ok(2), a(2), ok(b(c)), a(b(c)).
```

4.2 実装方法

反応したグラフ構造の履歴の管理は，ルールを表す構造体に追加したハッシュテーブルによって行う．反応したグラフ構造を一意に識別できる形で履歴表に保持させる必要があるため，グラフ構造を文字列に置き換えたものを識別子とした．ルールのマッチングが行われる際は，ヘッドにマッチしたグラフ構造から識別子を生成する．その識別子が履歴表に含まれていない場合には過去に反応したことがない構造だと判断し，ルール適用と共に履歴表に識別子を挿入する．履歴表に識別子が含まれていた場合には過去に反応したことのある構造だと判断し，ルールは適用されない．

この `uniq` 制約を用いた LMNtal プログラムをモデル検査器で実行させる場合，ある 2 つの状態における階層グラフ構造が同型であっても，ルールが持つ履歴表の内容が異なればその状態から遷移できる状態は異なる．そのためモデル検査においてルールが持つ履歴表の内容が異なれば別の状態と判定するよう，状態展開の際のグラフの同型性判定に，各ルールが持つ履歴表の等価性判定を併せて行うようにした．現在の実装では，一方の履歴表が保持する要素を取り出し，他方の履歴表内にそれが含まれるか検査する．履歴の個数 N に対する計算量は $O(N)$ であり，これを双方の履歴表に対して行う．

```
{goal}, {a,obj(box)}, {b,obj(stick)}, {c,monkey(none)}.
cond(none).

move@@ {monkey(X), $p}, {$q} :- unary(X) |
   {$p}, {monkey(X), $q}.
hold@@ {monkey(none), obj(X), $p} :- unary(X) |
   {monkey(X), $p}.
put@@ {monkey(X), $p}, cond(C) :-
   \+($p=(obj(Y),$pp)), X\=C |
   {monkey(none), obj(X), $p}, cond(C).
get_on@@ {monkey(X), obj(box), $p} :- unary(X) |
   {monkey(X,box), $p}.
get_off@@ {monkey(X,box), $p} :- unary(X) |
   {monkey(X), obj(box), $p}.
goal@@ {goal, monkey(stick,box) } :- got_banana.
```

図 2: monkey and banana

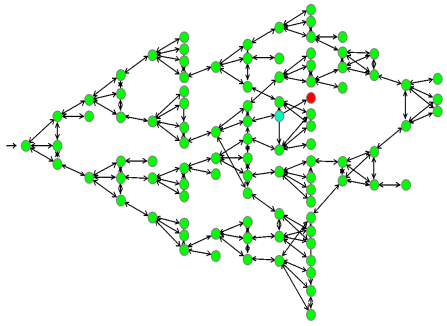

図 3: monkey and banana の全状態遷移

5. 記述例とその評価

本節ではいくつかの例題を解く LMNtal プログラムを紹介し，その評価を行う．これら以外の LMNtal による状態空間探索やシステム検証の記述例は [5][6] を参照されたい．

5.1 monkey and banana

本例題では非決定実行によって生成され得る全状態遷移を可視化することで，システムの挙動の理解が容易になることを示す．

部屋の中に一匹の猿がおり，床には道具 (棒，箱) があり，天井からはバナナが吊るされている．猿がバナナを食べるためには，箱をバナナの下まで運び，棒を持った状態で箱に乗る必要がある．今回は初期条件として猿，棒，箱，バナナをそれぞれ部屋の中の異なる地点に設置し，これらの 4 地点以外の地点に猿は移動できないものとする．また猿は道具を持つことができるが，同時に二つの道具を持つことはできない．一地点に置ける道具は一つだけである．

この例題を LMNtal プログラムで記述すると図 2 のようになる．`\+($p=(obj(Y),$pp))` は `$p` で表されるプロセス集団に 1 個の `obj` アトムが無いという条件を表す．

図 3 は本例題の全状態遷移を可視化したものである．最も左にある円が初期状態，矢印が状態遷移を表しており，赤い円が最終状態 (猿がバナナを手にしたとき) である．4 つの状態からなる三角形が多く表示されているが，これは猿が部屋の中の 4 地点間を移動できることを示している．また最終状態よりも右側に状態があることから，猿はバナナを手にするまでに必ずしも最短な方法を取るわけではなく，冗長な行動を取りながら最終状態へと向かう場合もあることが分かる．

```
copy_rule@@
graph{$p},make_rule{@p} :- graph{$p,@p},make_rule{@p}.
get_subgraph@@
graph{graph{$p[]},$q,@q}/ :- graph{$q,@q},graph{$p[]}.
rule{  make_subgraph@@
         edges{edge(X, Y), $p[]}, num(N) :-
           unary(X), unary(Y), N>0, uniq(X, Y) |
             edges{edge(X, Y), $p[]}, num(N),
             graph{edges{$p[]}, num(N-1)}.          }.
```

図 4: Spanning Spider (抜粋)

5.2 Spanning Spider

本例題は，LMNtal の通常実行において，`uniq` 制約を用いることによってこれまで実現が非常に困難であった例題が簡潔に記述できた例として取り上げた．

与えられた初期グラフから Spanning Spider (全域クモ) を見つける．全域クモとは，Spanning Tree (全域木：あるグラフから 0 本以上の枝を取り除いて閉路を除いたグラフ) であり，かつクモであるグラフを指す．ここでのクモとは，枝を 3 本以上持つ頂点が高々一つしか含まれないグラフのことである．

この例題では，X，Y の 2 頂点を結ぶ枝を `edge`(X, Y) で表し，この `edge` の集合をグラフとする．与えられた初期グラフから `edge` を任意の数だけ取り除き，残った `edge` から形成されるグラフに対して全域木かつクモであるかを判定することで，全域クモであるグラフを見つける．通常実行を前提とした LMNtal プログラムでは，ある要素の集合が持ち得る全ての組合せの部分集合を生成するような記述は大変煩雑になるが，`uniq` 制約を用いることでこれを簡潔に記述できる．図 4 は全域クモを見つける LMNtal プログラム中から，`uniq` 制約を用いて網羅的に部分集合を生成している箇所を抜粋したものである．初期グラフは `graph` 膜の中にさらに `edges` 膜があり，その中に `edge` の集合が入っている状態で与えられる．図 4 の `rule` 膜およびルール群と同じ階層に `graph` 膜がある場合には，`copy_rule` ルールが `graph` 膜内に `make_subgraph` ルールをコピーし，`graph` 膜内の `edge` 集合から任意の `edge` を 1 つだけ取り除いたサブグラフを生成する．`graph` 膜内でルール適用が行われなくなると，`get_subgraph` ルールが `graph` 膜内からサブグラフ膜を取り出し，以降は再帰的にルール適用が行われていく．

5.3 ペア生成

本例題では `uniq` 制約を含むルールを用いることで，非決定実行中に生成され得る状態数が削減できることを示す．

n 人以上のメンバが所属しているグループが m 個存在する．これらのグループから 2 人を選びペアを作る．ただしペアは (i) 同じグループのメンバで構成されてはならず，かつ (ii) 同じ組合せのペアは 2 つ以上存在してはならない．今回は簡単のために $m = 4$ に固定し，各グループのメンバ数は等しく n 人とした．図 5 は $(m, n) = (4, 3)$ とした場合に与える初期グラフであり，$\mathrm{p}(X)$ はグループ X に属するメンバを表す．またさらにグループ X，Y によって生成される (X, Y) ペアと (Y, X) ペアは異なる組合わせとした．この例題を解くプログラムを `uniq` 制約を用いない場合と用いた場合のルールによって記述し，条件 (i)，(ii) を満たしつつ最も多くのペアを作成できる組合せを探す過程で生成される状態数を，n を変化させながら比較した．

`uniq` 制約を用いない場合，用いた場合のペアを生成するルールをそれぞれ図 6，7 に示す．図 6 ではペア生成のルールの他に既に生成されているペアを生成してしまった場合にはペアを

```
p(1), p(1), p(1), p(2), p(2), p(2),
p(3), p(3). p(3), p(4), p(4), p(4).
```

図 5: $(m, n) = (4, 3)$ としたペア生成の初期グラフ

```
p(X), p(Y) :- X\=Y | pair(X, Y).
pair(X1, Y1), pair(X2, Y2) :-
  X1=X2, Y1=Y2 | pair(X1, Y1), p(X2), p(Y2).
```

図 6: uniq 制約を用いないペア生成のルール

```
p(X), p(Y) :-
  X\=Y, uniq(X, Y) | p(X), p(Y), pair(X, Y).
```

図 7: uniq 制約を用いたペア生成のルール

メンバ数	2	3	4	5	6
uniq 制約無し	330	2730	16320	76896	302230
uniq 制約有り	294	1806	6561	15717	27344
削減率 [%]	10.91	33.85	59.8	79.56	90.95

図 8: ペア生成が取り得る全状態数

再び解体するルールが必要であるが，図 7 のように uniq 制約を用いることで一度生成した組合わせのペアはそれ以上生成されないため，一本のルールで記述できる．

非決定実行によってこのプログラムが実行中に取り得る状態数を $2 \leq n \leq 6$ の場合について求め，表 8 に示す．表 8 から分かるように，全ての n の値において uniq 制約を用いない場合に対して状態数が削減できており，特に $n = 6$ では 90% 以上の状態数を削減できている．

5.4 プレゼント交換

本例題は uniq 制約を含むルールによるモデル検査の例として取り上げる．

n 人の人が各自プレゼントを 1 つずつ用意し，他の人と互いにプレゼントを交換する．一度の交換でプレゼントを交換できるのは 2 人だけであり，同じ組合わせの 2 人の間での交換は一度だけである．この交換を可能な限り繰り返すと，最終的に全ての人が自分が用意したプレゼントを保持している可能性があるか検証する．今回は $n = 4$ の場合について検証する．

このモデルも uniq 制約を用いることで，図 9 に示すように全ての組合せに対して一度だけ交換を行う動作を一本のルールで表すことができる．$\mathrm{p}(X, P)$ は X という人が P というプレゼントを持つことを表している．命題記号は，

```
p = p(1, book), p(2, toy),
    p(3, ball), p(4, candy) :- |
```

と定義し，“全員が最終的に自分が用意したプレゼントを持つことはない”，すなわち $!(\Box\Diamond \mathrm{p})$ に対する反例が無いか検証する．LMNtal モデル検査器によって状態空間探索を行った結果を可視化すると図 10 のようになる．図の最も右にある状態は反例を表しており，全員が最終的に自分が用意したプレゼントを持つ場合があることが分かった．

6. まとめと今後の課題

LMNtal モデル検査器に uniq 制約を導入し，例題を通してその有用性を示した．今回紹介した uniq 制約を用いた例題以外にも，状態数の削減，およびプログラム記述量の削減を実現できる例題は数多くあると予想される．

また uniq 制約導入のため，階層グラフ構造の同型性判定にルールが持つ履歴表の等価性判定を追加した．履歴の個数に対して線形の時間計算量で判定を行えるため，同型性判定を多数回行うような状態数の多いモデルを実行する際にも，状態数削減の効果によって実行時間を大幅に短縮することが可能であると予想される．

```
p(1, book), p(2, toy), p(3, ball), p(4, candy).
p(A, PA), p(B, PB) :-
  A>B, uniq(A, B) | p(A, PB), p(B, PA).
```

図 9: プレゼント交換

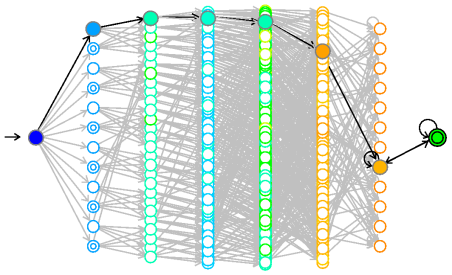

図 10: プレゼント交換の反例検出結果

これらの例題の蓄積，および実行時間の測定は，uniq 制約の有用性をより強く実証することにつながるため，今後の重要な課題である．

謝辞 早稲田大学上田研究室を卒業した岡部亮氏，佐々木隆之氏，および現在の開発メンバである堀泰祐氏，石川力氏にはモデル検査器の開発，uniq 制約の導入に協力していただいた．また同言語班の小林史佳氏，八鍬豊氏 (08 年度卒)，斉藤和佳子氏，岩澤宏希氏，後町将人氏らとの活発な議論も，上記の開発を行う上で非常に有益なものであった，ここに感謝の意を表する．本研究の一部は，科学研究費補助金（特定 18049015）の補助を得て行った．

参考文献

[1] 上田和紀, 加藤紀夫: 言語モデル LMNtal, コンピュータソフトウェア, Vol. 21, No. 2(2004), pp. 126–142.

[2] Holzmann, G., *The SPIN Model Checker*, Addison-Wesley, 2004.

[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, コンピュータソフトウェア, Vol. 25, No. 1(2008), pp. 124–150.

[4] 村山敬, 工藤晋太郎, 櫻井健, 水野謙, 加藤紀夫, 上田和紀: 階層グラフ書換え言語 LMNtal の処理系, コンピュータソフトウェア, Vol. 25, No. 2(2008), pp. 47–77.

[5] 堀泰祐, 佐々木隆之, 綾野貴之, 岡部亮, 上田和紀: LMNtal に基づくモデル検査環境，第 5 回システム検証の科学技術シンポジウム, No. 59, pp. 21–32，2008.

[6] 綾野貴之, 堀泰祐, 岩澤宏希, 小川誠司, 上田和紀: 統合開発環境による LMNtal モデル検査，第 11 回プログラミングおよびプログラミング言語ワークショップ論文集, pp. 1–15, 2009.

[7] Thom Frühwirth: Theory and Practice of Constraint Handling Rules，Journal of Logic Programming, Special Issue on Constraint Logic Programming, Vol 37(1-3), October 1998.

[8] Oddoux, D., Gastin, P.: LTL 2 BA, 2007. http://www.lsv.ens-cachan.fr/gastin/ltl2ba/