# 定理証明支援系 Coq によるグラフ書換え言語の性質証明

山本 直輝　上田 和紀

グラフ書換え言語 LMNtal は，グラフの書換えによって計算を表現するプログラミング言語としての側面と，一般の複雑なグラフ構造を扱えるモデリング言語としての側面とを兼ね備えた言語である．LMNtal プログラムは，読者の理解を助けるために，そのプログラムに対応するグラフの図的表現と併記されることがある．しかし，LMNtal の項はア・プリオリにグラフとして意味づけされているわけではなく，λ 計算や π 計算といった計算モデルと同様に，構文主導の意味論によって定義されている．そのため，LMNtal プログラムとその図的表現との対応関係は自明でない．そこで，LMNtal の項上の同値関係である構造合同関係と，グラフ理論におけるグラフ同型性との対応を明らかにするための準備として，本論文ではまず定理証明支援系 Coq の上で LMNtal の抽象構文と意味論を実装し，LMNtal 言語に関する性質の証明を Coq 上で行った．

## 1　はじめに

一般のグラフ構造は，代数的データ型（リストや木構造）よりも複雑な構造をもち，それ故に既存のプログラミング言語では扱うことが難しい．例えば，C 言語のようなポインタを扱える言語であれば，ポインタの接続構造としてグラフを表現することができるが，依然としてダングリングポインタなどの不正なポインタが発生する危険性がある．そこで，グラフを第一級オブジェクトとしてもち，それを書き換えることによって計算を表現する，**グラフ書換え言語**というパラダイムが提案されている．

LMNtal[11] は，グラフ書換え言語の一つであり，アトム（ノード）やルール（書換え規則）といった最小限の言語要素からなる言語モデルである．しかしながら，LMNtal は十分な表現力・計算能力をも持ち合わせている．実際，LMNtal はチューリング完全であり，λ 計算などの他の言語体系を LMNtal プログラムにエンコードする手法が存在している [5] [6]．さらに，LMNtal グラフは well-formed であるかぎり，不正なポインタを含まない．この意味で，LMNtal は言語仕様のレベルにおいて不正なポインタを排除している（すなわち，ポインタ安全である）といえる．

また，LMNtal はプログラミング言語としての側面の他に，モデリング言語としての側面を持ち合わせている．実際，LMNtal の実行時処理系である SLIM [13] [10] には，これらの 2 つの側面とそれぞれ対応する 2 種類の実行モードがある．すなわち，LMNtal をプログラミング言語とみなす場合には SLIM は実行時処理系となり，モデリング言語とみなす場合には SLIM はモデル検査器となる．特にモデル検査器としての SLIM にはグラフの書換えによって遷移しうる全状態を出力する機能があるほか，LTL モデル検査などの機能も充実している．

LMNtal の言語仕様，すなわち抽象構文と意味論は，数学的な形で簡潔に記述されているが，その言語仕様自体の妥当性は自明でない．例えば，LMNtal プログラムは，読者の理解を助けるために，そのプログラムに対応するグラフの図的表現と併記されるこ

---

* Proving Properties of Graph Rewriting Languages with the Coq Proof Assistant.



Naoki Yamamoto, Kazunori Ueda, 早稲田大学基幹理工学研究科情報理工・情報通信専攻, Dept. of Computer Science and Communications Engineering, Graduate School of Fundamental Science and Engineering, Waseda University.

とがある．しかし，LMNtal の項はア・プリオリにグラフとして意味づけされているわけではなく，$\lambda$ 計算や $\pi$ 計算といった計算モデルと同様に，構文主導の意味論によって定義されている．そのため，LMNtal プログラムとその図的表現との対応関係は自明でない．そこで我々は，定理証明支援系を用いた形式的検証により，LMNtal の項上の同値関係である構造合同関係と，グラフ理論におけるグラフ同型性との対応を明らかにすることを目指している．

定理証明支援系 Coq[4] では，OCaml ライクなプログラミング言語である Gallina により，データ構造や関数・述語の定義およびそれらに関する定理を記述できる．また，定理を証明する際は，定義の展開（`unfold`）や既存定理の適用（`apply`），簡単なゴールの自動証明（`auto`）等を行うタクティックと呼ばれる命令を処理系に与えることにより，対話的に証明を構築していく．

Coq においては数学の各分野と対応する形で様々なライブラリが開発されており，本論文と関連するグラフ理論の分野においても，Graph Theory[2] というライブラリが開発中である．グラフ理論におけるグラフの定式化には様々な類型があるが，このライブラリでは有向・無向の別や，多重辺を許すか否かなど様々な種類のグラフがカバーされているほか，メンガーの定理などの重要定理とその証明も合わせて提供されている．

LMNtal の形式的検証に関する研究には，他に [12] がある．この研究では LMNtal のコンパイラによって生成される中間命令列を簡潔な形で再設計した上で定理証明支援系 Coq により形式化し，その中間命令列によるグラフの書換えが仕様を満たすことを確認している．しかし，LMNtal のコンパイラの動作，すなわち LMNtal プログラムと中間命令列との関係性については形式的には議論されていない．

本論文ではまず Coq の上で LMNtal の抽象構文と意味論を実装し，例題を基にして動作確認を行うとともに，LMNtal 言語に関する性質の証明を Coq 上で行った．

**グラフ**

| | |
|---|---|
| $G ::= \mathbf{0}$ | （空） |
| $\mid\ p(X_1, \ldots, X_m)$ | （アトム, $m \geq 0$） |
| $\mid\ G, G$ | （分子） |

**ルールセット**

| | |
|---|---|
| $R ::= \mathbf{0}$ | （空） |
| $\mid\ G :\!\!-\ G$ | （ルール） |
| $\mid\ R, R$ | （分子） |

**図 1　LMNtal の構文**

## 2　グラフ書換え言語 LMNtal

本節では，グラフ書換え言語 LMNtal について簡単に説明する．なお，本来 LMNtal には膜による階層や，ルールの発火に関する制約条件を記述するためのガード及びプロセス文脈といった機能が備わっているが，本論文では簡単のためこれらを除外したサブセット言語を扱う．また，LMNtal の拡張言語である HyperLMNtal[7] において導入された拡張機能である，超辺を表すハイパーリンクについても扱わない．しかしながら，これらの言語機能をもたない LMNtal もまたチューリング完全であり，プログラミング言語としてもモデリング言語としても十分強力である．

### 2.1　構文

LMNtal の構文を図 1 に示す． LMNtal プログラムは，**グラフ**（アトムの多重集合）と**ルールセット**（ルールの多重集合）の組として表現され，これを**プロセス**と呼ぶ．$p$ という名前で， $m$ 本のリンク $X_1, \ldots, X_m$ をもつアトムを $p(X_1, \ldots, X_m)$ と書く．これらのリンクをアトムの引数といい，順序がついている．英字の大文字から始まる名前はリンク名，そうでないものはアトム名として解釈される．アトム名と引数の本数（価数）の組を**ファンクタ**といい，$p/m$ のように書き,「$m$ 価の $p$ アトム」のように読む．

LMNtal における**アトム・リンク**は，それぞれグラフ理論における節点（ノード）・辺（エッジ）に相当するものであるが，引数の数が固定である点や，引数

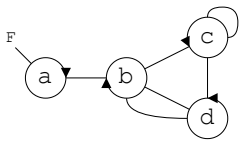


**図 2　LMNtal グラフの例**

の順序，コネクタの存在など，様々な点において定式化が異なる．通常のグラフ理論においては，グラフは節点の集合と辺の集合の組によって定義されるが，LMNtal においては，アトムの多重集合によって無向多重グラフを表現する（多重辺や自己ループの出現を許す）．例えば，

```
a(A,F), b(A,C,L1,L2), c(C,D,S,S), d(D,L1,L2)
```

というアトムの多重集合は，図 2 に示すような無向グラフを意味する．図中において矢印は第 1 引数の場所を指しており，そこから矢印の向きに従って引数に順序がついていることを示す．

グラフはいずれも次の**リンク条件**を満たす：

> グラフ中に同名のリンクは高々 2 回出現する．

グラフ中に 2 回出現するリンクは両端がアトムと繋がっている**局所リンク**であり，1 回のみ出現するリンクは一端が無接続である（または，外部と接続している）**自由リンク**である．自由リンクのないグラフは**閉じている**という．

**ルール**は，部分グラフから部分グラフへの書換えを表す書換え規則である．例えば，`a(X) :- b(X)` は 1 価の `a` アトムを，1 価の `b` アトムに書き換えるルールである．なお，書換えの前後において自由リンクの本数が増減することはあり得ないので，ルールの左辺と右辺における自由リンクの集合は等しい．

### 2.2　意味論

LMNtal の意味論は，**構造合同**と**遷移規則**からなる．これらについて順に説明する．

#### 2.2.1　構造合同

グラフ理論における “グラフ同型” と同様に，構文上の差異を吸収するための概念として，LMNtal にはグラフ間の構造合同関係 “$\equiv$” が存在する．この関係

| | | |
|---|---|---|
| (E1) | $\mathbf{0}, P$ | $\equiv P$ |
| (E2) | $P, Q$ | $\equiv Q, P$ |
| (E3) | $P, (Q, R)$ | $\equiv (P, Q), R$ |
| (E4) | $P$ | $\equiv P[Y/X]$ |
| | （$X$ は $P$ の局所リンク） | |
| (E5) | $P \equiv P' \Rightarrow P, Q$ | $\equiv P', Q$ |
| (E7) | $X{=}X$ | $\equiv \mathbf{0}$ |
| (E8) | $X{=}Y$ | $\equiv Y{=}X$ |
| (E9) | $X{=}Y, P$ | $\equiv P[Y/X]$ |
| | （$P$ はアトム，$X$ は $P$ の自由リンク） | |

**図 3　LMNtal グラフ上の構造合同関係**

は，図 3 の規則を満たす最小の同値関係として定義される．構造合同な LMNtal プロセスは互いに 0 ステップで変換可能である．なお，$P[Y/X]$ はグラフ $P$ に出現するリンク $X$ をリンク $Y$ に置き換えることを意味する．ただし，(E6), (E10) は膜に関する規則であるので省略した． (E1)–(E3), (E5) はプロセス計算にもみられる一般的な規則であり，(E4) は局所リンク名の $\alpha$ 変換である．(E7) から (E9) までは特別な 2 価のアトムである**コネクタ**に関する規則である．$=(X, Y)$ は二つのリンク $X, Y$ を接続するという意味を持ち，中置記法で $X = Y$ とも書く．(E7) は一つのリンクの両端が繋がって輪になっているものは空とみなして良いこと，(E8) はリンクが対称である（つまり無向である）こと，(E9) は直接接続されている二つのリンクが一つのリンクに縮約できることを表している．

#### 2.2.2　遷移規則

LMNtal における本質的な計算ステップを表す，グラフ間の二項関係として，**ルール $T$ :- $U$ によるグラフの遷移関係** “$\xrightarrow{T :- U}$” が存在する．これは，図 4 に示す遷移規則をみたす最小の二項関係として定義される．ただし，(R2), (R4), (R5) は膜に関する規則であるので省略した． 最も本質的で重要な規則は (R6) である．これは，「ルールと，その左辺にパターンマッチするプロセスが存在したら，その左辺を右辺に書き換えてよい」ということを言っている．

$$\text{(R1)}\quad \frac{G_1 \xrightarrow{T :\!\!- U} G_1'}{G_1, G_2 \xrightarrow{T :\!\!- U} G_1', G_2}$$

$$\text{(R3)}\quad \frac{G_2 \equiv G_1 \quad G_1 \xrightarrow{T :\!\!- U} G_1' \quad G_1' \equiv G_2'}{G_2 \xrightarrow{T :\!\!- U} G_2'}$$

$$\text{(R6)}\quad T \xrightarrow{T :\!\!- U} U$$

**図 4　LMNtal グラフ上の遷移関係**

また，この定義を自然にルールセットへと拡張することができる．すなわち，$\exists r \in R.\ G \xrightarrow{r} G'$ が成り立つとき「ルールセット $R$ によって，グラフ $G$ が（1 ステップで）$G'$ に遷移する」といい，単に $G \xrightarrow{R} G'$ と書く．

なお，通常の LMNtal においては，アトムとルールの多重集合であるプロセスが，遷移関係に基づいて**おのずから書き換わっていく**．そのため，「ルールによる作用を受けてグラフが書き換わる」とは考えない．しかしながら，書換えに関する静的な性質について定義および議論をするにあたっては，書換えの主体たるルールと客体たるグラフとは明確に分離されていたほうが都合が良い．よって，本論文では LMNtal の構文・意味論を簡明な形に等価変換して扱っている．

## 3　Coq による LMNtal の構文および意味論の実装

本節では，前節で述べた LMNtal の構文および意味論の Coq による形式化について述べる．

### 3.1　データ構造の定義

まず，図 1 で紹介した抽象構文に沿って，Coq のデータ構造を図 5 のように代数的データ型として再帰的に定義した．

また可読性や記述性のため，Coq の Custom Entry および Notation の機能を用いて，`{{` と `}}` で囲まれた箇所では図 1 で導入した “`:-`” などの各種記号を用いてグラフやルールを記述できるようにした．例えば，`{{"a"("X"):-"b"("X")}}` と書いた場合は `React (GAtom (AAtom "a" ["X"])) (GAtom (AAtom "b" ["X"]))` と解釈される．これは，通常の LMNtal の構文でいう `a(X):-b(X)` に相当する．なお，`{{a(X):-b(X)}}` と書いた場合は，ここに登場する `a`, `b`, `X` は特定のアトム名やリンク名を表す記号ではなく Coq の変数として解釈される．

```
Definition Link := string.

Inductive Atom : Type :=
  | AAtom (name:string) (links:list Link).

Inductive Graph : Type :=
  | GZero
  | GAtom (atom:Atom)
  | GMol (g1 g2:Graph).

Inductive Rule : Type :=
  | React (lhs rhs:Graph).

Inductive RuleSet : Type :=
  | RZero
  | RRule (rule:Rule)
  | RMol (r1 r2:RuleSet).
```

**図 5　Coq による LMNtal グラフ・ルールセットの定義**

### 3.2　Well-formedness の定義

続いて，グラフとルールそれぞれについて well-formedness を定義する．

グラフに関する well-formedness は，前述のリンク条件，すなわち「グラフ中に同名のリンクは高々 2 回出現する」というものであり，ここでは述語 `wellformed_g`（図 6 上部）により表現した．ここで `link_multiset` はグラフを受け取ってそこに出現するリンクの多重集合を返す関数であり，`unique_links` はグラフ中に出現するリンクのリストから重複要素を除いて返す関数，`multiplicity` は多重集合と要素を受け取ってその要素の重複度（出現回数）を返す関数である．

また，以降の証明での便宜のために補題として `wellformed_g_forall`（図 6 下部）も証明した．なお `links` はグラフ中に出現するリンクを単に（重複を省かず）リストで返す関数である．

一方，ルールに関する well-formedness は，「書換え

```
Definition wellformed_g (g:Graph) : Prop :=
  forallb (fun x => Nat.leb
    (multiplicity (link_multiset g) x) 2)
    (unique_links g) = true.

Lemma wellformed_g_forall:
  forall g, wellformed_g g <->
    forall x, In x (links g) ->
    (multiplicity (link_multiset g) x) <= 2.
```

**図 6　LMNtal グラフの Well-formedness**

```
Definition wellformed_r (r:Rule) : Prop :=
  match r with
  | {{lhs :- rhs}} =>
    link_list_eq
      (freelinks lhs) (freelinks rhs)
  end.
```

**図 7　LMNtal ルールの Well-formedness**

の前後において自由リンクの本数が増減することはあり得ないので，ルールの左辺と右辺における自由リンクの集合は等しい」ことである．ここでは，図 7 に示す述語 `wellformed_r` としてこれを定義した．なお，`link_list_eq` は受け取った 2 つのリンクのリストを（多重）集合として等価か判定する述語であり，`freelinks` は受け取ったグラフ中に出現する自由リンクのリストを返す関数である．

### 3.3　構造合同関係

次に，図 3 で導入した構造合同関係を，二項関係 `cong` として定義した（図 8）．ここでは，`Reserved Notation` コマンドにより定義と同時に中置記法の “==” を `cong` に割り当てている．

図 3 に示した構造合同関係では，== の両辺が well-formedness，すなわちリンク条件を満たすことを前提条件として暗黙に要求していたが，Coq における定義ではこれらの前提条件を明示的に記述している．また，前掲の定義では「図 3 の規則を満たす最小の同値関係」として構造合同関係を定義しているが，これも Coq における定義では明示的に（wellformed なグラフについて）同値関係であること，すなわち反射律（`cong_refl`）・推移律（`cong_trans`）・対称律（`cong_sym`）をそれぞれ記述している．

### 3.4　遷移関係

同様に，図 4 で導入した遷移関係を，二項関係 `rrel` として定義した（図 9）．ここでも，定義と同時に $p$ `-[` $r$ `]->` $q$ という記法を導入し，`rrel` $r$ $p$ $q$ に割り当てている．また，書換え前後のグラフおよびルールの well-formedness を前提条件として明示的に記述している．

### 3.5　例題

次節で普遍的な性質の証明に移る前に，ここまでに定義した構造合同関係および遷移関係を具体例に適用して動作確認を行う．

まず構造合同関係の例として `p(X,X)` ≡ `p(Y,Y)` を示す．これは非形式的には (E4) から直ちに導かれる．Coq ではこれを図 10 のように記述・証明する．

一方，ここにおいて `p`, `X`, `Y` が任意の場合，すなわち

```
Example cong_example' :
  forall p X Y,
    {{ p(X,X) }} == {{ p(Y,Y) }}.
```

はアトム名やリンク名が固定の場合に比べると自明でない．これは，途中過程で変数同士の比較が行われているため，（たとえ同じ変数同士の比較であっても）Coq の処理系が自動で判定結果を算出しないことが主な原因である．そのため，定義の展開（`unfold`）と自明な比較の簡約（`rewrite`）を繰り返し行うタクティック `solve_refl` を新たに作成した：

```
Ltac solve_refl :=
  repeat unfold wellformed_g, wellformed_r,
    freelinks, locallinks, unique_links;
  simpl; repeat ((rewrite Leq_dec_refl
    || rewrite eqb_refl); simpl); auto.
```

これを `auto` タクティックの代わりに用いることで，アトム名やリンク名が固定の場合と同様に証明できる（図 11）．

```
Reserved Notation "p == q" (at level 40).
Inductive cong : Graph -> Graph -> Prop :=
  | cong_E1 : forall P, wellformed_g P -> {{GZero, P}} == P
  | cong_E2 : forall P Q, wellformed_g {{P, Q}} -> {{P, Q}} == {{Q, P}}
  | cong_E3 : forall P Q R, wellformed_g {{P, (Q, R)}} -> {{P, (Q, R)}} == {{(P, Q), R}}
  | cong_E4 : forall P X Y, wellformed_g P -> wellformed_g {{ P[Y/X] }} ->
              In X (locallinks P) -> P == {{ P[Y/X] }}
  | cong_E5 : forall P P' Q, wellformed_g {{ P,Q }} -> wellformed_g {{ P',Q }} ->
              P == P' -> {{ P,Q }} == {{ P',Q }}
  | cong_E7 : forall X, {{ X = X }} == GZero
  | cong_E8 : forall X Y, {{ X = Y }} == {{ Y = X }}
  | cong_E9 : forall X Y (A:Atom), wellformed_g {{ X = Y, A }} -> wellformed_g {{ A[Y/X] }} ->
              In X (freelinks A) -> {{ X = Y, A }} == {{ A[Y/X] }}
  | cong_refl  : forall P, wellformed_g P -> P == P
  | cong_trans : forall P Q R, wellformed_g P -> wellformed_g Q -> wellformed_g R ->
                 P == Q -> Q == R -> P == R
  | cong_sym   : forall P Q, wellformed_g P -> wellformed_g Q -> P == Q -> Q == P
  where "p '==' q" := (cong p q).
```

**図 8　Coq における LMNtal の構造合同関係の定義**

```
Reserved Notation "p '-[' r ']->' q"
  (at level 40, r custom lmntal at level 99, p constr, q constr at next level).
Inductive rrel : Rule -> Graph -> Graph -> Prop :=
  | rrel_R1 : forall G1 G1' G2 r,
                wellformed_g {{G1,G2}} -> wellformed_g {{G1',G2}} -> wellformed_r r ->
                G1 -[ r ]-> G1' -> {{G1,G2}} -[ r ]-> {{G1',G2}}
  | rrel_R3 : forall G1 G1' G2 G2' r,
                wellformed_r r ->
                G2 == G1 -> G1' == G2' -> G1 -[ r ]-> G1' -> G2 -[ r ]-> G2'
  | rrel_R6 : forall T U,
                wellformed_g T -> wellformed_g U -> wellformed_r {{ T :- U }} ->
                T -[ T :- U ]-> U
  where "p '-[' r ']->' q" := (rrel r p q).
```

**図 9　Coq における LMNtal の遷移関係の定義**

続いて，遷移関係の例として，

$$\texttt{a(),b(Z),c(Z)} \xrightarrow{\texttt{b(X),c(X):-d()}} \texttt{a(),d()}$$

を示す．これは Coq では以下のように記述する：

```
Example rrel_example :
  {{ "a"(), "b"("Z"), "c"("Z") }}
  -[ "b"("X"),"c"("X") :- "d"() ]->
  {{ "a"(), "d"() }}.
```

この証明の Coq スクリプトは比較的長大となるのでここでは割愛するが，概略としてはまず遷移規則 (R3) により証明のゴールを

$$\texttt{b(Z),c(Z),a()} \xrightarrow{\texttt{b(X),c(X):-d()}} \texttt{d(),a()}$$

に変形し，(R1) により

$$\texttt{b(Z),c(Z)} \xrightarrow{\texttt{b(X),c(X):-d()}} \texttt{d()}$$

に変形して，これを (R6) により示す，という流れになる．なお，途中 (R3) の適用により

$$\texttt{a(),b(Z),c(Z)} \equiv \texttt{b(Z),c(Z),a()}$$

を示すことになるが，この部分が証明全体の半分程度を占める．また，上掲の `solve_refl` タクティックを使うことで，アトム名とリンク名を変数にした場合も

```
Example cong_example : {{ "p"("X","X") }} == {{ "p"("Y","Y") }}.
Proof.
  replace ({{ "p"("Y","Y") }}:Graph) with {{ "p"("X","X")["Y"/"X"] }}; auto.
  apply cong_E4; unfold wellformed_g; auto.
  simpl. auto.
Qed.
```

**図 10　Coq における構造合同関係の例とその証明（アトム名・リンク名が固定の場合）**

```
Example cong_example_var : forall p X Y, {{ p(X,X) }} == {{ p(Y,Y) }}.
Proof.
  intros p X Y. replace ({{ p(Y,Y) }}:Graph) with {{ p(X,X)[Y/X] }}.
  - apply cong_E4; solve_refl.
  - solve_refl.
Qed.
```

**図 11　Coq における構造合同関係の例とその証明（アトム名・リンク名が変数の場合）**

同様に証明可能できた.

## 4　LMNtal プログラムの性質とその証明

本節では，LMNtal プログラムにおける普遍的な性質について，Coq による形式的な証明を行う.

### 4.1　ルールの反転と逆実行

本論文で扱う範囲の，すなわち膜やガードといった高度な機能のない LMNtal においては，左辺と右辺を入れ替えた（反転した）ルールを適用すると，逆向きの実行を実現できる [9]．つまり，次の定理が成り立つ：

**定理 1.** LMNtal ルール $r = T$ :- $U$ の**反転** $r^{\mathrm{inv}}$ を $r^{\mathrm{inv}} \triangleq U$ :- $T$ で定めるとき，

$$G \xrightarrow{r} G' \Leftrightarrow G' \xrightarrow{r^{\mathrm{inv}}} G$$

この定理を Coq で記述・証明すると図 12 のようになる．なお，ここではまず定理 1 の右向きを定理 `inv_rrel` として示したのち，反転の反転が元のルールと同一になることを利用して，定理 1 を系 `inv_rrel_iff` として示している.

### 4.2　Admissible な構造合同規則

ある形式体系において，その推論規則のうち一つを除いても元の体系と表現力が同等である，すなわち他の推論規則から導出可能であるとき，そのような推論規則を admissible であるという．例えば，シーケント計算においてはカット規則が admissible であることがカット除去定理として知られている.

LMNtal の構造合同規則においても，(E8) は admissible である [8]．本節ではこの証明を Coq 上で行う.

まず構造合同関係 `cong` の定義から (E8) のみを取り除いた新たな同値関係 `congm` を定義する．以下，`cong` を`==`，`congm` を`==m` で表記する．証明すべき定理は以下である：

```
Theorem congm_cong_iff :
  forall P Q, P == Q <-> P ==m Q.
```

この証明は `cong` と `congm` に関する構造的帰納法により行うが，(E8) 以外の規則については互いに対応する規則を使うことにより自明であるので，実質的に証明すべきは次の補題である：

```
Lemma congm_E8 :
  forall X Y, {{X = Y}} ==m {{Y = X}}.
```

```
Theorem inv_rrel: forall r G G', G' -[r]-> G
  -> let inv_r := inv r in G -[ inv_r ]-> G'.
Proof.
  intros r G G' H. simpl.
  assert (A: wellformed_r r -> wellformed_r (inv r)).
  { simpl. destruct r. apply link_list_eq_commut. }
  induction H.
  - apply rrel_R1; auto.
  - apply rrel_R3 with (G1') (G1); auto.
    + apply cong_sym; auto; apply cong_wellformed_g in H1; destruct H1; auto.
    + apply cong_sym; auto; apply cong_wellformed_g in H0; destruct H0; auto.
  - apply rrel_R6; auto.
Qed.

Lemma inv_inv : forall r, inv (inv r) = r.
Proof. intros [lhs rhs]. reflexivity. Qed.

Corollary inv_rrel_iff: forall r G G', G' -[r]-> G
  <-> let inv_r := inv r in G -[ inv_r ]-> G'.
Proof.
  intros r G G'. split.
  - apply inv_rrel.
  - simpl. intros H. apply inv_rrel in H. rewrite inv_inv in H. apply H.
Qed.
```

**図 12　逆向き実行とルール反転の関係性の証明**

congm_E8 の証明の概略は次のとおりである．まず X, Y と異なるリンク名 Z をとる．すると，

$$\begin{aligned} \texttt{X=Y} \;\; &\texttt{==m} \;\; \texttt{Z=X,Z=Y} \quad \cdots \text{(E9)} \\ &\texttt{==m} \;\; \texttt{Z=Y,Z=X} \quad \cdots \text{(E2)} \\ &\texttt{==m} \;\; \texttt{Y=X} \qquad \cdots \text{(E9)} \end{aligned}$$

がいえる．このように非形式的な証明は簡潔であるが，Coq の証明ではいくつかの補題が必要となる．

まず，上記の証明では「X, Y と異なるリンク名 Z をとる」としたが，そのためにはこのような Z が実際に取れることを補題として証明する必要がある：

```
Lemma get_fresh_link_X_Y:
  forall (X Y: Link),
    exists Z, X <> Z /\ Y <> Z.
```

また，==m の両辺は well-formed である必要があるが，上記の証明では明示的に記述していなかった．これについても，Coq の証明では逐一示す必要がある．例えば，コネクタ 1 つのみからなるグラフは，コネクタの価数が 2 であることから必ず well-formed である：

```
Lemma connector_wellformed_g :
  forall X Y, wellformed_g {{ X = Y }}.
```

加えて，Z=X，Z=Y が well-formed であることも確認する．ただし，Z が X と Y のいずれかと等しい場合には Z=X，Z=Y は well-formed でなくなるので，前提条件が必要となる：

```
Lemma wellformed_g_zx_zy:
  forall X Y Z,
    X <> Z -> Y <> Z ->
    wellformed_g {{Z = X, Z = Y}}.
```

以上のように，Coq で証明を行う際には人間が非形式的な証明を書く際には自明視してしまいがちな細部にまで気を配って証明する必要がある．そのため，非形式的な証明を書く場合に比べると作業量は大きく増大するが，その分定義や証明における細かな誤謬を見落とすことがなくなるため，言語仕様の研究においても定理証明支援系は大いに役立つと言える．

## 5 まとめと今後の課題

本論文では，定理証明支援系 Coq の上で LMNtal の抽象構文と意味論を実装し，例題を基にして動作確認を行うとともに，LMNtal 言語に関する性質の証明を Coq 上で形式的に行った．

今回記述した Coq スクリプトは，定義と定理および証明すべてを合わせた全体で約 750 行である．また，定義した関数は 15 個であり，4 節で紹介した逆実行とルール反転に関する定理 `inv_rrel` とその系 `inv_rrel_iff`，および構造合同規則 (E8) の除去定理 `congm_cong_iff` を示すために，計 25 個の補題を証明した．なお，以上は本論文に直接関係のある部分のみを抜粋したデータであり，後述する構造合同規則とグラフ同型の関係性の証明に向けて実装中の Coq スクリプトを含めると，全体は約 1940 行である．

今後の課題としては冒頭で挙げたとおり，今回実装した構造合同規則とグラフ同型との関係性の形式化および証明を目標にしている．すなわち，テキストベースで記述された LMNtal プログラムをグラフ構造に変換する関数 $\mathcal{T}$ を Coq 上に実装したうえで，

$$P \equiv Q \Leftrightarrow \mathcal{T}(P) \simeq \mathcal{T}(Q)$$

（ただし $\simeq$ はグラフ同型を表す）を証明することが目標である．しかし，変換先のグラフ構造の定式化は一意に定まるものではなく，Coq での証明に適した形を選択する必要がある．ここで，1 節で紹介した Graph Theory [2] のようなライブラリを活用することも考えられるが，LMNtal におけるグラフは通常のグラフ理論におけるグラフの定式化とは大きく異なるため，そのままの形で利用することは難しい．また，変換関数 $\mathcal{T}$ やグラフ同型 $\simeq$ の定式化にあたっても同様の注意が必要である．

一方で，項の等価性判定問題をグラフ同型性判定問題に帰着させる研究 [1] が存在することから，この研究を足がかりとし，構造合同規則と項の等価性判定問題の対応関係を明らかにするという手法も考えられる．実際，$\pi$ 計算を項の等価性判定問題に帰着させることにより，間接的に $\pi$ 計算をグラフ同型性判定問題に帰着させた研究 [3] も存在する．この手法の場合は，前述の通りグラフの定式化の違いに十分留意する必要があることに加え，構造合同規則には LMNtal 独自の言語要素であるコネクタが使われていることにも注意が必要である．

**謝辞** 本研究の一部は，科学研究費基盤研究 (B) 18H03223 および早稲田大学特定課題研究費 (2021C-142, 2021R-016) の助成を受けて実施した．

## 参 考 文 献

[1] Basin, D. A.: A term equality problem equivalent to graph isomorphism, *Information Processing Letters*, Vol. 51, No. 2(1994), pp. 61–66.
[2] Doczkal, C. and Pous, D.: Graph Theory in Coq: Minors, Treewidth, and Isomorphisms, *Journal of Automated Reasoning*, (2020).
[3] Khomenko, V. and Meyer, R.: Checking pi-Calculus Structural Congruence is Graph Isomorphism Complete, *Proc. ACSD 2009*, 2009, pp. 70–79.
[4] Team, T. C. D.: The Coq Proof Assistant, (2021).
[5] Ueda, K.: Encoding the Pure Lambda Calculus into Hierarchical Graph Rewriting, *Proc. RTA2008*, LNCS, Vol. 5117, Springer, 2008, pp. 392–408.
[6] Ueda, K.: Towards a Substrate Framework of Computation, *Concurrent Objects and Beyond*, LNCS, Vol. 8665, Springer, 2014, pp. 341–366.
[7] Ueda, K. and Ogawa, S.: HyperLMNtal: An Extension of a Hierarchical Graph Rewriting Model, *KI - Künstliche Intelligenz*, Vol. 26, No. 1(2012), pp. 27–36.
[8] 佐野仁, 上田和紀: ハイパーグラフ書き換え系への構文駆動で compositional な構文・意味論の提案, 日本ソフトウェア科学会第 *38* 回大会 *(2021* 年度*)* 講演論文集, 2021.
[9] 山本直輝, 上田和紀: グラフ書換え言語における数値制約を伴う型の静的型検査, 日本ソフトウェア科学会第 *37* 回大会 *(2020* 年度*)* 講演論文集, 2020.
[10] 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発展, コンピュータ ソフトウェア, Vol. 28, No. 4(2011), pp. 4_137–4_157.
[11] 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.
[12] 信夫裕貴, 田辺良則, 上田和紀: LMNtal におけるグラフ書換え操作の Coq による形式化, 日本ソフトウェア科学会第 *30* 回大会 *(2013* 年度*)* 講演論文集, 2013.
[13] 石川力, 堀泰祐, 村山敬, 岡部亮, 上田和紀: 軽量な LMNtal 実行時処理系 SLIM の設計と実装, 情報処理学会第 *70* 回全国大会講演論文集, (2008), pp. 153–154.