プログラミング-言語・基礎・実践- 21－4
(1995. 3. 9)

並列離散事象シミュレーション言語「もえぎ」の構想

大澤 範高, 弓場 敏嗣
電気通信大学 情報システム学研究科
〒182 東京都調布市調布ヶ丘 1-5-1
E-mail: {osawa, yuba}@is.uec.ac.jp

あらまし 高度なシステムの設計のためには、大規模シミュレーションが不可欠になっている。シミュレーションを高速に行なうためには、単一プロセッサでは限界があり、並列計算機の活用の必要性が高まっている。並列シミュレーションの性能改善には、システムによる解析だけではなく、ユーザの注釈情報の活用が重要である。ユーザの注釈情報を柔軟に扱える機能を持った離散事象シミュレーション用並列オブジェクト指向言語「もえぎ」の構想を述べ、その処理系の実装状況を報告する。

和文キーワード 離散事象シミュレーション, 並列オブジェクト指向言語, プロトタイプベース, 注釈

A Parallel Language for Discrete Event Simulation: "Moegi"

Noritaka OSAWA, Toshitsugu YUBA
Graduate School of Information Systems
The University of Electro-Communications
Chofugaoka 1-5-1, Chofu, Tokyo 182, JAPAN
E-mail: {osawa, yuba}@is.uec.ac.jp

Abstract Simulation is an indispensable tool for designing and verifying advanced systems. A parallel computer is expected to be a good vehicle for simulation because of performance limitation of a single processor system. A simulation system has to be capable of analyzing programs by collecting runtime information. Moreover, the system should permit programmers to annotate simulation programs in order to speed up the system execution. A parallel language for discrete event simulation "Moegi" is proposed for these objectives. The language can help programmers to add annotations for performance improvement to programs. The implementation status of the language system is also explained.

英文 key words discrete event simulation, parallel object-oriented language, prototype-based, annotation

1 はじめに

高度通信システム、製造システム等の設計のためには、シミュレーションが不可欠になっている。対象は、年々大規模化、複雑化しており、より高速なシミュレーションが必要とされている。

単一プロセッサ計算機では十分な速度でシミュレーションを実行することができない。したがって、複数プロセッサから構成される並列計算機によってシミュレーションを高速に実行できるようにすることが重要性を増している。

また、より多数の実体を記述する必要が増えており、その記述性を向上することも必要である。

並列・分散離散事象シミュレーションの方式は、大きく、保守的(Conservative)方式と楽観的(Optimistic or Aggressive)方式に分類できる[5][11][15]。保守的方式では、因果律違反(Causality Error)が発生しないようにシミュレーションを進める。シミュレーション実行の際に、保守的方式では因果律違反を発生させるか否かの判定を効率よく行なう必要がある。一方、楽観的方式では、並列に処理を進め、因果律違反の発生を検出した場合には巻戻し(Rollback)を行なう。楽観的方式では、因果律違反の発生による巻戻しおよびその準備のオーバーヘッドが小さくなるように処理することが重要である。

どちらの方式においても、事象発生の予測精度を高めることによって、高速化を図ることができる。これには、シミュレーション対象の記述をコンパイラが解析する方法、実行時情報から予測を行なう方法、ユーザからの注釈を利用する方法などがある。自動的にコンパイラや実行時処理系が高い精度の予測を行なうことが望ましいが、ユーザからの注釈の活用も重要である。

ここで、注釈とは、正しさには影響しないが性能を向上させるためにユーザが行なう指示のことをいう。並列計算機におけるプロセッサの指定なども注釈と考える。

本稿では、高性能シミュレーション技術の研究環境の基礎として利用する並列オブジェクト指向離散事象シミュレーション処理用言語「もえぎ」の構想およびそのプロトタイプ処理系の現状を報告する。

2 設計思想

並列オブジェクト指向離散事象シミュレーション処理用言語「もえぎ」の設計では以下を重視している。それぞれについて簡単に説明する。

1. 柔軟な注釈機能
2. オブジェクト指向(プロトタイプベース)
3. 並列性の有効利用
4. 離散事象向き機能
5. 表現性向上

柔軟な注釈機能

シミュレーションの並列実行を高速化するためには、予測情報を有効に利用するシミュレーションアルゴリズムの開発が必要である。精度の高い予測情報を基にすることによって、並列性を引き出し、シミュレーション処理を高速に行なうことができる。

予測精度を高めるためには、システムがプログラムを静的に解析する方法とシステムが統計情報を動的に収集する方法、ユーザからの注釈に基づく方法がある。従来はシステムによる方法への依存度が大きかった。「もえぎ」においては、ユーザからの注釈を受け入れやすくし、より高速な実行が可能となるようにする。

注釈は対象等に依存する部分もあり、研究の進展にしたがって、注釈の種類を増やしたり、注釈方法を改善する必要がある。そのような変更が容易になる構文をめざす。

オブジェクト指向(プロトタイプベース)

オブジェクト指向の考え方に基づいた言語とし、シミュレーションモデルにおける実体との対応をとりやすくする。また、クラスとインスタンスを区別しないプロトタイプベースの言語とする。プロトタイプベースの言語では、オブジェクトの生成は、複製によって行なわれる。複製を基本とすることによって、オブジェクトに対して行なわれた注釈を継承することが容易になる。たとえば、オブジェクトを配置するプロセッサの指定の継承が自然にできる。プロトタイプベース言語としては Self [14] に近い。

並列性の有効利用

シミュレーションの高速化のために並列計算機を有効に利用できる機能が必要である。処理を並行して実行できる機能を持たせる。

離散事象向き機能

離散事象シミュレーションに不可欠な仮想時間 (シミュレーション時間) を管理する機能、および、シミュレーションの際に必要な乱数生成、データの整理に必要な統計処理機能を標準で備える。

表現性向上

シミュレーションプログラムを文書としても利用可能な構文とする。将来的には視覚的プログラミング言語 (Visual Programming Language) への拡張が容易になるようにする。

3 従来のシステム

従来の離散事象シミュレーションシステムは、既存の言語に離散事象シミュレーションライブラリを追加して構成されるシステムと新しく設計された離散事象シミュレーション用言語から構成されるシステムに大きく分けられる。

ライブラリ

既存の言語を利用し、ライブラリを追加するシステムでは、シミュレーションのために新たな言語を覚える必要がなく、ライブラリの仕様を理解するだけでよい。一方、ライブラリで実現できない部分は基礎となる言語で記述する必要があり、プログラミングが繁雑になる場合がある。注釈の研究では、ライブラリだけではなく、コンパイラ等の言語処理系を変更する必要があり、ライブラリの追加によるシミュレーションシステムの構築は注釈の研究には適していない。

C++ のシミュレーションライブラリとしては、たとえば、OLPS [1]、YANSL [9]、Awesime [7] があり、Smalltalk-80 への機能追加を行なったシステムとしては、SimTalk [10] 等がある。ライブラリで実現する追加機能は、キューイングサポート、統計情報収集、シミュレーション指向グラフィックス等である。

言語

GPSS をはじめとして種々のシミュレーション言語がある。そして、最近のシミュレーション言語の多くは、シミュレーション対象のモデルとの対応を取りやすくするためにオブジェクト指向言語になっている。

並列シミュレーション言語としては、Maisie [2] 等があり、オブジェクト指向を採り入れたシミュレーション言語としては、MODSIM II [3] などがある。しかし、これらの言語には、注釈の表現法がなかったり、特定の注釈を固定的な形式でのみ指定可能であったりする。

4 構文

「もえぎ」の構文は、注釈を柔軟に付加でき、自然言語風の表記によるプログラムの文書性の向上ができることを目標としている。拡張 BNF による予備的な構文規則を付録 Aに示す。

注釈および文書性

注釈はハードウェア技術、ソフトウェア技術 (コンパイル技術) の進展によって変わる。このため、注釈の形式を固定的に定めるのは、注釈の研究には不適切である。また、標準組み込みの機能とユーザの拡張を表記上区別されないようにしたい。このため、C 言語等の制御構造とライブラリ (関数) が明確に区別される言語は適していない。また、関数形式や Lisp 言語の S 式は、統一的で簡潔であるが文書性で劣り、やはり目的に合致しない。

プログラムをシミュレーション対象の文書としても利用可能にできるように、従来のプログラミング言語のような表記だけではなく、日本語、英語等の自然言語に似た表現を許す構文とする。これによって、プログラムの記述性、文書性を高められると考える。(文書性向上の可能性は、プログラムの読みやすさの向上を必ずしも意味するものではない)

セレクタ

「もえぎ」の構文は、Smalltalk-80 [6] のように引数間にキーワードを挿入する形式が基礎となっている。ただし、Smalltalk-80 と異なり、キーワードを最後の引数の後にも付加できる。キーワードと引数を表す特殊な文字の連結からセレクタが構成される。

キーワードは、「:」「.」 で終る通常文字列 (空白および予約文字を含まない文字の列)、もしくは、あらかじめユーザが指定した文字で始まるか終わる通常文字列である。以下の例では、「:」「=」「%」「、」「。」で終わる通常文字列をキーワードとする。

細かい点であるが、キーワードが、メッセージ式の

最後であることが明らかである場合には、キーワード最後のユーザが指定した文字(読点等)を省略できる。ただし、メッセージ式の最後のキーワードの最後のユーザ文字のみが異なるセレクタ定義はできない。

その他

ブロックには、Smalltalk-80の[]ではなく{}を利用する。また、コメントは、#から行の終りまでである。

5 機能

「もえぎ」は、並列オブジェクト指向言語(プロトタイプベース言語)としての基本的な機能を備える。それに加えて離散事象シミュレーション向き機能を持つ。

5.1 オブジェクト指向

「もえぎ」は、プロトタイプベース言語である。プロトタイプベース言語としてはSelf[14]に近いので、その用語をここでは用いる。スロットは、変数、メソッドを統合したものであり。オブジェクトは、複数のスロットを持つことができる。

オブジェクトは、複製(cloning)によってのみ生成される。複製時には、複製元のスロットを引き継ぐ。これによって単純継承を実現できる。複製後に、スロットを追加することが可能である。

「もえぎ」では、スロットは次の特性を持つ。

- 仮想時間依存性
- 非同期性

仮想時間依存性は、シミュレーションの進行に依存するか否かについての特性である。楽観的実行を行なう場合に、回復する必要がある状態か否かを指定できることになる。メッセージの受信回数(手続き呼び出し回数)などの統計情報を保存している情報は仮想時間非依存として指定することによって、巻戻しの際に、値が戻ってしまうことを防ぐことができる。デフォルトは、仮想時間依存である。

非同期性は、メッセージ送信時に、送信側が戻り値の返信を待つか否かについての特性である。

5.2 並列処理

メッセージの送信を行なった際に戻り値を待つか否かは、スロットの特性によって決められる。この戻り値待ちを行なわないことによって、複数のオブジェクトの並列実行が可能となる。

オブジェクト内の実行は、排他的に連続して処理される。ただし、他のオブジェクトへメッセージを送信し、待ちに入る場合には、その時点で他のメッセージが処理される可能性がある。このことは、シミュレーションの保守的実行を行なう場合には、オブジェクトをモニタとして実装することによって実現できる。楽観的実行を行なう場合には、排他性を満たさなかった場合には巻戻しが発生する。

オブジェクト間のメッセージ送受信順序は保存される。すなわち、あるオブジェクト O_s が別のオブジェクト O_r へメッセージ M_1、M_2 をこの順序で送信した場合には、O_r は、M_1、M_2 の順に受信する。

5.3 等価表現

名前(セレクタ)の等価性を指定することによって、処理系がそれらの間の変換を行なうことができる。セレクタの部分だけではなく、引数の順序を変更することが可能である。たとえば、英語風表記(プログラミング言語風表記)と日本語風表記を変換できる。

```
equiv: { plus: x minus: y }
  to: { から、 y を引き、 x を足す。 };
```

のように対応関係を設定すれば、

```
1 plus: 2 minus: 3;
```

と

```
1 から、 3 を引き、 2 を足す。
```

は等価となる。どちらの表記でも入力できるだけではなく、表記の優先順位を環境に指定しておくことによって、どちらの表記でも出力することが可能である。将来的には図的な表現との等価性を指定できるようにし、視覚的プログラミングを可能とする予定である。

自然言語風表記の例として、最大公約数を求めるプログラム例を示す。処理系に組み込まれた変換機能を用いることによって、適切な等価表現指定を行なえば、図1のプログラムを図2のように変換することができる。この逆変換も同様に可能である。

5.4 離散事象シミュレーション向き機能

仮想時間

離散事象シミュレーションに必要な仮想時間(シミュレーション時間)を管理する機能を持つ。オブジェクト間メッセージ送信時に仮想時刻がつけられ、

```
手続き: { 引数: m と、n の最大公約数を求める。} を、 {
    m に、 ( m の絶対値 ) を代入する。
    n に、 ( n の絶対値 ) を代入する。
    条件: { m が、0 ではない } が成立している限り、 {
        初期化: { t に、 m を代入する } を行ない、 {
            m に、 ( n を、 m で割った剰余) を代入する。
            n に、 t を代入する。
        } を実行する。
    } を繰り返す。
    n ; # 戻り値
} と定義する。
```

図 1: 最大公約数を求める (日本語風)

```
define: { GCD: m and: n } as: {
    m:= (abs: m );
    n:= (abs: n );
    while: { m != 0 } do: {
        let: { t:= m } in: {
            m:= ( n % m );
            n:= t ;
        };
    };
    n ;
};
```

図 2: 最大公約数を求める (プログラム言語風)

受信側オブジェクトの仮想時刻が、送信の仮想時刻になるまでメッセージは処理されない。

離散事象シミュレーションを容易にするために、シミュレーションのための標準オブジェクトを用意する。また、離散事象シミュレーションに必要な仮想時間を扱う機能を持つ。さらに、シミュレーションの統計計算向きのスロット定義用標準メッセージを用意する。これらによって、シミュレーションを柔軟かつ容易に制御できる。

洗車問題の記述例を図 3に示す。受付係、洗車係に定義されている手続きは、非同期に実行されるとする。

ここでは、洗車問題として、1 人の受付係と 2 人の洗車係から構成される洗車場を考える。受付係は、車が到着すると、空きリストから空いている洗車係を見つけ、その洗車係に仕事を依頼する。洗車係は、依頼された洗車を一定時間かかって行ない、完了するとそれを受付係に知らせる。受付係は、洗車完了を知ると空きリストにその洗車係を登録する。洗車問題におけるオブジェクト間メッセージの流れを 図 4 に示す。

標準オブジェクト

離散事象シミュレーションのために以下のオブジェクトを用意する。

- 乱数生成
- 統計計算

一様分布、正規分布、2 項分布、ポアソン分布、ガンマ分布等の標準的な乱数を生成するためのオブジェクトを用意する。

また、シミュレーション結果を解析するために統計計算機能をもったオブジェクトを用意する。

スロット定義

スロット定義では、通常の値の代入、参照だけではなく、アクセス回数のカウントおよび値の集計なども行なう定義を簡潔にできるようにする。これによって、統計の基礎的データを容易に収集することができる。

仮想時間に関連した標準メッセージとしては、下記を基本とする。

仮想時間待ち 指定した仮想時間だけ実行を中断する。図 3 の例では、

X が時間: T の経過を待つ。

に相当する。

仮想時刻待ち 指定した仮想時刻になるまで実行を中断する。

5.5 注釈機能

シミュレーションにおける注釈の効果の評価、および、ユーザによる注釈とシステムによる予測との統合法については今後の研究課題であるが、現在検討している注釈の例をいくつか以下に示す。

オブジェクトの配置指示 オブジェクトをどの PE で処理するかを指示できる機能を持たせる。

状態保存単位指示 状態保存単位を変更することによって楽観的実行の際のオーバーヘッドを変化さ

```
# 変数定義
変数: 受付係 を定義する。
変数: 洗車係A を定義する。
変数: 洗車係B を定義する。

# 受付係の生成
受付係:= ( オブジェクト を複製 );
        # 受付係:= は代入を表す。
# 受付係の定義
受付係 変数: 空きリスト
    を初期値: ( キュー を複製 ) として定義する。
受付係 の手続き、{ に、車 が到着する。 } を、{
    初期化: { 洗車係 に、
        ( 空きリスト の先頭を取り出す ) を代入する }
    を行ない、{
        洗車係 が、車 を洗車する。
    } を実行する。
} と定義する。
受付係 の手続き、{ が、洗車係 の洗車完了を知る。 } を、{
    空きリスト に、洗車係 を登録する。
} と定義する。

# 洗車係 A の生成
洗車係A:= ( オブジェクト を複製);
# 洗車係の定義
洗車係A の定数: 洗車時間 を、10 として定義する。
洗車係A の手続き、{ が、車 を洗車する。 } を、{
    自分 が時間: 洗車時間 の経過を待つ。
    受付係 が、自分 の洗車完了を知る。
        # 自分 は Smalltalk-80 の self を表す。
} と定義する。

# 洗車係 B の生成
洗車係B:= ( 洗車係A を複製);

# 初期化
受付係 が、洗車係A の洗車完了を知る。
受付係 が、洗車係B の洗車完了を知る。

# 実行
受付係 に、( 車 を複製) が到着する。
受付係 が時間: 5 の経過を待つ。
受付係 に、( 車 を複製) が到着する。
受付係 が時間: 3 の経過を待つ。
受付係 に、( 車 を複製) が到着する。
```

図 3: 洗車問題の記述例

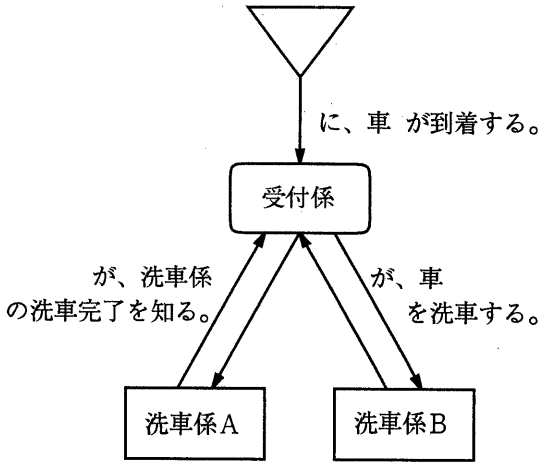

図 4: 洗車問題におけるメッセージの流れ

せることができるようにする。Time Warp[8] の場合のイベントやページング機構を利用する際のページなどを単位として指定できるようにする。

仮想時間同期指示 仮想時間 (シミュレーション時間) 同期の時間単位を変化させることによって、楽観的な実行形態から、保守的な実行形態まで、変化させることができるようにする。

巻戻し予測実行指示 巻戻しが発生するか否かを統計的に調べ、巻戻しの発生を予測して、それに応じて保守的実行するか楽観的実行するかを変更することが可能である。この調査、予測、変更を行なうか否かを指示できるようにする。

6 プロトタイプ処理系

UNIX オペレーティングシステム、および、分散記憶型並列計算機上で動作するプロトタイプ処理系を作成中である。Sun OS および 並列計算機 Cenju-3 上を対象としている。Sun OS 上では、プロセスを仮想プロセッサとして利用している。プロセス間通信には、Message-Passing Interface(MPI)[12] ライブラリを利用している。

プロトタイプシステムは、インタプリタ処理系としており、仮想時間管理には、まず保守的方式を利用している。楽観的方式も利用できるようにする予定である。

また、多種の文字コードを認識できないキャラクタベースの表示環境を想定しているため、日本語と英語のみが現時点では利用可能である。処理系内部の(文字)コードには、16ビットを単位とする可変長文字コードを使用している[13]。

Sun OS 上での単一プロセス版から複数プロセス版を作成途上である。

7 今後の予定

今後の予定・課題は以下の通りである。

- Cenju-3 上への移植
- コンパイラの作成
- GUI(Graphical User Interface) を利用した多国語対応、視覚的言語対応
- 注釈機構の評価

本研究は、郵政省 電気通信フロンティア研究開発・高度統合網のための高性能シミュレーション技術の研究の一部として行なわれた。

参考文献

[1] Abrams, M., "The Object Library for Parallel Simulation (OLPS)," Proc. of the 1988 Winter Simulation Conf., pp.210–219.

[2] Bagrodia, R. L. and W. T. Liao, "A Language for Interactive Design of Efficient Simulations," Technical Rep. UCLA-CSD-920044, CSD, UCLA, Oct. 1992.

[3] Belanger, R. and A. Mullarney, Modsim II tutorial, CACI Products Company, La Jolla, CA, 1990.

[4] Bezivin, J., "Some Experiments in Object-Oriented Simulation," OOPSLA 87, pp.394–405, Oct. 1987.

[5] Fujimoto, R. M., "Parallel Discrete Event Simulation," Comm. of ACM, Vol.33, No.10, pp.30–53 , 1990.

[6] Goldberg, A. J. and D. Robson, Smalltalk-80^{TM}: The Language and Its Implementation, Addison-Wesley Pub. Co., 1983.

[7] Grunwald, D., A Users Guide to Awesime: An Object Oriented Parallel Programming and Simulation System, CU-CS-552-91, University of Colorado, 1991.

[8] Jefferson, David R., B. Beckman, F. Wieland, L. Blume, M. DiLoreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J. Wedel, H. Younger and S. Bellenot, "Distributed Simulation and the Time Warp Operating System," ACM 11th Symp. on Operating Systems Principles, pp.77–93, 1987.

[9] Joines, J. A., K. A. Powell, Jr. and S. D. Roberts, "Object-Oriented Modeling and Simulation with C++," Proc. of the 1992 Winter Simulation Conf., pp.145–153.

[10] Knapp, V. E., "The Smalltalk Simulation Environment, Part II," Proc. of the 1987 Winter Simulation Conf., pp.146–151.

[11] Misra, J., "Distributed Discrete-Event Simulation," ACM Computing Surveys, Vol.18, No.1, pp.39–56, 1986.

[12] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, March 1991.

[13] 大澤範高, 木村憲雄, "プログラミング言語「きなり」," トロン技術研究会, Vol.5, No.2, pp.39–50, 1993.

[14] Ungar, D. and R. B. Smith, "Self: The Power of Simplicity," OOPSLA '87, pp.227–242, 1987.

[15] 米澤 明憲, 柴山 悦哉, モデルと表現, 岩波書店, 1992.

A 構文

拡張したBNFによって「もえぎ」の予備的な構文を示す。<、> で囲まれた記号が、非終端記号。要素をすべて列挙することが困難な場合には、< > 中に説明している。| によって選択肢の区切りを示す。(、) はグルーピングを意味する。[、] で囲まれた部分は、省略可能であることを示す。* は 0 回以上の繰り返し。+ は 1 回以上の繰り返し。

2進数字 ::= **0** | **1**

8進数字 ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7**

数字 ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

16進数字 ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** | **a** | **b** | **c** | **d** | **e** | **f**

符号 ::= + | −

2進数 ::= [<符号>] **0b** <2進数字>+

8進数 ::= [<符号>] **0o** <8進数字>+

10進数 ::= [<符号>] <10進数字>+

16進数 ::= [<符号>] **0x** <16進数字>+

整数 ::= <2進数> | <8進数> | <10進数> | <16進数>

仮数部 ::= [<符号>] <10進数>+ . [<10進数>+]

指数部 ::= (**e** | **E**) [<符号>] <10進数>+

小数 ::= <仮数部> [<指数部>] | <10進数> [<指数部>]

文字定数要素::= <' と \ を除いた文字> | \' | \t | \n | \0 | \\

文字定数 ::= ' <文字定数要素> '

文字列要素 ::= <" と \ を除いた文字> | \" | \t | \n | \0 | \\

文字列 ::= " <文字列要素>* "

空白文字 ::= <スペース文字、タブ文字、改行文字>

コメント ::= <#から行末まで>

空白 ::= (<空白文字> | <コメント>) +

キーワード先頭文字::= <ユーザがキーワード先頭文字として指定した文字集合>

キーワード末尾文字::= <: とユーザがキーワード末尾文字として指定した文字集合>

キーワード文末文字::= <. とユーザがキーワード文末文字として指定した文字集合>

予約文字 ::= { | } | (|) | [|] | " | ' | # | ; |

通常文字 ::= <空白文字および予約文字を除いた文字>

文末キーワード::= <通常文字>*<キーワード文末文字>

キーワード ::= <キーワード先頭文字><通常文字>* | <通常文字>*<キーワード末尾文字>

識別子先頭文字::= <空白文字および予約文字、数字、符号、キーワード先頭文字を除く文字>

識別子末尾文字::= <空白文字および予約文字、数字、キーワード末尾文字、キーワード文末文字を除く文字>

識別子 ::= <識別子先頭文字> [<通常文字>* <識別子末尾文字>]

リテラル ::= <整数> | <小数> | <文字> | <文字列>

引数リスト ::= [<空白>] (<識別子> <空白>)* [<識別子>]

ブロック ::= { [<引数リスト> |] <文リスト> }

リスト要素 ::= <リテラル> | <キーワード> | <識別子> | <ブロック> | <リスト>

リスト ::= (* [<空白>] (<リスト要素><空白>)* [<リスト要素>])

1次式 ::= <識別子> | <リテラル> | <ブロック> | <リスト> | (<空白文字>* <式> <空白文字>*)

引数 ::= <1次式>

オブジェクト::= <1次式>

メッセージ式::= <オブジェクト> (<空白><キーワード><空白><引数>)* [<空白>(<キーワード>) | <文末キーワード>)] | <キーワード> (<空白><引数><空白><キーワード>)* [<空白><引数> [<空白><文末キーワード>)]] | <文末キーワード>

完結メッセージ式::= <オブジェクト> (<空白><キーワード><空白><引数>)* <空白><文末キーワード> | <キーワード>(<空白><引数><空白><キーワード>)* <空白><引数> <空白><文末キーワード> | <文末キーワード>

式 ::= <1次式> | <メッセージ式> | <完結メッセージ式>

文 ::= <文末記号> | <式> [<空白>] <文末記号> | <完結メッセージ式> | <ブロック>

文リスト ::= [<空白>] (<文> [<空白>]) * [<文>]

プログラム ::= <文リスト>