言語ゲーム

LanguageGame – interactive parser generator

山宮 隆
Takashi YAMAMIYA
株式会社クリプトワンソフト
(〒564-0063 大阪府吹田市江坂町 1-6-1 スペース・江坂 I 3F Email: yamamiya@qriptone.ne.jp)

ABSTRACT: LanguageGame is a parser generator that has graphical user interface. Contrary to popular professional parser generator like yacc/lex in UNIX platform, it is intended to use non-professional computer users, students and children. Playing with LanguageGame, users can learn and build simple parser with pretty graphical interface, drag-and-drop and graphical syntax tree representation.

１．背景

コンピュータに指示を与える為には、メニュー選択、ボタン、ドラッグなどの様々な手段がある。近年の研究、とりわけ、WYSWIG(What You See Is What You Get) と呼ばれる思想により、ユーザインタフェースはより豊かになり、エンジニア以外の人々にも馴染みやすいものとなった。しかし、コンピュータと対話する最も基本的な手段はやはりプログラミング言語のような一連のシンボルによるものである。

進化した紙として代替的にコンピュータを使う限りWYSWIG方式は有効かもしれない。しかし、より抽象的で複雑な作業においては、プログラミング言語による操作に利点がある。なぜなら、このような言語のシンボルはより信頼性と柔軟性を持ち、再帰や参照などの強力な概念が使えるからである。GUIがいかに発達してもシンボルによるプログラミング言語の重要性は変わらない。

世界には数多くのさまざまな言語があり、それぞれ興味深い性質と特徴を備えている。これらプログラミング言語の文法には非常に興味を惹かれるものであると同時に、言語自身の開発をしてみたいと云う誘惑も大きい。もちろん、言語の開発は非常に困難である。そこで、人々が彼ら自身のプログラミング言語を簡単に作れるような方法を考える事にした。

パーサジェネレータはコンピュータ言語を開発する為の最も基本的なツールのひとつである。歴史的にさまざまなプログラミング言語が開発されてきたが、これらの言語のパーサを簡単に開発する為に、自動パーサ生成系の技術が開発されてきた。いまやパーサジェネレータは確立された技術と言えるだろう。

２．目的

ここでは、**「言語ゲーム」**と名づけられた画像ベースのパーサジェネレータについて述べる。言語ゲームの目的は、分かりやすいユーザインタフェースを備え、専門家以外でも使えるようなたパーサジェネレータを、コンピュータを深く学ぶ為のツールとして提供する事である。コンピュータにとって、プログラミング言語は不可欠なものである。プログラミング言語を深く学ぶ事によって、コンピュータをより深く知り、有効に活用する事が出来る。

言語ゲーム開発の第一段階として、最初にこのプロジェクトに備わるべきユーザインタフェースについて調査した。インタフェースの開発に専念する為に、再利用できるGUIライブラリがすでに大量に存在し、開発が容易なSqueak[1]を選んだ。また、パーサ生成エンジンについては、Squeakで利用できる既存のソフトウェアであるT-Gen[2]をベースに開発を行った。

開発が進むにつれて、言語ゲームが備えるべき性質についても明らかになった。特に重要な点として以下のものが挙げられる。

（１）操作のリアルタイム性

ユーザの操作を出来るだけすぐに反映させる事により効率の良い開発が期待できる。このリアルタイム性の為に必要なデータ構造について設計を行った。

（２）プログラムの外部表現

通常プログラミング言語では一連のテキストを記述する事によって意味を記述する。しかし、解析可能な情報であれば特にテキストにこだわる必要はない。言語ゲームでは、ユーザにより親しみやすい印象を与える為、絵文字によるプログラミングをサポートした。

（３）意味解析

構文を定義する為には、すでにBNF(Backus-Naur Form)が実用的な記法として確立している[3][4]。しかし構文ツリーを生成したあとに、文の意味を定義する方法に標準はない。言語ゲームにおいてはSmalltalkのブロック文記法をそのまま使用する事によって柔軟な意味表現を可能とした。

（４）デバッグ

プログラマだけでなく一般ユーザにとってもコンピュータがどのようなプロセスで動作しているのかを把握する事は重要である。コンピュータの動作を追跡し、可能であれば動的に

振る舞いを変更する事によってユーザはシステムの動作を直感的に把握する事が出来る。この目的を実現する最低限シンプルなデバッガについて開発した。

（5）外部オブジェクトとの通信

自由に外部のオブジェクトを操作する事によって、ユーザは楽しいアプリケーションを作成する事が出来る。単なる教材としてではなく、気の利いた道具として言語ゲームを利用できるように、シンプルな外部オブジェクトとのインタフェースを設計した。

3．詳細

（1）設計

a） メインウインドウ

図 1 メインウインドウ

メインウインドウ(図1)は上下二つの領域に分けられる。上部を文法ペインと呼び、BNF記法によって文法を記述する。下部を入力ペインと呼び、これから解析する文を入力する。それぞれのインタフェースとして、テキストモードとタイルモードが存在し、タイルモードではドラッグアンドドロップを使って直感的に文法及び文を構築する事が出来る

図 2 文法ペインメニュー

図 3 入力ペインメニュー

右クリックによって、文法ペインでは図2、 入力ペインには図3のメニューが現れる。コマンドの意味は以下の通りである。

表 1 メニュー

text	テキストモードにトグル
palette	その文法で使用可能なシンボルを表すパレットを表示する
images	シンボルに結びつけられた画像を閲覧する。
tree	文法ツリーの表示
inspect	文法ツリーのインスペクタを表示
explore	文法ツリーを辿る為にカスタマイズされたエキスプローラを表示
fire	現在の文法にアクションが設定されていれば実行する。
save … (文法ペインのみ)	現在の文法を記録(GGrammarUsers クラスのメソッドとして記録される)
load … (文法ペインのみ)	保存してあった文法を呼び出す。
build（文法ペインのみ)	現在の文法を再構築する。
step（入力ペインのみ)	現在の入力をステップ実行する。
monad（入力ペインのみ)	現在の入力によって構築される遅延実行オブジェクトを表示(開発時デバッグ用)

b） 文法の記法

言語ゲームで使われるBNF記法は、T-Genで使用されるもののサブセットになっている。なお、T-Genと異なりスキャナを独自に作成する事は出来ない。スキャナに関しては言語ゲームの用意する標準のスキャナを使用する。文法は以下のように記述する。

非終端記号 ： 記号 … [アクションブロック]

- 右辺はいずれも省略できる。
- 記号は非終端記号か終端記号のいずれかである。
- 記号にはアルファベットを先頭とし、空白を含まないする任意の文字列が使える。
- ルールは複数記述できる。それぞれのルールは改行で区切る。
- アクションブロックには任意の Smalltalk 文が記述できる。

c） タイル

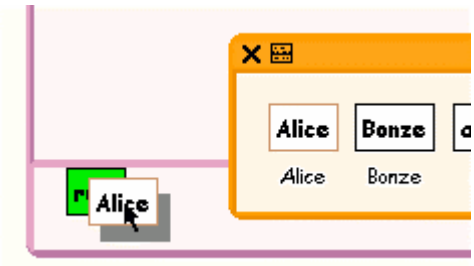

図 4 終端記号のドラッグ

図 5 入力補完

文法ペイン、入力ペインのどちらでもタイルモードによる入力が出来る。タイルモードでは、ドラッグアンドドロップによるシンボルの追加が可能である。

入力ペインでタイルを使うには、まずパレットを右クリックメニューからパレットを呼び出す。パレットには、現在の文法で使用可能な終端記号が現れる。終端記号をドラッグして、入力ペインの非終端記号の上に重ねる。もしも該当する非終端記号がドラッグしたタイルを解釈できる場合、非終端記号は緑色に変化する。

非終端記号が、ドラッグされた終端記号に対してルールを一意に決定する事が出来る場合、そのルールの残りの終端記号は自動的に補完される。

d) ブロックに SqueakToys のタイルを使う

図 6 SqueakToys のフレーズタイルをドラッグ

図 7 コードに変換されたフレーズタイル

文法ペインに関しても同じようにタイル操作が出来る。ただ、ルールの一番右側の要素であるブロックについては、SqueakToysのビューワからフレーズタイルを直接ドラッグできる。この場合、タイルはブロックコンテキストの表現形式に自動的に変換される。

e) タイルにイメージを使う

図 8 externalName を編集する

図 9 画像を終端記号として使う

図 10 登録された画像の操作

タイルとして、単なる文字列だけではなく任意のモルフを使用する事が出来る。言語ゲームはそのモルフのexternalName をタイルの終端記号として扱う。具体的には、モルフに名前を付け、タイル上にドラッグする事により任意のモルフがタイルのシンボル文字列として扱われる。モルフと文字列は言語ゲーム内部で登録され、登録された文字列はいつでもモルフに変換される。

この機能により、絵文字によるプログラミングが実現できる。英語を話せないユーザの助けになる他、将来的には別の言語を持つ人々が共同作業を行う際に役立つ可能性がある。

メニューのimagesから登録されたモルフを閲覧/削除する為のツールを呼び出す事が出来る。

f) アクションブロック

解析された文に意味を付加させる為、アクションブロックと呼ぶ仕組みを導入している。例を示す。

carton : [0]

carton : egg carton [:a :b | 1 + b]

この文法は入力された文字列の中で、”egg” の個数を数え、それを返すと云う機能を持つ。例えば、egg egg egg と云う入力があった場合、以下のツリーが生成される。

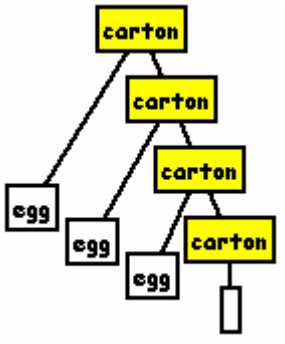

図 11 **文法ツリー**

この文が実行されると、まず最初のルールにマッチしたブロックが実行される。この場合、図11 の右下で空記号が **carton** にマッチし、**[0]**が実行される。これは、単にゼロを返すSmalltalk文である。

次に2行目のルール **carton : egg carton** がマッチする。このブロックで、egg は変数aに、 **carton** は変数 **b** に割り当てられ、ブロックの右辺 **1 + b** が実行される。この場合、bに割り当てられたものは、最初のブロックで実行された答えの0になる。

以下これを繰り返し、最終的に **egg** が現れた文だけ加算され答えの3が返される。

g） デバッガ

コンピュータの動作は、文で説明されるよりも実際に示されるほうがずっと分かりやすい。言語ゲームでは楽しく文法のデバッグを行う為、Moleと呼ばれる簡単なデバッガを用意している。Moleはツリーの非終端記号の上で右クリックメニューで登場させる。

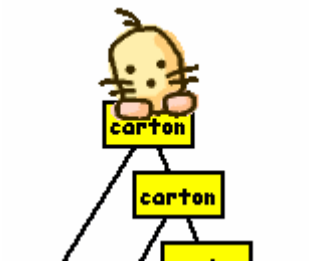

図 12 Mole

Moleをクリックすると、実行すべきシンボルの上に移動し、そのアクションから返される答えを表示する。MoleをSqueakのSqueakToysのメカニズムを使用して制御する事も出来る。

（2）実装

a） タイルインタフェースと文法ツリー

ユーザからの入力にSqueakToysと同じようなタイル形式のインタフェースを使う事にはいくつかの利点がある。ひとつは、ユーザにより親しみを持ってもらう事。もうひとつは、ユーザの操作をある程度適切に制限できる事である。操作の制限は誤操作を減らし、ユーザを問題の本質に集中させる事が出来る。適切な操作の制限を行う為にはまた複雑なタイル間の相互作用が必要である。これらを実現する為に以下の構造が必要であった。

■ 画面上でタイルを表示する為の GTile オブジェクト

GTile自身は文の意味を知らない。他のモルフがGTileにドラッグされると、GTileは下記GNodeにその処理を問い合わせる。GNodeはそのモルフの内容によって、シンボルとして受け入れるか、無視するかを決める。

■ ツリーの内部構造 GNode。

他のパーサジェネレータが文法定義を一括して処理し、ツリーを生成するのに比較して、言語ゲームではユーザの操作に応じて動的にツリーのを変更してゆく。この機能を実現する為にノード自身がパーサへの参照を持つ。ある非終端記号を表すノードAに終端記号Bがドラッグされた場合、次のような操作が行われる。

まず、Aは自身のパーサとBのパーサが同一であるかを比較する。異なっていた場合、Bを自分のパーサで”汚染”させ、Bをスキャンしなおす。Bの文法クラスを受け入れる場合。Aは自身のパーサから適合するルールを選択し、そのルールに従ってBを子ノードにする。もしもそのルールによって他の子ノードが一意に定まる場合、自動的に適合する子ノードを生成し接続する(入力補完として働く)。

b） アクションとデバッガ

入力ペインやツリーのメニューで「fire」が選択された場合、言語ゲームはそのシーケンスを実行する。単純な四則計算器を作成する場合なら、これは単に与えられたブロックを入力に従って実行するだけで実現できる。しかし、ループなどの実行制御やデバッガ等の機能を考慮した場合、遅延実行をサポートする為のシンプルなラッパーが必要になる。

遅延実行の複雑さをユーザから隠蔽する為に、言語ゲームの遅延実行オブジェクトGMonadは「求められてから実行する」モデルを採用している。例えば、[:a :b ¦ b + 1]と云うブロックが与えられた時、実際にaやbに束縛されるのはシンボルの値ではなく、シンボルをラップしたGMonadオブジェクトである。計算は即時実行されずブロックの形のままGMonadにラップされて次のルールの変数に束縛される。実行が完了し、画面に表示する為に asStringメソッドが呼ばれた瞬間、GMonadは文字列として振る舞い始める。多くの場合は自分の持つ値やブロックを評価し、文字列として結果を返す。例外としてvalueメソッドが呼ばれた時のみブロックをそのまま返す。これによって制御構造が実現できる。

GMonadが自身を評価する際、毎回デバッグ用の例外が発生する。この例外は通常無視されるが、Moleによってデバッグ実行される際。Moleはこの例外を受け取り、処理を中断させ実行中の変数等のコンテキストをユーザに知らせる。この機能とE-Toysの持つステップ実行機能を組み合わせて、音楽の自動演奏等の面白い使い方をする事が出来る。

c） 外部オブジェクト

言語ゲームはGGame クラス変数として外部オブジェクトへの参照を保持している。下記の2種類がある。

■ VariableTable

ブロック内で使われる変数を表す辞書。テキストモードでペインに任意のモルフをドラッグすると、モルフはVariableTableに登録される。これにより任意のモルフをブロック内で操作できる。

■ SymbolCostume

終端記号とモルフを結びつける辞書。一度登録されたモルフは、すべての言語ゲームウインドウから同じように使う事が出来る。

４．今後の課題、展望

現在の実装には冗長、不安定な要素も多い。今後はさらに目的を絞り、使いやすい実装を目指したいと思う。現時点での技術的な改善点及び想定している具体的な応用について述べる。

（１）技術的な改善

a） ドラッグアンドドロップ機能の強化

タイルモードにおいて、現在ドラッグアンドドロップによるシンボル操作の機能は弱く、シンボルの追加と削除は可能だが、挿入が出来ない。挿入を可能にする為には文法の中でリスト構造を表すパターン

list :

list : first list

を検出し挿入ポイントを表示する必要がある。

b） リアルタイム文法解析の完成

言語ゲームの特徴であるリアルタイムな文法解析についてはまだまだ未完成である。特に、通常のパーサジェネレータが完成した文法ツリーしか扱わないのと異なり、言語ゲームは作成途中の未完成な文法ツリーを保持しなくてはならない為、動作が不安定な部分が多い。この部分を完成しなくてはならない。また、現在LL文法しか扱う事が出来ないが、実用的にはもう少し広いクラスの文法を扱える必要がある。T-Gen に依存しないシステムが必要になるだろう。

c） 拡張 BNF 文法のサポート

言語ゲームがサポートするBNF文法は再帰を利用する為、特に任意長リストを表す際に、一般の人々に分かりづらい記法となっている。正規表現的な拡張BNFを採用する事により、もっと分かりやすい文法の表記が可能になる。

（２）教育的な応用

数学や化学などの分野で再帰的な構造を扱う場合に、言語ゲームは直感的なモデルを提供する。ある概念の複雑さの原因が、概念そのものから来るものではなく、その表記法による場合、言語ゲームによってより単純な表記法を設計する事で無用な混乱を避ける事が出来る。

この方面での先行事例としては、LispやHaskellなどの関数型言語群がある。それらの言語はより数学的な記法に近く、記号処理に適している[5]。言語ゲームもまた、それ自体としては処理の順番に依存せず、状態を持たないと云う点で関数型言語に近い側面を持ち、関数型言語の記述性にSqueakの持つ親しみやすさを持ち込む環境だとも言える。また、言語ゲームのフレームワークを利用して関数型言語の実行環境自身をSmalltalkで実装し、これらの言語の持つ論理的な特徴とSqueakのマルチメディア性を融合させる事も出来る。

（３）実用的な応用

言語ゲームの持つ動的な構文ツリー生成機能は興味深い応用が考えられる。例えば、近年の統合開発環境においてはユーザの入力するコードを検知し、適切な入力補完やヘルプの提供が必須となっている。言語ゲームのツリー生成系を発展させ、入力支援機能の包括的なフレームワークを開発する事が出来る。また、これらは開発環境のみならず、一般的なアプリケーションにおける入力支援に応用する事が出来るだろう。

一連のテキストから複雑なオブジェクトを生成する考え方は、アプリケーションの生産性と云う観点から見ても非常に有効である。例えば近年のWEBアプリケーションでは、画面生成にHTML言語を介在させる事によりメンテナンス性やポータビリティを向上させている。最近ではXMLによる抽象的な意味データの使用の方向に向かっているが、さらにアプリケーションに特化した抽象度の高い言語を使う事により、ビジネスルールからのインタフェースの自動生成も可能になる。

また、システム開発の中で、実はコーディングそれ自体よりもルールの客観的な記述と云った点により多くのコストが発生する。言語ゲームのもうひとつの可能性として、このようなルールの記述を支援するようなシステムの開発も考えられる。

５．成果物

現在の成果物はこちらからダウンロードできる。
http://languagegame.org:8080/ggame/9

６．参加企業及び機関

株式会社クリプトワンソフト

７．参考文献

[1] Squeak. http://www.squeak.org/
[2] T-Gen. http://minnow.cc.gatech.edu/squeak/1419
[3] 青木峰郎: Ruby を 256 倍使うための本無道編(2001)
[4] A.V.エイホ R.セシイ J.D.ウルマン著 原田賢一訳:コンパイラ I p30 (1990)
[5] 山下伸夫 Programming in Haskell. http://www.sampou.org/haskell/