Hackシステムにおけるコンパイラの教育を目的とした教育支援システムの改良とその実現

Kubota, Yuuki / 久保田, 祐貴

(出版者 / Publisher)
法政大学大学院理工学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学・工学研究科編
(巻 / Volume)
60
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2019-03-31
(URL)
https://doi.org/10.15002/00022046

Hack システムにおけるコンパイラの教育を目的とした教育支援システムの改良とその実現

IMPROVEMENT OF THE HACK SYSTEM FOR EDUCATION AND UNDERSTANDING OF THE COMPILERS AND ITS IMPLEMENTATION

久保田祐貴
Yuuki Kubota
指導教員　和田幸一

法政大学大学院理工学研究科応用情報工学専攻修士課程

In this thesis, we improve the Hack system[1], which has been used as an educational support system for several concepts of computer science. In the improvement, we focus on the education and understanding for compilers. The extended system includes visualization of basic concepts and behavior on compilers, and the language extension. This system provides more efficient compiler learning environment than the original Hack system.

Key Words *: Compiler, Educational System, Hack*

1.　はじめに

情報系の学科において，ハードウェア，アーキテクチャ，オペレーティングシステム，プログラミング，コンパイラ，データ構造，アルゴリズムやエンジニアリングなどはカリキュラムとして組み込まれている．これらは計算機科学におけるコンピュータシステムに現れる諸概念であり，コンピュータ技術が発達して複雑化するにつれて，各々を深く理解することは難しくなっている．コンピュータシステムを理解するには，これら密接する諸概念の理解が求められる．しかしながら，諸概念は互いに関連しており，コンピュータシステム全体を理解することは難しい．

そこで，計算機科学の教育を目的として The Elements of Computing Systems[1]が開発された．その目的をはじめとした多方面で利用されており，その教育支援ツールとしての完成度と評価は高い．ハードウェアとソフトウェアを構築することを通して，コンピュータサイエンスにおける理論をや応用技術を学ぶことができるものである．まず NAND 回路を用いて論理回路，加算器，ALU と CPU の設計によってハードウェアを構築し，さらにアセンブラとコンパイラ，オペレーティングシステムの設計によりソフトウェアを構築する．これらを 1 から作り上げる作業により，計算機科学に現れる諸概念を理解することができるのである．[1]は情報工学やその他の工学部系の学部生と大学院生を対象としている．筆者の研究室でも，配属された情報系の学部生を対象として講義内でも利用している．しかしながら，半期の講義内の限られた時間の中で全部のテーマを学び終えることは難しいのが現状である．多くがコンパイラのテーマの途中で終わってしまう．これは学部生の多くが各テーマの初学者であるがゆえに，特に理解するための難易度が高いテーマに多く時間を費やしてしまうことが理由としてあげられる．現状のシステムに加えて効果的な教育支援システムを活用することで，初学者でも難易度の高いテーマを効率よく学習できる環境を提供する必要がある．

現状のシステムの理解の難易度が高いテーマは，回路の動作[3]，アセンブラの動作[2][8]，VM の動作[4]，コンパイラ[6][7]の動作である．
本研究では[1]のコンパイラのテーマの教育支援を行う．本稿では，これをコンパイラの教育設計支援としてシステムを提案して実装を行う．

提案するシステムは，コンパイラのフロントエンドを構成する基礎概念の理解，一般的なコンパイラに倣った解析，パーサジェネレータを用いたコンパイラの生成の 3 つの機能を提供する．また，Jack の言語仕様を拡張する機能と Jack プログラム中でマクロを定義できる機能を提供する．

2.　Hack コンピュータシステム

[1]のコンピュータシステムを Hack と呼ぶ．Hack コン

ピュータシステムはハードウェアとソフトウェアの２つの階層から構築される． ハードウェアとソフトウェアで取り扱うテーマは， 初学者でも簡単にコンピュータサイエンスを理解できる必要最小限に限られている． 以下にHack コンピュータシステムで取り扱うテーマとその詳細を示す．

●ハードウェアで取り扱うテーマ

ブール演算，論理回路，コンピュータアーキテクチャ

●ソフトウェアで取り扱うテーマ

機械語，アセンブリ言語，アセンブラ，バーチャルマシン(VM)，コンパイラ，高水準言語，オペレーティングシステム(OS)，データ構造とアルゴリズム，ソフトウェアエンジニアリング

●論理回路

論理回路は組み合わせ回路と順序回路から構成される．組み合わせ回路はNAND ゲート，順序回路はD 型フリップフロップ(DFF)を最小単位として構成される．

●コンピュータアーキテクチャ

16 ビットのノイマン型アーキテクチャをベースとしている．論理回路から構築された２つのメモリモジュール(命令メモリとデータメモリ)，中央処理装置(CPU)を搭載する．CPU は演算装置(ALU)とレジスタ，制御装置を含む．さらに，アドレッシングモードとメモリマップドI/O の機能を提供する．

●回路の定義方法

回路の構築はハードウェア記述言語(HDL)を用いて定義する．これにより定義した回路をシミュレーションしてテストすることができる．

●アセンブラ

VM で出力されたアセンブリ言語をコンピュータアーキテクチャ上で実行可能な機械語に変換している．

●高水準言語

高水準言語は Jack と呼ばれ，オブジェクト指向ベースの言語である．Jack の構文は Java や C＃と似ているが，基本的な性質を備えたより単純化された言語である．単純ではあるが，テトリスなどのインタラクティブなゲームを作成できるなど，一般用途に用いることもできる．

●VM

抽象化された仮想的なコンピュータである．コンパイラによって出力された VM コードを VM アーキテクチャ上で実行する．実際には VM コードがコンピュータアーキテクチャ上で実行可能なアセンブリ言語に変換されている．

●コンパイラ

コンパイラはフロントエンドとバックエンドの２段階変換による方法を採用する．フロントエンドで高水準言語から中間コードへ変換される．そして，バックエンドでVM が中間コードを実行することで機械語へと変換される．

●OS

OS は一般的な OS とは異なり，Jack の標準ライブラリとしての位置づけである．ハードウェアに特化したサービスはカプセル化して，ソフトウェアから使いやすいサービスを提供する．また Jack を様々な抽象データ型で拡張することを目的とする．

●提供されるツール群 ：

ハードウェアシミュレータ ― HDL で記述された回路をシミュレートしてテストを行う．

CPU エミュレータ ― 機械語で記述されたプログラムをシミュレートしてテストを行う．

アセンブラ ― アセンブリ言語を機械語に変換する．

VM エミュレータ ― VM 言語で書かれたプログラムをシミュレートしてテストを行う．

Jack コンパイラ ― Jack プログラムを VM プログラムに変換する．

3. 拡張システムの設計

本システムでは３つのモードを提供する．１つ目はコンパイラの構造を学習するモードである．このモードは，字句解析，構文解析，意味解析，コード生成の４つの基礎概念を理解するモードである．ここでは忠実な実行の流れではなく，各概念の動作を知るために抽象的な可視化を行う．また，Jack の構文の拡張機能にも対応しており，構文の拡張なし，マクロによる構文の拡張，拡張 BNF の拡張の３つの場合のコンパイラの動作の違いを可視化する．２つ目はコンパイラの動作の学習モードである．このモードは抽象的な可視化ではなく，コンパイラの動作を忠実に再現して実行するモードである．構文解析では再帰下降構文解析と非再帰な予測構文解析を用いた解析方法で実行できる．このモードも同様に，Jack の構文の拡張に伴った可視化を行う．３つ目はコンパイラの生成モードである．このモードはパーサジェネレータを用いたコンパイラの生成を行うことができる．一般的にコンパイラの開発をゼロから行うことは少なく，既存のパーサジェネレータを使う場合が多い．パーサジェネレータを利用することで，初学者でも簡単にコンパイラを生成することができる．まず，字句解析器と構文解析器が自動生成され，その解析器に追加する形で意味解析器とコード生成器の生成の補助を行う．これにより，コンパイラの開発手法と各解析器に対する理解は十分に得られる．パーサジェネレータは LL 文法を扱うことができて機能が豊富である ANTLR4[5]を採用する．

4. システムの詳細

前章で述べた設計方針に従って提案したシステムの詳細を示す．

（１）Jack コンパイラの構造を学習するモード

字句解析， 構文解析と意味解析， コード生成の処理の可視化機能をサポートする．

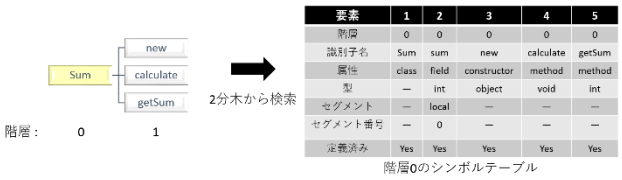
a) 字句解析の処理の可視化

入力した字句要素を定義した拡張 BNF，Jack ソースコード，Jack ソースコードから解析中の１文字、トークンの生

成状況が提示される．Jack ソースコードは定義された字句要素に従い解析が行われ，トークンが生成される．Jack ソースコードが 1 文字ごとに読み込まれ，字句要素の定義を参照しながらトークンが生成される流れを可視化する．ユーザが 1 文字単位，トークン単位で指定して解析を行うことができる．

b) 構文解析と意味解析の処理の可視化

構文解析では Jack の構文を定義した拡張 BNF，字句解析で出力されたトークン群，読み込み中のトークン，構文木が提示される．意味解析ではシンボルテーブル，変数の型の検査状況が提示される．トークン群からトークンを読み込み，文法構造と一致しているかを拡張 BNF を読み進めて調べる．もし，一致していれば構文木に生成規則を追加する．この際に拡張 BNF で定義した意味解析のアクションを認識した場合は，シンボルテーブルの設定，もしくは変数の型のチェックが行われる．シンボルテーブルはスコープごとに独立しており，クラスの階層とシンボルテーブル名から判断する．スコープの階層，識別子名，属性，型，VM のセグメント，VM のセグメント番号，定義済みか否かの要素を設定する．また，テーブルの探索は 2 分木探索であり，通常の 2 分木，もしくは平衡 2 分木である赤黒木から行われる．構文解析が完了したら，構文木は非終端記号などの不要な要素を取り除いた抽象構文木へと変換される．解析は 1 トークン単位で行うことができる．



要素	1	2	3	4	5
階層	0	0	0	0	0
識別子名	Sum	sum	new	calculate	getSum
属性	class	field	constructor	method	method
型	—	int	object	void	int
セグメント	—	local	—	—	—
セグメント番号	—	0	—	—	—
定義済み	Yes	Yes	Yes	Yes	Yes

階層0のシンボルテーブル

図 1　シンボルテーブルと検索の可視化

c) コード生成の処理の可視化

構文解析で出力された抽象構文木，シンボルテーブル，VM コードの生成状況が提示される．抽象構文木に対して深さ優先探索の後順走査を適用して，それに対応する VM 関数を出力する．解析は抽象構文木を 1 ノードずつ後順走査で行うことができる．

(2) Jack コンパイラの動作を学習するモード

Jack コンパイラの構造を学習するモードの字句解析，構文解析，意味解析，コード生成のフェーズを基本として，より一般的なコンパイラに倣った動作を学習するモードである．字句解析から構文解析を一連の操作で解析する．構文解析には再帰下降構文解析と非再帰下降構文解析の 2 種を用いる．

a) LL(1)文法の提示

Jack の構文を定義した拡張 BNF の左辺にある非終端記号ごとに first 集合，follow 集合，director 集合から LL(1)文法であるか判断が可能である．第一段階として，生成規則，first 集合，follow 集合，director 集合の表，集合を求めるアルゴリズムが提示される．各左辺の非終端記号ごとにアルゴリズムを 1 命令ずつ実行して，集合表を求めていく．この操作はプログラムのデバッグ操作を似せている．director 集合が求めれたら第二段階では director 集合から LL(1)構文解析表を求める．生成規則，集合表，LL(1)構文解析表が提示される．左辺の非終端記号ごとに director 集合から次の生成規則を判断して，解析表を構築する．

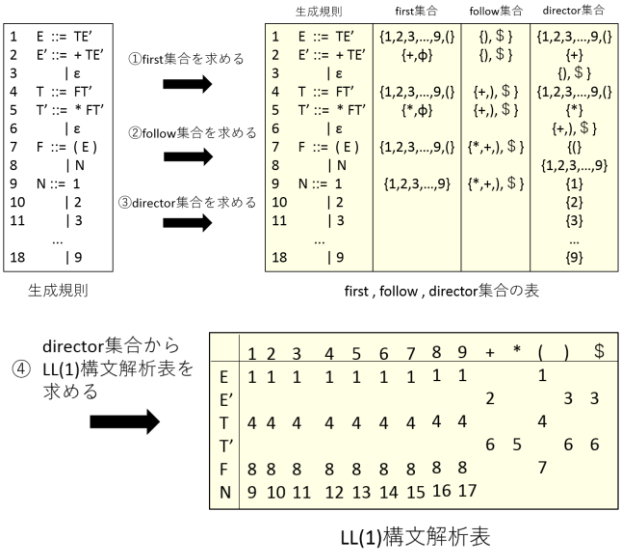


図 2　集合表と LL(1)構文解析表による可視化

b) 再帰下降構文解析による構文解析

再帰下降構文解析は生成規則の左辺の非終端記号ごとに関数を定義，右辺はその関数に対して処理として定義され，その定義された関数を用いて解析を行う方法である．再帰下降構文解析を理解するためには，生成規則を定義した拡張 BNF から非終端記号に対応した関数とその処理が生成される過程と生成された関数を用いた解析の過程を提示することが必要である．前者の過程は拡張 BNF からトップダウン的に終端記号と非終端記号を読み込み，終端記号であればトークンを読み進める処理を設定する．非終端記号の場合は呼び出しと宣言の 2 つのパターンがあり，呼び出しは対応する関数の呼び出しを行い，宣言では関数の定義を行う．後者の過程は字句解析から得られたトークンを，定義した関数をもとに解析される過程を示す．解析方法はプログラムのデバッグ操作を模擬した各処理ごとに関数を読み進めていく．第一段階では生成規則から関数が定義される過程を提示する．生成規則，生成される関数とその処理の生成状況が可視化される．生成規則の左辺の非終端記号ごとに実行され，最初に関数を生成する．関数は返り値は void 型，引数が void 型の非終端記号名で定義する．そして，右辺の生成規則を終端記号と非終端記号単位で読み進めて，非終端記号であれば関数を呼び出す．終端記号であればプログラミングで用いられる if 文の構造に従って比較演算子を用いて

定義する．0 回以上の繰り返し記号は while 文で定義される．もし，複数の生成規則の選択肢がある場合は director 集合と if-else 文を用いて定義される．拡張 BNF に定義した意味解析はアクションとして適切な位置に semantic()の呼び出しコマンドとして定義される． 現在のトークンは Token 型の変数 token に格納される．token は nextToken()が実行されると次のトークンに更新される．

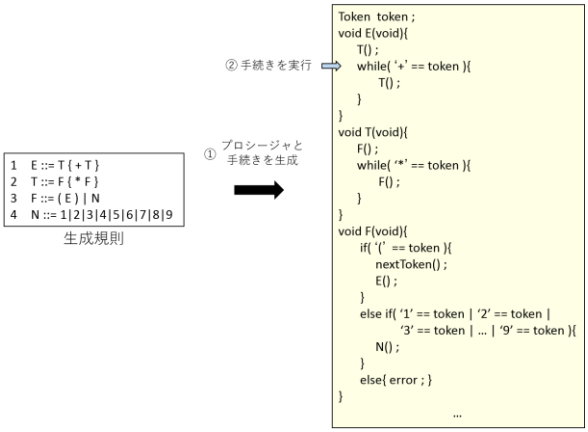


図 3　生成規則から関数とその処理の定義の可視化

第二段階では字句解析から構文解析と意味解析，コード生成までを一連の操作としてその動作を可視化する．構文解析では定義された関数を用いる．字句解析部，定義された関数，意味解析部，構文木，コード生成部が提示される．字句解析では Jack ソースコードから字句要素の定義を参照して 1 トークンを切り出す．1 トークンが生成された段階で関数とその処理による構文解析を行う．プログラムベースの記法となっているため，デバッグ操作で読み進めていく．最初の非終端記号に対応した関数から始めて 1 命令単位で実行する．解析単位は 1 命令(1 行)ごとに解析，最後まで解析，設定したブレークポイントまで解析をサポートする．トークンが生成規則と一致した場合は構文木を構築する．構築方法は Jack コンパイラの構造を学習するモードと同様である．意味解析はシンボルテーブルと型の検査のいずれかの操作であり，これも Jack コンパイラの構造を学習するモードと同様である．構文木が構築されて nextToken()の命令で，字句解析の操作に戻り次のトークンを生成する操作を繰り返す．最後に構文木の構築が完了した段階で抽象構文木へと変換されて，コード生成でこの抽象構文木を走査して VM コードを生成する．字句解析ではソースコードから 1 文字ごとに読み進めてトークンを生成する場合と 1 文字ごとに読み進めずにトークンの出力のみを行う場合の 2 つの場合をサポートする．

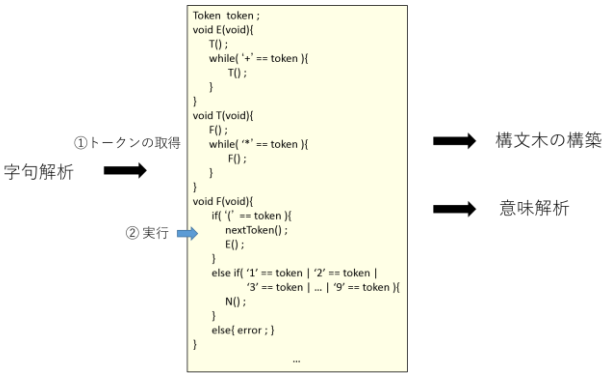


図 4　関数とその処理を用いた解析の可視化

c) 非再帰下降構文解析による構文解析

非再帰下降構文解析は LL 構文解析表とスタックを用いた予測的な構文解析である．対象とする LL(1)構文解析表は前述の LL(1)文法の提示で既に生成されているためそれを用いる．ここでは再帰下降構文解析と同様に字句解析，構文解析と意味解析，コード生成の動作の可視化を一連の操作で行う．生成規則，LL(1)構文解析表，スタックによる解析状況が提示される．スタックによる解析状況は生成規則を置き換えるスタックと入力記号列(1 トークン)，生成規則の導出番号，出力の状況が提示される．スタックの初期状態は最初の非終端記号である．トークンが得られた段階で，スタックの先頭が終端記号でない場合は次の生成規則に置き換える．このとき，複数の生成規則の可能性がある場合は LL(1)構文解析表を参照することで一意に定まる．スタックの先頭に終端記号が出現するまでこの操作を繰り返す．終端記号が出現した場合は入力記号列と比較して正しいならば出力する．同時に構文木の構築を行う．スタックの先頭が意味解析のアクション([semantic])だった場合は，意味解析の操作に移る．スタックと入力記号列が空になるまでこの操作を繰り返す．解析単位はスタックの先頭を置き換える操作ごとに解析，出力が行われるまで解析，最後まで解析をサポートする．

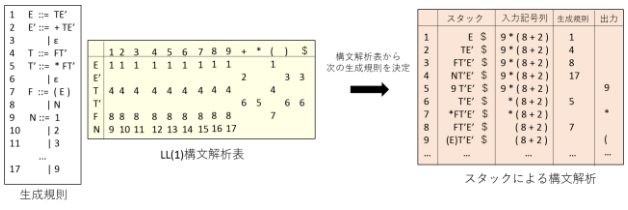


図 5　スタックによる構文解析の可視化

(3) Jack コンパイラを生成するモード

通常，コンパイラを作る場合はゼロから実現することはほとんどない．パーサジェネレータと呼ばれる字句解析と構文解析を行うプログラムを自動生成するプログラムが用いられる．文法の種類には依存するが操作は単純であり，対象とする言語の BNF を入力して出力操作を行

うことでプログラムが自動生成される．Jack コンパイラを生成するモードではパーサジェネレータを用いて，完全な Jack コンパイラを生成するために操作を段階的に導き，コンパイラの作成方法を学習させる目的とする．ANTLR4 を用いて Java 製のコンパイラプログラムを出力する．導く過程は 4 段階から構成されている．第一段階では拡張 BNF を入力として字句解析と構文解析のプログラムを生成する．この段階で Jack プログラムを入力することで構文木が出力される．第 2 段階では意味解析の操作を追加する．拡張 BNF に対して意味解析の実際のプログラムを直接追加していく．しかし，プログラムが膨大になほど全体の見栄えが悪くなり作業効率が下がってしまうため，プログラムを挿入する位置を指定して別の位置に間接的に記述できる環境を提供する．最終的には直接的に挿入された形で ANTLR4 に入力される．これで意味解析の追加が完了する．第 3 段階ではコード生成のプログラムを追加する．意味解析と同様にプログラムを挿入する位置を指定して間接的追加していく．ここまでの操作で Jack コンパイラが完成する．第 4 段階はマクロによる Jack 言語の拡張を行う．これはマクロの解析の部分を Jack コンパイラと同様に字句解析と構文解析，コード生成(プログラムの置き換え)として導く．詳細は後述する．

(4)拡張 BNF とその構文図の可視化

Jack コンパイラの構造を学習するモードと Jack コンパイラの動作を学習するモードでは字句解析もしくは構文解析の際に拡張 BNF を参照して解析を行う．このとき，拡張 BNF だけでは構文の分岐や繰り返しの構造を知ることは難しい．そこで，拡張 BNF に対応した拡張 BNF 構文図を表示して，現在どの生成規則を実行しているのかを 1 目でわかるような機能を提供する．

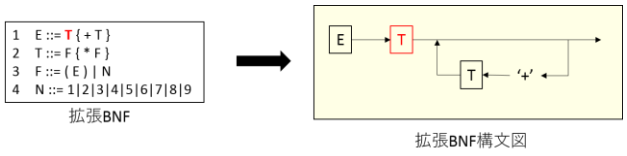


図 6　拡張 BNF とその構文図の可視化

(5) 言語機能の拡張

内部関数の適用，マクロな拡張，拡張 BNF の拡張による Jack 言語の拡張をサポートする．

a) 内部関数の適用

内部関数は多くの言語で用いられている機能である．関数の定義の中でさらに関数を定義することができる．デフォルトの Jack 言語であれば，クラスのスコープと関数ごとのスコープの 2 階層であるが，これを用いることで多階層を実現することができ，シンボルテーブルの動作をより理解することができる．

```
class Sum{
    field int sum;
    constructor Sum new(){
        let sum = 0 ;
        return this ;
    }
    method void calculate(int value){
        let sum = sum + value ;
        printValue(value) ;
        method void printValue(int value){
            Output.printInt(value) ;
        }
    }
    method int getSum(){
        return sum ;
    }
}
```

図 7　内部関数の適用例

この使用例では赤字で記述されている Jack プログラム部分に内部関数が適用されている．内部関数の適用によりシンボルテーブルの階層は深くなる．関数の定義方法は Jack 言語の仕様に準拠する．

b) マクロな Jack 言語の拡張

マクロな Jack 言語の拡張は Jack 言語の構文はそのままで，別途で定義したマクロ構文を Jack ソースコードで使用することができる．このとき，マクロ構文の動作に対応するデフォルトの Jack 構文を定義する必要がある．コンパイラでは字句解析で解析中にマクロ構文を検知した場合に，対応するデフォルトの Jack 構文に置き換えて再度解析を進める．

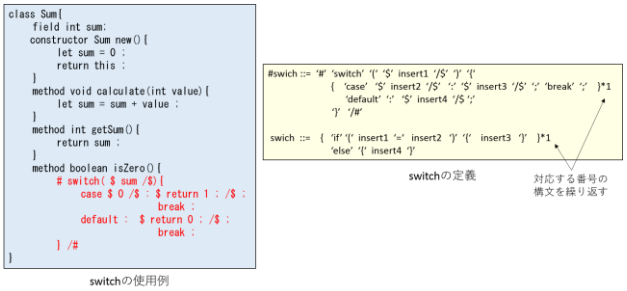


図 8　マクロ構文の使用例

マクロの定義は Jack ソースコードとは別途に専用の定義ファイルに宣言する．このプログラム例では赤字の部分が使用箇所である．マクロの定義方法は拡張 BNF の記法をベースとしている．定義は#に続く形で構文名を指定する．この際，置き換える Jack 構文の構文名はマクロの#を除いた構文名と一致させる必要がある．構文を#と/#で囲むのは処理を簡単にしてマクロの構文であるところを明示するためである．insert は任意の文字列であり，マクロ構文内で対応する箇所のプログラムが置き換え後の Jack 構文の同名の insert 部分に置き換えられる．このとき，insert を用いる場合は&と/&で囲む．また，対応する番号の箇所の繰り返しは 0 回以上の繰り返し(中括弧)，グループ化(丸括弧)の終わりにアスタリスク記号

(*)に続く形で指定する．これは指定された部分が同じ回数だけ置き換えられる．

c) 拡張 BNF の拡張

Jack 言語の構文をベースとして，それに新規の構文を拡張することで拡張 Jack 言語を作る．拡張 BNF を変更することで拡張された新規の言語を作成でき，コンパイラがその構文に伴った構文解析がされることを学習する．しかしながら，Jack コンパイラで新規の Jack 言語を解析するため，最終的には拡張した構文に対応したデフォルトの Jack 言語のプログラムを定義する必要がある．構文木における拡張された構文の部分木は対応した Jack 言語の構文木に置き換えられる．

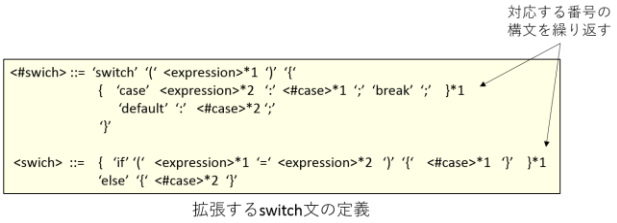


図 9　switch 構文の定義例

左辺の非終端記号名の先頭に#記号が追加されているものが拡張された構文である．#記号がない同名の非終端記号は拡張構文に対応した最終的に置き換えるための構文である．マクロの構文に従って構文木は構築される．この際に非終端記号や 0 回以上の繰り返し(中括弧)，グループ(丸括弧)，省略可能(鍵括弧)に付加されるアスタリスク記号(*)はそのまま構文木に追加される．そして，拡張部の部分木の構築が完了した段階で，別途に置き換える構文に従って構文木を構築する．このとき，アスタリスク記号に対応する部分は拡張構文の部分木がそのまま置き換える構文木の対応する部分木となる．この置き換える操作は拡張構文に対応した VM コードを生成する際に，VM の知識が未熟であってもコンパイラだけで VM コードの生成まで行うことができるようになっている．

5. システムの実現

Jack コンパイラの構造を学習するモードと Jack コンパイラの動作を学習するモードは設計が完了しており，Jack コンパイラを生成するモードはほぼ設計が完了している．システムの開発は Java と Javascript を使用する．以下に示すテクノロジーを利用する．

表 1　テクノロジーとその概要

テクノロジー	概要
Spring Boot	Spring Framework フレームワーク
JavaFX8	GUI ライブラリ
ControlsFX	JavaFX の外部ライブラリ
Ace Editor	Javascript テキストエディタ
Nashorn	Javascript エンジン

6. まとめと今後の課題

本稿では[1]の Jack コンパイラの学習に関して，初学者でもわかりやすく学習できるように，基礎の学習としてコンパイラの構造の学習，基礎を踏まえた次の学習としてコンパイラの動作の学習，パーサジェネレータを用いたコンパイラの生成手法の学習と段階的に導き，効率的に理解させる環境を提供している．今後の課題は Jack コンパイラを生成するモードの設計を完成させると同時に，3 つのモードの未実装な機能を実装して提案したシステムを完成させることである．そして，提案したシステムを教育現場で実際に運用を行い，システムの有用性の検証を行うとともに，得られた課題からさらなる改良を施す．

謝辞:
本研究を進めるに当たり，指導教官として懇切なご指導を戴いた法政大学理工学部応用情報工学科 和田幸一教授に深く感謝の意を表します．また，所属する計算機科学研究室の皆様には，研究の実施に際して多くの協力を戴き感謝申し上げます．

参考文献

1)N. Nisan and S. Schocken, *"The elements of Computing Systems: Building a Modern Computer from First Principles"*, The MIT Press, 2005.
2)岩本和也,"Hack システムにおけるアセンブラの教育とその作成方法を支援するツールについて"，法政大学理工学部応用情報工学科卒業論文，2018.
3)猪狩淳,"Hack 教育支援システムにおける順序回路の可視化について"，法政大学理工学部応用情報工学科卒業論文，2019.
4)大村優斗，"Hack 教育支援システムおける仮想計算機の可視化ツールについて"，法政大学理工学部応用情報工学科卒業論文，2019.
5)Terence Parr, *"The Definitive ANTLR4 Reference*. The Paragmatic Bookshelf", 2012.
6)久保田祐貴，和田幸一,"Hack システムにおけるコンパイラの教育を目的とした教育支援システムの改良とその実現" 2018 年度電子情報通信学会総合大会学生ポスターセッション，2018.
7)久保田祐貴，和田幸一，"Hack システムにおけるコンパイラの教育を目的とした教育支援システムの改良とその実現" 2019 年度電子情報通信学会総合大会，2019.
8)岩本和也，和田幸一，"Hack システムにおけるアセンブラの教育とその作成方法を支援するツールについて"2019 年度電子情報通信学会総合大会学生ポスターセッション,2019.