EXTLex: UM ANALISADOR LÉXICO EXTENSÍVEL CAPAZ DE DETECTAR ERROS LEXICAIS

EXTLex: AN EXTENSIBLE LEXICAL ANALYSER CAPABLE OF DETECTING LEXICAL ERRORS

PAETZOLD, Gustavo Henrique
Doutorando na área Processamento de Linguagem Natural na Universidade de Sheffield, é graduado do Curso de Ciência da Computação da Universidade Estadual do Oeste do Paraná - UNIOESTE Câmpus de Cascavel
ghpaetzold@outlook.com

SCHEMBERGER, Elder Elisandro
Professor Assistente em regime de Dedicação Exclusiva na Universidade Tecnológica Federal do Paraná - Câmpus de Toledo
eschemberger@utfpr.edu.br

Resumo

Analisadores léxicos são componentes fundamentais dos compiladores de linguagens de programação. No desenvolvimento de um compilador, a funcionalidade do analisador léxico, assim como a funcionalidade do restante dos componentes, comumente se limita apenas à linguagem de programação para qual o compilador foi desenvolvido. Com o objetivo de promover reusabilidade no âmbito de desenvolvimento de compiladores, este artigo apresenta um analisador léxico extensível. Por meio de dois arquivos de configuração, o analisador léxico EXTLex é capaz de realizar a análise lexical de múltiplas linguagens distintas. A ferramenta inova por utilizar autômatos finitos na representação de linguagens, e também por fornecer uma plataforma de detecção de erros lexicais. O resultado dos experimentos conduzidos revela que a ferramenta é capaz de enriquecer as informações obtidas na análise lexical sem grande comprometimento em desempenho.

Palavras-chave: Análise Lexical, Compiladores, Linguagens de Programação.

Abstract

Lexical analysers are fundamental components of programming language compilers. When developing a compiler, the functionality of its lexical analyser, such as the functionality of the rest of its components, is commonly restrained to the programming language for which it was designed. In an effort to promote reusability in the context of compilers, this article presents an extensible lexical analyser. By using two configuration files, the featured lexical analyser EXTLex is capable of performing the lexical analysis of multiple distinct languages. The tool innovates by employing finite automata to represent programming languages, and also by providing a platform for the detection of lexical errors. The experiment results reveal that the tool is capable of enriching the data produced during lexical analysis without major compromises in performance. Key-words: Lexical Analysis, Compilers, Programming Languages.

Key-words: Lexical Analysis, Compilers, Programming Languages.

INTRODUÇÃO

Sempre que uma nova linguagem de programação é proposta, se faz necessário a construção de um compilador/interpretador capaz de traduzir códigos representados em seus formalismos em códigos compreensíveis por máquinas. (ALFRED et al., 1995) define a compilação como um processo composto por duas etapas: a análise, que divide o programa fonte nas partes constituintes e cria uma representação intermediária do mesmo, e a síntese, que constrói o programa alvo desejado, a partir da representação intermediária do mesmo.

Em (TOSCANI e ALENCAR, 2008), a análise do programa fonte é descrita como um procedimento que se divide em múltiplas etapas, sendo elas a análise lexical, análise sintática e análise semântica. Estas etapas são realizadas em sequência, e cada uma recebe como entrada os dados produzidos como saída pela etapa anterior.

Na construção de um compilador, devem ser desenvolvidos algoritmos ou módulos que realizem individualmente cada uma das etapas de análise, e que, ao fim, produzam a representação intermediária do código necessária para que o processo de síntese do mesmo seja realizado.

O analisador léxico é um dos componentes que constituem um sistema de compilação, o qual, assim como o nome sugere, é responsável por realizar a etapa de análise lexical do programa fonte. A função do analisador léxico em um compilador, assim como descrito por (ALFRED et al., 1995), é "ler os caracteres do código e produzir uma sequência de tokens para ser posteriormente utilizada na análise sintática".

Os chamados tokens são componentes lexicais de uma linguagem de programação. Estes representam os elementos fundamentais de uma linguagem de programação, como, por exemplo, identificadores de variáveis e funções, números inteiros, números reais, expressões de comparação, etc.

Comumente, o analisador léxico é desenvolvido individualmente para cada nova linguagem de programação, entretanto, atualmente existem ferramentas cuja função é construir um analisador léxico com base na estrutura lexical da nova linguagem.

Na literatura, estas ferramentas são popularmente conhecidas como compiladores de analisadores léxicos ou "Lexers", termo em inglês que faz referência aos analisadores léxicos.

Tais ferramentas requerem que o usuário se familiarize com expressões regulares (AHO e ULLMAN, 1992), e, na maioria dos casos, requer que o usuário instale aplicações complementares, tornando o processo de uso mais oneroso.

No objetivo de sobrepor estas limitações e diminuir os custos no desenvolvimento de analisadores léxicos, este artigo apresenta o EXTLex, um analisador léxico em Java que, por meio de ajustes em seus arquivos de configuração, é capaz de fazer a análise lexical de diferentes

linguagens de programação, bem como uma análise de desempenho do mesmo.

Diferente das ferramentas previamente mencionadas, o EXTLex não tem como função compilar um analisador léxico, mas sim reconhecer os tokens de um código-fonte com base no autômato finito de estados e as palavras reservadas que descrevem a linguagem de programação em que está escrito. O EXTLex também provê uma plataforma de detecção de erros lexicais, funcionalidade esta não suportada pelos compiladores de analisadores léxicos.

EXTLEX E OS COMPILADORES DE ANALISADORES LÉXICOS

Como supracitado, os compiladores de analisadores léxicos são ferramentas que têm como objetivo agilizar o processo de construção de um compilador para uma nova linguagem.

A maioria destas ferramentas requer que o usuário descreva toda a estrutura lexical da linguagem por meio de expressões regulares, cuja forma de representação é, apesar de concisa e poderosa, prolixa, e frequentemente varia entre as ferramentas.

Isto obriga o usuário, antes que possa construir o analisador léxico, se familiarize com a sintaxe das expressões regulares da ferramenta.

O Antlr4 (PARR, 2013) é uma ferramenta de construção de compiladores em Java bastante utilizada atualmente. Esta provê algoritmos ágeis para a construção não apenas de analisadores léxicos, mas também de analisadores sintáticos. Assim como o EXTLex, o Antlr4 não requer qualquer tipo de instalação, entretanto, exige o uso de expressões regulares e não provê qualquer recurso dedicado à personalização do processo de detecção de erros lexicais em códigos fonte.

Existem também várias ferramentas escritas nas linguagens de programação C e C++ capazes de compilar analisadores lexicais, como, por exemplo, o YACC (JOHNSON, 1975), Lex (LESK e SCHMIDT, 1990) e Flex (LESTES, 2008).

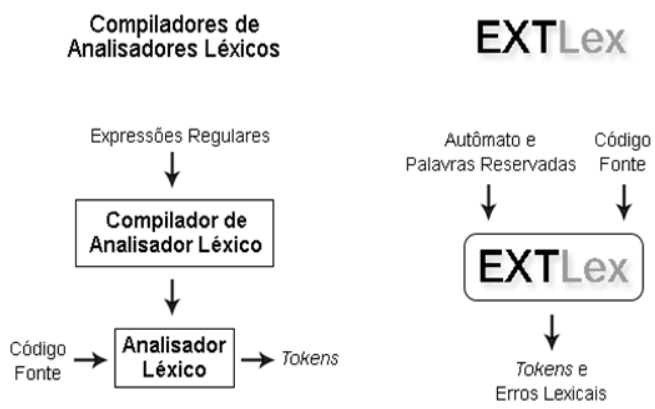
Estas também não fornecem recursos de detecção de erros lexicais, e, além de exigirem o uso de expressões regulares, requerem que o usuário escreva funções em C ou C++ que sejam ativadas toda vez que uma determinada expressão regular é encontrada no código-fonte.

Esta característica das ferramentas exige que o usuário seja familiarizado com a sintaxe destas linguagens de programação, o que pode tornar o processo de uso destas ferramentas ainda menos intuitivo e, consequentemente, custoso.

Deve-se lembrar, entretanto, que os analisadores léxicos gerados por estas ferramentas, por serem escritos em C e C++, são frequentemente muito mais ágeis do que analisadores equivalentes em Java.

A Figura 1 ilustra a diferença entre a estratégia de análise lexical empregada pelo EXTLex e as demais ferramentas mencionadas.

Figura 1 – Comparativo entre EXTLex e Compiladores de Analisadores Léxicos



Nota-se na Figura 1 que o EXTLex recebe, no lugar de expressões regulares, um autômato finito de estados equivalente e uma lista de palavras reservadas, os quais descrevem a estrutura lexical da linguagem de programação sendo analisada.

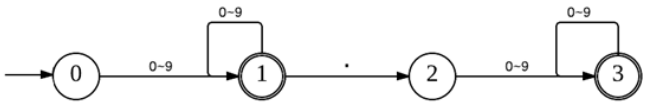
Quando uma estrutura lexical é representada por um autômato, o analisador lexical deve ler cada caractere do código e atualizar o estado atual de análise com base nas transições do autômato. No caso de não existirem mais transições possíveis, o analisador verifica qual é o estado atual de análise, e, de acordo com a especificação do estado, reconhece, ou não, um token bem formado da linguagem.

Autômatos finitos permitem uma abstração descomplicada dos tokens de uma linguagem de programação, e permitem também que o usuário determine quais estados representam tokens bem formados e quais caracterizam cenários de erro.

Considere, por exemplo, uma linguagem que reconhece tokens de números inteiros e reais. Neste exemplo, assume-se que números inteiros são caracterizados por uma sequência de um ou mais algarismos entre 0 e 9, enquanto números reais são constituídos por dois números inteiros separados por um ponto. Figura 2 ilustra um exemplo de autômato finito capaz de reconhecer ambos tokens, e que pode ser representado pelos arquivos de configuração do EXTLex.

No autômato ilustrado na Figura 2, estados finais são caracterizados por contornos duplos e o estado inicial é aquele identificado pelo número 0, à extrema esquerda.

Figura 2 – Autômato de estados finito que reconhece números inteiros e reais



Observa-se que os estados finais de número 1 e 3 representam números inteiros e reais bem formados, respectivamente. O estado de número 2, entretanto, caracteriza um número real incompleto, o qual não possui quaisquer algarismos após o ponto, o que mostra, de forma simples, como o uso de autômatos na análise lexical permite que erros específicos sejam reconhecidos.

A estratégia empregada pelo EXTLex explora tal característica dos autômatos, e permite que o usuário especifique mensagens de erro para cada estado que não representa um token bem formado.

ESPECIFICAÇÃO E ASPECTOS TÉCNICOS

O EXTLex realiza a análise lexical de um código-fonte com base em dois arquivos de configuração: um contendo o autômato finito

de estados que descreve a estrutura lexical da linguagem em questão, e outro que lista suas palavras reservadas. Tais arquivos são nomeados Automato.EXT e Palavras_Reservadas.EXT, respectivamente.

O processo de análise lexical da ferramenta se inicia pela leitura e interpretação do conteúdo dos arquivos de configuração. Em seguida, o EXTLex posiciona a variável de estado atual de análise no estado inicial do autômato da linguagem.

Concluído este procedimento, a ferramenta lê cada caractere do código a ser compilado e então navega pelos estados do autômato finito de acordo com as transições especificadas. Os caracteres lidos são armazenados em um buffer de lexema até que não existam mais transições de estado possíveis.

Quando não existirem mais transições para o estado atual da análise, o EXTLex interpreta o significado do estado atual de análise e realiza as ações adequadas de acordo com a forma como o estado foi especificado no arquivo Automato.EXT.

Ao final da análise, o EXTLex exibe em tela a estrutura de tokens do código lido e também todas as mensagens correspondentes aos erros encontrados no código.

Caso o usuário esteja usando o EXTLex como uma biblioteca em um projeto Java, o mesmo pode ter acesso a todos os dados resultantes da análise por meio das classes responsáveis por armazená-los.

Para que o EXTLex possa analisar códigos-fonte sem acusar erros de execução, ambos arquivos devem ser confeccionados pelo usuário com base na sintaxe especificada pelos autores.

ESTRUTURA DO ARQUIVO AUTOMATO.EXT

Para que o EXTLex seja capaz de reconhecer o autômato descrito no arquivo Automato.EXT, o mesmo deve conter todos os estados que representam a estrutura lexical da linguagem.

Os estados devem ser separados por uma linha em branco e podem, ou não, possuir transições para outros estados. Cada estado deve seguir a estrutura descrita na Tabela 1.

Tabela 1 – Estrutura de estados do arquivo Automato.EXT

Descrição	Entradas do arquivo
Nome do estado.	01: <nome>
Token de retorno do estado <nome>.	02: <*token*_resultante>
Ação lexical referente ao estado <nome>.	03: <ação_lexical>
Transição: número de caractere em ASCII e estado de destino.	04: <ASCII> <nome_estado>
Transição: número de caractere em ASCII e estado de destino.	05: <ASCII> <nome_estado>
Linha vazia indicando final da descrição do estado.	06:

De acordo com a Tabela 1, todos os estados devem conter um nome, o token que será reconhecido caso a análise se encerre no estado em questão, uma chave de ação lexical e suas transições para outros estados.

O estado inicial do autômato é aquele descrito por primeiro no arquivo Automato.EXT. Os campos de nome do estado e token resultante podem ser compostos por uma sequência de caracteres qualquer.

O campo de ação lexical consiste em uma palavra chave a qual leva o EXTLex a tomar uma determinada ação caso não existam mais transições possíveis a partir de um determinado estado.

A Tabela 2 lista quais palavras chave de ação lexical são reconhecidas pelo EXTLex e descreve os procedimentos correspondentes realizados pela ferramenta.

Tabela 2 – Ações lexicais reconhecidas pelo EXTLex

Ação lexical	Rotina correspondente
LEXEME	Salva o *token* do estado atual e o associa à palavra presente no *buffer* de lexema.
ERROR	Salva a mensagem do campo <*token*_resultante> à lista de erros lexicais encontrados.
NEWLINE	Descarta o lexema lido e incrementa o valor da variável de linha atual.
NULL	Descarta ambos lexema e *token* encontrados.

Independente do tipo de ação lexical do estado, após não encontrar mais transições possíveis, o EXTLex designa o estado inicial do autômato à variável de estado atual de análise.

A chave "NEWLINE" deve ser atribuída ao campo de ação lexical de todos os estados do autômato o qual são resultantes, a partir do estado inicial, da leitura de caracteres de quebra de linha.

Deve ser atribuído o valor "NULL" ao campo de ação lexical de estados os quais resultam no reconhecimento de tokens sem significado na linguagem de programação, ou que representem espaços em branco.

Caso o estado não represente um token bem formado, mas sim um estado de erro, o valor do token de retorno deve conter uma mensagem de erro, enquanto o valor de ação lexical deve conter a palavra "ERROR". Deve ser atribuído o valor "LEXEME" à ação lexical de quaisquer estados referentes à tokens bem formados.

Cada transição de estado deve ser constituída por um número decimal da tabela ASCII separado por um espaço do nome do estado de saída resultante.

Para saber o número decimal que representa um determinado caractere, o usuário deve consultar a tabela ASCII. O EXTLex não aceita códigos Unicode de caracteres em sua versão atual.

A Tabela 3, abaixo, apresenta um exemplo de autômato que reconhece comentários cercados por caracteres '#', o qual respeita a sintaxe do arquivo Automato.EXT e é constituído por diferentes tipos de estado.

Tabela 3 – Autômato que respeita a sintaxe do arquivo Auto-

Linha	Entradas do arquivo Automato.EXT
01:	Estado_Inicial
02:	Comentário vazio.
03:	ERROR
04:	35 Segundo_Estado
05:	
06:	Segundo_Estado
07:	Ausência do caractere "#" de fechamento do comentário.
08:	ERROR
09:	32~126 Segundo_Estado
10:	35 Terceiro_Estado
11:	
12:	Terceiro_Estado
13:	<comentario>
14:	LEXEME

No autômato da Tabela 3, é possível observar três estados:

1. **Estado_Inicial:** Estado inicial da construção de um bloco de comentário.

Token de retorno: Mensagem de erro para comentário vazio.

Ação lexical: ERROR

Transições:

• Caractere "#" de código ASCII 35 leva ao Segundo_Estado.

2. **Segundo_Estado:** Estado intermediário na construção de um comentário.

Token de retorno: Mensagem de erro para comentário

incompleto.

Ação lexical: ERROR

Transições:

• Caracteres de códigos ASCII de 32 a 126 levam ao Segundo_Estado.

• Caractere "#" de código ASCII 35 leva ao Terceiro_Estado.

3. **Terceiro_Estado:** Estado final na construção de um bloco de comentário.

Token de retorno: <comentario>

Ação lexical: LEXEME

O estado de nome "Segundo_Estado" faz uso de mais um recurso da sintaxe do arquivo Automato.EXT, que é a representação de intervalos de caracteres para transições.

A expressão "32-126 Segundo_Estado" significa que, caso qualquer caractere cujo código faz parte do intervalo [32, 126] da tabela ASCII seja encontrado durante a análise, o estado atual de leitura será atualizado para "Segundo_Estado".

Este recurso previne que o usuário tenha que criar uma transição individual para cada caractere do intervalo, e, consequentemente, diminui o tamanho do arquivo Automato.EXT.

ESTRUTURA DO ARQUIVO PALAVRAS_RESERVADAS.EXT

A estrutura que deve ser seguida para a confecção do arquivo Palavras_Reservadas.EXT é minimalista. O arquivo deve conter todas as palavras reservadas da linguagem de programação, uma por linha. Uma linha em branco ao final do arquivo representa o fim da lista de palavras reservadas.

A Tabela 4 demonstra um exemplo de arquivo estruturado da maneira adequada.

Tabela 4 – Estrutura do arquivo Palavras_Reservadas.EXT

Linha	Descrição	Entradas do arquivo
01:	Primeira palavra reservada.	for
02:	Segunda palavra reservada.	while
03:	Terceira palavra reservada.	int
04:	Linha vazia indicando o final do arquivo.	

Toda vez que um token bem formado é encontrado, o EXTLex busca pela palavra armazenada no buffer de lexema na lista de palavras reservadas. Caso o lexema seja uma palavra reservada, o valor do token resultante é substituído pela mesma.

EXPERIMENTOS E RESULTADOS

Dada a escassez de conjuntos de teste gratuitos para análise lexical de linguagens de programação populares, foi elaborada a linguagem de programação EXTLanguage. A EXTLanguage se adequa ao paradigma de programação procedural e reconhece alguns dos principais comandos básicos comumente reconhecidos por linguagens do mesmo paradigma, como, por exemplo, laços de repetição, cadeias if e else, expressões lógicas e similares.

Como o processo de análise lexical não requer consultas a quaisquer aspectos sintáticos ou semânticos da linguagem, descreve-se neste interim apenas a estrutura lexical da EXTLanguage.

Tabela 5 – Estrutura dos componentes lexicais da linguagem EXTLanguage

Componente lexical	Expressão regular
<inteiro>	[0-9]+
<real>	[0-9]+.[0-9]+
<caractere>	\'[A-z0-9]\'
<string>	\"[A-z0-9]+\"
<comentário>	#[A-z 0-9]#
<operador>	<(=) | >(=) | [~=]=
<identificador>	[A-z][A-z0-9]*
Caracteres reservados	; | (|) | * | + | - | / | ,
Palavras reservadas	int | float | char | string | if | or | and | not | while | for | write | read | else

Na Tabela 5 é descrita, por meio de expressões regulares, a forma que tomam os tokens, caracteres e palavras reservadas que constituem a estrutura lexical da linguagem.

Foram propostos aqui dois experimentos, cujo objetivo é avaliar como o desempenho da ferramenta EXTLex se compara, em diferentes aspectos, ao de dois dos mais utilizados compiladores de analisadores léxicos gratuitos disponíveis: as ferramentas Antlr4 (PARR, 2013) e Flex (LESTES, 2008).

Enquanto o Antlr4 gera analisadores léxicos escritos em Java, a ferramenta Flex gera analisadores escritos em C. Ambas ferramentas podem ser utilizadas em ambientes Windows e Unix e requerem que o usuário represente a estrutura lexical da linguagem de programação por meio de expressões regulares.

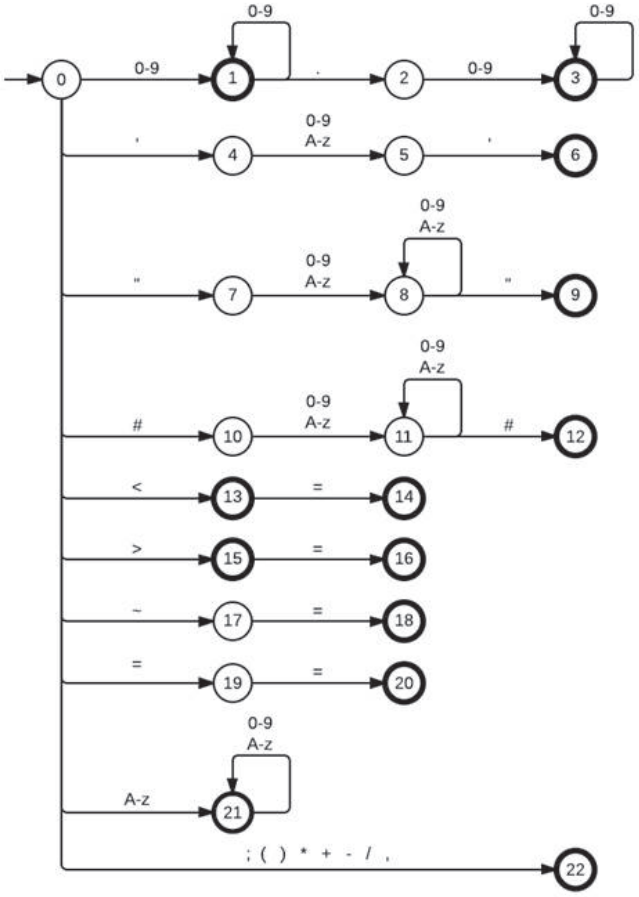
Os experimentos propostos são o de Análise de Desempenho e o de Análise de Eficiência na Detecção de Erros Lexicais. Enquanto este objetiva traçar um comparativo entre a capacidade das ferramentas de reconhecer e reportar diferentes tipos de erros lexicais, aquele objetiva avaliar como a agilidade com que o EXTLex reconhece tokens em códigos-fonte se compara à das demais ferramentas.

ANÁLISE DE DESEMPENHO

Neste experimento, é medida a quantidade de tempo necessário para que o EXTLex realize a análise lexical de códigos-fonte de diferentes tamanhos, e, em seguida, tais valores são comparados com os tempos de execução das ferramentas Antlr4 e Flex.

O conjunto de testes é composto de 200 arquivos de texto contendo códigos-fonte na linguagem EXTLanguage constituídos de 100 a 20000 comandos aleatórios que correspondem a tokens. Cada código-fonte é acompanhado de um arquivo que documenta a lista de tokens a qual os comandos correspondem, permitindo, desta forma, que seja verificada a capacidade das ferramentas de

Figura 3 – Autômato que representa a estrutura lexical da



identificar os tokens da forma correta.

Para cada uma das três ferramentas foram confeccionados os arquivos de configuração necessários para que estas pudessem reconhecer os tokens da linguagem EXTLanguage. Todas as ferramentas provêm os artifícios necessários para que possam reconhecer a EXTLanguage.

O arquivo Palavras_Reservadas.EXT do EXTLex foi confeccionado com base nas palavras reservadas da Tabela 5, enquanto o arquivo Automato.EXT foi construído com base no autômato ilustrado na Figura 3, o qual é equivalente às expressões regulares da Tabela 5.

No autômato da Figura 3, os estados com a borda mais espessa representam estados finais, e, portanto, tokens bem formados. A correspondência entre estados finais da Figura 3 e tokens bem formados da EXTLanguage é descrita na Tabela 6.

Tabela 6 – Estados finais do autômato da EXTLanguage e tokens correspondentes

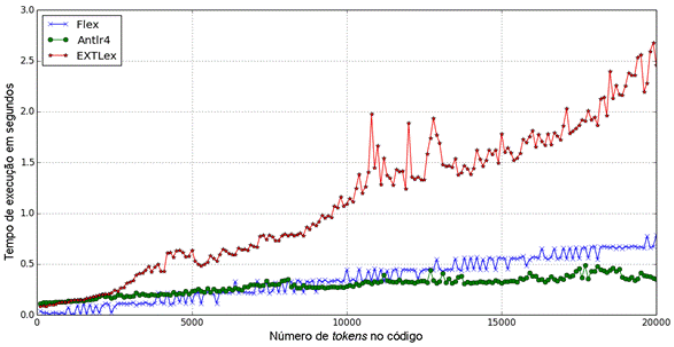
Expressão	Descrição
123.	Número real sem algarismos após o ponto
'A	Caractere sem aspas simples de fechamento
"Palavra	Cadeia de caracteres sem aspas duplas de fechamento
#	Cerquilha separada do restante da expressão de comentário
'	Aspas simples separada do restante da expressão de caractere
"	Aspas duplas separadas do restante da expressão de cadeia de caracteres
~	Expressão de diferenciação incompleta
#Comentario	Comentário sem cerquilha de fechamento
=	Expressão de comparação de igualdade incompleta

Uma vez configuradas, as ferramentas foram então submetidas à análise lexical de cada um dos arquivos do conjunto de teste.

O computador escolhido para o experimento possui um processador Intel i7-4500U com quatro núcleos de 1.80 Gigahertz cada, 8 Gigabytes de memória RAM operando a 1600 Megahertz de frequência, e também um disco rígido de 1 Terabyte com 5400 RPM de velocidade.

O Gráfico da Figura 4 ilustra o tempo de execução em segundos necessário para cada uma das ferramentas analisar os códigos-fonte de teste.

Figura 4 – Gráfico de desempenho das ferramentas EXTLex, Antlr4 e Flex



Ao observar o Gráfico da Figura 4, nota-se que, apesar do tempo de execução de todas as três ferramentas crescer linearmente conforme o número de tokens reconhecidos aumenta, a ferramenta Antlr4 é a que apresenta os menores tempos de execução para os códigos-fonte mais extensos.

É importante ressaltar que as três ferramentas produziram tokens corretos para todos os códigos-fonte do conjunto de teste.

A diferença no tempo de execução do EXTLex com relação às demais ferramentas evidencia que, apesar de permitir a detecção de erros lexicais, o uso de autômatos finitos de estados se mostra uma alternativa mais lenta ao uso de expressões regulares na tarefa de análise lexical.

ANÁLISE DE EFICIÊNCIA NA DETECÇÃO DE ERROS LEXICAIS

Diferente da Análise de Desempenho, este

experimento objetiva avaliar como a capacidade das ferramentas Antlr4 e Flex de relatar erros lexicais se compara aos recursos oferecidos pelo EXTLex.

Para este experimento foi confeccionado um código-fonte na linguagem EXTLanguage contendo, exclusivamente, sete expressões com erros lexicais, como, por exemplo, um número real sem quaisquer algarismos após o ponto.

A Tabela 7 lista, em ordem, todos os expressões com erros contidos no código-fonte de teste.

Tabela 7 – Expressões com erros estruturais presentes no código-fonte de teste

Expressão	Descrição
123.	Número real sem algarismos após o ponto
'A	Caractere sem aspas simples de fechamento
"Palavra	Cadeia de caracteres sem aspas duplas de fechamento
#	Cerquilha separada do restante da expressão de comentário
'	Aspas simples separada do restante da expressão de caractere
"	Aspas duplas separadas do restante da expressão de cadeia de caracteres
~	Expressão de diferenciação incompleta
#Comentario	Comentário sem cerquilha de fechamento
=	Expressão de comparação de igualdade incompleta

No código-fonte de teste, cada uma das expressões da Tabela 7 é separada por um espaço em branco. O código-fonte foi analisado lexicalmente pelas três ferramentas configuradas da mesma forma como feito no experimento de Avaliação de Desempenho.

A Tabela 8 apresenta a saída produzida pelas aplicações Antlr4 e Flex.

Tabela 8 – Saída produzida pelas ferramentas Antlr4 e Flex

Antlr4	Flex
<inteiro> line 1:3 token recognition error at: '.' line 1:5 token recognition error at: "'A' line 1:8 token recognition error at: "'Palavra ' line 1:17 token recognition error at: '# ' line 1:19 token recognition error at: "' ' line 1:21 token recognition error at: "'' ' line 1:23 token recognition error at: '~' line 1:25 token recognition error at: '#Comentario' line 1:37 token recognition error at: '='	<inteiro> . 'A "Palavra # ' " ~ #Comentario =

Avaliando a Tabela 8 nota-se que ambas as ferramentas identificaram, de forma equivocada, um número inteiro na expressão "123" do código-fonte de teste.

Nota-se também que, apesar das ferramentas identificarem todos os demais erros lexicais, nenhuma provê qualquer informação complementar sobre a natureza dos mesmos. A saída produzida pelo EXTLex é ilustrada na Tabela 9.

Tabela 9 – Saída produzida pelo EXTLex

```
Tokens:
Nenhum.

Erros:
Linha 1:25: Expressão de comentário sem cerquilha de fechamento.
Linha 1:3: Número real sem complemento após o ponto.
Linha 1:5: Expressão de caractere sem aspas simples de fechamento.
Linha 1:8: Expressão de cadeia de caracteres sem aspas duplas de fechamento.
Linha 1:17: Cerquilha sem complemento de comentário.
Linha 1:19: Aspas simples sem complemento de caractere.
Linha 1:21: Aspas duplas sem complemento de caractere.
```

Diferente das ferramentas Antlr4 e Flex, o EXTLex não encontrou quaisquer tokens bem formados no código-fonte de teste. Observa-se também que, pela associação de mensagens de erro aos estados não finais do autômato da EXTLanguage, o EXTLex foi capaz de detectar e informar detalhes sobre os erros referentes a cada expressão do código-fonte.

DISCUSSÃO E CONSIDERAÇÕES FINAIS

Este artigo apresentou a ferramenta EXTLex, a qual é capaz de realizar a análise lexical de diferentes linguagens de programação com base nos dois arquivos de configuração Automato.EXT e Palavras_Reservadas.EXT. Por ser desenvolvida

em Java, a ferramenta não requer instalação, e se diferencia dos compiladores de analisadores léxicos por prover uma plataforma de personalização do processo de detecção de erros, bem como por representar a estrutura lexical de linguagens de programação por meio de autômatos no lugar de expressões regulares.

Os resultados dos experimentos realizados neste trabalho revelam que, por meio da categorização de erros, o EXTLex é capaz de complementar a riqueza de informações produzidas durante a análise lexical enquanto mantendo desempenho competitivo em relação a alguns dos mais consolidados compiladores de analisadores léxicos disponíveis. Tais características fazem com que o EXTLex seja uma alternativa viável no desenvolvimento de compiladores em contexto acadêmico.

Como trabalhos futuros, destacam-se o uso de técnicas de otimização de código ao EXTLex, a introdução de códigos de caracteres Unicode aos arquivos de configuração, e também a implementação de ferramentas para análise sintática e semântica seguindo a mesma ideologia de desenvolvimento apresentada neste ínterim.

REFERÊNCIAS

AHO, A.V.; ULLMAN, J.D. Foundations of Computer science. W. H. Freeman, 1992.

ALFRED, V.A.; ULLMAN D.J.; SETHI, R. Compiladores: Princípios, Técnicas e Ferramentas. LTC, Rio de Janeiro, 1995.

JOHNSON, S. C. Yacc: Yet Another Compiler Compiler. Holt, Rinehart, and Winston, New York, NY, USA. 1979.

LESK, M. E.; SCHMIDT, E. Lex: A Lexical Analyser Generator. W. B. Saunders Company, Philadelphia, PA, USA. 1990.

LESTES, W. Flex: The Fast Lexical Analyser. 2008. Disponível em: <http://flex.sourceforge.net/>, acessado em 01/07/2014.

PARR, T. The Definitive ANTLR 4 Reference. Pragmatic Bookshelf, 2013.

TOSCANI, S.S.; ALENCAR, A.M. Implementação de Linguagens de Programação. Artmed. Editora Sagra-Luzzatto, 2008.

Artigo submetido em: 27.08.2014

Artigo aceite para publicação em: 29.12.2014