Um ambiente de desenvolvimento de reconhecedores sintáticos baseado em autômatos adaptativos

Joel Camargo Dias Pereira e João José Neto

Departamento de Sistemas Digitais - Escola Politécnica da Universidade de São Paulo
e-mail: joel@mandic.com.br , jjneto@pcs.usp.br

Endereço para correspondência:
R. Dr. Abelardo Vergueiro César 45, Apto 122 - CEP 04635-080, São Paulo, SP

Resumo

Com o surgimento de novas propostas teóricas na área de construção de reconhecedores de linguagens, adveio também a necessidade de ferramentas que possibilitassem a implementação prática de tais teorias propostas. As ferramentas de auxílio ao desenvolvimento de reconhecedores devem apresentar as mesmas características das ferramentas de programação disponíveis no mercado. A ferramenta RSW pretende proporcionar ao seus usuários um ambiente integrado para a edição, compilação, execução, depuração e otimização de reconhecedores baseados na teoria dos autômatos finitos, autômatos de pilha estruturado e nos autômatos adaptativos. RSW se encontra implementada em sua primeira versão, para uso em computadores compatíveis com IBM-PC.

Abstract

As new theoretical developments occur in the field of construction of language recognizers, corresponding tools became necessary to allow practical implementation of the proposed theories. Such recognizer development tools are required to present characteristics similar to commercially found programming environments. Our tool, called RSW, has been designed to provide its users with an integrated environment for editing, compiling, executing, debugging and optimizing recognizers based on the theory of finite-state-, pushdown- and adaptive-automata. RSW version 1.0 is now available for use on IBM-PC-compatible machines.

1. Introdução

As ferramentas de auxílio à construção de reconhecedores para linguagens de programação exercem um papel fundamental no processo do desenvolvimento teórico desta área, pois tais ferramentas possibilitam comprovar ou não as vantagens ou possibilidades práticas sugeridas pelos resultados teóricos, além de trazer ao mundo real os conceitos formulados na teoria.

Outro ponto importante do uso de ferramentas é o ganho de produtividade que pode ser obtido com esta prática para uma equipe que necessite desenvolver um reconhecedor, em um dado projeto. Neste caso, o principal papel da ferramenta é o de possibilitar que o usuário especifique seu reconhecedor através de uma linguagem simples e produtiva, e utilize o recurso, proporcionado pela ferramenta, de geração automática de código em uma linguagem de programação que tenha sido adotada pela equipe, obtendo-se assim o reconhecedor desejado, sem que a equipe deva para isso submeter-se ao processo de codificação do mesmo.

Uma vantagem adicional da geração automática de código, a partir da especificação de uma linguagem, é a ausência de erros adicionais de programação, freqüentemente introduzidos na atividade de codificação manual. Este fato também contribui para melhorar a produtividade do projeto. Entretanto, esta mesma vantagem carrega em si, gratuitamente, a desvantagem potencial de que o desempenho apresentado pelo reconhecedor gerado seja eventualmente inferior ao de um reconhecedor equivalente, porém desenvolvido manualmente pela equipe. Cada caso deve ser analisado tendo em vista estes fatores positivos e negativos. Na maioria das vezes, a pequena perda de desempenho é fartamente compensada pelo conseqüente ganho na produtividade do projeto.

Este trabalho apresenta uma ferramenta de programação e auxílio ao desenvolvimento de reconhecedores, denominada RSW, que se adequa ao quadro acima descrito, e pretende beneficiar os seus usuários fornecendo-lhes recursos de trabalho compatíveis com as especificadas para as ferramentas mais modernas de sua classe.

Entre as ferramentas tradicionais de desenvolvimento de reconhecedores, talvez uma das mais antigas e conhecidas, e até hoje amplamente utilizadas, seja o par Lex-Yacc. Um dos motivos desta ampla utilização é que tais ferramentas são distribuídas com a maioria dos sistema operacionais UNIX, nas suas diversas variantes, como por exemplo, Flex-Bison.

2. Especificação da Ferramenta

Esta seção dedica-se à análise de alguns pontos considerados fundamentais para uma ferramenta de programação ou ainda de auxílio à construção de reconhecedores. Cabe salientar a razão de incluirmos características de ferramentas de desenvolvimento, ou de programação: quando especifica um reconhecedor, que não deixa de ser um programa, o usuário percorre um processo muito semelhante ao do desenvolvimento de um outro programa aplicativo qualquer, enfrentando, entre outras, as necessidades usualmente manifestas na elaboração de compiladores, tais como as de: *edição de texto*, para preparar um programa-fonte que representa a especificação; verificação de sua *aderência às imposições da linguagem*, através das análises léxica, sintática e semântica do texto pelo compilador reconhecedor da gramática de especificação; da *execução* ou *teste* do reconhecedor gerado; da *depuração* para a localização e eliminação de eventuais erros contidos na especificação; e, finalmente, da realização de operações de *otimização* da especificação, destinadas a melhorar o desempenho do produto.

2.1 Conceitos

O presente trabalho tem como base o conceito de autômatos adaptativos. Nesta seção as principais idéias do formalismo básico dos autômatos adaptativos são apresentadas, de forma muito resumida, com a finalidade única de facilitar a leitura do restante do artigo. Para obter maiores detalhes sobre autômatos adaptativos, o leitor pode consultar [1] e [2]. Sobre meta-programação e assuntos correlatos à geração automática de reconhecedores sintáticos a partir de especificações baseadas em meta-linguagens, um texto introdutório pode ser encontrado em [6]. Boas introduções sobre notações para a especificação formal de linguagens de programação são encontradas em [7] e [8]. Estudos de casos, bem como outras aplicações, são disponíveis em [3], [4] e [5].

Autômatos de pilha estruturados[10] correspondem a coleções de sub-máquinas, que operam como autômatos finitos mutuamente recursivos. A chamada de uma sub-máquina por outra se dá sempre que for executada uma transição de chamada de sub-máquina. Nesta ocasião é empilhada uma indicação do estado para onde o retorno deverá ser feito ao final da operação da sub-máquina chamada. Isto ocorre quando, estando a sub-máquina em um estado final, não houver qualquer outra transição possível, sendo então executada uma transição de retorno à sub-máquina chamadora. Autômatos de pilha estruturados permitem o reconhecimento de linguagens livres de contexto. Linguagens dependentes de contexto podem ser tratadas por meio de uma extensão deste modelo, que constitue os autômatos adaptativos.

Autômatos adaptativos podem ser vistos como autômatos de pilha estruturados que têm a possibilidade de auto-modificação. No início de sua operação, os autômatos adaptativos se apresentam como uma máquina de estados inicial, na forma de um autômato de pilha estruturado. As dependências de contexto da linguagem que estiver sendo reconhecida, que não são tratadas pelo formalismo livre de contexto, podem ser então consideradas através da execução de transições adaptativas. Estas nada mais são que as transições normais de autômatos de pilha estruturados, às quais são acrescentadas indicações sobre as operações de modificação estrutural do autômato, necessárias para que seja tratada a dependência de contexto em questão.

Estas indicações de alteração são efetuadas através de ações adaptativas, que se assemelham a chamadas de procedimentos, em que se determinam as alterações a serem impostas ao autômato. São permitidas três espécies de ações adaptativas elementares: ações de consulta ao conjunto de transições existentes, ações de inclusão de uma nova transição nesse conjunto e ações de eliminação de uma transição desse conjunto. Por meio da combinação adequada dessas três operações básicas, são especificadas as operações de "edição" a serem impostas ao autômato na ocasião em que for executada a transição que especifica tais alterações.

A operação do autômato adaptativo pode ser interpretada como uma evolução sucessiva do autômato de pilha estruturado em que se baseia, promovida pela execução de ações adaptativas: partindo da configuração original, o autômato de pilha que implementa o autômato adaptativo em um dado instante opera normalmente, consumindo sucessivos símbolos da cadeia de entrada, até que seja executada uma transição adaptativa. As alterações impostas por esta evoluem o autômato de pilha estruturado corrente, formando um novo autômato, que irá continuar a tratar o restante da cadeia de entrada até que nova ação adaptativa seja executada, e assim sucessivamente, até que a cadeia de entrada seja totalmente consumida e o autômato atinja um estado final, dizendo-se neste caso que a cadeia de entrada foi aceita pelo autômato. Impasses de qualquer natureza na operação do autômato correspondem à rejeição da cadeia de entrada.

O modelo do autômato adaptativo tem uma característica muito desejável: sendo geral, e sendo também derivado do autômato de pilha estruturado por simples agrregação de recursos, sem alterar o formalismo

deste, torna possível que seja utilizado um único formalismo para a especificação e tratamento de linguagens das diversas categorias, sem que seja necessário usar formalismos diferentes.
Do ponto de vista prático, uma única ferramenta pode ser assim utilizada para o tratamento de linguagens regulares, liveres de contexto ou dependentes de contexto, sem que o seu usuário tenha a necessidade de assimilar outras notações, ou de operar novas ferramentas.
Outra vantagem marcante é que toda a simplicidade e eficiência que caracterizam os autômatos finitos e os autômatos de pilha estruturados podem continuar a ser usufruidos pelos usuários dos autômatos adaptativos, pois a forma de operação destes só se afasta da anterior nas específicas ocasiões, usualmente pouco freqüentes, em que ocorrem as alterações estruturais, permanecendo rigorosamente idêntica nos demais casos.

2.2 Linguagem RSW

A linguagem RSW permite a descrição de um autômato na sua forma mais primária, através de uma notação em que o usuário tem o controle dos estados e de suas respectivas transições, constituindo essencialmente uma forma alternativa para descrever um autômato, seus estado e transições.
Na maioria das ferramentas que auxiliam a construção de analisadores léxicos e sintáticos, a linguagem de descrição deste analisadores representam a gramática desejada. O benefício das linguagens que descrevem gramáticas é a produtividade e a simplicidade do código-fonte. As gramáticas condensam, em geral, em poucas linhas, o equivalente a vários estados e várias transições do autômato equivalente. Por isso é menor o esforço para se descrever uma linguagem através da uma gramática do que através de suas sub-máquinas, estados e transições.
Por outro lado, o controle sobre as transições e os estados do reconhecedor gerado é muito menor nas linguagens descritas através de gramáticas. Este controle muitas vezes beneficia o usuário em relação ao desempenho do reconhecedor. Um exemplo simples desta característica é a possibilidade que a Linguagem RSW dá ao usuário de associar uma ação semântica a alguma transição em vazio que parte de algum de seus estados.
Analogamente ao que acontece entre as linguagens de alto nível e as linguagens de máquina, em relação ao nível em que a programação é feita em cada caso, a quantidade de código escrito é maior em notações baseadas em autômatos que nas equivalentes baseadas em gramáticas. Entretanto, quando há a necessidade de otimizar ao máximo um procedimento, recorre-se à análise e alteração do código de mais baixo nível, pois este permite um controle mais fino da operaçào desejada, permitindo assim buscar o melhor desempenho.
Ambientes de programação baseados em linguagens de terceira geração atualmente possibilitam ao usuário a inserção de trechos de código em linguagem de máquina. Em outras palavras, o programador pode lançar mão do código de máquina em todos os trechos do programa em que houver necessidade de alta velocidade de execução.
A ferramenta RSW foi projetada com a permanente meta de disponibilizar recursos compatíveis com os encontrados nos ambientes de programação mais utilizados atualmente. Sendo assim, está prevista a implementação de uma linguagem RSW para descrição de gramáticas. Esta linguagem utilizaria o suporte hoje existente para a descrição de linguagens através de autômatos, incorporando assim ao sistema a potencial simplicidade e produtividade inerentes às notações gramaticais. Será também permitido ao usuário trabalhar com descrição de autômatos, dentro do mesmo projeto. Com isso, ficam disponiveis ao usuário as vantagens e os benefícios das duas formas de expressão.

2.3 Interface Homem-Máquina

A forma de o homem interagir com a maquina é, hoje em dia, uma das maiores preocupações do mercado de software. Cada vez mais desenvolvem-se maneiras mais amigáveis de o usuário transmitir ao computador suas intenções e desejos. A quantidade de programas dedicados a tentar determinar a intenção do usuário, sugerir melhores formas de executar um mesmo trabalho, ou, ainda, auxiliá-lo no andamento de um processo, vem ficando cada vez maior, e vem se generalizando para todos os tipos de programas, sejam eles aplicativos pessoais ou empresariais, ferramentas de programação ou ferramentas de publicação ou editoração eletrônica.
Um ponto, comum a todos os programas que se propõem a ser amigáveis ao usuário, certamente é o da utilização extensiva de uma boa interface gráfica. Atualmente, a presença de uma interface gráfica é, sem dúvida, uma exigência básica para que uma ferramenta possa ser utilizada com facilidade, e ser rapidamente dominada por seus usuários.
Atendendo a essa exigência, a ferramenta RSW foi desenvolvida para operar no ambiente gráfico Microsoft Windows 95, de 32 bits. Além de facilitar o uso da ferramenta, esta decisão proporciona a

possibilidade da programação gráfica, com animações, e o recurso de concorrência, motivos que fortemente influenciaram na escolha deste ambiente.
A exemplo do clássico par mais conhecido, Lex-Yacc, a maioria das ferramentas desta categoria utilizam uma interface textual como meio de interação com o usuário, embora existam ferramentas que operam em ambientes gráficos, tais como LISA.

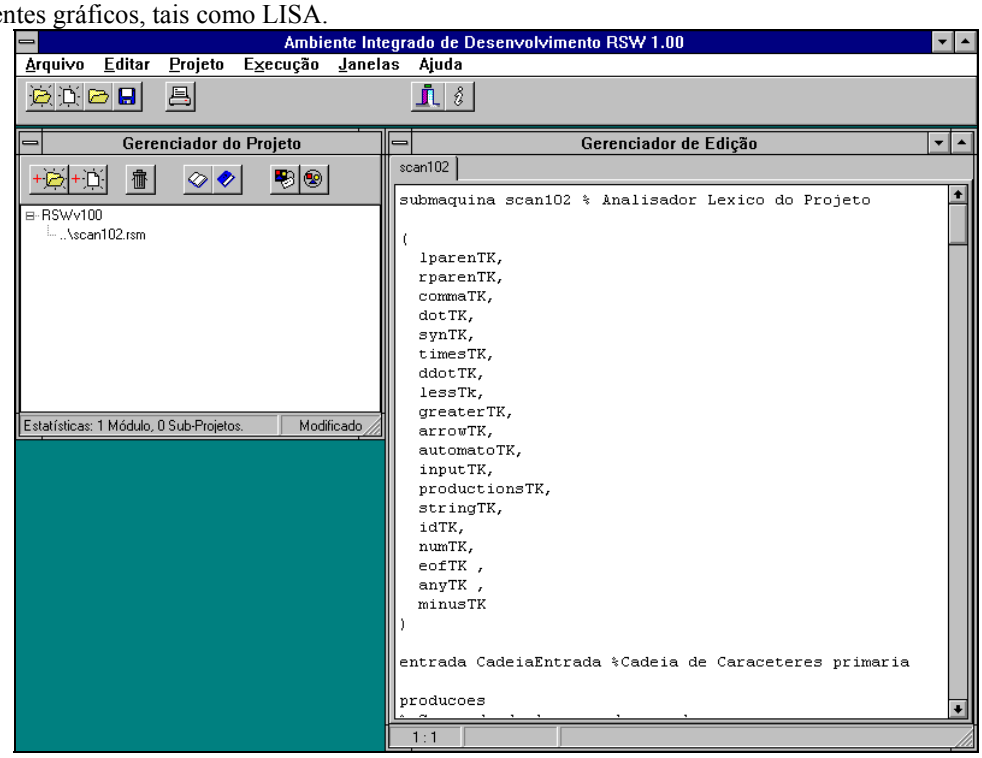

Figura 1 - Aspecto da tela de operação da ferramenta RSW

2.4 Ambiente Integrado de Desenvolvimento

Com o desenvolvimento das ferramentas de programação surgiram os Ambientes Integrados de Desenvolvimento (*IDE, Integrated Development Environment*). Estes ambientes visam proporcionar ao usuário um ponto único, central, no qual se encontram à disposição todos os recursos necessários para o processo de desenvolvimento.
Tais recursos eram encontrados, no passado, espalhados em diferentes programas. Assim, o usuário realizava a edição do código-fonte utilizando-se para isso de um programa especializado, que só realizava esta tarefa. Após ter o código em condições de ser compilado, fazia-o utilizando outra ferramenta, também construída com a única finalidade de compilar programas. O resultado desta operação, o código executável, era executado sob os cuidados do sistema operacional, ou então de uma ferramenta de depuração, que possibilitava ao usuário verificar simultaneamente a execução de um bloco de código e seu respectivo código-fonte, facilitando assim a detecção e eliminação de erros. Depois de o código executável estar estabilizado, isento portanto da grande maioria dos erros, ele poderia ser submetido a uma nova ferramenta, responsável por produzir um diagnóstico de execução, com informações tais como o tempo gasto pelos diversos procedimentos executados. Estas informações eram analisadas, e verificava-se a necessidade de otimizar ou melhorar o desempenho de algumas rotinas, críticas para o desempenho do sistema como um todo.
Resumindo, nos antigos sistemas de desenvolvimento de reconhecedores, diversas ferramentas independentes contribuíam neste processo de desenvolvimento. Por outro lado, os ambientes integrados, de concepção mais moderna, pretendem disponibilizar todos estes recursos de forma centralizada, aumentando assim o conforto do usuário e proporcionando recursos para a agilização de todo o processo de obtenção dos produtos desejados.

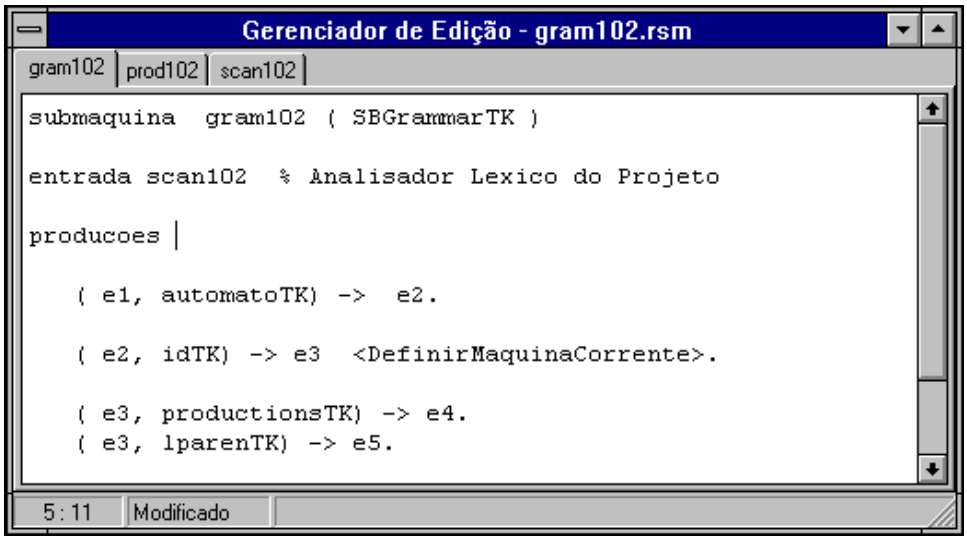

Figura 2 - Aspecto da tela do Gerenciador de Edição

A ferramenta RSW foi projetada de forma tal que incorporasse todas as características acima relacionadas. O gerenciador de edição permite a criação e alteração de arquivos-texto contendo o código-fonte, oferecendo recursos básicos de edição de texto, tais como copiar, recortar, colar e limpar um bloco de caracteres, voltar à execução da última operação, e encontrar ou substituir cadeias de caracteres.

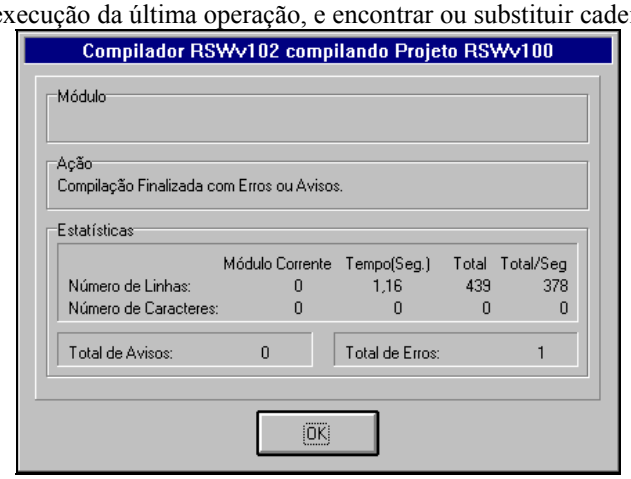

Figura 3 - Janela de estado de uma Compilação

A compilação é realizada, dentro deste ambiente, através da seleção da opção **Compilar Projeto** do menu **Compilar**. Caso o compilador encontre erros no código-fonte, um gerenciador de erros apresenta ao usuário uma listagem dos mesmos, possibilitando-lhe visualizar rapidamente o ponto do código-fonte em que foi detectado o problema.

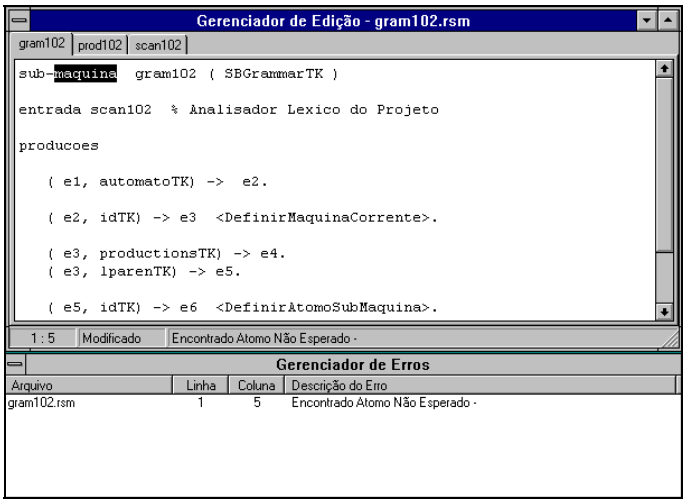

Figura 4 - Visualização de uma mensagem de erro

Continuando o processo de desenvolvimento, o usuário poderá executar o compilador desenvolvido, determinando um arquivo-texto a ser utilizado como cadeia de entrada. Esta execução poderá ser realizada de tal forma que possibilite a depuração do compilador que estiver sendo desenvolvido, ou seja, a cada transição realizada vai sendo visualizado o respectivo código-fonte.
Em relação às atividades ligadas à otimização, a ferramenta RSW oferece uma opção, em tempo de execução, que permite controlar a geração de uma listagem contendo a freqüência de utilização e o tempo gasto em cada sub-máquina do compilador. Através dessas informações, é possível ao usuário localizar e alterar a sub-máquina que se mostrar responsável pela maior parcela do tempo de processamento, constituindo portanto o gargalo do desempenho do reconhecedor. Outro meio de melhorar o desempenho do produto final é através da geração do código-fonte do compilador em linguagens de programação como Pascal. Este código pode ser modificado pelo usuário para que seja obtido o máximo desempenho através da eliminação manual de ineficiências introduzidas intrinsecamente pelo processo automático de geração de código.

3. Arquitetura

A ferramenta RSW foi desenvolvida utilizando-se o produto Borland Delphi 2.0. Apesar de a linguagem de programação deste produto ser uma versão modificada do Object Pascal, é possível a implementação de um sistema orientado a objeto.
A ferramenta RSW foi modelada em objetos de duas classes.
A primeira classe de objetos tem o objetivo de interagir diretamente com o usuário. Ela é responsável pelo funcionamento do ambiente integrado de desenvolvimento e poderá ser futuramente reutilizada.
Um exemplo desta funcionalidade seria a reutilização do objeto responsável pela edição do arquivo texto. Poderíamos adicionar uma serie de novos comportamentos, como, por exemplo, a apresentação do texto em diversas cores, conforme o significado sintático das palavras para o compilador.
Assim as palavras reservadas poderiam ser grafadas em negrito, os identificadores em azul e os comentários em vermelho.
Outra aplicação seria a edição de objetos visuais, em uma linguagem gráfica. No caso da linguagem RSW, cada estado seria representado por um símbolo gráfico do tipo quadrado ou círculo, com suas transições representadas por setas do estado origem ao estado destino.
A descrição completa da biblioteca de objetos relacionados com o ambiente integrado de desenvolvimento não está disponível na versão 1.0 da ferramenta RSW, e será, entre outras, disponibilizada na versão 2.0 .
A segunda classe de objetos é responsável pelas analises léxica e sintática, execução de funções adaptativas, e também pelas chamadas das rotinas semânticas apropriadas.
Dentre as várias classes e objetos disponíveis nesta biblioteca, os de maior destaque são: aTransicao, aTabelaTransicao, aEstado, aSubMaquina e CadeiaEntrada.
A classe aTransicao é responsável pelo armazenamento de todas as informações que caracterizam uma transição, tais como o próximo estado, as funções adaptativas pré e pós, a rotina semântica, o átomo a ser empilhado e a chamada a uma sub-máquina.

O átomo corrente de entrada é manipulado pela classe aTabelaTransicao. Esta classe é responsável pela procura de uma transição correspondente à configuração corrente do autômato. Ela pode decidir dinamicamente qual estrutura de dados adotar, conforme a quantidade de transições que um estado apresenta, no intuito de obter sempre a estrutura de dados mais adequada quanto a desempenho e espaço ocupado.
Assim, caso exista apenas uma transição, o procedimento mais eficiente é comparar o átomo de entrada corrente com o átomo de entrada previsto nessa única transição, determinando assim a aplicabilidade de tal transição. Se o número de transições for pequeno e com átomos de entrada numericamente esparsos, utiliza-se uma lista ligada para enumerá-los na estrutura de armazenamento. Caso o número de transições seja grande ou com átomos de entrada numericamente próximos, utiliza-se para representá-las um vetor em que o índice de acesso corresponde numericamente ao átomo de entrada. Assim, aTabelaTransicao explora a possibilidade de alterar a estrutura de armazenamento das transições em função de um melhor desempenho ou de economia de espaço, em resposta à solicitação do usuário.
A classe aEstado tem a responsabilidade de executar as operações relacionadas com uma determinada transição, tais como o consumo do átomo de entrada, se necessário, o empilhamento de um átomo de saída, a chamada de uma sub-máquina, a chamada de funções adaptativas, a chamada eventual de rotinas semânticas etc.
A classe aSubMaquina é responsável pela continuidade do funcionamento do autômato. Iniciando o reconhecimento da cadeia de entrada pelo estado inicial, aSubMaquina irá continuamente solicitar a informação de qual deverá ser o próximo estado, nele posicionando o autômato, e repetindo, para o novo estado corrente, a operação descrita, até que seja alcançado um estado final, quando então será devolvido o controle à sub-máquina chamadora. Uma condição de erro será apontada caso não seja possível determinar um próximo estado.
Cada instância de aSubMaquina possui um conjunto de instâncias de aEstado, que representam os estados daquela sub-máquina. Cada instância de aEstado possui uma instância de aTabelaTransicao, contendo um conjunto de instâncias de aTransicao. Cada instância de aTransicao representa uma transição válida para aquele estado.
Finalizando, o objeto CadeiaEntrada é a abstração de uma cadeia de entrada contendo o texto a ser reconhecido. Este objeto tem a habilidade de receber átomos para serem empilhados na cadeia de entrada original. Sua função é disponibilizar o átomo de entrada corrente para a sub-máquina que o ativar. Esta, por sua vez, deverá informar se o átomo corrente foi consumido, para que o objeto CadeiaEntrada possa disponibilizar corretamente o novo átomo corrente.
O objeto e as classes acima são descritos por meio de uma referência completa no arquivo de ajuda online do RSW. O funcionamento detalhado deste mecanismo pode ser observado pelo usuário através do recurso de geração de código em Pascal, pois o código gerado utiliza estas classes para implementar reconhecedores descrito na linguagem RSW.

4. Exemplos Ilustrativos

As aplicações mais triviais de uma ferramenta que possibilita a implementação de autômatos finitos, autômatos de pilha estruturados e de autômatos adaptativos, são apresentadas a seguir, com o propósito de ilustrar, através de três exemplos, o aspecto típico das especificações RSW. Nestes exemplos, são descritos, respectivamente, um reconhecedor de números inteiros e identificadores, um reconhecedor de expressões aritméticas simples e um reconhecedor de palíndromes ímpares, da forma $({}^{n} B)^{n}$. Esses exemplos reproduzem, em RSW, os autômatos adaptativos apresentados em [1], onde pode ser encontrada, por extenso, a descrição pormenorizada de seu funcionamento.

4.1 Autômato Finito

Listagem RSW de um reconhecedor de identificadores e números inteiros. Na segunda linha especifica-se que esta submáquina se alimenta diretamente de caracteres ASCII extraídos da cadeia de entrada, um por vez, automaticamente, a cada transição desta submáquina. Os estados estão designados como e1, e2 e e3. O alfabeto de entrada é o conjunto de caracteres alfanuméricos. Os símbolos ATIdentificador e ATNumeroInteiro designam átomos, referentes a Identificador ou Número Inteiro encontrados, e são gerados por esta submáquina para alimentar outras em que for referenciada. O símbolo ^ (acento circunflexo) denota a pilha, e o símbolo * (asterisco) designa o estado indicado no topo da pilha como estado de retorno.

```
submaquina IdentNum ( ATIdentificador,   ATNumeroInteiro )
entrada CadeiaEntrada
```

```
producoes

%Reconhecimento de Identificadores
( e1, ["_abcdefghijklmnopqrstuwvxyz"] ) -> e2.
( e2, ["_abcdefghijklmnopqrstuwvxyz0123456789"] ) -> e2.
( ^*, e2, vazio ) -> ( ^, *, ATIdentificador).

%Reconhecimento de Números Inteiros
( e1, ["0123456789"] ) -> e3.
( e3, ["0123456789"] ) -> e3.
( ^*, e3, vazio ) -> ( ^, *, ATNumeroInteiro).
```

Listagem 1 - Reconhecedor de Identificadores e Números Inteiros

4.2 Autômato de Pilha Estruturado

Listagem do programa de um reconhecedor de expressão aritmética simples. As convenções são as mesmas do exemplo anterior. Na segunda linha da listagem, indica-se que o texto-fonte corresponde às saídas do Analisador Léxico, uma submáquina pouco mais complexa, embora conceitualmente muito semelhante ao extrator de identificadores e inteiros do exemplo anterior, e não incluída neste texto. O Analisador Léxico é chamado automaticamente a cada transição realizada nesta submáquina. O conjunto de estados desta submáquina compreende os estados e1, e2, e3, e4, e seu alfabeto de entrada inclui os átomos ATNumero, ATParentEsq, ATParentDir, ATExpressaoAritmetica e os quatro sinais de operações aritméticas. Os átomos identificados por nomes iniciados por AT correspondem a códigos gerados como saídas de submáquinas do autômato adaptativo, em decorrência de ter sido encontrada uma cadeia válida. Em particular, ATParentDir e ATParentEsq correspondem, respectivamente, aos caracteres Parênteses, direito e esquerdo, respectivamente.

```
submaquina ExpressaoAritmetica (  ATExpressaoAritmetica  )
entrada AnalisadorLexico
producoes

( e1, ATNumero ) -> e2.
( e1, ATParentEsq ) -> e3.
( e2, [ "+-*/" ] ) -> e1.
( ^*, e2, vazio ) -> ( ^, *, ATExpressaoAritmetica ).
( e3, vazio ) -> ( ^ExpressaoAritmetica, e3 ).
( e3, ATExpressaoAritmetica ) -> e4.
( e4, ATParentDir ) -> e2.
```

Listagem 2 - Reconhecedor de expressões aritméticas

4.3 Autômato Adaptativo

Listagem do programa de um reconhecedor da palíndrome impar da forma $(^{n} B)^{n}$. Este autômato está representado na forma de uma submáquina que se alimenta de átomos gerados pelo Analisador Léxico. Seu conjunto de estados inclui estado1, estado2, estado3 e estado4. O alfabeto de entrada compreende os símbolos ATIdentificador, ATParentEsq, ATParentDir. Notar, na segunda produção abaixo, a presença de uma ação adaptativa AdicionarEstados com três argumentos. Esta é uma chamada da função adaptativa de mesmo nome, especificada quatro linhas adiante na listagem, na seção 'funcoes'. Destina-se a alterar o autômato através de quatro adições de transições novas (introduzida pelo sinal +) e da eliminação de uma (introduzida pelo sinal -). Notar a presença dos nomes NovoEstado1 e NovoEstado2. Referem-se a geradores, nomes que assumem valores novos, únicos, a cada vez que a função é chamada, permitindo assim que novos estados sejam criados sempre com nomes inéditos. Observar a sintaxe da notação: as produções entre colchetes correspondem àquelas que irão ser acrescentadas ou retiradas do conjunto de produções corrente do autômato. Sendo parametrizadas, as funções adaptativas podem referenciar, nessas ações elementares, os parâmetros ou os identificadores que simbolizam geradores. Nesses casos, a cada chamada da função adaptativa, novos valores nesses elementos irão instanciar novas produções, de modo que o comportamento da mesma ação adaptativa elementar irá modificar-se em função de tal dinâmica.

```
submaquina  Pilha (  ATPilha  )
entrada AnalisadorLexico
producoes
```

```
(estado1, ATIdentificador ) -> estado4.
(estado1, ATParentEsq ) -> estado2
                           : AdicionarEstados( estado2, estado3, estado1 ) .
(estado2, ATIdentificador ) -> estado3.
(estado3, ATParentDir ) -> estado4.
( ^*, estado4, vazio ) -> ( ^, *, ATPilha).

funcoes
AdicionarEstados ( origem, destino, anterior ) =
{  NovoEstado1, NovoEstado2 :
   + [ (NovoEstado1, ATIdentificador) -> NovoEstado2 ]
   + [ (NovoEstado2, ATParentDir) -> destino ]
   + [ (origem, ATParentEsq) -> NovoEstado1
                        : AdicionarEstados (NovoEstado1,NovoEstado2,origem)]
   - [ (anterior, ATParentEsq) -> origem
                           : AdicionarEstados( origem, destino, anterior) ]
   + [ (anterior, ATParentEsq ) -> origem]
}
```

Listagem 3 - Reconhecedor de uma palíndrome impar

5. Conclusão

A ferramenta RSW contribui significativamente para o estudo e desenvolvimento de linguagens dependentes de contexto, pois cria um ambiente que espelha a teoria dos autômatos adaptativos, fornecendo meios para a sua realização automática no computador.

Toda esta contribuição conceitual vem acompanhada das facilidades e dos recursos comumente utilizados por usuários de linguagens e ferramentas de programação.

A linguagem RSW permite ao usuário um altíssimo controle sobre o comportamento do autômato especificado, controle esse usualmente não proporcionado por ferramentas baseadas em especificações gramaticais das linguagens.

A orientação a objetos, como metodologia de desenvolvimento, possibilitou a ampla reutilização dos módulos construídos e armazenados através da bibliotecas que a ferramenta disponibiliza.

Em sua primeira versão, RSW se encontra plenamente implementada. Diversas melhorias e novos recursos já foram planejados para as futuras versões, dando uma boa idéia da amplidão do campo de aplicação desta ferramenta. Sua utilidade intrínseca fica mais acentuada quando se constata que RSW pode suprir, de forma bastante elegante, a evidente carência de ferramentas modernas similares que estejam disponíveis e que sejam práticas e fáceis de usar. Nesses termos, RSW tem a oportunidade de oferecer à comunidade uma ferramenta moderna e de largo espectro, um recurso poderoso e simples de grande potencial, a despeito da popularidade e da inegável utilidade de algumas ferramentas antigas e muito disseminadas, voltadas no entanto apenas para linguagens regulares e livres de contexto, e de algumas outras, mais recentes, baseadas em gramáticas de atributos, porém disponíveis de forma ainda restrita.

6. Referências

[1]. João José Neto
Contribuições à Metodologia de Construção de Compiladores
Tese de Livre-Docência - Escola Politécnica da USP
São Paulo, 1993

[2]. João José Neto
Adaptive Automata for Context-Dependent Languages
ACM SIGPLAN Notices, Vol. 29, n. 9, September 1994, p. 115-124

[3]. Margarete Keiko Iwai
Um Meta-Editor Dirigido por Sintaxe para Linguagens Estruturadas em Blocos
Dissertação de Mestrado - Escola Politécnica da USP
S. Paulo, 1995

[4]. P. Rechenberg, H. Mössenböck
A Compiler Generator for Microcomputers
Prentice Hall, 1989

[5]. Jean-Paul Tremblay, P. G. Sorenson

The theory and practice of compiler writing (capítulo 14)
McGraw-Hill, 1985

[6]. A.T.Schreiner, H. G. Friedman, Jr.
Introduction to Compiler Construction with UNIX
Prentice Hall, 1985

[7]. Andrew D. McGettrick
The definition of Programming Languages
Cambridge University Press, 1980

[8]. Frank G. Pagan
Formal Specification of Programming Languages - a panoramic primer
Prentice-Hall, 1982

[9]. IEEE
Translator Writing Systems
Special Issue of IEEE Computer - august 1980

[10]. José Neto, J.
Introdução à Compilação
LTC, Rio de Janeiro, 1987