atende a todos os requisitos das modernas técnicas de sinalização semafórica possibilitando a redução, ao mínimo, do tempo de reparo de equipamentos que, eventualmente, estejam fora de serviço. Sua alta flexibilidade operacional permite-lhe, por exemplo, administrar corredores exclusivos para transporte urbano garantindo o desempenho do sistema, tanto em termos de fluidez como em termos de segurança.

O estágio de desenvolvimento corresponde à fase de projeto preliminar do ciclo de vida do software, o que significa dizer que é possível, caso necessário, acrescentar outras funções ao sistema sem causar efeitos colaterais indesejáveis à arquitetura do sistema do software.

REFERÊNCIAS BIBLIOGRÁFICAS

1. CURY, P. **Projeto SEMCO:** Sistema de Controle de Tráfego em Área de São Paulo. São Paulo, CET, 1977. 63p. (Boletim Técnico da CET, 7).

2. PROJETO SEMCO: implantação com base na economia e segurança. **Trânsito**, São Paulo, (2):53-62, dez. 1976.

Data de entrega: dezembro/80

CONSIDERAÇÕES SOBRE O PROJETO DE UM COMPILADOR PARA UMA LINGUAGEM DE PROGRAMAÇÃO DE ALTO NÍVEL

João José Neto, Assistente-Doutor, EPUSP
Marcus Vinícius Midena Ramos, engenheiro, EPUSP

UNITERMOS: Compiladores. SDL. Engenharia de software. Linguagens de alto nível. Projeto de sistemas

RESUMO

Este texto apresenta de forma considerada os principais resultados de um trabalho de construção de um compilador para a linguagem LIA, destinada a serviços como ferramenta no desenvolvimento de software para centrais telefônicas do tipo CPA.

São descritos os aspectos relativos à implementação, metodologia de programação e desenvolvimento e de validação e testes do compilador construído.

ABSTRACT

The present text presents the main results of the design an implementation of the language LIA, created by the author as a tool for the development of software for CPA systems.

Implementation aspects are descrideb, and programming methodology and development, as well as validation and testing aspects of the compiler are considered.

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.

CONSIDERAÇÕES SOBRE O PROJETO DE UM COMPILADOR PARA UMA LINGUAGEM DE PROGRAMAÇÃO DE ALTO-NÍVEL

Autor : Marcus Vinícius Midena Ramos

Depto. de Eng. de Eletricidade da EPUSP

Este texto apresenta de forma condensada os principais resultados do trabalho do autor no projeto e construção de um compilador para uma linguagem de programação de alto-nível. A linguagem fonte denomina-se LIA (Linguagem de Implementação de Autômatos) e foi projetada também pelo autor deste trabalho.

A linguagem LIA ([17]) foi desenhada para servir como ferramenta destinada a aumentar a produtividade de programação no desenvolvimento do software de controle de centrais telefônicas digitais do tipo CPA (Controle por Programa Armazenado) especificadas na linguagem SDL (Specification and Description Language) definida pelo CCITT ([18]). Para alcançar este objetivo, a LIA adota como modelo conceitual o próprio modelo definido pela SDL, o qual se baseia no conceito de Máquinas de Estados Finitos Estendidas e Comunicantes. A linguagem LIA é algorítmica e possui semelhanças com a linguagem de prósito geral Pascal.

Neste texto são descritos os aspectos relativos à implementação, metodologia de programação e de desenvolvimento, e de validação e testes do compilador construído.

O projeto da linguagem bem como o projeto e construção do compilador são temas da Dissertação de Mestrado do autor, a qual se encontra atualmente em fase final de redação. O orientador do autor neste trabalho é o Professor Doutor João José Neto do Departamento de Engenharia de Eletricidade da EPUSP. A implementação do compilador está concluída e o mesmo está sendo utilizado experimentalmente no momento no projeto de uma CPA de pequeno porte.

Este trabalho está organizado sa seguinte forma :

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.

1 Aspectos de Implementação

Este item descreve detalhes da construção do compilador LIA. Inicialmente são consideradas as suas caraterísticas gerais. A seguir ele é estudado sob o ponto de vista da sua arquitetura interna a nível de blocos, do contexto de utilização (arquitetura externa) e por último das técnicas utilizadas na construção dos principais blocos.

1.1 Características Gerais

O compilador construído traduz programas escritos em LIA para programas equivalentes escritos em linguagem intermediária independente de máquina. Um segundo programa, denominado Tradutor, realiza a conversão final do programa intermediário para um programa objeto equivalente executável diretamente em um microprocessador comercial.

A linguagem intermediária utilizada pelo compilador sintetiza a abstração de uma máquina virtual que utiliza uma pilha como principal estrutura de operação. Esta linguagem possui nível suficientemente baixo para permitir a construção de tradutores simples para as várias máquinas reais distintas, assim como possui tambem nível alto o suficiente para ser independente de máquina e ainda assim facilitar a geração de código no compilador e permitir otimizações dependentes da máquina real por parte do programa tradutor.

A compilação é efetuada em um único passo, ao longo do qual são realizadas as tarefas de análise léxica, sintática, semântica e de geração de código. O núcleo do compilador é constituído por um reconhecedor recursivo descendente que implementa uma aproximação da linguagem LIA descrita formalmente por uma gramática livre de contexto do tipo LL (1). As dependências de contexto exibidas pela linguagem são tratadas por rotinas separadas (denominadas ações semânticas), que são ativadas sob o controle do núcleo livre de contexto.

1.2 Arquitetura Interna

Internamente o compilador é constituído por blocos que possuem funções bem definidas e interagem uns com os outros. A figura 1 ilustra estes aspectos.

Esta figura permite a identificação das entradas que são fornecidas ao programa bem como das saídas produzidas pelo mesmo. As entradas são :

- Um texto fonte escrito em LIA, correspondente ao programa à ser traduzido.
- Uma Biblioteca de Sinais, contendo as definições dos sinais utilizados pelo programa fonte.
- Um arquivo auxiliar contendo o texto correspondente a cada um dos códigos de erro que o compilador gera.

As saídas são :

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.

- O programa objeto em linguagem intermediária correspondente a compilação do programa fonte.

- Uma listagem formatada, contendo eventuais mensagens de erro, do programa fonte compilado.

- Uma tabela de símbolos, contendo informações variadas sobre o programa fonte compilado.

Os blocos que compõe o compilador são os seguintes :

- Analisador Léxico (ALe)

- Analisador Sintático (ASi)

- Analisador Semântico (ASe)

- Formatador de Listagens (FLi)

- Gerenciador da Tabela de Símbolos (TSi)

- Gerador de Código (GCo)

Macroscópicamente, a interação entre os blocos pode ser entendida da seguinte forma : o ALe lê o arquivo fonte, caracter a caracter, no disco. Estes são repassados ao FLi, que se encarrega de recompô-los para obter uma listagem formatada do programa fonte. Os átomos gerados pelo ALe são passados para o ASi, o qual informa o FLi, na presença de erros sintáticos (livres de contexto), dos códigos de erro correspodentes (para que este insira os mesmos na listagem formatada na posição adequada). As ações semânticas (ASe) são ativadas pelo ASi na medida em que a sintaxe do programa fonte assim o exigir. Assim como o ASi, o ASe também informa o FLi sobre a presença de outros erros no programa fonte. O GCo pode ser ativado diretamente pelo ASi ou então também pelo ASe (por exemplo, no caso de ser necessária alguma coerção de tipos), e produz código objeto armazenado diretamente no disco do computador hospedeiro. A inserção de identificadores (e de seus atributos) na tabela de símbolos (TSi) é realizada pelo ASi, que invoca no bloco correspondente a rotina de inserção de nomes. O TSi atende ao ASe e ao GCo, fazendo a pesquisa dos nomes solicitados por estes e devolvendo como resultado os atributos dos mesmos. O TSi obtém diretamente do disco o conteúdo da Biblioteca de Sinais e o insere na tabela de símbolos. Por último, o FLi e o TSi devolvem ao disco respectivamente a listagem e a tabela de símbolos do programa fonte.

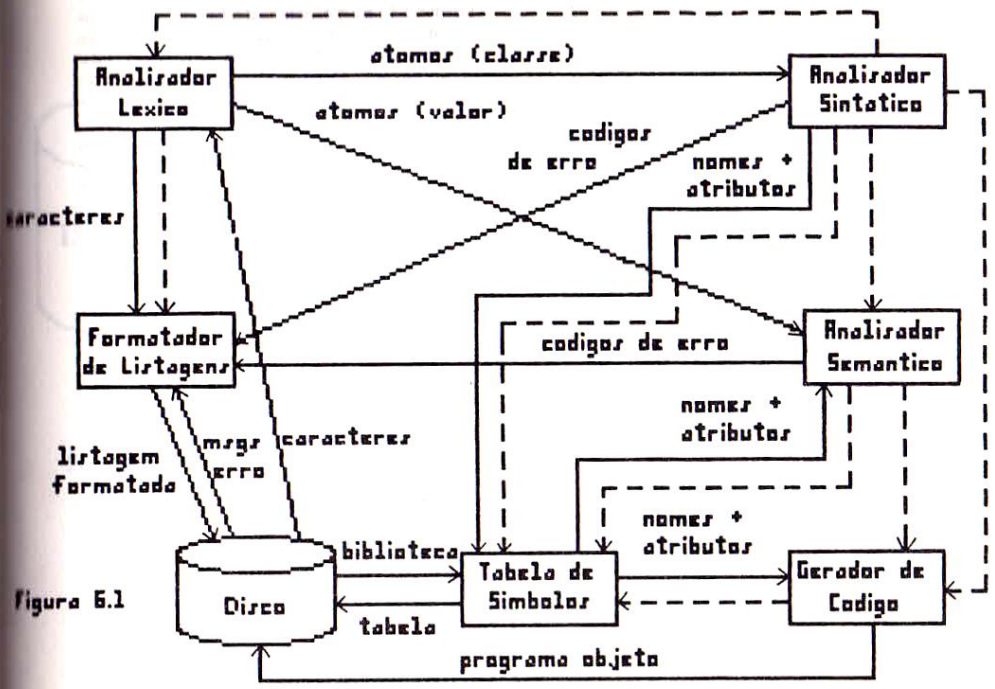

Figura 6.1

1.3 Arquitetura Externa

O compilador funciona em conjunto com outros programas que complementam a sua função. Eles são :

- Tradutor : encarregado de converter programas em linguagem intermediária gerados pelo compilador para a linguagem de montagem da máquina alvo a que ele se destina. Existem tantos tradutores quantas sejam estas máquinas alvo.

- Gerador de mnemônicos : gera uma listagem do programa intermediário usando mnemônicos. Notar que estes não são gerados pelo compilador para evitar a fase de analise léxica no tradutor.

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.

- Gerador de tabela de símbolos : produz uma tabela dos identificadores definidos num programa em conjunto com os seus atributos, a partir de arquivos temporários gerados pelo compilador.

A escolha pela implementação destas funções usando-se programas separados é justificada essencialmente pelo desejo de reduzir o tamanho do compilador obtendo-se a consequente redução na sua complexidade final.

A figura 2 ilustra o aspecto de independência entre os programas, ao passo que a figura 3 mostra como estes podem (e devem) se relacionar quando utilizados de forma correta.

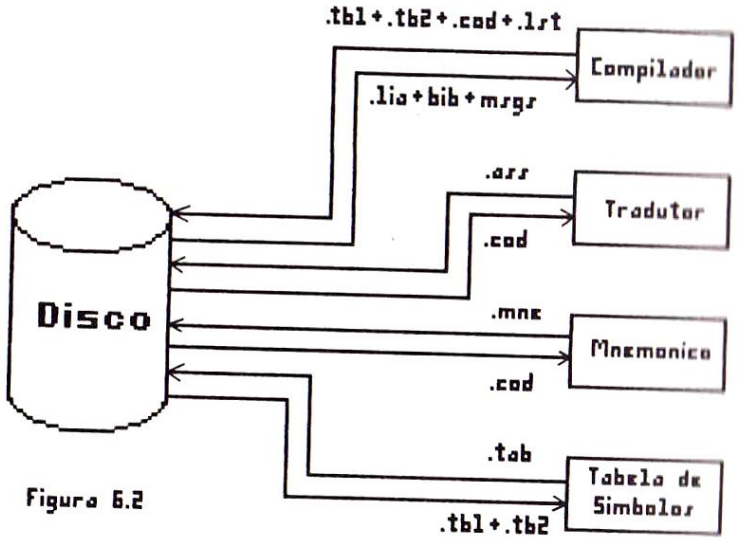

Figura 6.2

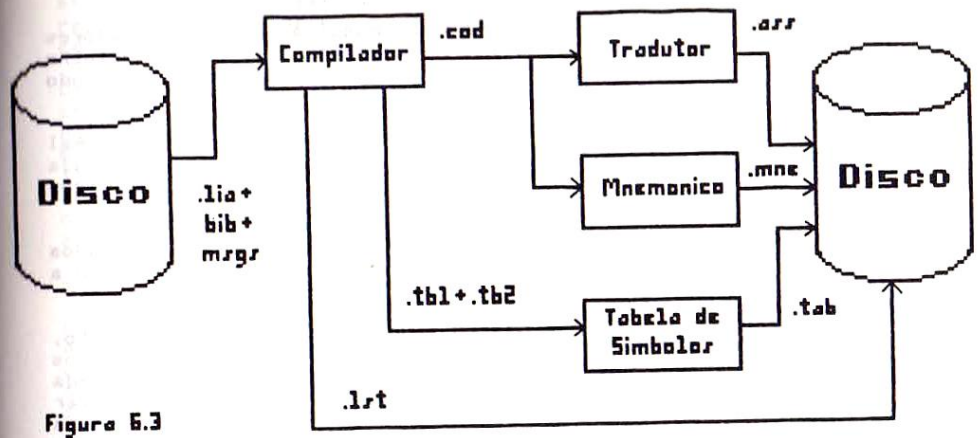

Figura 6.3

1.4 Técnicas Utilizadas

Este item detalha a forma escolhida para a implementação de cada um dos blocos do compilador.

- Analisador Léxico -

Através da interface com as rotinas de entrada e saída do ambiente de execução da máquina hospedeira, o ALe lê, individualmente, os caracteres do texto fonte necessários ao reconhecimento de um átomo da linguagem (segundo a ótica do ASi). Caracteres de controle (como mudança de linha e retorno do carro) são descartados pelo ALe durante esta operação. Cada átomo é classificado em termos de classe e valor. A classe, que representa a identificação do átomo, é passada para o ASi, ao passo que o valor, que representa uma eventual complementação a identificação informada na classe, é passada para o ASe e GCo. São apenas duas as classes de átomos que tem a sua identificação complementada por um valor : a dos identificadores (cujo valor corresponde a sequência dos caracteres que o definem) e a dos

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.

números inteiros (corespondente a conversão numérica, para o formato interno da máquina hospedeira, do valor expresso pelo conjunto dos caracteres que o definem).

Além da conversão numérica, o ALe pesquisa os identificadores reconhecidos na busca de palavras reservadas. Devido ao pequeno número de palavras reservadas, a pesquisa é efetuada de modo linear.

Não existe tratamento de erros a nível do ALe. Átomos mal formados fazem com que o ALe produza um átomo especial, cuja classe é ILEGAL, para o ASi. Desta forma, o tratamento de erros léxicos ocorre sempre a nível da análise sintática.

Os caracteres consumidos pelo ALe são individualmente passados para o FLi, o qual se encarrega de re-agrupá-los possibilitando a obtenção de uma listagem formatada do programa fonte.

O ALe corresponde a implementação, na forma de um procedimento, de um autômato finito (a linguagem definida pelo conjunto dos atomos é regular) que é ativado sob o comando do ASi. Cada transição deste autômato está associada ao consumo de um caracter do programa fonte, sendo que um conjunto de estados finais denota o reconhecimento de diferentes categorias léxicas (classes de átomos). Quando o ALe é ativado pelo ASi, o referido autômato inicia a sua operação a partir do estado inicial. Ao atingir um estado final, o controle retorna para o ASi, sendo que as informações de classe e valor são passadas por meio de uma área de dados global. Comentários e caracteres em branco são descartados pelo analisador.

O tratamento de fim de programa fonte é realizado exclusivamente pelo ASi. Toda vez que o ALe é ativado para o reconhecimento de mais um átomo, e este constatar o esgotamento da cadeia de entrada, ele produz para o ASi um átomo especial cuja classe é FIM_DE_TEXTO.

- Analisador Sintático -

O ASi constitui o núcleo do compilador, sendo ele o responsável pela ativação dos demais blocos de análise do texto fonte e de síntese do código objeto. Ele é implementado na forma de um conjunto de procedimentos recursivos, cada qual correspondente a uma sub-máquina associada ao reconhecimento de estruturas pertencentes a uma classe sintática particular.

O mecanismo de recuperação de erros consiste em se descartar o átomo que provocou a perda do sincronismo do reconhecedor com a cadeia de entrada. Eventualmente, a recuperação do sincronismo pode exigir que o reconhecedor descarte, além do átomo corrente, uma sequência de átomos que sigam o mesmo. O sincronismo é re-encontrado quando for localizado algum átomo que satisfaça a uma certa condição. Esta condição é a de pertencer a um conjunto de átomos, conjunto este que varia dinâmicamente com o progresso do reconhecimento. Essencialmente, este conjunto contém as classes dos átomos potencialmente encontráveis na sub-árvore correspondente a expansão do não-terminal representado pelo procedimento corrente, bem como daqueles associados aos procedimentos que ativaram este, desde a raiz da gramática. Desta

forma, a recuperação de um erro sintático é tentada inicialmente a partir do contexto mais restrito em torno do ponto de ocorrência do mesmo, e daí em direção a contextos mais externos até que seja possível encontrar um ponto de sincronismo do reconhecedor (pode ser a mesma ou então outra sub-máquina) com a cadeia de entrada.

Os erros de sintaxe detectados pelo ASi são informados ao FLi para que este providencie o registro da ocorrência dos mesmos nos pontos corretos da listagem do texto fonte. Os átomos descartados pelo ASi durante a fase de recuperação de erros sao realçados na listagem do programa, de modo que seja possível para o usuário do compilador estar a par dos eventuais trechos de seu programa que não tenham sofrido qualquer tipo de análise (e que portanto podem ainda conter novos erros).

- Analisador Semântico -

A tarefa principal do ASe é implementar a verificação das dependências de contexto exibidas pela linguagem e não tratadas pelo ASi. Além disso, ele promove coerção de tipos nos pontos onde cabíveis, controla a alocação de memória para os objetos do programa e interpreta o significado de objetos pré-definidos.

O ASe é composto por uma coleção de procedimentos, todos eles ativados exclusivamente pelo ASi, cada qual sendo responsável por uma tarefa bem específica, como por exemplo verificar a observação de uma determinada dependência de contexto. Estas ações semânticas não possuem definição formal de qualquer natureza , tendo sido derivadas a partir da descrição em linguagem natural da LIA. As dependências de contexto são várias e involvem, por exemplo, a verificação do uso coerente dos identificadores e da compatibilidade no uso de tipos.

Os erros detectados por estes procedimentos são passados para o FLi, da mesma forma que o ASi procede ao detectar erros no programa fonte. Porém, existe uma diferença fundamental entre ambos os casos, pois a recuperação de um erro nos procedimentos de análise semântica ocorre apenas ao nível da manipulação das estruturas de informação que este gerencia e vai construindo ao longo da análise, não envolvendo nunca, como no caso do ASi, a rejeição de trechos da cadeia de entrada na busca de um ponto de sincronismo. Esta característica permite que o super conjunto definido pela descrição livre de contexto da linguagem seja sempre reconhecido, mesmo na presença de erros relativos às dependências de contexto. Desta forma, a utilização do compilador se torna mais simplificada, uma vez que o usuário do mesmo recebe uma melhor orientação acerca dos erros cometidos. Além disso, a análise realizada é tão extensiva quanto possível, pois todos os átomos do programa são sempre considerados. Em outras palavras, do ponto de vista do usuário, a simples aderência à descrição gramatical livre de contexto é suficiente para que o seu programa seja completamente analisado.

O ASe se comunica com o TSi toda vez que ele necessitar conhecer os atributos de um nome para poder proceder a verificação quanto ao uso coerente do mesmo.

- Formatador de Listagens -

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.

Este bloco recebe dados provenientes de três fontes : o ALe, o ASi e o ASe. Do primeiro ele recebe individualmente os caracteres que este consome nas suas transições. Do segundo, ele recebe códigos associados aos tipos dos erros detectados pelo reconhecedor livre de contexto da linguagem, bem como informações sobre eventuais trechos de programa descartados durante um procedimento de recuperação de erros. Do terceiro são recebidos os códigos relativos aos erros que envolvem dependências de contexto.

A partir destes dados o FLi as combina de modo a produzir uma listagem formatada do programa fonte contendo numeração de páginas e de linhas, cabeçalho e indicações dos erros detectados. Estas indicações são inseridas em pontos próximos aos das suas ocorrências, uma vez que nem sempre existe coincidência do ponto de detecção com o ponto de ocorrência do mesmo. Para cada posição assinalada, o FLi anexa o código do erro correspondente. No final da listagem o FLi emite uma relação de mensagens relativas aos códigos gerados, a qual permite a elucidação mais imediata acerca da natureza dos erros cometidos. A listagem assim formatada não é diretamente impressa, sendo ao invés disso enviada para o disco do sistema no qual fica a disposição para impressão quando o usuário assim o desejar.

- Gerenciador da Tabela de Símbolos -

A tabela de símbolos é organizada na forma de uma pilha de árvores binárias não balanceadas em cujos nós são armazenados os identificadores e seus atributos. Cada árvore nesta pilha está associada ao conjunto dos identificadores declarados numa certa unidade de programa. O conjunto das árvores, num certo instante da análise, reflete a estrutura estática de aninhamento exibida pelo programa no respectivo ponto. A árvore mais antiga nesta pilha corresponde à unidade de programa mais externa e a mais recente à unidade correntemente sendo analisada.

Quando se inicia a análise de uma nova unidade de programa, cria-se uma árvore vazia no topo da pilha, ao passo que quando se atinge o final da mesma procede-se a remoção da respectiva árvore (e que está, por construção, sempre no topo da pilha). Este mecanismo permite que, em qualquer instante, a tabela de símbolos contenha exatamente o conjunto dos identificadores visíveis num certo ponto do programa sob análise.

Os procedimentos de gerenciamento da tabela de símbolos permitem a criação ou remoção de árvores no topo da pilha, a inicialização ou o armazenamento (em disco) da árvore no topo da pilha e também a inserção e pesquisa de identificadores na árvore situada no topo da pilha.

A inicialização é sempre solicitada pelo ASi e consiste na carga da Biblioteca de Sinais selecionada pelo programa e residente no disco, na árvore no topo da pilha. A referência a uma determinada Biblioteca é feita por meio de uma diretiva de compilação. A carga é realizada por meio da repetição, até que se esgote a Biblioteca, das operações de leitura de um sinal e da sua inserção (junto com seus atributos) na árvore do topo da pilha. Desta maneira, o compilador pode garantir o uso coerente dos

sinais e o usuário do compilador não precisa repetir a declaração dos mesmos em cada programa. Este tipo de inicialização pode ocorrer em árvores correspondentes a qualquer unidade de programa. Um segundo tipo de inicialização ocorre apenas em relação à árvore representante do nível mais externo de programa, que é o bloco de implementação. Neste caso, a árvore é inicializada com os identificadores pré-definidos da linguagem.

O pedido de inserção de nomes nas árvores é feito pelo ASi, que passa para o TSi o identificador a ser inserido e os seus atributos como parâmetros. O procedimento de inserção é responsável pela verificação da duplicidade de nomes num mesmo contexto, e, em caso de verificação desta condição, pela sinalização para o FLi da situação de erro.

O pedido de pesquisa de nomes provém das rotinas de análise das dependências de contexto e também das rotinas de geração de código, as quais passam o identificador que se pretende localizar como parâmetro de entrada. Em caso de localização, os atributos do mesmo são devolvidos como parâmetros de saída. Em caso contrário, é devolvida a informação de não localização, não sendo emitido nenhum código de erro pelo procedimento de pesquisa mas sim pelos procedimentos que o ativaram. Isto decorre do fato de que estes procedimentos conhecem o contexto no qual o identificador está sendo usado, sendo portanto capazes de emitir códigos de erro mais expressivos acerca da não localização do identificador.

Ao atingir o fim de uma unidade de programa, o ASi solicita a remoção da árvore de identificadores situada no topo da pilha. Esta remoção é acompanhada do armazenamento da mesma no disco. Isto é feito com o intuito de permitir uma posterior recuperação da mesma possibilitando a geração da listagem relativa à tabela de símbolos global do programa compilado.

- Gerador de Código -

É composto por rotinas que são chamadas pelo ASi e pelo ASe. Eventualmente, o GCo pode necessitar consultar a tabela de símbolos com o intuito de obter os atributos de um nome para poder gerar o código correspondente a utilização do mesmo.

Como saída, este bloco produz sequências de instruções de uma máquina de pilha virtual as quais são armazenadas no disco da máquina hospedeira. Cada instrução é representada como uma sequência variável de números inteiros, os quais indicam a presença ou não de um rótulo antecedendo a instrução, o código da instrução propriamente dito e os seus operandos. A representação mnemônica de um programa intermediário pode ser obtida com o auxílio de um programa utilitário encarregado de efetuar esta conversão. Esta característica do GCo permite um melhor desempenho do programa tradutor, uma vez que não existe necessidade de realização de análise léxica sobre o programa intermediário.

2 Metodologia de Programação e Desenvolvimento

O desenvolvimento de um compilador para uma linguagem do porte da LIA é uma tarefa que exibe razoável complexidade. Portanto, a

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, 1: 697-714, 1988.

necessidade de adoção de uma estratégia adequada para sistematizar a evolução do trabalho é indiscutível. Neste item são apresentadas as principais decisões de orientação do projeto e discutidas as repercussões da adoção efetiva destas.

Inicialmente, há que se destacar a necessidade da escolha de ferramentas adequadas para suportar o projeto. Devido as características deste, seria desejável que a linguagem de programação adotada exibisse propriedades tais como :

- Suporte a estruturação do código
- Suporte a estruturação dos dados
- Suporte a definição de novos tipos de dados
- Suporte a abstração de funções
- Suporte a abstração de tipos de dados
- Suporte ao desenvolvimento modular de programas
- Boa legibilidade
- Boa portabilidade
- Boa disponibilidade

Apesar de não ser satisfatória em alguns destes aspectos (principalmente os de abstração de tipos de dados e de desenvolvimento modular de programas), a linguagem escolhida foi a Pascal. Esta escolha se justifica pela baixa disponibilidade e suporte para linguagens mais modernas que incorporem satisfatóriamente todos os conceitos acima relacionados.

No entanto, a Pascal se mostrou bastante adequada para o desenvolvimento do projeto, principalmente pelos seus aspectos de estruturação, definição de novos tipos de dados, legibilidade e portabilidade. A estruturação é essencial, pois facilita a organização e consequentemente a manutenção do programa. Definir novos tipos de dados é bastante conveniente na medida em que os objetos manipulados pelo programa podem se mostrar mais aderentes à natureza do problema que está sendo resolvido. A legibilidade produz como efeito secundário uma elevação na confiabilidade do programa, na medida em que é mais fácil entender o seu significado. Finalmente, o aspecto de portabilidade se revelou de extrema importância, pois ao longo do desenvolvimento do projeto foram utilizados vários computadores hospedeiros, cada qual com a sua própria versão da linguagem Pascal. Por meio de uma estrita aderência à definição básica da mesma, foi possível transportar o programa nas suas formas intermediárias diversas vezes sem que isto representasse obstáculo mais sério para o andamento do trabalho.

O planejamento do trabalho foi efetuado de modo a que o mesmo iniciasse com tarefas pequenas e de baixa complexidade, evoluindo posteriormente no sentido das tarefas maiores e mais complexas.

Em particular, o primeiro bloco que foi construído foi o analisador léxico, devido ao seu pequeno porte, reduzida complexidade conceitual e independência dos demais blocos do compilador. Uma vez construído este analisador, já era possível executá-lo de forma autônoma e testá-lo completamente. Este aspecto é importante, pois permite que a próxima etapa do trabalho seja principiada partindo-se de uma unidade básica livre de erros.

A seguir, procedeu-se à construção do analisador sintático, que utilizava como único bloco adicional o léxico construído anteriormente. O sintático é um bloco que possui uma complexidade intermediaria, porém a sua construção foi facilitada pela existência de métodos sistemáticos para a construção de reconhecedores. Com o analisador sintático pronto, procedeu-se a integração deste com o analisador léxico, obtendo-se como resultado um reconhecedor completo e autônomo para o super-conjunto livre de contexto definido pela gramática da LIA. Com a verificação deste reconhecedor, criam-se condições para a anexação dos próximos blocos.

As rotinas de manipulação da tabela de símbolos foram as próximas a serem construídas e anexadas no programa, bem as chamadas destas nos pontos correspondentes. O cumprimento desta etapa induz naturalmente a realização seguinte, que consiste na incorporação das rotinas de análise das dependências de contexto e finalmente das de geração de código.

As vantagens da realização do trabalho segundo este plano são as seguintes :

- possibilidade de construção de partes que podem ser testadas autônomamente ou então com base em outras partes já testadas, o que confere um bom grau de confiabilidade ao processo de desenvolvimento.

- possibilidade de abordargem de um problema complexo no todo, partindo-se de sub-problemas inicialmente simples e posteriormente de complexidade crescente, até que o problema final tenha sido resolvido. Isto facilita a elaboração de uma solução final.

- possibilidade de se criar e manter um clima de entusiasmo durante todo o andamento do projeto, uma vez que os produtos das etapas intermediárias são sempre capazes de mostrar os resultados da sua implementação. Apesar de subjetivo, este aspecto tem uma importância particular, pois é importante que todo trabalho ofereca uma re-alimentação para aquele(s) que nele se empenha(m).

Com relação ao aspecto particular de implementação das rotinas de geração de código, pode-se mencionar ainda uma vantagem adicional. Esta decorre de que não existe necessidade de implementá-lo de uma vez só. Ao contrário, se a sua implementação for gradual, pode-se mais cedo obter versões do compilador que representam sub-conjuntos da linguagem original. Apesar de não traduzirem todos os comandos, estas versões tem a vantagem de facilitarem a divulgação e a familiarização da linguagem pela sua futura população de usuários, bem como propiciar a utilização das mesmas na produção de software experimental.

Um outro aspecto importante, é o que esta relacionado com a opção de geração de código para uma máquina abstrata ao invés de fazê-lo para uma máquina concreta. Além da vantagem evidente para os usuários do compilador em termos de portabilidade dos programas traduzidos, esta opção tem efeitos positivos sobre o próprio processo de desenvolvimento do compilador. Por um lado, faz com que o problema de tradução da linguagem fonte inicial para o

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.

código de máquina final seja simplificado uma vez que se reduz a distância entre as linguagens fonte e objeto. Por outro, evita que sejam incorporados no compilador detalhes que exigiriam a sua constante alteração e consequentemente manutenção de versões distintas, cada vez que fosse necessário criar programas para uma nova linguagem de máquina. Como resultado disso, criou-se um programa tradutor, encarregado de fazer a conversão do programa intermediário para a sua versão objeto final. Desta forma, para cada nova linguagem de máquina basta criar um novo tradutor, o que é uma vantagem, pois, em vez de se alterar um programa grande e complexo, constrói-se um novo programa pequeno e simples.

3 Validação e Testes

Os testes do compilador foram em grande parte simplificados devido a estratégia de desenvolvimento adotada, que se baseou na construção, teste e integração sucessiva de partes pequenas.

Em particular, os blocos de análise sintática e de geração de código também puderam ser desenvolvidos gradualmente. No caso do Asi, partindo-se de construções sintáticas independentes das demais e evoluindo-se para outras que usassem estas. Assim, o reconhecimento das estruturas sintáticas complexas pode ser feito a partir de outras mais simples, num processo que conferia maior segurança e consequentemente confiabilidade ao processo de desenvolvimento. No caso do GCo, o mesmo procedimento pode ser aplicado, conforme já mencionado.

Portanto, o desenvolvimento gradual e sistemático foram os principais fatôres que propiciaram a obtenção de um compilador que tem apresentado bom desempenho em termos de confiabilidade.

Por outro lado, houve um teste maior que envolveu tanto o compilador como o tradutor. Este teste consistiu na re-codificação de um processo de controle já existente num sistema CPA de 48 ramais por 16 troncos. Após a re-codificação, o processo original (escrito em linguagem de montagem) foi substituído, no software de um equipamento de utilizacao intensa e generalizada, pelo correspondente processo escrito em LIA. Desta forma, pode-se obter maior segurança com relação ao emprego do compilador nos aspectos de :

- funcionalidade da sua utilização.
- adequação aos objetivos iniciais de aderência à aplicação, legibilidade, produtividade, etc...
- correção do esquema de geração de código-objeto pelo compilador.
- correção do esquema de geração de código objeto pelo tradutor.
- compatibilidade da interface do código-objeto com o sistema operacional.
- velocidade de execução do código-objeto.

4 Conclusões

Este texto possui dois enfoques básicos. No primeiro deles, foi feita uma discussão técnica acerca do projeto de um compilador. Por se tratar de uma visão geral de um processo específico de tradução de uma linguagem de alto-nível, o trabalho pode ser interessante para aqueles que procuram estudar casos concretos com o intuito de compreender melhor a aplicação prática das inúmeras técnicas, conceitos e teorias pertinentes a este assunto. O segundo enfoque diz respeito à metodologia de desenvolvimento adotada e ao impacto ocasionado por esta. Trata-se de uma contribuição pessoal àqueles que pretendem iniciar projetos de natureza semelhante. Este aspecto não costuma ser abordado de forma mais profunda nos textos disponíveis sobre o assunto.

5 Referências Bibliográficas

As referências [8], [9] e [16] são introdutórias e apresentam grande valor quando tomadas como ponto de partida na obtenção de subsídios para um estudo mais aprofundado. As referências [1], [2], [4], [7], [14] e [15] possibilitam o aprofundamento do estudo por se tratarem de textos mais completos e detalhados. Em [5], [11], [13] e [16] são estudadas implementações de portes variados e também são contribuições importantes pois enfocam o lado prático do emprego da teoria e dos conceitos. Em particular, a referência [11] serviu como modelo tanto para o projeto do analisador léxico como das estruturas de dados que representam os objetos da LIA. A referência [3] tem uma conotação histórica pois sugeriu e adotou pela primeira vez a técnica da programação estruturada aplicada a construção de compiladores, com influência marcante em trabalhos subsequentes. Em [12] são introduzidos esquemas tradicionais de tradução das construções apresentadas pelas linguagens de programação convencionais. A referência [6] apresenta a essência da técnica utilizada neste trabalho para a construção do analisador sintático. Esta técnica é revisada e corrigida em alguns detalhes na referência [10].

A referência [17] representa o resultado do primeiro esforço de definição da LIA. Apesar de haver sofrido várias modificações, na essência o aspecto e o significado da linguagem continuam o mesmo. Detalhes sobre a SDL podem ser obtidos em [18].

[1] AHO, A.V.; ULLMAN, J.D. **The theory of parsing, translation and compiling.** Englewood Cliffs, Prentice-Hall, 1972. 2v.

[2] _____.; SETHI, R.; ULLMAN, J.D. **Compilers:** principles, techniques, and tools. Reading, Addison-Wesley, 1986. 796p.

[3] AMMANN, U. The method of structured programming applied to the development of a compiler. In: INTERNATIONAL COMPUTING SYMPOSIUM, Davos, 1973. **Computing 1973.** Amsterdan, ACM/ North-Holland, 1973. p.93-100.

[4] BACKHOUSE, R.C. **Syntax of programming languages:** theory and practice. Englewood Cliffs, Prentice-Hall, 1979. 301p.

[5] BARRON, D.W., ed. **PASCAL:** the language and its implementation. Chichester, Wiley, 1981. 301p.

[6] HARTMANN, A.C. Syntax analysis. In: ____. **A concurrent Pascal compiler for microcomputers.** Berlin, Springer, 1977. (Lectures Notes in Computer Science, 50) cap.5.

JOSÉ NETO, J. & RAMOS, M.V.M. Considerações sobre o projeto de um compilador para uma linguagem de programação de alto nível. **Anais EPUSP. Sér. B,** São Paulo, **1**: 697-714, 1988.