1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-3753-PRE/876		*Janeiro 1986*	☐ Interna ☒ Externa ☐ Restrita

4. Origem	Programa
DIN	*INFOR/INTAL*

6. Palavras chaves - selecionadas pelo(s) autor(es)

LISP
COMPILAÇÃO
INTERPRETAÇÃO

7. C.D.U.: *681.3.06*

8. Título *INPE-3753-PRE/876*

SISTEMA ALISP

10. Páginas: *16*

11. Última página: *11*

12. Revisada por

Sandra A. Sandri

9. Autoria *Edson Luiz França Senne*
Guilherme Bittencourt

13. Autorizada por

Marco Antonio Raupp
Diretor Geral

Assinatura responsável

14. Resumo/Notas

Neste trabalho descreve-se o sistema ALISP, composto de três módulos: um interpretador da linguagem LISP, um tradutor de rotinas escritas na linguagem LISP para a linguagem ALGOL e um montador cuja função é criar um novo interpretador no qual as diversas rotinas do usuário, anteriormente traduzidas, são processadas como primitivas. O sistema ALISP está escrito na linguagem ALGOL para o computador B6800 do INPE.

15. Observações *Este trabalho foi apresentado no 5º Simpósio sobre Desenvolvimento de Software Básico, 25-27 de novembro de 1985 em Belo Horizonte-MG.*

ABSTRACT

This work describes the ALISP system, consisting of three modules: a LISP interpreter, a translator of routines written in LISP into ALGOL and an assembler, whose function is to create a new interpreter in which the various user's routines previously translated are processed as primitives. The ALISP system is written in ALGOL for INPE's B6800 computer.

SUMÁRIO

SISTEMA ALISP

Edson Luiz França Senne
Guilherme Bittencourt

Ministério da Ciência e Tecnologia - MCT
Instituto de Pesquisas Espaciais - INPE
Caixa Postal 515 - 12225 - São José dos Campos - SP

SUMÁRIO

Neste trabalho descreve-se o sistema ALISP, composto de três módulos: um interpretador da linguagem LISP, um tradutor de rotinas escritas na linguagem LISP para a linguagem ALGOL e um montador cuja função é criar um novo interpretador no qual as diversas rotinas do usuário, anteriormente traduzidas, são processadas como primitivas. O sistema ALISP está escrito na linguagem ALGOL para o computador B6800 do INPE.

1. INTRODUÇÃO

A linguagem LISP, desde que foi introduzida por McCarthy (1960), vem sendo largamente utilizada para problemas de manipulação simbólica e inteligência artificial. Entretanto, a despeito de ser considerada eficiente para qualquer tipo de processamento simbólico, sempre carregou a "fama" de ser uma linguagem lenta. Isto se deve principalmente ao fato de que, em geral, os programas LISP são interpretados, uma vez que, pela própria estrutura da linguagem, a implementação de compiladores apresenta algumas complicações (Honschopp et alii, 1983).

Este trabalho descreve o sistema ALISP, desenvolvido com o objetivo de alcançar eficiência no processamento de programas LISP. Para isto, optou-se por escrever em ALGOL: (i) um novo *interpretador* da linguagem LISP, com estrutura flexível a ponto de ser possível aumentá-lo com novas primitivas;

(ii) um programa *tradutor* de rotinas LISP para o ALGOL; e (iii) um programa *montador* para incorporar as rotinas traduzidas ao interpretador.

Com a utilização deste sistema, um programa LISP (em geral um conjunto de funções) é convertido num programa ALGOL equivalente, o qual pode então ser compilado e processado eficientemente.

Apresenta-se a seguir uma breve descrição de cada um dos três módulos que compõem o sistema. Na Seção 2 descrevem-se o módulo interpretador, a representação interna das entidades LISP e o método de avaliação. Na Seção 3 descrevem-se os módulos tradutor e montador e compara-se o desempenho de funções quando executadas de forma compilada e de forma interpretada. Na Seção 4 são apresentadas algumas conclusões sobre o sistema ALISP e o que se pretende no futuro.

2. MÓDULO INTERPRETADOR

A linguagem LISP, reconhece dois tipos de estruturas: ÁTOMOS e LISTAS. Os átomos podem ser de três tipos: símbolos alfanuméricos, números ou "strings". Já as listas são entidades iniciadas por "(" e terminadas por ")", e formadas por um número qualquer de elementos que podem ser tanto ÁTOMOS como novas LISTAS. Por exemplo:

A, Z123, XY2, 12, "ISTO E UM STRING" : são átomos;

(A B C), (1 (A B) ((C))), ("22" 12 Z) : são listas.

No sistema ALISP as listas são representadas em uma matriz bidimensional, com duas colunas e tantas linhas quantas forem necessárias para o usuário. Para cada linha da matriz de listas, a primeira coluna representa a cabeça da lista (CAR), e a segunda coluna o resto, ou cauda, da lista (CDR). Sempre que durante uma operação faltar espaço nesta matriz (inicialmente,

existe espaço para 1000 Células), ela é aumentada através do comando RESIZE do ALGOL (Segre, 1981), com a particularidade de que uma vez terminada a operação o espaço disponível é remanejado, numa operação de "garbage collection" (Cohen, 1981), de modo que possa ser novamente utilizado.

Os diferentes tipos de átomos são representados em diferentes vetores. Assim existe um vetor para símbolos, um vetor para números e um vetor para "strings". A armazenagem dos átomos é feita de tal maneira que não existem átomos duplicados em nenhum destes vetores. Além destes, existe um vetor que contém os valores dos símbolos e um vetor específico para o armazenamento de propriedades de símbolos, uma estrutura presente na linguagem LISP que auxilia muito o processamento simbólico.

Tal qual a matriz de listas, os vetores responsáveis pelo armazenamento de átomos são aumentados em tamanho sempre que as necessidades do usuário exigirem.

O programa INTERPRETADOR do sistema ALISP foi desenvolvido de forma modular e, portanto, cada função da linguagem LISP é realizada através de um procedimento ALGOL adequado. Além destes, existem os procedimentos específicos para a manipulação das estruturas internas e para entrada e saída, isto é, para a transformação de átomos e listas na representação interna (vetores e matrizes) e vice-versa.

Uma rotina importante do interpretador é a que cuida dos valores de símbolos, isto porque a linguagem LISP faz uso de escopo dinâmico e os valores são determinados pelo ambiente de avaliação (Winston and Horn, 1981). Esta rotina utiliza o espaço de listas para salvar os valores de símbolos toda vez que se inicia a execução de uma função externa (valores de parâmetros) ou uma função PROG (valores de variáveis locais), restaurando os valores salvos ao término da execução.

A rotina "EVAL" é responsável pela avaliação de qualquer entidade (LISTAS ou ÁTOMOS) submetida ao interpretador. Através dela é feito o encaminhamento do controle para uma das rotinas internas que implementam a semântica das primitivas LISP, ou para alguma rotina traduzida.

3. MÓDULOS TRADUTOR E MONTADOR

Os módulos TRADUTOR e MONTADOR são utilizados para estender o interpretador acrescentando novas primitivas definidas pelo usuário. O módulo tradutor recebe como entrada um arquivo de funções LISP do tipo EXPR (Marti et alii, 1979) e produz os arquivos: TRADUTOR/SAIDA, que contêm os procedimentos ALGOL equivalentes às funções LISP definidas no arquivo de entrada e TRADUTOR/FUNÇOES, que armazena os nomes e o número de parâmetros das funções traduzidas e também os nomes das variáveis globais que deverão ser declaradas. Devido ao arquivo TRADUTOR/FUNÇOES é possível utilizar o módulo tradutor repetidas vezes para vários arquivos de programas LISP.

No processo de tradução, as referências às funções primitivas são transformadas em chamadas para as rotinas internas do interpretador, e as referências às funções definidas pelo usuário são transformadas em chamadas para os procedimentos ALGOL traduzidos (ou declarados como FORWARD, caso não tenham sido traduzidos ainda). Algumas primitivas LISP, no entanto, recebem tratamento especial. Dentre elas têm-se:

a) COND - traduzida para uma estrutura IF-THEN-ELSE;

b) PROG - traduzida como um procedimento do tipo real que retorna o valor da variável VOLTA (atualizado pela tradução da função RETURN);

c) PROGN - traduzida também como um procedimento do tipo real, mas que retorna o valor da última função executada;

d) RETURN - traduzida como uma estrutura BEGIN-END que atualiza o valor da variável VOLTA e transfere o controle para o fim do procedimento;

e) PLUS, TIMES, AND, OR, MAX, MIN - que, devido ao fato de poder ter um número variável de parâmetros, são traduzidas numa sequência de chamadas com dois parâmetros. Por exemplo: (PLUS 1 2 3 4) é traduzido para PLUS (1, PLUS (2, PLUS (3, 4))).

O módulo montador recebe como entrada os arquivos TRADUTOR/SAIDA e TRADUTOR/FUNCOES e o arquivo LISP/INTERPRETADOR que contém o módulo interpretador e produz como saída um novo interpretador, com a incorporação dos procedimentos do arquivo TRADUTOR/SAIDA como primitivas. A Figura 1 mostra o esquema de utilização do sistema ALISP.

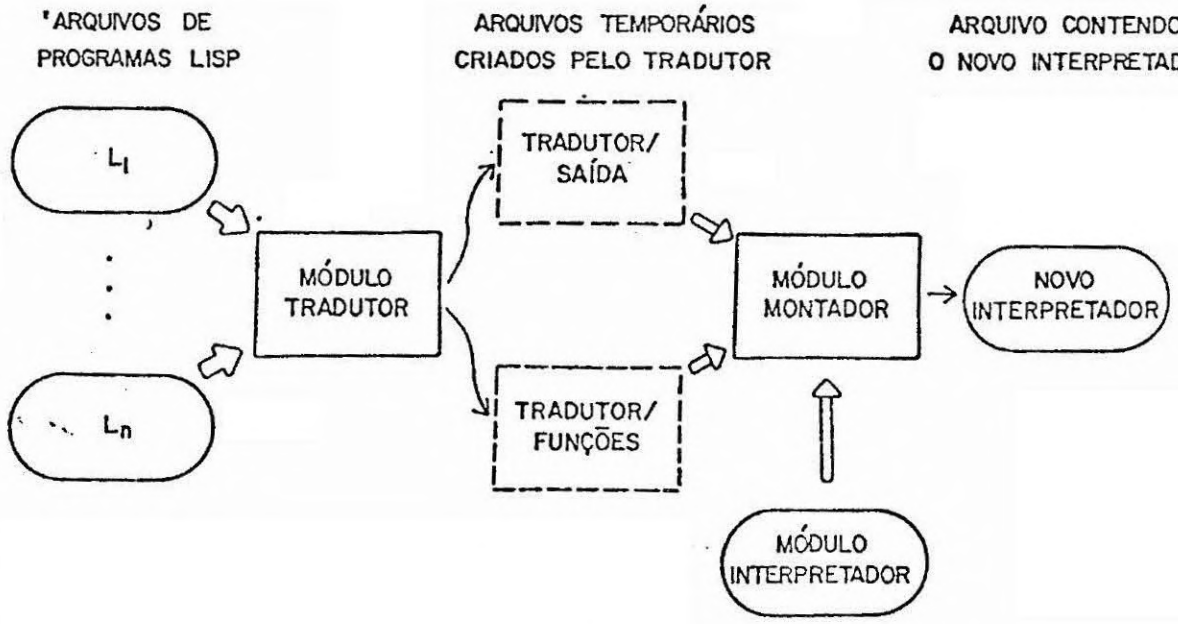

Fig. 1 - Esquema de utilização do sistema ALISP.

Para ilustrar, seja o arquivo EXEMPLO com as funções:

```
(DE FATORIAL (X)
  (COND ((ZEROP X) 1)
        (T (TIMES X (FATORIAL (SUB1 X))))))

(DE CIRCULAR (X N)
(PROG (K)
   (SETQ K 1)
LOOP
   (COND ((GREATERP K N)(RETURN X)))
   (SETQ X (APPEND (CDR X)(LIST (CAR X))))
   (SETQ K (ADD1 K))
   (GO LOOP)))
```

Apresenta-se a seguir trechos de sessão de utilização do sistema ALISP.

Inicialmente mostra-se a execução destas funções por interpretação:

```
SISTEMA ALISP - INTERPRETADOR - VERSAO: 11 JUN 85

> (RDS (OPEN 'EXEMPLO 'INPUT))
0
FATORIAL
CIRCULAR
*EOF*
> (FATORIAL 10)
3628800
> (FATORIAL 20)
2.43290200819E+18
> (CIRCULAR '(A B C D) 50)
(C D A B)
> (CIRCULAR '(A B C D) 100)
(A B C D)
> $ST
```

```
TEMPO DE SESSAO ..... = 00:37      MINUTOS: SEGUNDOS
ESPAÇO DE LISTAS UTILIZADO .... = 278      DISPONIVEL = 2722
SIMBOLOS UTILIZADOS ..... = 25      DISPONIVEL = 475
NUMEROS UTILIZADOS ..... = 118      DISPONIVEL = 382
ESPAÇO DE STRINGS UTILIZADO ... = 0      DISPONIVEL = 3000

> (QUIT)
# END SESSION - ET=43.5 PT=11.3 IO=0.4
```

Nota-se neste caso que o espaço de listas foi aumentado para 3000 células e foram necessárias operações de "garbage collection".

Obtém-se a tradução das funções do arquivo EXEMPLO invocando o módulo tradutor:

```
SISTEMA ALISP - TRADUTOR - VERSAO: 11 JUN 85

TRADUCAO DO ARQUIVO .:.> EXEMPLO
FATORIAL
CIRCULAR
# END SESSION - ET=36.0 PT=2.7 IO=1.4
```

O arquivo TRADUTOR/SAIDA produzido neste caso apresenta-se como:

```
REAL X, N, K;

REAL PROCEDURE FATORIAL ( LSP1 );
VALUE LSP1;
REAL LSP1;
BEGIN
  LOCAIS (CONS (X, NIL));
  SET(X, LSP1);
  FATORIAL := RETURN ( IF ZEROP (VALOR (X)) NEQ NIL
  THEN
    QNUM (1)
```

```
    ELSE
    TIMES (VALOR (X), FATORIAL (SUB1. (VALOR (X))))).
  END;

  REAL PROCEDURE CIRCULAR ( LSP1, LSP2 );
  VALUE LSP1, LSP2;
  REAL LSP1, LSP2;
  BEGIN
    REAL PROCEDURE PROG1;
    BEGIN
      REAL VOLTA;
      LABEL FIM, LOOP;
      VOLTA := NIL;
      LOCAIS (CONS (K, NIL));
      SET (K, QNUM (1));
      LOOP: IF GREATERP (VALOR (K), VALOR (N)) NEQ NIL
      THEN
        BEGIN
          VOLTA := VALOR (X);
          GO TO FIM;
        END
      ELSE
      NIL;
      SET (X, APPEND (CDR (VALOR (X)), CONS (CAR (VALOR (X)), NIL)));
      SET (K, ADD1 (VALOR (K)));
      GO TO LOOP;
      FIM: PROG1 := RETURN (VOLTA);
    END;
    LOCAIS (CONS (X, CONS (N, NIL)));
    SET (X, LSP1);
    SET (N, LSP2);
    CIRCULAR := RETURN (PROG1);
  END;
```

Para conseguir um novo interpretador com as funções traduzidas, incorporadas como primitivas, usa-se o módulo montador:

```
SISTEMA ALISP - MONTADOR - VERSAO: 11 JUN 85

MONTAGEM NO ARQUIVO ...> TESTE
## END SESSION - ET=59.2 PT=7.3 IO=6.2
```

Seja agora a execução das funções já compiladas no interpretador TESTE:

```
TESTE - VERSAO: 20/AGO/1985 (TERCA-FEIRA) 08:31:27

> (FATORIAL 10)
3628800
> (FATORIAL 20)
2.43290200819E+18
> (CIRCULAR '(A B C D) 50)
(C D A B)
> (CIRCULAR '(A B C D) 100)
(A B C D)
> $ST

TEMPO DE SESSAO              ... = 00:08   MINUTOS: SEGUNDOS
ESPAÇO DE LISTAS UTILIZADO   ... = 894     DISPONIVEL = 106
SIMBOLOS UTILIZADOS ..... = 21     DISPONIVEL = 479
NUMEROS UTILIZADOS ..... = 118     DISPONIVEL = 382
ESPAÇO DE STRINGS UTILIZADO ... = 0       DISPONIVEL = 3000

> (QUIT)
## END SESSION - ET=28.0 PT=1.4 IO=0.1
```

Neste caso o espaço de listas não precisou ser aumentado. Nota-se também que não foi preciso efetuar operações de "garbage collection". O tempo de sessão é bem menor, neste caso, devido ao fato de não ser preciso ler o arquivo EXEMPLO para definir as funções FATORIAL e CIRCULAR, e o ganho em velocidade de processamento é de aproximadamente 8 (11.3/1.4) vezes.

4. CONCLUSÃO

Existem muitas vantagens em se ter um sistema que permita a compilação de procedimentos escritos na linguagem LISP sem perder as vantagens inerentes ao sistema de interpretação característico desta linguagem. Dentre elas podem-se citar:

- aumento na velocidade de execução de programas LISP,
- economia de memória no espaço de listas,
- diminuição do esforço despendido em operações de "garbage collection".

Em testes preliminares realizados obtiveram-se tempos de processamento cerca de sete vezes menores que os tempos gastos pela mesma rotina executada por um interpretador (sem considerar o tempo necessário para definir as rotinas no caso interpretado).

Utilizando a experiência obtida com este sistema pretende-se implementar um sistema semelhante na linguagem C, com o objetivo de tornar o sistema transportável para microcomputadores.

5. REFERÊNCIAS BIBLIOGRÁFICAS

COHEN, J. Garbage Collection on Linked Data Structures. *Computing Surveys*, 13(3):341-367, Sept. 1981.

HONSCHOPP, U.; LIPPE, W.M.; SIMON, F. Compiling Functional Languages for Von Neumann Machines. Proceedings of the Sigplan'83 Symposium on Programming Languages Issues in Software Systems. *ACM Sigplan Notices*, 18(6):22-27, June 1983.

MARTI, J.B.; HEARN, A.C.; GRISS, C. *Standard LISP Report*. Salt Lake City, UT., University of Utah. 1979.

McCARTHY, J. Recursive Functions of Symbolic Expressions and their Computation by Machine, Part. I. *Communications of the ACM*, 12(3):184-195, Dec. 1960.

SEGRE, L.M. *Linguagem de Programação ALGOL*. Campus, Rio de Janeiro, RJ, 1981.

WINSTON, P.H.; HORN, B.K.P. *LISP*. Addison-Wesley, Reading, MA, 1981.