1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-4816-TDL/362		Junho 1989	☐ Interna ☒ Externa ☐ Restrita

4. Origem	Programa
PG-DCG	*FRH/ECO*

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

COMPUTAÇÃO INCREMENTAL *MULTIPROCESSAMENTO* *ARQUITETURA MIMD*
PROCESSAMENTO PARALELO *LINGUAGEM*

7. C.D.U.: **681.322.0**

8. Título

L - UMA LINGUAGEM PARA SOLUÇÃO INCREMENTAL DE SISTEMAS EM UM AMBIENTE DE PROCESSAMENTO PARALELO

10. Páginas: *121*

11. Última página: *B.12*

12. Revisada por

Eduardo W. Bergamini

9. Autoria *João Benedito Diehl*

Assinatura responsável

13. Autorizada por

Ralf Gielow
Presidente Conselho de Pós-Graduação

14. Resumo/Notas

Descrição de uma linguagem de alto nível, denominada linguagem L, apropriada para a programação de um Computador Incremental com capacidade de processamento paralelo para a solução de sistemas representáveis por modelos matemáticos. É introduzida a linguagem L e são também feitas considerações para a sua implementação. Recursos implementados para simulação da execução de programas em L em um ambiente de monoprocessamento são também apresentados, assim como um exemplo de aplicação.

15. Observações *Dissertação de Mestrado em Eletrônica e Telecomunicações - Opção Sistemas Digitais e Analógicos, aprovada em 21 de Dezembro de 1988.*

AGRADECIMENTOS

Ao Dr. Eduardo Whitaker Bergamini, pela orientação e apoio no desenvolvimento do trabalho.

Ao Dr. Alderico Rodrigues de Paula Jr. pelo apoio e esforço na manutenção dos cursos de ECO/SDA.

ABSTRACT

Description of a high level language, denominated L language, appropriate for programming an Incremental Computer with parallel processing capability, for the solution of systems wich can be represented by mathematical models. The L language is introduced and aspects of its implementation are commented as well. Resources implemented for simulation of execution of programs written in L in a monoprocessing environment are also presented and an application example is given.

Aprovada pela Banca Examinadora
em cumprimento a requisito exigido
para a obtenção do Título de Mestre
em Eletrônica e Telecomunicações

Dr. Alderico Rodrigues de Paula Júnior

Presidente

Dr. Eduardo Whitaker Bergamini

Orientador

Dr. Wilson Vicente Ruggiero

Membro da Banca
-convidado-

Dr. Cláudio Kirner

Membro da Banca
-convidado-

Candidato: João Benedito Diehl

São José dos Campos, 21 de dezembro de 1988

SUMÁRIO

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

A Atividade COMINC (COMputação INCremental), executada pela Linha de Pesquisa e Desenvolvimento em Engenharia de Computação do Instituto de Pesquisas Espaciais (INPE-SJC) tem por objetivo a pesquisa e desenvolvimento de máquinas com capacidade de processamento paralelo, para a realização de Computação Incremental. A pesquisa cobre as áreas de arquitetura, software básico e software de aplicação para essas máquinas.

Atualmente a atividade COMINC tem, já construído, um Computador Incremental que permite processamento paralelo com capacidade máxima para 64 processadores. A versão em operação, atualmente, contém 4 processadores.

Este trabalho tem por objetivo o desenvolvimento de uma linguagem de programação, chamada linguagem L, que facilite aos usuários a tarefa de programação de máquinas baseadas na arquitetura do Computador Incremental para solução incremental de sistemas representáveis por modelos matemáticos.

O **capítulo** 2 descreve brevemente a arquitetura com capacidade de processamento paralelo do Computador Incremental.

No **capítulo** 3 são descritas as principais características das linguagens de simulação de sistemas contínuos, nas quais será baseada a linguagem proposta.

No **capítulo** 4 são analisados os requisitos de uma linguagem para programação do Computador Incremental e, a partir

deles, é proposta uma linguagem que satisfaça tanto quanto possível esses requisitos.

No **capítulo** 5 são feitas considerações para implementação da linguagem L, tratando de aspectos relativos à compilação de programas em L e relativos ao suporte da execução de programas em L.

No **capítulo** 6 é descrito o processo de simulação da execução de programas em L em máquinas com monoprocessamento. Neste capítulo é descrito o protótipo do tradutor L e as rotinas de suporte da execução de programas em L que foram implementados.

No **capítulo** 7 são apresentadas as considerações finais sobre o trabalho desenvolvido.

CAPÍTULO 2

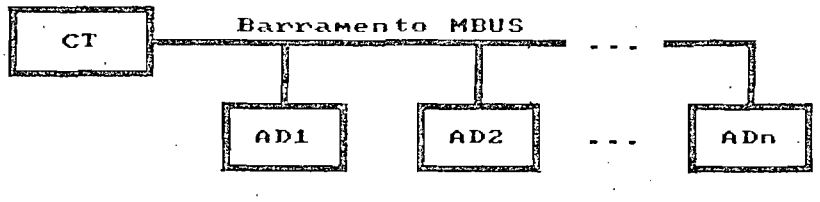
ARQUITETURA COM RECURSOS DE PROCESSAMENTO PARALELO

Neste capítulo é descrita a arquitetura do Computador Incremental, que servirá de subsídio para o desenvolvimento da Linguagem L. A linguagem L deverá ter características que permitam utilizar, de maneira apropriada, os recursos de processamento paralelo do Computador Incremental.

A seguir são descritas: a arquitetura do Computador Incremental, o modo de execução de processamento paralelo nessa máquina, e as características de comunicação entre os seus processadores.

2.1 - ARQUITETURA

A arquitetura do Computador Incremental consiste, basicamente, de diversas unidades de processamento autônomas conectadas a um barramento único, como mostra o diagrama de blocos simplificado da Figura 2.1.



CT: Controlador
AD: Analisador Digital

Fig. 2.1 - Diagrama em blocos simplificado da arquitetura do Computador Incremental

As unidades de processamento que compõem esta arquitetura são de dois tipos:

1) Unidade Mestre, denominada Controlador (CT);

2) Unidade Escrava, denominada Analisador Digital (AD);

O CT é a unidade responsável pelo gerenciamento do sistema, pela comunicação entre as unidades de processamento e pela comunicação com os dispositivos de Entrada e Saída externos. Esta última propriedade não é necessariamente exclusiva do CT.

Os ADs são as unidades responsáveis pela execução de tarefas de uma determinada computação. O início da execução dessas tarefas é comandado pelo CT. O AD pode, eventualmente, realizar tarefas de Entrada e Saída de dados, se as restrições de tempo real o exigirem.

O barramento que interliga as unidades de processamento do Computador Incremental foi denominado Barramento MBUS. Por ser um barramento apropriado para este fim, o MBUS facilita a inclusão de novos ADs no computador. Embora o potencial de cálculo de uma configuração cresça linearmente com o número de ADs, o barramento único pode ser um gargalo no desempenho da máquina, conforme o tipo de comunicação exigida entre os processadores e a carga de cálculo alocada a cada um deles.

Um sistema de computação baseado na arquitetura descrita na Figura 2.1 pode ser classificado como sendo do tipo MIMD ("Multiple Instruction Multiple Data") segundo a classificação de Flynn (1972).

2.2 - PROCESSAMENTO PARALELO

A execução do processamento paralelo no Computador Incremental é feita segundo o diagrama de tempo do fluxo de processamento apresentado na Figura 2.2.

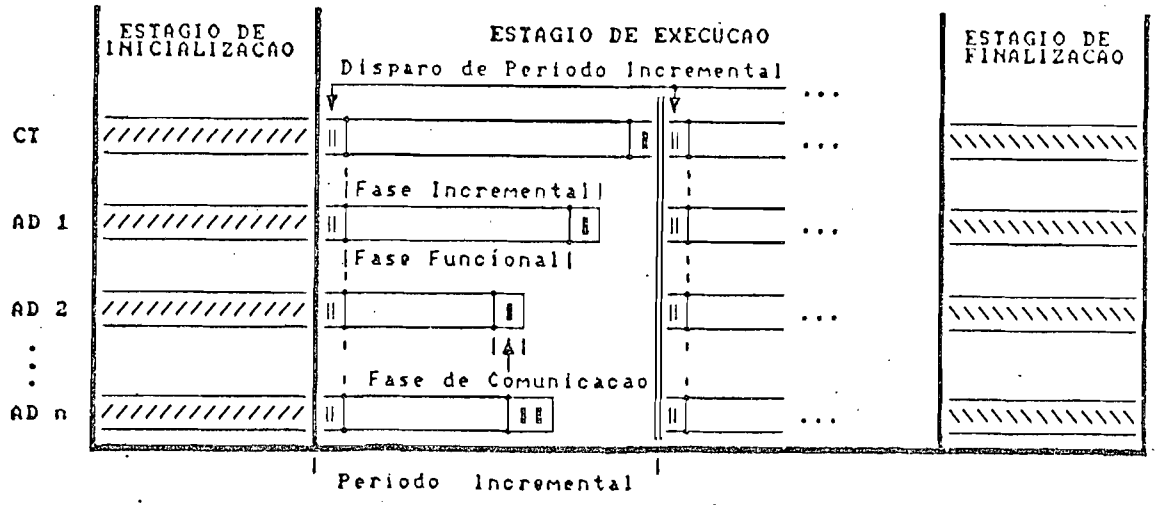


Fig. 2.2 - Diagrama de tempo do fluxo de processamento

O processamento no Computador Incremental é dividido em tres estágios: ESTÁGIO DE INICIALIZAÇÃO, ESTÁGIO DE EXECUÇÃO e ESTÁGIO DE FINALIZAÇÃO.

No **Estágio de Inicialização,** cada unidade de processamento do Computador Incremental é carregada com as tarefas que lhe couberam na partição do problema a ser resolvido e que deverão ser executadas no Estágio de Execução. No **Estágio de Execução,** ocorre a resolução propriamente dita do problema, em que cada unidade de processamento executa as tarefas que lhe foram destinadas, cabendo ao CT a tarefa de realizar a comunicação e a sincronização dessas unidades de processamento. O Estágio de Execução dura até que a solução do problema tenha sido completada ou interrompida. No Estágio

de Finalização são realizados os cálculos e operações de Entrada e Saída que devem ser executados após o término da execução do problema.

O Estágio de Execução é caracterizado pela repetição de PERÍODOS INCREMENTAIS.

Um **Período Incremental** é iniciado com um sinal de Disparo, comandado pelo CT. Quando ocorre o Disparo, todas as unidades de processamento iniciam a execução de uma FASE FUNCIONAL. Na **Fase Funcional** são efetuados os cálculos relativos às tarefas que cada unidade deve executar. Ao terminar a Fase Funcional, cada unidade realiza uma FASE DE COMUNICAÇÃO. Na **Fase de Comunicação**, cada unidade envia às demais os resultados pertinentes aos cálculos que ela executou em sua Fase Funcional. A execução de uma Fase Funcional e de uma Fase de Comunicação por uma unidade caracteriza a ocorrência de uma FASE INCREMENTAL nessa unidade. Quando todas as unidades terminaram uma Fase Incremental, encerra-se o PERÍODO INCREMENTAL em execução, e o próximo Período Incremental pode ser iniciado.

As Fases Funcionais são executadas em paralelo pelas unidades de processamento do Computador Incremental. As Fases de Comunicação são realizadas serialmente, quando cada unidade ganha acesso ao barramento MBUS.

O fluxo de processamento do Computador Incremental é adequado para solução de problemas que permitem uma decomposição em tarefas paralelas relativamente uniformes quanto ao tempo de execução e que devem ser executadas iterativamente, de maneira síncrona, exigindo comunicação entre as tarefas em toda iteração.

Um problema desse tipo é a solução numérica de um sistema de N equações diferenciais de primeira ordem por N ADs (Bergamini e Diehl, 1987). Neste caso, em cada ponto de discretização da variável independente, cada AD calcula a solução da equação que lhe

coube na partição do problema, e envia a solução obtida para os demais ADs, para que o cálculo possa prosseguir no próximo ponto.

Considere, como exemplo, o sistema de equações diferenciais ordinárias descrito abaixo:

$$y'_1 = f_1(t, y_1, y_2, \ldots, y_N)$$
$$y'_2 = f_2(t, y_1, y_2, \ldots, y_N)$$
$$\vdots$$
$$y'_N = f_N(t, y_1, y_2, \ldots, y_N)$$

de modo que a solução possa ser calculada nos pontos:

$$t_n = t_0 + n\ dt$$

onde: - dt é o incremento da variável independente t
- n é um inteiro

Um método simples para obter a solução aproximada desse sistema de equações é a aplicação do método de integração de Euler em cada equação:

dada a equação

$$y'_j = f_j(t, y_1, y_2, \ldots, y_N), \text{ onde } j = 1,2, \ldots, N,$$

a solução aproximada de y_j^{n+1} (aproximação da solução real $y_j(t_{n+1})$) é dada por:

$$y_j^{n+1} = y_j^n + f_j(t_n, y_1^n, y_2^n, \ldots, y_N^n) * dt$$

Supondo que este sistema de N equações seja distribuído para cálculo em N processadores, para calcular a solução em t_{n+1}, o

processador j deverá receber os resultados obtidos pelos outros processadores no ponto t_n . Se todos os processadores tem acesso aos resultados obtidos pelos outros processadores no ponto t_n , eles poderão realizar em paralelo o cálculo de suas equações no ponto t_{n+1} .

Este problema pode ser solucionado naturalmente no Computador Incremental, pois na Fase Funcional, os processadores calculam, em paralelo, a solução de suas equações, e na Fase de Comunicação, cada processador envia aos demais a solução obtida no ponto atual, de modo que o cálculo no próximo ponto pode ser realizado em paralelo logo que ocorrer o próximo sinal de disparo de início de Fase Funcional. O sinal de disparo de início de Fase Funcional serve como sinal de sincronismo para os processadores, já que um processador só pode realizar o cálculo no ponto atual quando todos os demais terminarem o cálculo no ponto anterior.

A seguir são definidos formalmente os termos utilizados na descrição do fluxo de processamento do Computador Incremental:

Estágio de Inicialização: estágio que precede a realização do cálculo pelo Computador Incremental, em que cada unidade de processamento que participa no cálculo é carregada com dados de inicialização e com as tarefas que lhe couberam na partição do problema;

Estágio de Execução: estágio em que as unidades de processamento do Computador Incremental realizam iterativamente suas tarefas. O Estágio de Execução dura até que a solução do problema tenha sido completada, ou interrompida;

Estágio de Finalização: estágio em que são realizados os cálculos e operações de Entrada e Saída que devem ser executados após o término do cálculo no Estágio de Execução;

Fase Funcional: período de tempo em que uma unidade de processamento (CT ou ADs) realiza as tarefas que lhe couberam na partição do problema. Todos os ADs iniciam uma Fase Funcional ao mesmo tempo, sincronizados pelo CT através de um **Sinal de Disparo** de Fase Funcional. As Fases Funcionais são realizadas em paralelo entre as diversas unidades de processamento;

Fase de Comunicação: é aquela realizada por cada unidade de processamento após o término de sua Fase Funcional. Na Fase de Comunicação, cada unidade de processamento envia às demais os dados que foram gerados na Fase Funcional, recém-executada. As Fases de Comunicação são realizadas serialmente, à medida em que cada unidade ganha acesso ao MBUS. Eventuais conflitos no uso do MBUS são resolvidos pelo seu árbitro, que determina a unidade com maior prioridade dentre as que aguardam a execução da Fase de Comunicação;

Fase Incremental: corresponde à execução de uma Fase Funcional e de uma Fase de Comunicação por uma unidade de processamento. O início de uma Fase Incremental corresponde, por definição, ao início de uma Fase Funcional;

Período Incremental: é o período compreendido entre dois sinais consecutivos de disparo de início de Fase Incremental. A execução de um Período Incremental é definida como sendo a execução de uma Fase Incremental por todas as unidades de processamento.

Na solução iterativa de problemas pelo Computador Incremental, os seguintes termos são definidos:

Variáveis de Estado: são as variáveis que armazenam os resultados de cada iteração do processo de solução de um problema pelo Computador Incremental.

Iteração Incremental: uma Iteração Incremental corresponde à execução de um Período Incremental.

Iteração Funcional: uma Iteração Funcional corresponde à execução, uma vez, das tarefas que caracterizam o problema que está sendo resolvido pelo Computador Incremental. A execução de uma Iteração Funcional implica na atualização, pelo menos intermediária, de todas as variáveis de estado do problema. Uma Iteração Funcional é composta por uma ou mais Iterações Incrementais.

Iteração Funcional Completa: uma Iteração Funcional Completa corresponde à execução de uma iteração no processo de solução de um problema pelo Computador Incremental. Uma Iteração Funcional Completa implica, necessariamente, na atualização de todas as variáveis de estado do problema, e pode ser composta por uma ou mais Iterações Funcionais.

Método Funcional: é o método utilizado para solução iterativa do problema. O Método Funcional define o número de Iterações Funcionais necessário para execução de uma Iteração Funcional Completa. O Método Funcional é denominado Método Funcional Monofase quando uma Iteração Funcional Completa é realizada com a execução de apenas uma Iteração Funcional e é denominado Método Funcional Multifase quando uma Iteração Funcional Completa é realizada com a execução de mais de uma Iteração Funcional.

O exemplo seguinte ilustra essas definições:

- Considere que o problema a ser resolvido pelo Computador Incremental seja calcular a solução numérica do sistema de equações diferenciais ordinárias descrito a seguir:

$$y_1' = f_1(t, y_1, y_2, \ldots, y_N)$$
$$y_2' = f_2(t, y_1, y_2, \ldots, y_N)$$
$$\vdots$$
$$y_N' = f_N(t, y_1, y_2, \ldots, y_N)$$

As **variáveis de estado** são $y_1, y_2, \ldots, y_N$.

As equações acima vão ser utilizadas para descrever as tarefas a serem realizados em uma **Iteração Funcional.**

O **Método Funcional** (neste caso, o método usado para integração das equações) vai definir o número de **Iterações Funcionais** necessário para realizar uma **Iteração Funcional Completa.**

Neste caso, uma Iteração Funcional Completa corresponde ao avanço do cálculo da solução do sistema de equações diferenciais do ponto t_n para o ponto t_{n+1}. O método de integração de Euler, por exemplo, é um Método Funcional Monofase, pois exige a execução de apenas uma Iteração Funcional para a realização de uma Iteração Funcional Completa. Há métodos de integração de equações diferenciais que exigem a execução de mais de uma Iteração Funcional para a realização de uma Iteração Funcional Completa; por exemplo, o método de integração Runge Kutta clássico exige a execução de quatro Iterações Funcionais para a realização de uma Iteração Funcional Completa. O método de integração Runge Kutta clássico é, portanto, um

Método Funcional Multifase, sendo que cada uma de suas fases exige a execução de uma Iteração Funcional.

2.3 - COMUNICAÇÃO ENTRE PROCESSADORES

As exigências de comunicação entre as unidades de processamento do Computador Incremental são rígidas, pois cada unidade deve enviar a todas as demais aqueles resultados obtidos na Fase Funcional atual que se fazem necessários para que a próxima Fase Funcional possa ser iniciada. Por isso, o processo de comunicação adotado foi o de **Difusão** ("broadcast"), em que uma unidade envia os dados necessários a todas as demais, que os recebem através de uma área de memória reservada para esse fim, denominada Memória de Comunicação.

Cada unidade de processamento do Computador Incremental possui uma área de memória destinada a ser utilizada como Memória de Comunicação. Todas as variáveis cujos valores são comunicados entre as unidades de processamento do Computador Incremental devem ser alocadas nessa área de Memória de Comunicação em todas as unidades de processamento. As variáveis alocadas na Memória de Comunicação são funcionalmente locais à unidade de processamento para leitura e globais para escrita, no sentido de que são atualizadas simultaneamente em todas as unidades de processamento por Difusão.

Na solução iterativa de sistemas de equações em paralelo, um processador, para realizar os cálculos na Iteração Funcional atual, pode precisar dos resultados obtidos na Iteração Funcional anterior por todos os outros processadores que participam do cálculo. Por isso, a comunicação por Difusão é bastante eficiente para este tipo de problema, já que um processador pode atualizar a Memória de Comunicação de todos os demais simultaneamente. Por outro lado, surge um problema de sincronismo, já que, na Iteração Funcional atual, os processadores precisam dos resultados dos outros processadores na

Iteração Funcional anterior, e esses dados podem estar sendo atualizados.

Para solucionar este problema, as variáveis de estado de um sistema de equações são duplicadas na Memória de Comunicação de todos os processadores. Assim, em uma determinada Iteração Funcional, os resultados da Iteração Funcional anterior são lidos de uma cópia da variável de estado e os resultados gerados nesta Iteração Funcional, e que serão utilizados na Iteração Funcional seguinte são escritos na outra cópia. Na Iteração Funcional seguinte os papéis se invertem, e a cópia utilizada como entrada é utilizada como saída. Esse esquema, embora duplique a demanda de memória das variáveis de estado, elimina o problema de sincronismo citado, sendo que o único tipo de sincronismo necessário é o de início de Iteração Funcional, para que a inversão ao acesso às cópias das variáveis de estado possa ser efetuada.

Neste trabalho, a duplicação das variáveis de estado é feita, funcionalmente, como descrito a seguir:

- Seja Y uma variável de estado. Sua declaração será:

```
var Y : array[boolean] of (tipo);
    pg: boolean;
```

- A utilização da variável Y na Iteração Funcional n será:

```
pg := not pg;
variável Y usada como entrada: Y[pg];
variável Y usada como saída:   Y[not pg];
```

A descrição do Computador Incremental feita neste capítulo caracteriza o modo de processamento paralelo nessa máquina. A linguagem L deverá prover recursos que facilitem ao programador a

tarefa de descrever apropriadamente os problemas a serem resolvidos em paralelo pelo Computador Incremental.

CAPÍTULO 3

LINGUAGENS PARA SIMULAÇÃO DE SISTEMAS CONTÍNUOS

Entre as linguagens de programação disponíveis, as mais apropriadas para descrição de problemas a serem executados pelo Computador Incremental são as linguagens de simulação de sistemas contínuos.

Este capítulo descreve as principais características de linguagens utilizadas para simulação de sistemas contínuos, cujas construções servirão como base para a definição da linguagem L.

3.1 - VISÃO GERAL

Em princípio, uma linguagem de uso geral pode ser usada para resolver problemas de simulação de sistemas contínuos, porém, isso obriga o usuário a desenvolver e depurar suas próprias rotinas, a serem utilizadas para realizar a simulação (Speckhart and Green, 1976).

As linguagens de simulação de sistemas contínuos provêm um conjunto de subrotinas que facilitam a implementação de simulações, já que livram o usuário das tarefas de escrever e depurar essas subrotinas.

Essas linguagens são usadas para simular sistemas físicos ou abstratos que são definidos ou modelados através de sistemas de equações matemáticas, permitindo que se resolvam equações num determinado intervalo de suas variáveis independentes, após a definição dos parâmetros que caracterizam o sistema ou modelo.

Em geral, os compiladores das linguagens de simulação de sistemas contínuos são escritos com base em compiladores de linguagens de uso geral, ou seja, eles traduzem comandos da linguagem de simulação em comandos de uma linguagem de alto nível. Efetuada esta tradução, esses comandos passam pelo compilador da linguagem de alto nível, gerando o programa objeto final, num esquema semelhante ao exemplo mostrado na Figura 3.1.

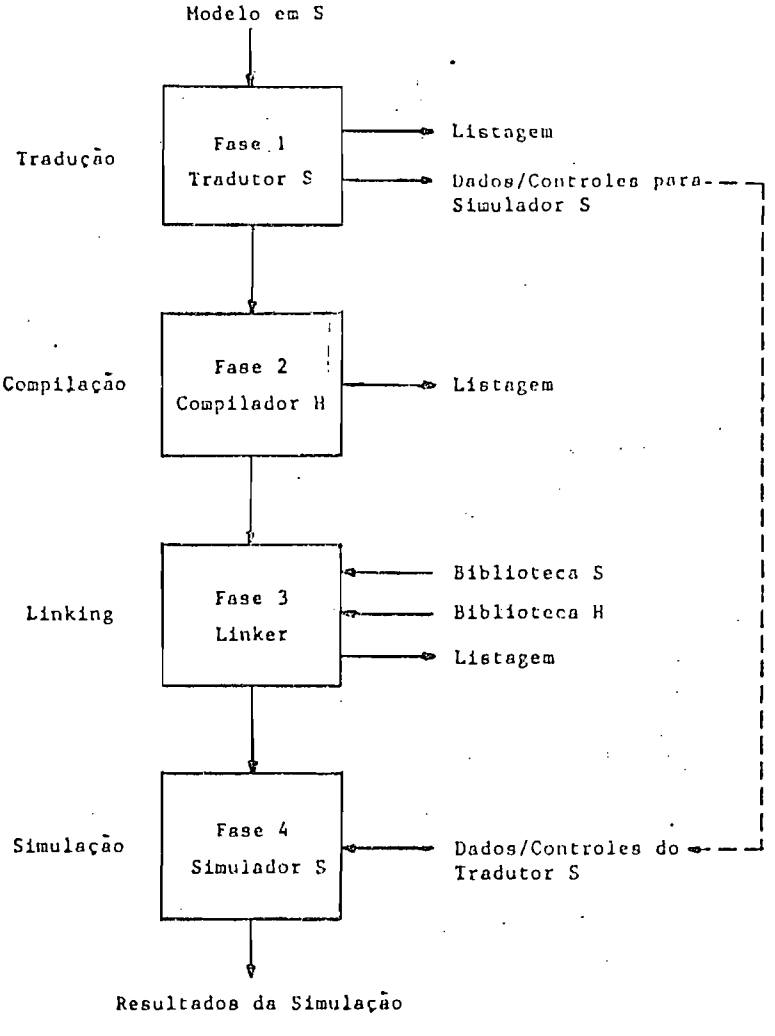


Fig. 3.1 - Compilação de um programa escrito em linguagem de simulação, baseado em Garzia (1986), p.5

Neste exemplo, a linguagem de simulação foi chamada linguagem S, e a linguagem de alto nível, para a qual os comandos escritos na Linguagem S serão traduzidos, linguagem H.

O usuário descreve o modelo a ser simulado através de comandos da linguagem S. A execução do programa passa por 4 fases:

Fase 1- Tradução: Fase em que o programa escrito na linguagem S é convertido para um programa na linguagem H. Esta fase gera também a listagem do programa do usuário, com indicações de possíveis erros sintáticos e um arquivo de dados/controles a serem utilizados na fase de Simulação;

Fase 2- Compilação: Fase em que o resultado da tradução feita na fase anterior é compilado, gerando o código objeto do programa de simulação. Essa fase é realizada com o uso do compilador da linguagem H, disponível no sistema. Opcionalmente, uma listagem do processo de compilação também poderá ser gerada.

Fase 3- Linking: Fase em que o código objeto gerado na fase anterior é ligado às bibliotecas da linguagem S e da linguagem H, gerando o código executável. Opcionalmente, uma listagem do processo de ligação também poderá ser gerada nesta fase.

Fase 4- Simulação: Fase em que a simulação é realizada, com o código executável, gerado na fase anterior, associado ao arquivo de dados/controles gerados na Fase 1.

A implementação da linguagem S através da tradução de comandos dessa linguagem em comandos da linguagem H tem como vantagens: 1) a facilidade de implementação, visto que o compilador para a linguagem H está disponível, e o processo de tradução é

consideravelmente mais simples que o processo de compilação; 2) a possibilidade de prover o usuário de um conjunto maior de comandos, uma vez que os comandos da linguagem H, em geral, podem também ser utilizados no programa escrito na linguagem S; 3) a disponibilidade de uma biblioteca de rotinas da linguagem H. As desvantagens desse esquema são: 1) os programas que vão passar pelo compilador da linguagem H já passaram pelo processo de tradução e, portanto, não devem conter erros de sintaxe, porém o compilador da linguagem H faz a verificação de erros sintáticos, o que resulta em maior tempo total de compilação; 2) como o código executável é aquele gerado através do compilador da linguagem H, há dificuldade em associar erros em tempo de execução ao programa fonte, escrito na linguagem S.

A seguir serão descritas brevemente as principais características das linguagens de simulação CSSL e CSMP.

3.2 - LINGUAGEM CSSL ("Continuous System Simulation Language")

A linguagem CSSL, apresentada em 1968 (Strauss ed. 1968), constitui um padrão para as linguagens de simulação que a seguiram. Na realidade, CSSL foi uma síntese e um direcionamento das diversas linguagens para simulação existentes até aquela data.

Um programa de simulação em CSSL é definido segundo a estrutura mostrada na Figura 3.2.

Essa estrutura mostra a divisão do programa em três regiões: INITIAL, DYNAMIC e TERMINAL.

Na região INITIAL são feitos os cálculos, operações de E/S, e procedimentos de inicialização que devem ser efetuados antes do início da simulação. Além disso, o programa pode chamar um interpretador (rotina intrínseca), que permite ao usuário interagir com o programa, fazendo, dentre outras coisas: ajuste de valores de

variáveis declaradas no programa; leitura do valor das variáveis; controle de parâmetros do algoritmo de integração; operações aritméticas simples para facilitar as mudanças dos parâmetros do programa, com base em resultados passados; controle do início e término da execução do programa de simulação.

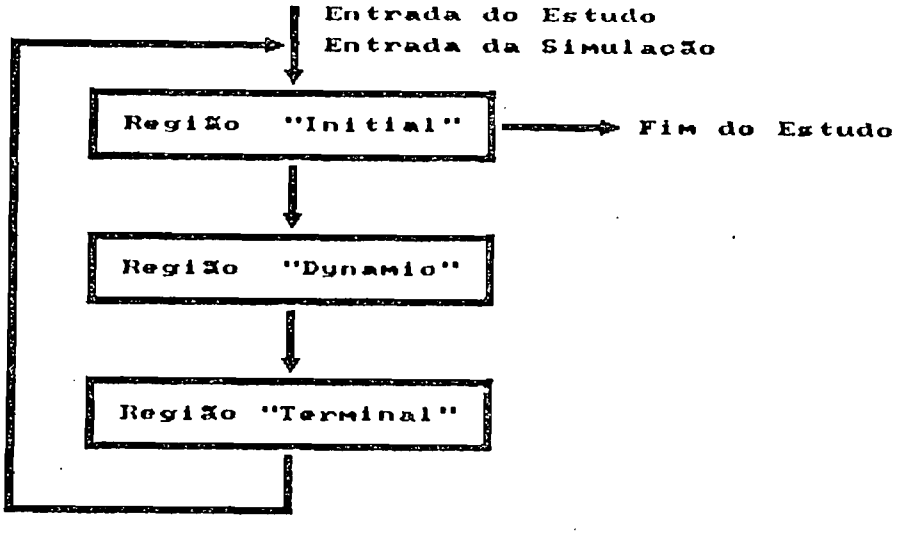


Fig. 3.2 - Estrutura de um programa de simulação.

FONTE: Strauss ed. (1968) p.285

Na região DYNAMIC são efetuados os cálculos relativos à solução do modelo descrito pelo programa de simulação. Esse cálculo é repetido para cada valor de discretização da variável independente. A região DYNAMIC é executada iterativamente até que a condição de término da simulação seja verificada, caso em que a região TERMINAL é executada.

Na região TERMINAL são efetuados os cálculos associados ao fim da simulação.

Por "simulação", entende-se a execução de um programa de simulação apenas uma vez. Por "estudo", entende-se um conjunto de

execuções consecutivas de um programa de simulação (variando-se os parâmetros ou entrada e saída).

Um sistema de equações diferenciais, como o descrito abaixo:

```
dr/dt = 2*r - a*r*f        r(0)=r0
df/dt = -f + a*r*f         f(0)=f0

a = 0.001
r0 = 300
f0 = 150
```

pode ser simulado através de CSSL com o seguinte programa:

```
PROGRAM
  COMMENT SIMULAÇÃO DE UM ECOSSISTEMA SIMPLIFICADO EM CSSL
  DATA [A=0.01; R0=300; F0=150]
  CINTERVAL [,0.01]
  INITIAL
    INTERPRETER
  END
  DYNAMIC
    R = INTEG [2*R - A*R*F, R0]
    F = INTEG [-F + A*R*F, F0]
  END
  TERMINAL
    PRINT [R, F]
  END
END
GO;
STOP;
```

Este exemplo mostra a chamada da rotina INTEG, utilizada para integração de equações. O primeiro parâmetro da rotina INTEG define a expressão da equação a ser integrada e o segundo parâmetro define o valor inicial da variável dependente. Na região INITIAL é chamada a rotina do interpretador, que permite a interação do usuário com o programa. As variáveis a serem listadas são especificadas através do comando PRINT. O período de listagem dessas variáveis é determinado através do comando CINTERVAL, e este é também o período usado para incremento da variável independente, se outro período não for especificado.

Por este exemplo pode ser verificada a facilidade de se descrever um modelo usando a linguagem CSSL em contraste com a utilização de uma linguagem de programação de uso geral. As rotinas que realizam o processo de integração das equações são pré-programadas, juntamente com a estrutura de controle da execução da simulação.

3.3 - LINGUAGEM CSMP ("Continuous System Modeling Program")

A linguagem CSMP foi desenvolvida pela IBM, em resposta à necessidade de uma linguagem de programação para simulação de sistemas contínuos que não exigisse conhecimento extensivo de FORTRAN e de técnicas numéricas de seus usuários (Speckhart and Green, 1976).

A implementação de CSMP é baseada no esquema da Figura 3.1, sendo que a linguagem de alto nível utilizada é FORTRAN. A linguagem CSMP permite, por isso, que o usuário utilize em seu programa construções da linguagem FORTRAN.

A vantagem da utilização de CSMP é a facilidade de se descrever um problema nessa linguagem e o fato de existir um grande número de funções pré-programadas, além das próprias funções intrínsecas da linguagem FORTRAN, que também podem ser utilizadas.

O sistema de equações diferencias citado no item 3.1 seria programado em CSMP através do programa a seguir:

```
LABEL SIMULAÇÃO DE UM ECOSSISTEMA SIMPLIFICADO EM CSMP
*
*
INITIAL
  CONSTANT A = 0.01, R0 = 300, F0 = 150
DYNAMIC
  T = 2*R - A*R*F
  U = -F + A*R*F
  R = INTGRL (R0, T)
  F = INTGRL (F0, U)
TERMINAL
  TIMER FINTIM = 10.0, PRDEL = 0.1
  PRINT R, F
*
*
END
STOP
ENDJOB
```

A estrutura da linguagem CSMP é semelhante à da linguagem CSSL. Um programa é composto pelas regiões INITIAL, DYNAMIC e TERMINAL. Na região INITIAL são feitos os cálculos que precisam ser executados apenas uma vez, antes do início da simulação. Na região DYNAMIC são realizados, iterativamente, os cálculos das equações que descrevem o modelo simulado, a cada valor de discretização da variável independente. Na região TERMINAL são efetuados os cálculos e operações de E/S que devem ser executados no final da simulação.

A integração de equações é realizada através da chamada da função intrínseca INTGRL, cujo primeiro parâmetro é a condição inicial da variável dependente e segundo parâmetro é a expressão que

define a equação. O comando TIMER especifica neste caso a condição de término do programa (FINTIM) e o período de listagem de resultados (PRDEL). Outros parâmetros podem ser modificados, tais como, o incremento da variável independente, o método usado para integração, ou o intervalo para plotagem das variáveis. Para listagem de variáveis é usado o comando PRINT, com a relação das variáveis a serem listadas.

A simulação de um sistema contínuo em CSMP é bastante simples, devido às facilidades que a linguagem fornece. Para escrever um programa que simula um sistema contínuo, o usuário deve, em primeiro lugar, transformar as equações diferenciais que descrevem o referido sistema em um sistema de equações diferenciais de primeira ordem. Este sistema de equações diferenciais comporá o segmento DYNAMIC do programa. Usando as construções fornecidas pela linguagem, o usuário fornece os parâmetros, condições iniciais e condições para execução da simulação. Uma listagem e/ou plotagem das variáveis de interesse pode ser pedida nos intervalos específicos determinados pelo usuário.

A utilidade das linguagens de simulação de sistemas contínuos está na facilidade que provêm ao usuário para descrever seu modelo. Além disso, por ter as rotinas de cálculo pré-programadas, essas linguagens permitem um desenvolvimento rápido e seguro de um programa para simulação.

A linguagem proposta neste trabalho tem sua estrutura geral baseada na estrutura dessas linguagens, acrescida de construções que permitem a exploração adequada da arquitetura e do modo de funcionamento do Computador Incremental.

CAPÍTULO 4

LINGUAGEM L

Neste capítulo é descrita a linguagem L. Inicialmente são enumerados os requisitos básicos dessa linguagem; em seguida é feita a descrição de suas estruturas.

4.1 - REQUISITOS BÁSICOS DA LINGUAGEM

A linguagem L deverá ter construções específicas que facilitem a programação dos problemas típicos a serem executados no Computador Incremental. Esses problemas típicos pertencem à família de problemas que podem ser decompostos em tarefas paralelas, que são executadas iterativamente, de maneira síncrona, exigindo comunicação entre as tarefas em toda iteração.

Os seguintes termos, introduzidos no capítulo 2 são utilizados para descrição da linguagem L:

a) **Iteração Incremental**: uma Iteração Incremental corresponde à execução de um Período Incremental (Fig. 2.2).

b) **Iteração Funcional**: uma Iteração Funcional corresponde à execução, uma vez, das tarefas que caracterizam o problema que está sendo resolvido pelo Computador Incremental. A execução de uma Iteração Funcional implica na atualização, pelo menos intermediária, de todas as variáveis de estado do problema. Uma Iteração Funcional é composta por uma ou mais Iterações Incrementais.

c) **Iteração Funcional Completa:** uma Iteração Funcional Completa corresponde à execução de uma iteração no processo de solução de um problema pelo Computador Incremental. Uma Iteração Funcional Completa implica, necessariamente, na atualização de todas as variáveis de estado do problema, e pode ser composta por uma ou mais Iterações Funcionais.

d) **Método Funcional:** é o método utilizado para solução iterativa do problema. O Método Funcional define o número de Iterações Funcionais necessário para execução de uma Iteração Funcional Completa. O Método Funcional é denominado Método Funcional Monofase quando uma Iteração Funcional Completa é realizada com a execução de apenas uma Iteração Funcional e é denominado Método Funcional Multifase quando uma Iteração Funcional Completa é realizada com a execução de mais de uma Iteração Funcional.

As principais características da arquitetura do Computador Incremental que influenciarão no projeto da linguagem L são:

1) **Processamento:** o processamento é iniciado simultaneamente em todos os processadores, em cada Iteração Incremental. Isso obriga a uma distribuição equitativa de carga de processamento entre os processadores, pois a taxa de iterações é limitada pelo processador programado com maior volume de cálculo. Se este processador tiver um tempo de cálculo muito maior que os demais, ele comprometerá o desempenho global da máquina.

2) **Comunicação:** a comunicação entre os processadores é realizada por Difusão ("broadcast"), através de Memória de Comunicação, em cada Iteração Incremental. Essa característica faz com que todos os processadores tenham

acesso aos resultados obtidos pelos demais em cada Iteração Incremental, sem que haja aumento considerável no tempo de comunicação entre os mesmos.

Os requisitos que nortearão o projeto da linguagem L são introduzidos e analisados a seguir.

4.1.1 - FACILIDADES DE PROGRAMAÇÃO DE CADA PROCESSADOR

A linguagem L deverá prover construções que permitam ao usuário, na descrição de um problema, alocar parcelas do cálculo para cada um dos processadores que estão envolvidos na solução do problema.

A paralelização do cálculo deverá ser feita pelo programador, explicitando quais parcelas do cálculo deverão ser executadas por cada processador.

A comunicação entre os processadores, bem como a sincronização dos processadores para início de uma Iteração Incremental deverão ficar transparentes ao usuário.

4.1.2 - BALANCEAMENTO DE CARGA DE PROCESSAMENTO

A solução de um sistema de equações poderá, naturalmente, apresentar problemas quanto à distribuição da carga de processamento nos processadores.

Considere o sistema de equações diferenciais descrito abaixo, que apresenta anotado, à esquerda de cada equação, o número de operações normalizadas (ops.) necessário para cálculo da expressão que define cada derivada, considerando-se que o cálculo seja efetuado em um coprocessador aritmético INTEL 80387 (Intel, 1987), com

representação numérica de 32 bits e ponto flutuante. Neste coprocessador, o tempo médio gasto para cálculo de funções seno ou coseno é, em média, 16 vezes maior que o tempo gasto em operações de soma ou multiplicação. A normalização é feita considerando-se que operações de soma ou multiplicação correspondem a uma operação normalizada, enquanto que funções seno ou coseno correspondem a 16 operações normalizadas.

```
 0 ops.: y1' = y3                          eq. 1
 0 ops.: y2' = y4                          eq. 2
34 ops.: y3' = sen(a * y1) - cos(y4)       eq. 3
34 ops.: y4' = sen(b * y2) - cos(y3)       eq. 4
```

Se alocarmos um processador para o cálculo de cada uma das equações, vamos ter um forte desbalanceamento de tempo de cálculo dos processadores 1 (eq. 1) e 2 (eq. 2) e dos processadores 3 (eq. 3) e 4 (eq. 4).

Uma maneira de minimizar este problema consiste em particionar de forma equitativa (balanceada), o cálculo das equações dos processadores 3 e 4 com aquelas dos processadores 1 e 2.

Assim poderíamos ter, por exemplo, a seguinte distribuição de cálculo:

```
17 ops.: y1' = y3;  x1 = sen(a * y1)
17 ops.: y2' = y4;  x2 = sen(b * y2)
17 ops.: y3' = x1 - cos(y4)
17 ops.: y4' = x2 - cos(y3)
```

Neste caso, apesar do inconveniente da atribuição de valores inicias para as variáveis x1 e x2 antes do início do processamento (o cálculo é iniciado simultaneamente em todas as unidades), não há prejuízo da dependência temporal do cálculo das

equações, já que o valor dessas partições intermediárias a serem usadas no passo atual refletem os valores das variáveis de estado já calculadas no passo anterior.

Há sistemas de equações em que não é possível fazer um balanceamento razoável na carga de processamento, como, por exemplo, o sistema de equações descrito a seguir:

```
 1 op. : y1' = a * y2                              eq. 1
 1 op. : y2' = b + y3                              eq. 2
35 ops.: y3' = sen(y1 + y2) + cos(y3 + y4)         eq. 3
35 ops.: y4' = sen(y1 + y3) + cos(y1 + y4)         eq. 4
```

Não é possível obter particionamentos diretos das equações que possam ser calculados em outros processadores, sem interferir na dependência temporal do cálculo.

Neste caso, a solução considerada consiste em criar fases intermediárias, com o propósito de possibilitar o balanceamento do cálculo das equações por cada processador. A adoção deste critério resulta nas seguintes fases de cálculo:

Fase 1:

```
17 ops.: x1 = sen(y1 + y2)
17 ops.: x2 = cos(y3 + y4)
17 ops.: x3 = sen(y1 + y3)
17 ops.: x4 = cos(y1 + y4)
```

Fase 2:

```
1 op.: y1' = a * y2
1 op.: y2' = b + y3
1 op.: y3' = x1 + x2
1 op.: y4' = x3 + x4
```

Na fase 1 são calculados os valores de parcelas das equações que serão utilizados na fase 2 para o cálculo completo do sistema de equações.

As fases 1 e 2 descrevem, cada uma delas, os cálculos que devem ser efetuados em uma Iteração Incremental. As fases 1 e 2 juntas descrevem os cálculos que devem ser efetuados em uma Iteração Funcional.

Quanto maior o tempo total de execução do problema, mais importante é a distribuição equitativa das tarefas entre os processadores.

4.1.3 - SOLUÇÕES EM TEMPO REAL

O Computador Incremental pode ser utilizado para cálculos em tempo real, por exemplo, fazendo parte de uma malha de controle, ou ainda, no processamento de sinais.

A linguagem L deverá prover recursos de programação que possibilitem ao usuário a solução de problemas em tempo real.

4.2 - DESCRIÇÃO DA LINGUAGEM L

A sintaxe da linguagem L é descrita no apêndice A, com a utilização de diagramas de sintaxe. Os diagramas de sintaxe são convenientes para descrição das gramáticas de linguagens, pois facilitam a dedução da linguagem gerada, bem como a compreensão de alguns métodos de análise sintática para essas linguagens. A definição formal de gramáticas, linguagens, e as regras de construção de diagramas de sintaxe podem ser encontradas, entre outros, em (Wirth, 1976), (Kowaltowski, 1979) e (Setzer e Melo, 1983).

Neste item é descrita a semântica das construções da Linguagem L. Os comandos básicos da linguagem L são os mesmos da linguagem Pascal (Wirth, 1971). A descrição feita neste item se concentra nas construções particulares para a linguagem L.

É importante notar que um programa em L é escrito para o CT do Computador Incremental.

4.2.1 - DADOS E DECLARAÇÕES

Os tipos de dados básicos permitidos na linguagem L são:

a) **integer:** um valor do tipo integer pertence ao subconjunto dos números inteiros, representável em uma palavra de memória do computador.

b) **long:** um valor do tipo long pertence ao subconjunto dos números inteiros, e contém, necessariamente, o subconjunto integer.

c) **real:** um valor do tipo real pertence ao subconjunto dos números reais, representável em, no mínimo, 32 bits.

d) **double:** um valor do tipo double pertence ao subconjunto dos números reais, representável em duas vezes o número de bits usados para representar um valor do tipo real.

e) **char:** um valor do tipo char pertence ao conjunto de caracteres ASCII.

f) **boolean:** um valor do tipo boolean assume um dos valores, verdadeiro ou falso, representados pelas palavras reservadas **true ou false**, respectivamente.

Os dados básicos podem ser agrupados em **arranjos.** Um arranjo em L pode ter qualquer dimensão e seus elementos são sempre do tipo básico.

Os **identificadores** devem ser declarados antes de serem utilizados.

A declaração de identificadores de constantes dos tipos básicos é feita como na linguagem Pascal. Por exemplo:

```
const K = 20;
```

A declaração de identificadores de variáveis é feita também como na linguagem Pascal. Exemplo:

```
var x  : real;
    i,j: integer;
```

Para comunicação entre processadores, foram criados dois tipos de variáveis, que, quando declaradas, são alocadas na Memória de Comunicação de todos os processadores:

a) **variáveis 'State'**: são variáveis utilizadas para armazenar os resultados de cada Iteração Funcional. Essas variáveis são duplicadas na Memória de Comunicação, para permitir a solução iterativa de sistemas, como discutido no capítulo 2, ou seja, elas permitem a manipulação do estado corrente da variável, ao mesmo tempo em que é gerado o seu próximo estado.

b) **variáveis 'Global'**: são variáveis utilizadas para controle do sequenciamento do cálculo, ou para armazenar parcelas intermediárias do cálculo.

Variáveis do tipo State e Global são funcionalmente locais à um processador para leitura e globais para escrita, ou seja, a leitura do valor de uma variável desses tipos é feita na sua Memória de Comunicação, enquanto que a atualização é feita na Memória de Comunicação de todos os processadores, por Difusão. Este fato otimiza o tempo de comunicação entre processadores, pois em cada Iteração Incremental são feitos muitos acessos para leitura de variáveis do tipo State ou Global, enquanto é feito apenas um acesso para atualização.

A declaração de identificadores de variáveis do tipo State e Global é feita utilizando a mesma sintaxe de declaração de variáveis em Pascal, substituindo-se a palavra reservada "var" por "state" ou "global", como descrito a seguir:

```
state y1,y2: real;

global m,t: integer;
```

As variáveis do tipo State, embora sejam duplicadas, são utilizadas como variáveis comuns nos programas. O compilador é responsável para, a cada nova Iteração Funcional, inverter a utilização das cópias dessas variáveis, ou seja, na iteração n+1, a cópia que era utilizada para leitura na iteração n é usada para escrita, e vice-versa.

A atualização de variáveis do tipo Global e State por Difusão pode ser feita depois do final da Fase Funcional de cada processador (Figura 2.2), em uma Fase de Comunicação explícita, ou logo que a variável receber uma atribuição, dependendo das facilidades específicas do hardware em que a linguagem L for implementada. De qualquer forma, funcionalmente o efeito é o mesmo, pois um processador poderá utilizar a variável atualizada por outro processador na

Iteração Incremental corrente, somente na próxima Iteração Incremental.

4.2.2 - MODIFICADOR "@"

As referências a variáveis do tipo Global e State estão sujeitas a um modificador "@". A utilização desse modificador tem os efeitos descritos a seguir:

a) **Escrita:** a utilização de @X, onde X é uma variável do tipo State ou Global, no lado esquerdo de um comando de atribuição, faz com que a atualização dessa variável seja local ao processador, ou seja, não ocorre a Difusão. O modificador "@" deve ser utilizado para que resultados intermediários do cálculo, dos quais os outros processadores não precisam tomar conhecimento, não sejam difundidos. A variável X, neste caso, funciona como se fosse local à unidade de processamento.

b) **Leitura:** a utilização de @X, onde X é uma variável do tipo State, faz com que a cópia utilizada para leitura do valor da variável seja a mesma que está sendo utilizada para atualização nesta iteração. A utilização do modificador "@" em operações de leitura em variáveis do tipo Global não tem efeito. O modificador "@" deve ser utilizado para fazer referência à cópia da variável que foi atualizada nesta iteração, mas que só estará disponível para leitura na próxima iteração.

A utilização do modificador "@" em variáveis comuns não tem efeito.

4.2.3 - COMANDOS

Os comandos permitidos na linguagem L formam um subconjunto dos comandos da linguagem PASCAL acrescido de comandos direcionados para o controle da execução de problemas no Computador Incremental, compondo a seguinte relação:

a) Comando de **Atribuição**;

b) Comando de **Invocação de Procedimento**;

c) Comando "**If-Then-Else**";

d) Comando "**While**";

e) Comando "**For**";

f) Comandos de **Controle de Execução** (Terminate e Restart).

Os comandos de controle de execução serão descritos no item 4.2.9. No apêndice A são fornecidos os diagramas de sintaxe dos comandos da linguagem L.

4.2.4 - PROCESSOS EM L

Um dos objetivos da definição da linguagem L é que seus programas permitam também a solução de problemas em tempo real. Para atingir esse objetivo, foram criados 3 tipos de processo em L: (Os nomes usados para os processos foram os definidos em "Linguagem Orientada para Controle", (Siebert and Winning, 1987)).

a) **Processos "Repeat"**: são processos cuja execução é repetida continuamente num período de tempo definido pelo usuário.

b) **Processos "Interrupt"**: são processos cuja execução está condicionada à ocorrência de uma interrupção.

c) **Processos "Background"**: são processos que são executados quando não há processos do tipo "repeat" ou "interrupt" para serem executados.

Cada um dos processos tem uma estrutura parecida com a estrutura de um programa de simulação em CSSL (Strauss ed., 1968), acrescida de uma região chamada **"Communication"**, onde são agrupados os comandos que devem ser executados apenas nos intervalos de comunicação definidos pelo usuário (Crosbie et all, 1985). Por exemplo, um processo típico do tipo Repeat tem a forma descrita a seguir:

```
repeat "intervalo" " nome";

  declarações

  initial
    comandos da região Initial
  end;   (* initial *)

  dynamic
    comandos da região Dynamic
  end;   (* dynamic *)

  communication
    comandos da região Communication
  end;   (* communication *)

  terminal
    comandos da região Terminal
  end;   (* terminal *)

end;   (* repeat *)
```

O parâmetro "intervalo" indica o período de ativação da execução do processo.

As declarações feitas dentro de um processo são locais a esse processo.

A **região Initial** é executada apenas uma vez, quando a execução do processo Repeat é iniciada.

A **região Dynamic** é executada periodicamente, num período definido pelo parâmetro "intervalo".

A **região Communication** é executada nos intervalos de comunicação definidos pelo usuário.

A **região Terminal** é executada quando a execução da região Dynamic terminar.

Os processos Interrupt e Background têm estrutura semelhante, apenas o modo de execução da região Dynamic de cada um deles é diferente. Um processo do tipo Interrupt tem sua região Dynamic executada sempre que ocorre a interrupção à qual ele está associado. Um processo do tipo Background tem sua região Dynamic executada sempre que não houver processos Repeat ou Interrupt em execução. Os exemplos a seguir ilustram a estrutura dos processos Interrupt e Background:

Exemplo de processo Interrupt:

```
interrupt "número" "nome";
  declarações
  initial
    .
    .
  dynamic
    .
    .
  communication
    .
    .
  terminal
    .
    .
end;
```

O parâmetro **"número"** indica à qual interrupção a execução do processo Interrupt está associada.

Exemplo de processo Background:

```
background "nome";
  declarações
  initial
    .
    .
  dynamic
    .
    .
  communication
    .
    .
  terminal
    .
    .
end;
```

4.2.5 - PROGRAMAS EM L

Um programa típico em L tem a estrutura descrita a seguir:

```
declaração
declaração de procedimento
processo em L;
  .
  .
processo em L;
```

Os identificadores declarados a nível de programa (identificadores globais), são reconhecidos em todos os processos.

A ativação da execução dos processos vai ser feita por ordem de declaração no programa.

4.2.6 - EXECUÇÃO NOS ADs

Cada comando da região Dynamic de um processo pode ser alocado para execução em um determinado processador. A construção que permite que isso seja feito é descrita a seguir:

Rótulo da Unidade: Comando;

O usuário rotula através da palavra reservada AD e de um número inteiro a unidade em que será executado o comando. Comandos não rotulados são executados pelo CT. O trecho de programa a seguir ilustra a aplicação deste conceito de rotulação de comandos.

```
dynamic
  AD 1: r := integ (2 * r - a * r * f);
  AD 2: f := integ (-f + a * r * f);
        x := r + f;
end;   (* dynamic *)
```

No exemplo citado, o AD 1 realiza o cálculo da variável dependente r, o AD 2 realiza o cálculo da variável dependente f e o CT realiza o cálculo da variável x.

Todos os processadores iniciam a execução de seus comandos ao mesmo tempo (em paralelo). Os comandos alocados a um determinado processador do Computador Incremental são executados serialmente.

As variáveis que podem ser referenciadas nos comandos dos ADs são as dos tipos Global e State, pois o escopo das variáveis comuns é limitado ao CT.

As constantes têm escopo global a todos os processadores e, portanto, podem ser utilizadas pelos comandos alocados nos ADs.

4.2.7 - DISTRIBUIÇÃO DE CARGA DE PROCESSAMENTO

Os comandos de uma região Dynamic devem descrever os cálculos a serem efetuados em uma Iteração Funcional.

Para permitir o balanceamento da carga de processamento nos processadores do Computador Incremental, uma Iteração Funcional pode ser constituída de várias Iterações Incrementais. Na linguagem L, cada Iteração Incremental é uma "fase" da Iteração Funcional, e pode

ser definida utilizando a declaração formal **phase "número"**, tal como no exemplo seguinte:

```
dynamic

  phase 1;
    rótulo da unidade: comando;
      .
      .
    rótulo da unidade: comando;
  end;   (* phase 1 *)

  .
  .

  phase n;
    rótulo da unidade: comando;
      .
      .
    rótulo da unidade: comando;
  end;   (* phase n *)

end;   (* dynamic *)
```

Essa construção permite o uso do mecanismo proposto no item 4.1.2 que permite uma distribuição mais equitativa da carga de processamento entre os ADs do Computador Incremental.

4.2.8 - COMUNICAÇÃO E SINCRONIZAÇÃO ENTRE PROCESSOS

Processos podem se comunicar através de variáveis globais (que são declaradas a nível de programa).

A sincronizaçao entre os processos é feita através de semáforos, com a utilização das primitivas WAIT e SIGNAL.

A declaração de um identificador de variável do tipo semáforo só é permitida a nível de programa, e tem a sintaxe abaixo:

```
var s: semaphore;
```

A declaração de um identificador de variável do tipo semáforo associa a esse identificador um contador que armazena o número de sinais enviados e não utilizados e uma fila onde são suspensos os processos aguardando sinal neste semáforo.

Formalmente, as operações WAIT e SIGNAL numa variável s do tipo semáforo são definidas como segue: (ref.: Peterson e Silberchatz (1985)).

WAIT:

```
s.Contador := s.Contador - 1;
se s.Contador < 0
então "coloca processo na fila de espera
       e altera seu estado para pendente"
```

SIGNAL:

```
s.Contador := s.Contador + 1;
se s.Contador <= 0
então "retira um processo da fila de espera
       e altera seu estado para pronto"
```

A utilização primordial de ferramentas de sincronização no contexto da linguagem L é a sinalização da ocorrência de eventos entre processos, portanto, a utilização de semáforos como ferramenta de sincronização deve cobrir as necessidades de sincronização dos programas em linguagem L.

4.2.9 - CONTROLE DE EXECUÇÃO

A execução de um processo em L é controlada da seguinte forma:

1) **Nas regiões INITIAL:**

Os comandos da região INITIAL são executados apenas uma vez, quando a execução do processo é iniciada.

2) **Nas regiões DYNAMIC:**

Os comandos da região Dynamic de um processo em L descrevem os cálculos a serem executados em uma Iteração Funcional.

Uma Iteração Funcional Completa pode ser composta de mais de uma Iteração Funcional, conforme o Método Funcional que estiver sendo utilizado.

Para controlar a execução de uma Iteração Funcional Completa, as seguintes **variáveis de controle** pré-declaradas são utilizadas:

a) **Method:** variável que designa o Método Funcional que está sendo utilizado para solução do problema.

b) **FM:** variável que designa qual fase do Método Funcional Multifase esta sendo executada.

c) **TFC:** variável booleana, indica quando uma Iteração Funcional Completa foi realizada.

d) **IFlag:** variável booleana, indica a primeira execução do Método Funcional.

A variável **Method** deve ser iniciada na região Initial, conforme o Método Funcional a ser utilizado. A variável **FM** é controlada automaticamente, de maneira transparente ao usuário, durante a execução da região Dynamic, conforme o Método Funcional utilizado, para indicar qual fase do Método Funcional Multifase deve ser executada. A variável **TFC** também é controlada automaticamente, de maneira transparente ao usuário, durante a execução da região Dynamic, conforme o Método Funcional utilizado, para controlar a execução de uma Iteração Funcional Completa. A variável **IFlag** tem o valor verdadeiro apenas na primeira Iteração Funcional, para permitir a iniciação dos Métodos Funcionais.

Para controlar a execução de uma Iteração Funcional, as seguintes variáveis de controle pré-declaradas são utilizadas:

a) **dt:** valor do incremento da variável independente. Se não for especificada pelo usuário, ela assume o valor padrão igual a 0,1;

b) **t:** variável independente, incrementada automaticamente a cada Iteração Funcional, conforme o método de integração que está sendo utilizado. Valor inicial padrão: 0;

As variáveis de controle **t** e **dt** têm relevância na determinação dos pontos de execução dos comandos da região Communication e na descrição e controle do cálculo a ser realizado na Iteração Funcional. Essas variáveis de controle são atualizadas automaticamente a cada Iteração Funcional, de acordo com o Método Funcional utilizado.

Todas as variáveis de controle são locais aos processos.

O término da execução da região Dynamic deve ser determinado com a utilização do **comando de controle** descrito a seguir:

terminate EXPRESSÃO BOOLEANA;

Quando a expressão booleana for verdadeira, a execução da região Dynamic é encerrada.

A expressão booleana do comando de controle **terminate** pode envolver, além das variáveis declaradas pelo usuário, as variáveis de controle pré-declaradas. Se o comando de controle Terminate não aparecer numa região Dynamic, sua execução será terminada quando a condição (t ▷ 10) for verdadeira.

A execução de uma **iteração da região Dynamic** envolve, além da execução de uma Iteração Funcional Completa, a ativação dos comandos da região Communication quando for atingido um ponto de comunicação definido pelo usuário e a verificação da condição de término da execução da região Dynamic, dada pelo comando de controle Terminate.

O **modo de execução** das iterações da região Dynamic é que caracteriza os processos Repeat, Background e Interrupt. Assim, processos do tipo Repeat têm uma iteração da região Dynamic executada a cada período definido pelo parâmetro **"intervalo"** de seu cabeçalho; um processo do tipo Interrupt tem uma iteração da região Dynamic executada toda vez que surgir a interrupção à qual ele estiver associado; um processo do tipo Background tem uma iteração da região Dynamic executada sempre que o processador lhe for concedido.

3) Regiões COMMUNICATION:

A cada iteração da região Dynamic é verificado se a execução do problema atingiu um ponto de ativação da região Communication e, em caso positivo, os comandos dessa região são executados.

Para controle do período de ativação das regiões Communication, existe uma variável de controle pré-declarada, denominada **cinterval**, com valor padrão igual a 0,1. O usuário pode modificar o valor dessa variável segundo sua conveniência na região Initial.

Por exemplo, se 1) o Método Funcional que estiver sendo utilizado para solução de um sistema de equações diferenciais for o método Runge Kutta clássico, 2) a variável cinterval tiver valor 0,1 e 3) a variável t for iniciada com valor 0, então os comandos da região Communication serão executados em t=0,1, t=0,2, t=0,3, etc. Se a variável dt tiver valor 0,05, isso significa que são realizadas duas Iterações Funcionais Completas para cada ativação da região Communication, e, como no método de integração Runge Kutta clássico uma Iteração Funcional Completa é composta por quatro Iterações Funcionais, isso significa que são realizadas oito Iterações Funcionais para cada ativação da região Communication.

4) Regiões TERMINAL:

Depois de atendida a condição de término da região Dynamic (dada pelo comando de controle Terminate) de um processo, ocorre a execução de sua região Terminal.

Nesta região, além dos cálculos relativos ao final da execução de sua região Dynamic, o usuário poderá mudar condições iniciais e parâmetros e reinicializar a região Dynamic, através do comando de controle:

restart

O comando de controle "restart" é útil em simulação, pois permite que um mesmo problema seja executado diversas vezes, com alteração nas condições de execução.

4.2.10 - ENTRADA E SAÍDA

A entrada e saída de dados em dispositivos externos será feita através de arquivos.

A sintaxe da declaração de identificadores de variáveis do tipo arquivo é mostrada no Apêndice A. A seguir é mostrado um exemplo de declaração de uma variável do tipo arquivo:

```
var F : file of integer;
```

Os procedimentos para operações com variáveis do tipo arquivo são os procedimentos **read, readln, write** e **writeln** definidos

em Pascal, além do procedimento **print**, que é semelhante ao print encontrado em CSMP, ou seja, permite ao usuário uma simplificação na listagem de variáveis, liberando-o de especificar os formatos de impressão.

4.2.11 - PROCEDIMENTOS

Nesta versão da linguagem L não é permitida a declaração de procedimentos. No entanto, é permitida a inclusão de **procedimentos externos**, na fase de conexão ("linking"). Para isso, o usuário deve declarar no programa o cabeçalho do procedimento externo, como ilustra o exemplo a seguir:

```
external procedure Soma(x,y:real; var z:real);
```

A sintaxe da declaração do cabeçalho de procedimentos (e funções) externos é a utilizada em Pascal para declaração de procedimentos (e funções), porém acrescida da palavra reservada "external".

4.3 - EXEMPLO DE PROGRAMA EM L

O sistema de equações diferenciais descrito no item 4.1.2 e particionado para cálculo em duas fases pode ser programado em L como segue:

```
background "nome";
  const a = 1; b = 2;
  state  y1,y2,y3,y4:  real;
  global x1,x2,x3,x4:  real;
  initial
    y1 := 0;  y2 := 4;
    y3 := 2;  y4 := 0;
  end;
  dynamic
    terminate (t > 5) or (y1 > 2);
    phase 1;
      AD 1: x1 := sin(y1 + y2);
      AD 2: x2 := cos(y3 + y4);
      AD 3: x3 := sin(y1 + y3);
      AD 4: x4 := cos(y1 + y4);
    end;
    phase 2;
      AD 1: y1 := integ(a * y2);
      AD 2: y2 := integ(b + y3);
      AD 3: y3 := integ(x1 + x2);
      AD 4: y4 := integ(x3 + x4);
    end;
  end;    (* dynamic *)
  communication
    print(t,y1,y2,y3,y4);
  end;    (* communication *)
  terminal
    print (t,y1,y2,y3,y4);
  end;    (* terminal *)
end;   (* background *)
```

A execução desse programa faz com que as variáveis de controle **t**, **dt** e **method** assumam seus valores padrão, uma vez que não foram atribuídos valores a essas variáveis na região Initial.

A cada intervalo de comunicação, são listados os valores da variável de controle t e das variáveis de estado do problema. (Neste caso, para determinar o intervalo de comunicação, é utilizado o valor padrão da variável **cinterval**, igual a 0,1).

Quando a condição de término da região Dynamic, dada pelo comando de controle Terminate, for verdadeira, a execução da região Dynamic é encerrada, e a região Terminal é executada.

As estruturas de controle que o Compilador L deve criar para execução dos programas escritos em L serão descritas no capítulo 6.

Note que na execução da fase 2, a equação alocada para cálculo no AD 1 faz uso da variável y2, alocada no AD 2. Como o cálculo é efetuado em paralelo, a variável y2 pode ser atualizada antes de ser utilizada pelo AD 1. Daí a necessidade dessa variável ser do tipo State, pois dessa maneira, o AD 2 atualiza a cópia de y2 que será utilizada pelo AD 1 na próxima iteração, não interferindo no cálculo efetuado no AD 1.

4.4 - ROTINAS INTRÍNSECAS ESPECIAIS

As rotinas intrínsecas da linguagem L (tais como funções trigonométricas, logarítmicas, etc.) são invocadas normalmente em programas escritos em L, obedecendo a sintaxe descrita no apêndice A. Neste trabalho não é apresentada a lista completa das rotinas intrínsecas que deverão estar disponíveis na linguagem L, pois ela depende da implementação da Biblioteca L.

As rotinas que implementam os Métodos Funcionais são **Rotinas Intrínsecas Especiais.** Devido ao fato de que a execução das Rotinas Intrínsecas Especiais tem estreita ligação com o controle da

execução das iterações da região Dynamic, a invocação dessas rotinas pode estar sujeita a restrições.

Por exemplo, a invocação da Rotina Intrínseca Especial "integ(expressão)" só pode ser feita em comandos simples de atribuição, da forma:

variável := integ(expressão);

onde: 1) na definição do parâmetro "expressão" não pode haver invocação da rotina Integ;

2) a variável que recebe a atribuição deve ser do tipo State.

Essas restrições se devem ao fato de que o processo de integração numa determinada Iteração Funcional deve ser feito com o valor das variáveis de estado na Iteração Funcional anterior. A atribuição do resultado da invocação da rotina Integ a uma variável do tipo State faz com que esse resultado só possa ser utilizado na Iteração Funcional seguinte, já que a cópia da variável do tipo State utilizada para leitura é diferente da cópia utilizada para atualização. Como o processo de integração é feito em sincronismo com a execução de uma Iteração Funcional, a invocação da rotina Integ não pode ser feita internamente a comandos "while", "for" ou "if-then-else"; esses comandos podem ser utilizados em uma região Dynamic para definir o parâmetro "expressão" da invocação "integ(expressão)". O valor inicial de todas as variáveis do tipo State deve ser atribuído a essas variáveis na região Initial; esse valor inicial vai ser utilizado para iniciar o algoritmo de integração na primeira chamada da rotina Integ.

A flexibilidade da linguagem L na solução incremental de problemas está ligada à variedade de Métodos Funcionais fornecidos na

implementação da Biblioteca L. Os Métodos Funcionais que realizam a integração de equações diferenciais de primeira ordem, por exemplo, permitem a simulação e implementação em tempo real de sistemas que podem ser descritos por essas equações.

4.5 - DEPURAÇÃO DE PROGRAMAS EM L

A depuração de programas em L é muito mais simples do que a depuração de programas paralelos em geral, pois a forma de comunicação entre os processadores é bem determinada, e o programador não pode interferir nesse processo de comunicação.

Os erros que poderão ocorrer mais frequentemente são aqueles afeitos à familiarização do programador com a forma de processamento do Computador Incremental, refletida na estrutura da região Dynamic da linguagem L.

Um **exemplo de erro** que pode ocorrer facilmente é o mostrado no trecho de programa a seguir:

```
initial
   y1 := 0;
   y2 := 5;
end;
dynamic
 AD 1: begin
         y1 := integ( y1 + y2);
         x := @y1 * @y1;
       end;
 AD 2: y2 := integ(x + y2);
end;
```

Como os comandos da região Dynamic são efetuados em paralelo, a utilização da variável x na equação do AD 2 é imprópria na primeira iteração da região Dynamic, visto que seu valor não foi iniciado ainda. O compilador deverá detectar esse erro, verificando se todas as variáveis utilizadas para "leitura" na região Dynamic foram iniciadas na região Initial.

Para depuração do programa em tempo de execução, o compilador deverá prover uma opção **"debug"**, que permite que comandos sejam compilados só quando essa opção estiver ativa.

Para compilar os comandos utilizados para depuração do programa, o usuário deverá **ativar** a opção "debug" no início do programa, da seguinte maneira:

(*$ DEB $*)

Quando o analisador léxico do compilador L encontrar os caracteres **(*$ D $*)** em uma linha do programa do usuário e a opção "debug" não estiver ativada, ele assume o restante dessa linha como comentário. Se a opção debug estiver ativada, ele despreza os caracteres (*$ D $*).

4.6 - RESTRIÇÕES DE TEMPO REAL

A aplicação ideal da linguagem L na solução de problemas em tempo real envolve a utilização de apenas um processo Repeat ou Interrupt para realizar o cálculo iterativo do problema. Neste caso, desde que o período de ativação da região Dynamic do processo Repeat, ou o período entre duas ocorrências de interrupções que ativam a execução da região Dynamic do processo Interrupt seja suficiente para realização do cálculo de uma iteração da região Dynamic desses

processos, não haverá atrasos na execução iterativa dessas regiões Dynamic.

Processos do tipo Background podem ser utilizados para cálculos afeitos ao problema, mas que não têm restrições de tempo real. Processos do tipo Interrupt podem ser usados para cuidar de condições de exceção na solução do problema.

Quando for utilizado mais de um processo do tipo Repeat, os períodos de ativação desses processos devem ser múltiplos entre si, para que não haja atraso na execução das iterações da região Dynamic desses processos.

4.7 - MEDIDAS DE TEMPO DE EXECUÇÃO

Para que o programador possa fazer um balanceamento ótimo da carga de processamento nos processadores do Computador Incremental, há a necessidade do conhecimento do tempo gasto por cada unidade para realizar uma Fase Incremental (Figura 2.2).

O balanceamento de carga de processamento de cada processador do Computador Incremental tem maior importância quanto mais longo for o tempo total de execução de uma iteração do problema.

Para medir o tempo gasto pelos processadores do Computador Incremental, a linguagem L deve contar com uma função intrínseca para leitura do relógio do sistema. Essa função, denominada R_Time, pode ser chamada em todas as unidades de processamento do Computador Incremental. O uso típico da função de leitura do relógio do sistema é seu disparo antes do início e depois do final do trecho de código cujo tempo de execução se deseja medir, e o cálculo da diferença entre as duas medidas obtidas.

A leitura do relógio do sistema deverá ser feita em uma unidade próxima a microsegundos. A chamada da rotina R_Time é mostrada a seguir:

```
Tempo := R_Time;
```

A rotina R_Time é reservada para cálculo de períodos de tempo curtos, como é o caso do período transcorrido na execução de uma Fase Incremental numa unidade de processamento do Computador Incremental. Se, por exemplo, a leitura do relógio do sistema for feita em microsegundos, e Tempo for uma variável inteira de 32 bits, o máximo período de tempo que pode ser medido através de R_Time é aproximadamente 35 minutos.

CAPÍTULO 5

CONSIDERAÇÕES PARA IMPLEMENTAÇÃO

A implementação que será considerada neste trabalho é a tradução de um programa escrito em L para um programa em uma linguagem de alto nível H.

5.1 - PROCESSO DE TRADUÇÃO

A Figura 5.1 mostra o método para a tradução e execução de um programa em linguagem L em uma máquina alvo com processamento paralelo.

A partir do programa L são gerados **Programas Parciais** para cada unidade de processamento que participa do cálculo, sendo cada programa parcial um arquivo que contém os comandos que devem ser executados por um determinado processador do Computador Incremental, tal como especificado no programa L. Além dos arquivos de programas parciais é gerado um arquivo de **Índice**, que será utilizado para acionar o **Compilador H**, o **Conector ("linker")** e o **Carregador** dos programas.

Depois da fase de tradução, cada arquivo de programa parcial é compilado com a utilização do compilador H. O resultado da compilação de cada programa parcial é conectado à **Biblioteca L**, à **Biblioteca H** e às rotinas de suporte aos ambientes paralelo e concorrente, formando um arquivo de código executável para cada unidade de processamento.

Os arquivos de código executável são carregados, através de um **Programa Carregador**, nas unidades de processamento onde deverão

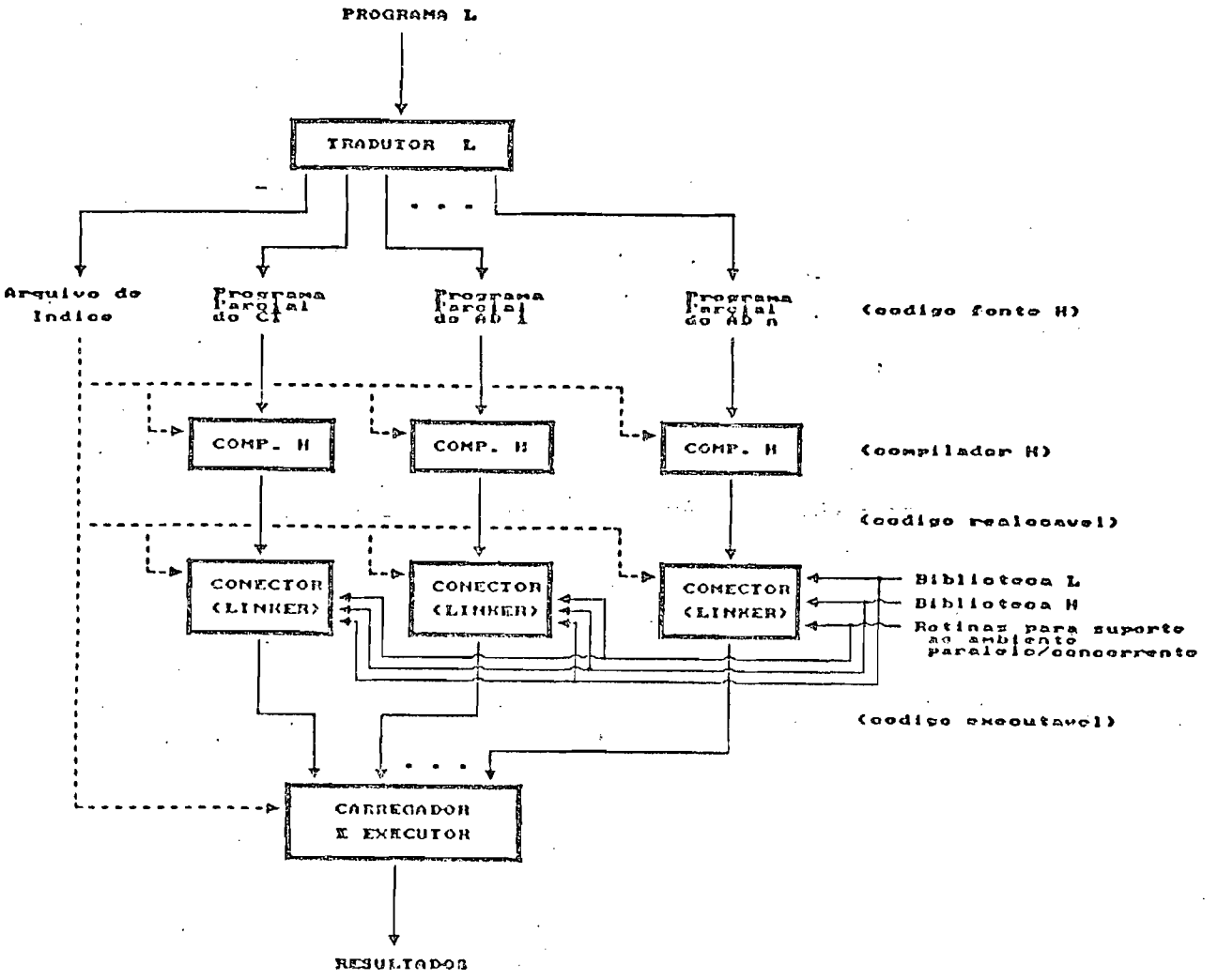


Fig. 5.1 - Método de tradução e execução de um programa em L para uma máquina alvo com processamento paralelo.

ser executados. A execução é iniciada a partir do programa parcial do CT, que por sua vez aciona a execução dos programas parciais nos ADs.

O processo de **compilação** de um programa em linguagem L pode ser feito, em relação a uma **máquina alvo**, com **residência própria** ou com **residência cruzada**, isto é, em uma outra máquina que não é a máquina alvo.

Uma possibilidade que pode também ser considerada é a execução de um programa em L em uma máquina alvo com monoprocessamento. A Figura 5.2 mostra o método para tradução de um programa em L, para este caso.

Note que os processos de tradução do programa em L para programas parciais em linguagem H, e de compilação dos programas parciais H são idênticos àqueles descritos na Figura 5.1. O processo de conexão é diferente no caso de monoprocessamento, pois todos os programas parciais devem ser conectados, já que todos serão executados no mesmo processador. Além disso, os programas parciais são conectados às **rotinas que simulam os ambientes paralelo e concorrente** a serem encontrados na máquina alvo.

5.2 AMBIENTES PARALELO E CONCORRENTE

A existência de comandos que são executados em paralelo na região Dynamic dos processos em L implica na necessidade de um ambiente que suporte a execução desses comandos. Os comandos que cada unidade de processamento deve executar são agrupados em seus respectivos programas parciais. As rotinas que suportam a existência de um ambiente paralelo no Computador Incremental são invocadàs por esses programas parciais para controle do fluxo de processamento. As rotinas de suporte ao ambiente paralelo implementam o fluxo de processamento do estágio de execução, mostrado na Figura 2.2.

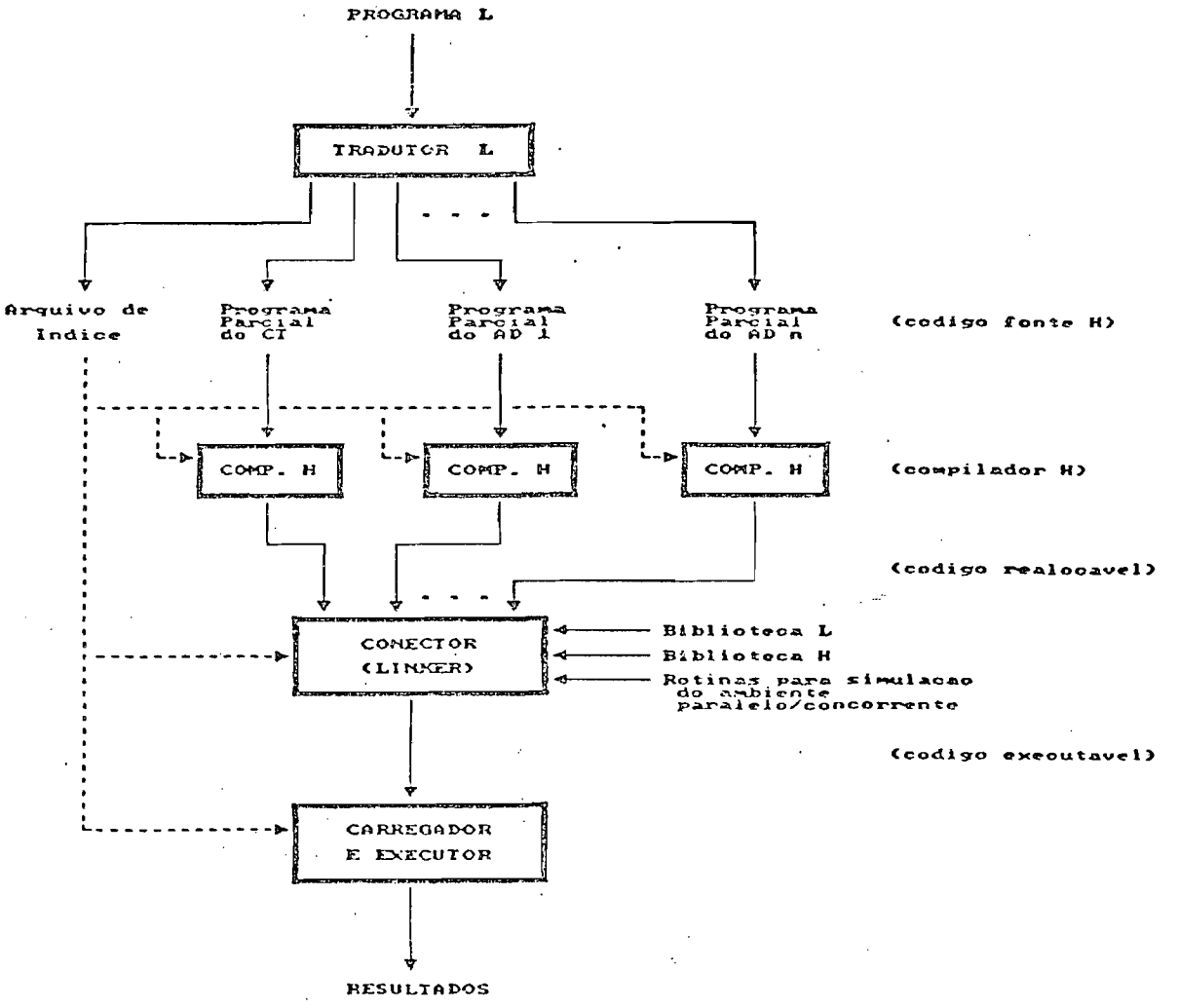


Fig. 5.2 - Método de tradução e execução de um programa em L para uma máquina alvo com monoprocessamento.

A existência de processos Repeat, Interrupt e Background na linguagem L requer a existência de um **Monitor** que garanta a execução concorrente desses processos. As primitivas desse Monitor são invocadas pelo programa parcial do CT para disparo e controle da execução desses processos.

5.2.1 - AMBIENTE PARALELO

As rotinas que suportam a execução de um programa em L em um ambiente paralelo devem fazer o disparo de uma Período Incremental (Figura 2.2) e gerenciar o término da execução de uma Fase Incremental nas diversas unidades de processamento do Computador Incremental.

Duas rotinas para suporte ao ambiente paralelo são invocadas pelo programa parcial do CT, a saber:

a) **Rotina ADs(palavra_controle, página):** envia a todos os ADs, por Difusão, os parâmetros "palavra_controle", que indica qual Iteração Incremental deve ser executada e "página", que indica qual cópia das variáveis do tipo State está sendo utilizada para atualização; depois disso, envia um sinal de disparo de Período Incremental, que faz com que cada AD que participa do cálculo passe a executar as tarefas que lhe couberam na Iteração Incremental especificada. Depois de executar esta rotina, o CT passa a executar as tarefas que lhe couberam na Iteração Incremental em questão. Após o término das tarefas do CT, ele aguarda que todos os ADs terminem também suas tarefas, para dar início à execução de uma nova Iteração Incremental.

b) **Rotina Esp_F_Fase:** rotina executada pelo CT quando termina as tarefas da Iteração Incremental em execução. Esta rotina

aguarda o aviso de todos os ADs que participam do cálculo de que também terminaram as tarefas da Iteração Incremental em execução.

Para suporte ao ambiente paralelo em cada um dos ADs, as rotinas descritas a seguir são utilizadas:

a) **Rotina Executa_Fase:** rotina que lê os parâmetros "palavra_controle" e "página" enviados pelo CT através da rotina ADs(palavra_controle,página) e aciona o programa parcial do AD com os parâmetros lidos. A partir do parâmetro "palavra_controle", o programa parcial determina que Iteração Incremental deverá ser executada.

b) Rotina Termina_Fase: rotina executada por cada AD ao terminar as tarefas da Iteração Incremental em execução. Esta rotina envia um sinal ao CT, indicando que o AD terminou a execução das tarefas que lhe couberam na Iteração Incremental em questão.

5.2.2 - AMBIENTE CONCORRENTE

As **primitivas** de um Monitor mínimo para suporte de execução dos processos em L são descritas em seguida:

a) **Start(processo):** coloca o processo definido pelo parâmetro "processo" na tabela de processos, em estado de pronto para execução. Utilizada para iniciar a execução dos processos Interrupt e Background.

b) **Repeat(processo,intervalo):** coloca o processo definido pelo parâmetro "processo" na tabela de processos e ativa sua

execução a cada intervalo definido pelo parâmetro "intervalo". Utilizada para iniciar a execução dos processos Repeat.

c) **Termina:** desativa o processo que invocou esta primitiva. Utilizada para terminar a execução dos processos Repeat, Interrupt e Background.

d) **Suspende:** suspende a execução do processo que invocou esta primitiva. Utilizada para encerrar a execução de uma iteração da região Dynamic dos processos Repeat e Background.

e) **Wait(semáforo):** operação de wait no semáforo definido pelo parâmetro "semáforo". Utilizada para sincronização entre processos.

f) **Signal(semáforo):** operação de signal no semáforo definido pelo parâmetro "semáforo". Utilizada para sincronização entre processos.

As rotinas e primitivas de suporte aos ambientes paralelo e concorrente constituem um conjunto mínimo de rotinas necessárias para suporte à execução de programas em L. No item seguinte é ilustrada a utilização dessas rotinas no processo de tradução dos programas em L.

5.3- ESTRUTURA DOS PROGRAMAS PARCIAIS

Dado um programa L, como o descrito a seguir:

Programa em L:

```
declaração
repeat "período" "nome"
  declaração
  initial
    .
    .
  end;
  dynamic
    fase 1
      1: comandos AD 1
      2: comandos AD 2
      comandos CT
    end;
    fase 2
      .
      .
    end;
      .
      .
  end;
  communication
    .
    .
  end;
  terminal
    .
    .
  end;
end;
```

Os arquivos a serem gerados para o CT e para os ADs do Computador Incremental, considerando-se disponíveis as rotinas discutidas no item anterior, são os seguintes:

Programa Parcial do CT:

```
begin
  declaração
  procedimento "nome"
  begin
   "código da região Initial"
    repeat
     Suspende;          (* encerra a execução de uma
                           iteração da região Dynamic *)
     TFC := false;
     while not TFC do
     begin
       pg := not pg;  (* inverte utilização das cópias
                         de variáveis do tipo State *)
       ADs(1,pg);     (* disparo dos ADs para execução
                         da Fase 1 *)
       "código da Fase 1 do CT"
       Esp_F_Fase     (* Espera Fim de Fase Funcional dos ADs *)
       ADs(2,pg);     (* disparo dos ADs para execução
                         da Fase 2 *)
       "código da Fase 2 do CT"
       Esp_F_Fase     (* Espera Fim de Fase Funcional dos ADs *)
        .
        .
       "atualiza variáveis de controle"
     end;
     "se chegou ao ponto de comunicação
       executa código da região Communication"
     "se chegou ao fim
       executa código da região Terminal
       termina;       (* encerra a execução do processo *)"
    until false;
  end;
  repeat("nome","período");  (* ativa o processo repeat *)
end;
```

Programa Parcial do AD i:

```
declaração
procedimento ADi(Fase,página)
begin
  case Fase of
   1: begin
        código da Fase 1 do AD i
        Termina_Fase;
      end;
   2: begin
        código da Fase 2 do AD i
        Termina_Fase;
      end;
   .
   .
  end;
end;
```

No programa parcial do CT, o processo repeat é transformado em um procedimento da linguagem H, que é acionado através da primitiva **repeat** do monitor de suporte ao ambiente concorrente.

No início desse procedimento é inserido o código da região Initial, que será executado apenas uma vez, quando a execução do procedimento for habilitada.

Em seguida, é inserido o código da região Dynamic, que será executado até que a condição de término da execução da região Dynamic seja verificada.

O código da região Dynamic é colocado em um comando "Repeat", que faz com que esse código seja executado repetidamente, até ser interrompido por uma invocação da primitiva Termina do

monitor, o que ocorrerá quando a condição de término de execução, dada pelo comando de controle "terminate" for satisfeita. A cada iteração do comando "repeat", é invocada a primitiva Suspende, que provoca a suspensão da execução do processo até o próximo período de ativação, definido pelo parâmetro **"período"** da invocação **"repeat("nome","período")"**.

Os comandos de uma região Dynamic constituem uma Iteração Funcional. O número de Iterações Funcionais necessário para realizar uma Iteração Funcional Completa é controlado através do comando "**While**". A variável de controle TFC é atualizada de acordo com o Método Funcional utilizado, para refletir o término da execução de uma Iteração Funcional Completa.

No início da execução de uma Iteração Funcional, é trocada a cópia das variáveis do tipo State que será utilizada para armazenar os dados desta Iteração Funcional (comando "pg := not pg"). Depois disso, é disparada a execução dos comandos de uma Iteração Incremental nos ADs (comandos de uma **fase** da região Dynamic), através da rotina ADs(palavra_controle,página).

Durante a execução de uma Iteração Incremental, o CT e os ADs realizam em paralelo as tarefas que lhes couberam nessa Iteração Incremental. Quando o CT termina a execução de suas tarefas, espera que todos os ADs também terminem, através da rotina "Esp_F_Fase". Isso completa a execução de uma Iteração Incremental. Se houver mais Iterações Incrementais, o processo se repete, até que a Iteração Funcional tenha sido completada. Depois disso, as variáveis de controle são atualizadas, entre elas a variável TFC, que determina quando uma Iteração Funcional Completa foi realizada.

Após o término da execução de uma Iteração Funcional Completa, o CT verifica se chegou ao ponto em que deve ocorrer a comunicação e, em caso positivo, executa os comandos da região Communication. Depois disso, verifica se chegou ao fim do cálculo e,

em caso positivo, executa os comandos da região Terminal e termina o processo, através da invocação da primitiva "Termina" do monitor.

O programa parcial de cada AD vai compor, com sua rotina Executa_Fase, o código a ser executado nesse AD. Quando o CT dispara o processamento de uma Iteração Incremental nos ADs, através da rotina ADs(palavra_controle,página), a rotina Executa_Fase de cada AD aciona o programa parcial desse AD, passando os parâmetros recebidos. Através do parâmetro "palavra_controle", que indica qual Iteração Incremental vai ser executada, cada AD executa os comandos que lhe couberam na Iteração Incremental em questão e depois de terminar, aciona a rotina Termina_Fase, que avisa ao CT que o cálculo da Iteração Funcional em execução nesse AD terminou.

A **tradução de processos do tipo Background** é idêntica à tradução de processos repeat, exceto que a forma de ativação desses processos é:

```
start("nome");
```

A invocação da primitiva Suspende no início de cada iteração da região Dynamic dos processos Background permite a ativação da execução de outros processos.

Na tradução de processos do tipo Interrupt, deve ser inserida, no início de cada iteração da região Dynamic, a invocação da primitiva Wait, no semáforo associado à interrupção que deve ativar a execução desse processo. A invocação da primitiva Suspende neste caso não é necessária, pois o processo ficará suspenso pela execução da primitiva Wait, até que ocorra a interrupção que o ativa.

5.4 - COMUNICAÇÃO ENTRE PROCESSADORES

A forma de execução de comandos paralelos pelos ADs do Computador Incremental e a comunicação entre processadores é bem determinada, como visto na Figura 2.2. A região Dynamic da linguagem L reflete o modo de execução descrito nessa Figura, ou seja, cada Iteração Incremental (fase da linguagem L) é equivalente à execução de um Período Incremental descrito na Figura 2.2.

A execução da região Dynamic, portanto, é a execução de uma sequência de Períodos Incrementais, que são caracterizados, em cada processador, pela execução de uma Fase Incremental.

No item anterior foi discutida a forma de execução de uma região Dynamic pelos ADs do Computador Incremental.

A comunicação entre processadores é realizada, a nível de gerenciamento de execução, pelas rotinas que suportam o ambiente paralelo do Computador Incremental.

A comunicação de dados entre os processadores é realizada com o uso de dois tipos especiais de variáveis, que são discutidas a seguir:

a) **variáveis do tipo Global:** são alocadas no espaço de endereçamento da Memória de Comunicação de todos os processadores, ou seja, há uma cópia dessas variáveis na Memória de Comunicação de todos os processadores. Uma variável do tipo Global é lida como uma variável local ao processador. A atualização de uma variável do tipo Global por um determinado processador implica na atualização da cópia dessa variável na Memória de Comunicação de todos os processadores.

b) **variáveis do tipo State:** são também alocadas na Memória de Comunicação de todos os processadores. A diferença entre uma variável Global e uma variável State é que esta última é duplicada na Memória de Comunicação de todas as unidades, para permitir a preservação do valor da variável durante uma iteração, enquanto um novo valor da mesma é calculado e armazenado. A forma adotada neste trabalho para manipular variáveis do tipo State foi discutida no capítulo 2 e consiste na declaração de cada variável State como um vetor de duas posições e o uso de cada posição desse vetor, alternadamente, como entrada e saída em iterações contíguas. A atualização de uma cópia da variável State também implica na atualização dessa cópia na Memória de Comunicação de todos os processadores.

Uma variável do tipo Global ou State deve ter o mesmo endereço na Memória de Comunicação de todos os processadores. Além disso, o hardware do Computador Incremental deve permitir a escrita por Difusão ("broadcast"). Dessa forma, para que um processador atualize uma variável do tipo State ou Global na memória de comunicação dos demais processadores, basta que ele faça uma escrita por Difusão no endereço dessa variável.

A escrita por Difusão poderá ser feita no momento em que a variável for atualizada, ou em uma Fase de Comunicação explícita, depois de efetuados todos os cálculos. A opção por um desses esquemas deverá levar em conta o desempenho do hardware específico em que a linguagem L será implementada, pois a Difusão a cada atualização das variáveis do tipo Global ou State implica em um número maior de operações de arbitragem no barramento que interliga os processadores, enquanto que a Difusão em uma Fase de Comunicação explícita, no final do cálculo, implica em concentração de operações de Difusão no final de um Período Incremental.

As considerações feitas neste capítulo mostram a estrutura dos programas parciais para cada processador do Computador Incremental, gerados pelo Tradutor a partir de um programa em L. Além disso, ajudam a esclarecer algumas características da linguagem L, discutidas no capítulo 4.

CAPÍTULO 6

SIMULAÇÃO DA EXECUÇÃO DE PROGRAMAS EM L EM MÁQUINA MONOPROCESSADORA

Foi implementado, como parte deste trabalho, o método para a tradução e execução de programas em L, como descrito na Figura 5.2, em uma máquina monoprocessadora.

Os componentes básicos desenvolvidos para a implementação do método são os seguintes:

a) **Protótipo do Tradutor L:** programa que gera, a partir de um programa L, os programas parciais de cada unidade de processamento do Computador Incremental e o arquivo para ativação do processo de compilação dos programas parciais em linguagem H e do processo de conexão dos programas realocáveis resultantes deste processo de compilação.

b) **Rotinas de simulação dos ambientes paralelo e concorrente:** rotinas que simulam a execução das rotinas de suporte aos ambientes paralelo e concorrente descritas no capítulo 5.

c) **Biblioteca L:** subconjunto da Biblioteca L que permite, no caso, a execução do processo de integração de equações diferenciais de primeira ordem.

O software implementado permite que programas em L sejam compilados e executados em microcomputadores do tipo IBM-PC. A seguir são descritos cada um dos itens da implementação citados acima.

6.1 - PROTÓTIPO DO TRADUTOR L

O protótipo do Tradutor L, chamado a partir de agora simplesmente de Tradutor, foi escrito em Pascal e gera programas parciais para cada unidade de processamento do Computador Incremental em linguagem C. Portanto, a linguagem de alto nível H citada na Figura 5.2 é, no caso, a linguagem C.

Para acelerar o processo de implementação do Tradutor, foi utilizado como ponto de partida um compilador para Pascal-S (Wirth, 1981). Esse compilador que gera, a partir de um programa escrito num subconjunto de Pascal, um programa em código intermediário, que depois é interpretado, sofreu modificações, para receber como entrada um programa em L e, então, gerar programas parciais para cada unidade de processamento do Computador Incremental em linguagem C.

Devido ao fato de que muitas construções da linguagem L são idênticas ou semelhantes às da linguagem Pascal, a utilização desse compilador como ponto de partida representou uma economia considerável no tempo de implementação do Tradutor, particularmente no processo de análise léxica, que é praticamente o mesmo, assim como no processo de análise sintática das construções L baseadas em Pascal.

A seguir são descritos alguns pontos importantes que caracterizam a geração dos programas parciais pelo Tradutor L. Um exemplo completo de tradução de um programa em L para programas parciais em C pode ser visto no apêndice B.

6.1.1 - CONSTANTES E VARIÁVEIS STATE E GLOBAL

Os identificadores de constantes e de variáveis do tipo State e Global devem ser visualizados pelo programa parcial do CT e

pelos programas parciais dos ADs. Como o programador pode declarar esses identificadores localmente a um processo e eles devem ser visualizados pelos programas parciais dos ADs, o Tradutor transforma um identificador local a um processo em um identificador global, acrescido de um sufixo, que o relaciona ao processo do qual ele foi extraído.

O exemplo a seguir clarifica essa operação. Seja o programa em L submetido ao Tradutor, na forma:

```
var x1,x2:real;
state y1,y2:real;
background nome;
 var i,j:integer;
 state y,y1:real;
  .
  .
 dynamic
    .
    .
  1: y:= integ(y+y1);
    .
    .
 end;
  .
  .
end;
```

O programa parcial gerado pelo Tradutor L para o CT é, então, o seguinte:

```
float x1,x2;
float y1[2],y2[2];
float y_00[2],y1_00[2];
void nome
  int i,j;
```

O programa parcial gerado para o AD 1 é o seguinte:

```
extern float y1[2],y2[2];
extern float y_00[2], y1_00[2];
void AD01(int palavra_controle, int pg)
  switch (palavra_controle)

    case nro: y_00[!pg]= integ(y_00[pg]+y1_00[pg], ...);
              break;
```

Para o tratamento de identificadores de constantes e de variáveis do tipo State e Global, foi criado mais um campo na tabela de símbolos do Tradutor, onde é colocado o nome do identificador, acrescido de seu sufixo. Desta forma, o tratamento de constantes e de variáveis do tipo State ou Global é o mesmo de uma variável normal, exceto que, na tradução para a linguagem C, é utilizado o nome armazenado nesse campo especial da tabela de símbolos do Tradutor.

Se uma variável X do tipo State é utilizada em comandos da linguagem L, na tradução para linguagem C, a referência a essa variável para operações de leitura é acrescida do sufixo [pg], e a referência a essa variável para operações de escrita é acrescida do

sufixo [!pg] ([not pg]). Isso permite que, numa Iteração Funcional, uma cópia da variável seja usada para leitura (X[pg]), enquanto a outra é usada para escrita (X[!pg]).

No ambiente paralelo, deverá haver uma cópia das variáveis Global e State na Memória de Comunicação de cada processador. Essas cópias deverão ter o mesmo endereço na Memória de Comunicação de todos os processadores, para permitir o processo de difusão. O programa parcial gerado para o CT determina o endereço das variáveis Global e State. Os programas parciais dos ADs fazem referência ao mesmo endereço, em suas respectivas memórias de comunicação. Na simulação da execução de programas em L, em um ambiente de monoprocessamento, há apenas cópia das variáveis Global e State no CT.

As alterações sofridas por variáveis do tipo State ou Global em um processador deverão ser enviadas aos outros processadores por Difusão. Esse processo de Difusão poderá ser feito no final da Fase Incremental de cada processador (na Fase de Comunicação), ou logo que a variável sofrer alteração. A escolha de uma dessas duas opções apontadas depende do desempenho do hardware específico em que a linguagem venha a ser implementada. Essas duas formas de comunicação por difusão são funcionalmente equivalentes, pois um processador só poderá utilizar uma variável atualizada por difusão por outro processador depois que terminar o Período Incremental em que essa atualização ocorreu. A simulação da execução do programa em L em um ambiente de monoprocessamento considera que a atualização de variáveis do tipo Global e State é feita logo que elas são alteradas, já que existe apenas uma cópia dessas variáveis no CT.

6.1.2 - TRADUÇÃO DE UM PROGRAMA EM L

A tradução de um programa em L deve gerar um programa parcial para o CT e programas parciais para cada um dos ADs envolvidos

no cálculo.

Cada programa parcial é escrito em um arquivo. Devido ao fato de que o Tradutor é executado em um computador do tipo IBM-PC, cujo Sistema Operacional permite que os nomes de arquivos tenham apenas oito caracteres, para que possam ser gerados os nomes dos arquivos dos programas parciais, o nome do arquivo do programa fonte em L deve ter, no máximo, cinco caracteres, e a extensão "l". Os outros três caracteres vão diferenciar os nomes dos arquivos dos programas parciais. O nome do arquivo que contém o programa parcial do CT é constituído pelo nome do arquivo do programa fonte em L, seguido do sufixo "_ct" e extensão "c". Os nomes dos arquivos que contêm os programas parciais dos ADs são constituídos, cada um deles, pelo nome do arquivo do programa fonte em L, seguido do sufixo "_nn" e a extensão "c", onde "nn" é o número do AD.

A tradução de comandos da linguagem L para comandos da linguagem C não será discutida neste trabalho.

Os programas parciais gerados para cada processador são semelhantes aos descritos no capítulo 5.

Na tradução de um processo, ao encontrar a palavra reservada que indica o início do processo (Repeat, Interrupt ou Background), o tradutor gera o cabeçalho de uma função C, que conterá os comandos descritos nesse processo. Além disso, armazena o nome, o tipo do processo e outras informações relevantes em uma tabela de processos para, no final, gerar o código referente à ativação desses processos.

As declarações são processadas normalmente, exceto as declarações de constantes e variáveis do tipo State e Global, que recebem o tratamento descrito no item 6.1.1.

O código da região Initial é inserido no início da função; em seguida, é inserido o código da região Dynamic, juntamente com as estruturas que controlam sua execução.

Ao encontrar um comando que deve ser executado em um dos ADs, o tradutor muda o arquivo de saída (que era o do programa parcial do CT) e escreve a tradução do comando no arquivo que contém o código do programa parcial desse AD, juntamente com as estruturas de controle que permitirão ao CT a ativação da execução desse comando durante a execução do programa.

As regiões Communication e Terminal, se existirem, são traduzidas a seguir, com a inserção de comandos para controle da execução dessas regiões.

Quando termina a tradução de um processo, o tradutor encerra a funçao C correspondente ao processo.

Se houverem outros processos, eles são traduzidos da mesma forma.

No final do programa, o tradutor encerra os programas parciais dos ADs e gera os comandos de ativação dos processos, a partir dos dados armazenados na tabela de processos no início da tradução de cada processo.

O apêndice B mostra os arquivos gerados pelo Tradutor a partir de um programa fonte em L.

6.2 - ROTINAS DE SIMULAÇÃO DOS AMBIENTES PARALELO E CONCORRENTE

As rotinas discutidas neste item permitem a simulação da execução de um programa L em uma máquina monoprocessadora. As principais restrições da simulação estão no comportamento concorrente

dos processos em L, pois a simulação do comportamento paralelo é mais simples, bastando, para tanto, executar serialmente os comandos que deveriam ser executados em paralelo.

6.2.1 - ROTINAS DE SIMULAÇÃO DO AMBIENTE PARALELO

As **rotinas de simulação** do ambiente paralelo são descritas a seguir:

a) **Rotina ADs(palavra_controle,página)**: esta rotina, quando executada no CT da máquina paralela, deve enviar, por difusão, os parâmetros **"palavra_controle"**, que indica qual Iteração Incremental será executada pelos ADs e **"página"**, que indica qual cópia das variáveis do tipo State será utilizada para armazenar os dados gerados nessa Iteração. Depois disso, a rotina deve enviar aos ADs um sinal de início de Período Incremental, que indica a esses ADs que eles devem ler os parâmetros enviados e iniciar a Iteração Incremental corrente. A simulação desta rotina deve acionar serialmente cada procedimento que contém os comandos a serem executados pelos ADs com os parâmetros "palavra_controle" e "página". Deste modo, a execução paralela dos cálculos nos ADs é transformada em execução sequencial.

b) **Rotina Esp_F_Fase**: esta rotina, quando executada no CT da máquina paralela, deve contabilizar a execução de um Período Incremental, verificando que cada AD terminou a execução de sua Fase Incremental. A simulação desta rotina não precisa realizar nenhuma operação, pois os cálculos que os ADs realizariam em paralelo foram efetuados serialmente e quando esta rotina é acionada, em verdade, cada AD já terá realizado sua Fase Incremental.

c) **Rotina Executa_Fase:** esta rotina, ao ser executada por um AD da máquina paralela, deve ler os parâmetros "palavra_controle" e "página", enviados pelo CT, através da rotina ADs(palavra_controle,página) e acionar a execução do programa parcial do AD, que contêm os comandos a serem executados por esse AD, com os parâmetros lidos. A simulação da execução desta rotina é coberta pela rotina ADs(palavra_controle,página), portanto, esta rotina não precisa ser utilizada na simulação.

d) **Rotina Termina_Fase:** esta rotina, quando executada em um AD da máquina paralela deve enviar um sinal ao CT, indicando que o AD terminou a execução da Fase Incremental corrente. A simulação da execução desta rotina não precisa realizar nenhuma operação, pois a execução da Fase Incremental de cada AD é acionada serialmente pela rotina ADs(palavra_controle,página), ou seja, depois da simulação da execução de uma Fase Funcional de um AD, o controle da simulação volta para a rotina ADs(palavra_controle,página).

6.2.2 - ROTINAS DE SIMULAÇÃO DO AMBIENTE CONCORRENTE

Para simular o ambiente concorrente, foram programadas as primitivas que compõem um monitor que simula o funcionamento do monitor descrito no capítulo 5.

A principal restrição do monitor implementado é que ele não permite a execução de processos em tempo real, fazendo apenas a simulação da ocorrência de interrupções para ativação de processos Interrupt e a simução da passagem de intervalos de tempo para ativação de processos Repeat.

O monitor é composto por uma **tabela descritora de processos** (tabela de processos), um **conjunto de semáforos gerais** e **primitivas** para manipulação dessas estruturas.

A tabela de processos contém as seguintes informações:

a) **Estado do Processo**, cujos valores são:
   0: entrada nula;
   1: processo terminado;
   2: processo suspenso;
   3: processo pronto.

b) **Próximo**: número da entrada, na tabela de processos, do próximo processo suspenso no mesmo semáforo.

c) **Timer**: contador utilizado para acionar processos Repeat. É iniciado com o valor do campo Intervalo da tabela de processos (descrito em seguida) e decrementado cada vez que o monitor é ativado. Quando seu valor é zero, o processo Repeat associado é colocado em estado de pronto para execução.

d) **Intervalo**: contém o intervalo de ativação do processo Repeat associado, ou seja, se o valor do campo Intervalo é n, na n-ésima ativação do monitor, o processo Repeat associado é colocado em estado de pronto para execução. Considera-se uma ativação do monitor, a invocação de qualquer uma de suas primitivas.

e) **Contexto**: armazena as informações necessárias para reativação do processo.

Os semáforos gerais são estruturas compostas de dois campos:

a) **Contador:** armazena o número de sinais recebidos e não utilizados.

b) **Primeiro:** armazena o endereço, na tabela de processos, do primeiro processo suspenso neste semáforo. Os processos suspensos a seguir são encadeados através do campo Próximo da tabela de processos.

Há 16 semáforos, de um total de 32, reservados para ativação de processos "interrupt", os outros podem ser utilizados para sincronização de processos em L.

Cada execução de primitiva do monitor implica na escolha de um novo processo para execução, cujas prioridades são:

a) **Processos Repeat:** são os de maior prioridade.

b) **Processos Background:** são os de menor prioridade, selecionados através de estratégia de prioridade rotativa ("round-robin").

Cada execução de primitiva do monitor faz com que o campo Timer da entrada na tabela de processos de cada processo Repeat seja decrementado de uma unidade, simulando a passagem de um intervalo de tempo.

É simulada a ocorrência de interrupção em aproximadamente 30% das ativações do monitor, através da geração de números aleatórios. Se o semáforo associado à interrupção ativada tem a ele associado um processo do tipo Interrupt, é realizada uma operação de "signal" nesse semáforo.

A seguir são descritas as **primitivas do monitor:**

a) **Wait(semáforo):** realiza a operação de "wait" no semáforo "semáforo", descrita a seguir:

```
Wait(s):
    s.contador := s.contador - 1;
    se s.contador < 0
    então "coloca processo na fila de espera
          e altera seu estado para suspenso"
```

b) **Signal(semáforo):** realiza a operação de signal no semáforo "semáforo", descrita a seguir:

```
Signal(s):
    s.contador := s.contador + 1;
    se s.contador <= 0
    então "retira um processo da fila de espera
          e altera seu estado para pronto"
```

c) **Start(processo):** procura uma entrada livre na tabela de processos e aloca essa entrada para o processo definido pelo parâmetro "processo", colocando no campo Estado o valor "pronto".

d) **Repeat(processo,intervalo):** procura uma entrada livre na tabela de processos e aloca essa entrada para o processo definido pelo parâmetro "processo", colocando no campo Estado o valor "pronto". A partir do parâmetro "intervalo", calcula o valor do campo Intervalo, atualizando-o também. Em seguida, preenche o valor do campo Timer com o valor do campo Intervalo.

e) **Termina:** altera o estado do processo que invocou esta primitiva para "terminado".

f) **Suspende:** suspende o processo que invocou esta primitiva. Se o processo é do tipo Interrupt, muda seu estado para "suspenso". Se o processo é do tipo Background, muda seu estado para "pronto".

A implementação deste monitor permite a simulação do comportamento concorrente de um programa L num ambiente que, em princípio, não permite processamento concorrente.

6.3 - BIBLIOTECA L

A biblioteca L agrupa as rotinas intrínsecas e as rotinas intrínsecas especiais da linguagem L, que são invocadas pelos programas em L, e também as rotinas que são utilizadas de maneira transparente, para controle da execução da região Dynamic.

As rotinas intrínsecas foram implementadas pela simples tradução da invocação da rotina em L para invocação da rotina em C.

As rotinas intrínsecas especiais implementam os Métodos Funcionais utilizados para solução incremental de problemas no Computador Incremental. Entre esses Métodos Funcionais estão os métodos de solução de sistemas de equações diferenciais de primeira ordem.

As rotinas utilizadas para controle da execução da região Dynamic devem atualizar as variáveis de controle, de acordo com o Método Funcional que está sendo utilizado, e verificar quando foi alcançado um ponto de ativação da região Communication.

As rotinas implementadas para utilização nos programas gerados pelo Tradutor são:

a) **Integ:** faz a interface entre a chamada **"integ(expressão)"** do programa em L com a rotina de integração definida pela variável de controle **method.**

b) **RK4:** implementa o algoritmo de integração Runge Kutta clássico.

c) **Update:** atualiza as variáveis de controle da execução da região Dynamic, de acordo com o Método Funcional utilizado.

d) **Comm:** verifica se foi atingido um ponto de ativação dos comandos da região Communication.

O apêndice B mostra a execução de um programa L com a utilização dos recursos de simulação apresentados neste capítulo.

CAPÍTULO 7

CONSIDERAÇÕES FINAIS

A principal vantagem da linguagem L é que ela permite a programação do Computador Incremental de maneira simples. Embora o Computador Incremental seja um sistema multiprocessador, não é necessário escrever um programa para cada processador e preocupar-se com a comunicação entre eles. O programador escreve apenas um programa para o CT, alocando parcelas do cálculo para serem executadas nos ADs. A comunicação entre os processadores é transparente, através da utilização de variáveis do tipo State e Global. A sincronização dos processadores na execução dos comandos em paralelo também é transparente ao programador.

Em uma região Dynamic de um programa em L, é possível realizar cálculos em paralelo, dentro de uma fase de cálculo, ou serialmente, com a execução de mais de uma fase.

Para que comandos possam ser executados em paralelo, de maneira previsível, é necessário que as variáveis alteradas por um comando não sejam referenciadas pelos outros; caso contrário, o resultado da execução dos comandos vai depender da ordem em que eles forem executados. A utilização de variáveis do tipo State na linguagem L permite que comandos que são executados em paralelo pelos processadores do Computador Incremental façam referência ao mesmo conjunto de variáveis do tipo State para atualização ou leitura, pois essas variáveis são duplicadas internamente, sendo que, em uma Iteração Funcional, uma cópia dessas variáveis é utilizada para leitura, enquanto a outra é utilizada para atualização. Deste modo, o conjunto de variáveis utilizadas para atualização é, na realidade, disjunto do conjunto de variáveis utilizadas para leitura,

preservando, portanto, o determinismo na execução paralela de comandos pelos processadores do Computador Incremental.

A estrutura de controle da execução da região Dynamic apresentada no capítulo 6, e implementada para simulação em um ambiente de monoprocessamento mostrou-se adequada para os Métodos Funcionais que realizam integração de equações diferenciais. No entanto, essa estrutura de controle é facilmente adaptável, e pode ser modificada para cobrir eventuais necessidades de outros Métodos Funcionais.

A linguagem L é adequada para a programação do Computador Incremental, já que os programas em L refletem o modo de processamento desse computador. A eficiência do processamento paralelo deve ser medida quando a linguagem L for implementada em um ambiente paralelo.

Futuros desenvolvimentos deste trabalho devem considerar, além da implementação da linguagem L em um ambiente paralelo, a programação de Métodos Funcionais, de modo a dotar a linguagem de um maior número de ferramentas para solução incremental de sistemas representáveis por modelos matemáticos.

Futuras extensões deste trabalho devem incluir a possibilidade de execução de Iterações Incrementais nas regiões Initial, Communication e Terminal.

REFERÊNCIAS BIBLIOGRÁFICAS

BERGAMINI, E.W; DIEHL, J.B. Uma arquitetura com processamento paralelo para computação incremental. In: SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES/PROCESSAMENTO PARALELO, 1., Gramado, 1987. Anais. Gramado, RS, Sociedade Brasileira de Computação, 1987, p. 115-128.

CROSBIE, R.E.; JAVEY, S.; HAY, J.L.; PEARCE, J.G. ESL - A new continuous system simulation language. Simulation, 44(5):242-246, May 1985.

FLYNN, M.J. Some computer organizations and their effectiveness. IEEE Transactions on Computers, C-21(9):948-960, Sept. 1972.

GARZIA, R.F. Experience with SYSL - System Simulation Language. Modeling; Simulation Technical Committee Newsletter, 25:4-12, Dec. 1986.

KOWALTOWSKI, T. Implementação de linguagens de programação. São Paulo, Escola de Computação. Instituto de Matemática de Estatística da USP, 1979.

PETERSON, J.L.; SILBERCHATZ, A. Operating system concepts. Reading, Massachusetts, Addison-Wesley, 1985.

SETZER, V.W.; MELO, I.S.H. A construção de um compilador. Rio de Janeiro, Ed. Campus, 1983.

SIEBERT, J.P; WINNING, D.J. The control oriented language. Simulation, 48(1):6-10, Jan. 1987.

SPECKHART, F. H.; GREEN, W. L. A guide to using CSMP. Englewood Cliffs, New Jersey, Prentice-Hall, 1976.

STRAUSS, J.C. (ed.) The SCi continuous system simulation language (CSSL). Simulation, 9(6):281-303, Dec. 1968.

WIRTH, N. Algorithms + data structures = programs. Englewood Cliffs, New Jersey, Prentice-Hall, 1976.

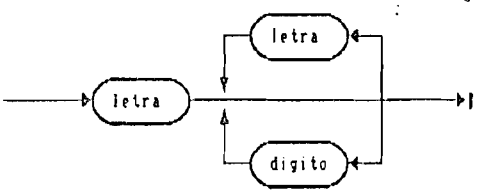
WIRTH, N. Pascal S: a subset and its implementation. In: BARROW, D.W. (ed.) Pascal: the language and its implementation. Chichester, John Wiley & Sons, 1981, p. 199-259.

WIRTH, N. The programming language Pascal. Acta Informatica, 1:35-63, 1971.

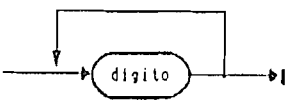
APÊNDICE A

DIAGRAMAS DE SINTAXE DA LINGUAGEM L

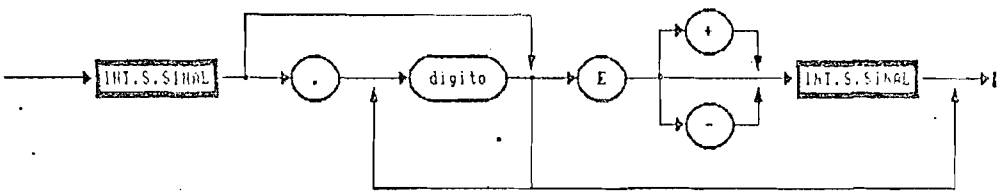
IDENT.



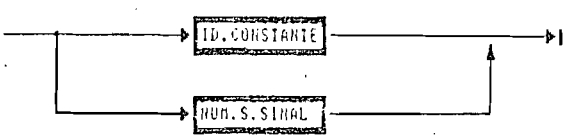
INT.S.SINAL



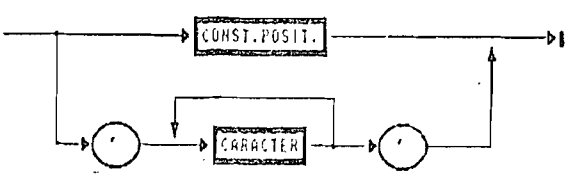
NUM.S.SINAL



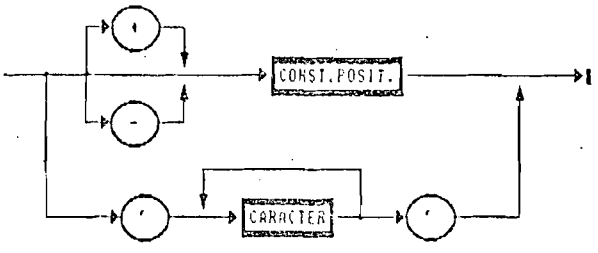
CONST.POSIT.



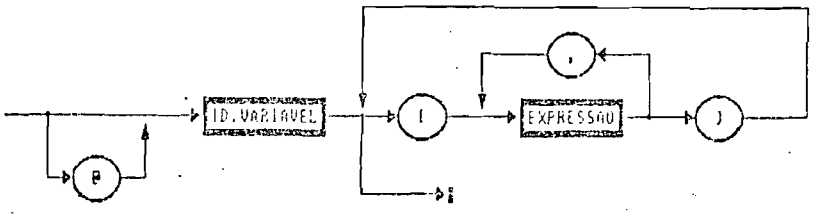
CONST.S.SINAL



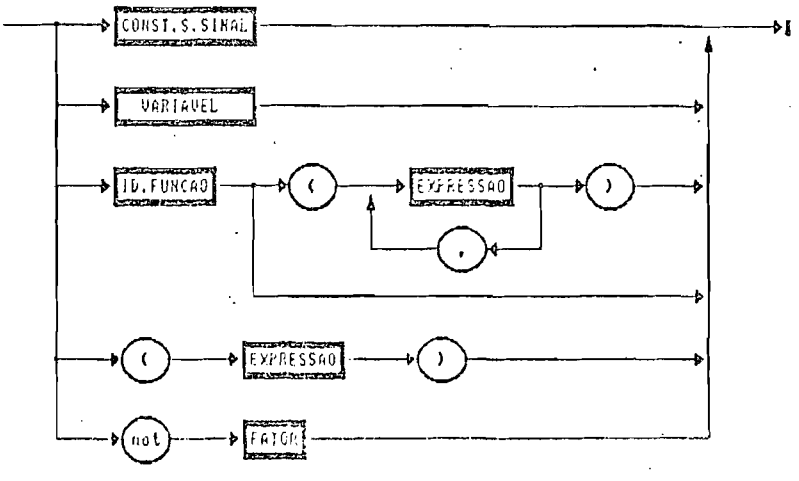
CONSTANTE



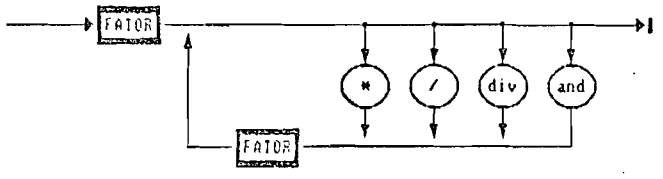
VARIAVEL



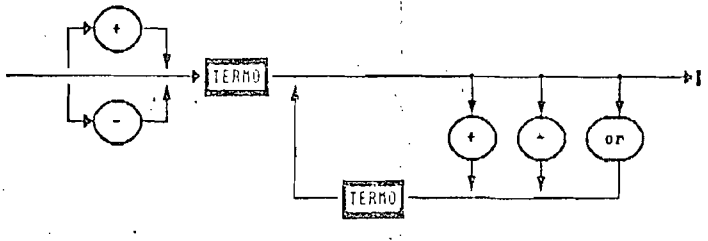
FATOR



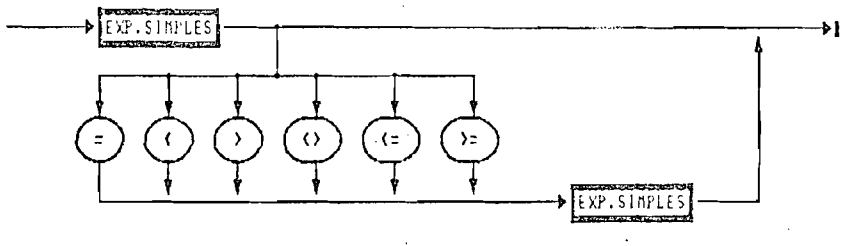
TERMO



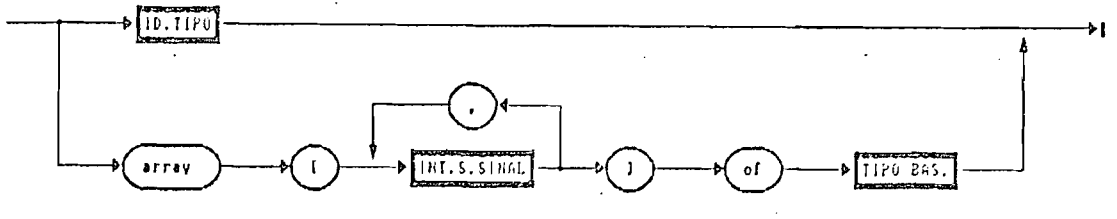
EXP.SIMPLES



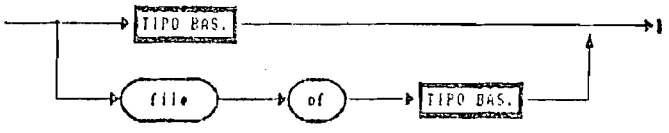
EXPRESSAO



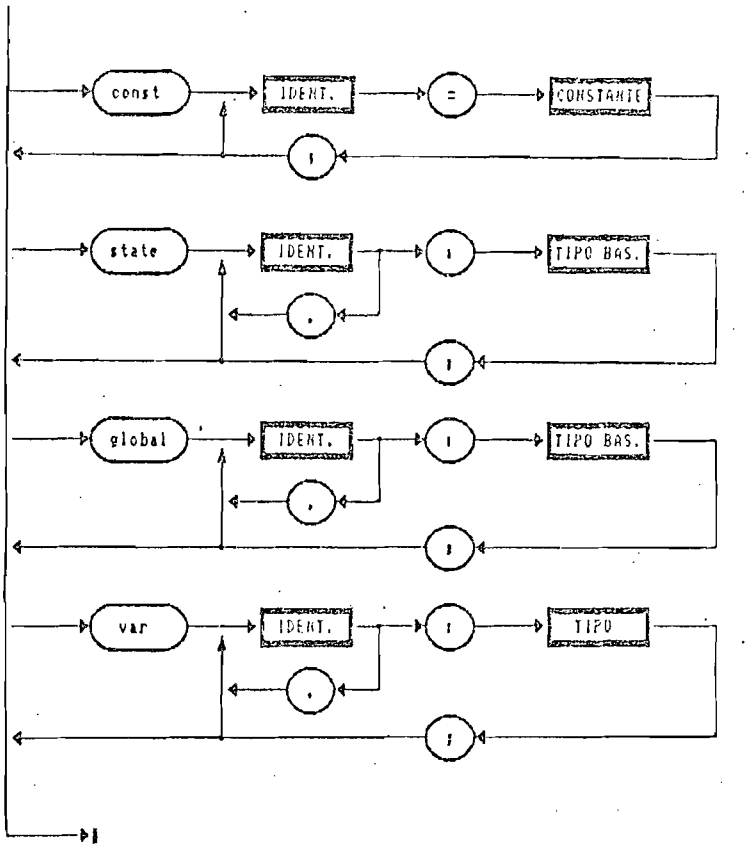
TIPO BAS.



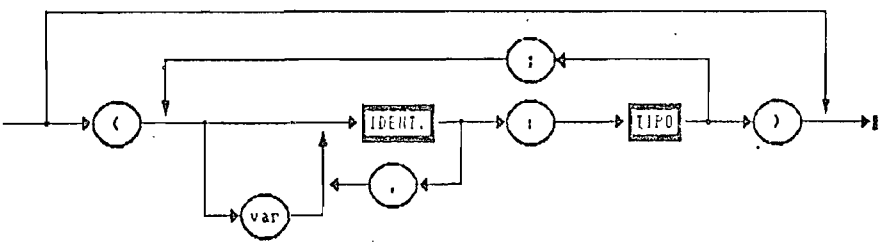
TIPO



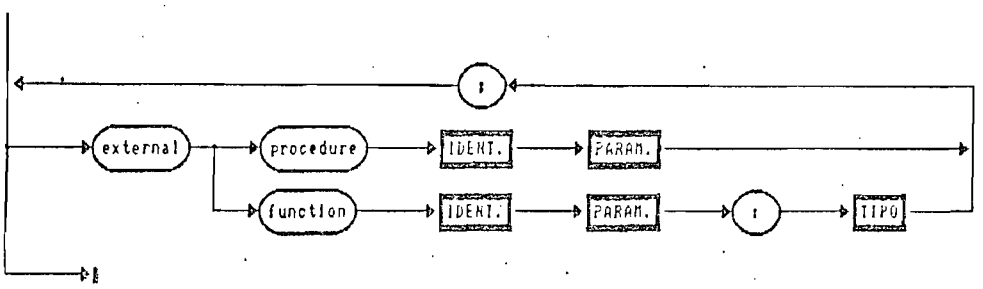
DECLARACAO



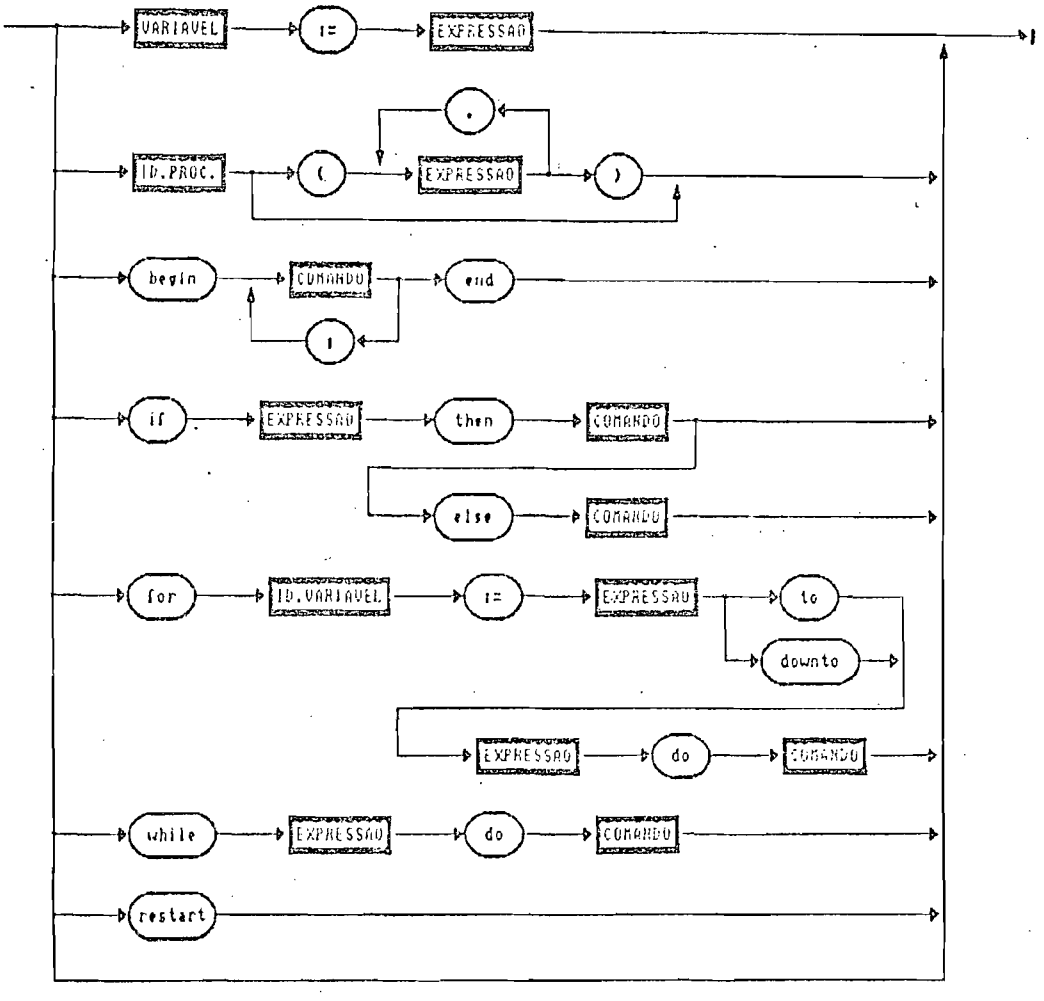
PARAM.



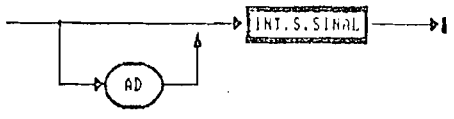
DECL. PROCED.



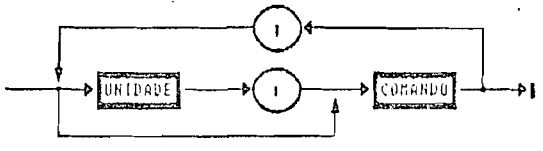
COMANDO


VARIAVEL
:=
EXPRESSAO
ID.PROC.
(
,
EXPRESSAO
)
begin
COMANDO
;
end
if
EXPRESSAO
then
COMANDO
else
COMANDO
for
ID.VARIAVEL
:=
EXPRESSAO
to
downto
EXPRESSAO
do
COMANDO
while
EXPRESSAO
do
COMANDO
restart

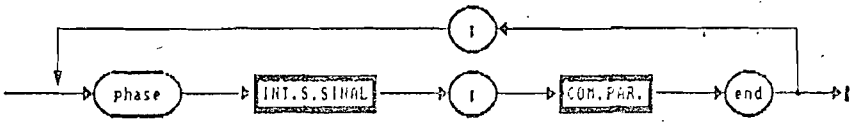
UNIDADE



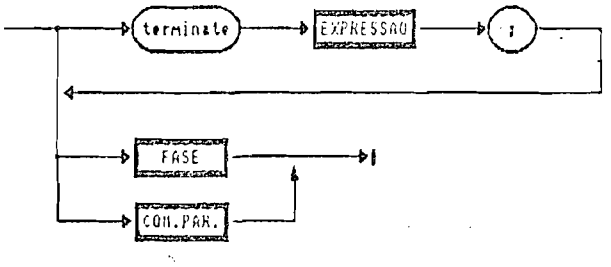
COM.PAR.



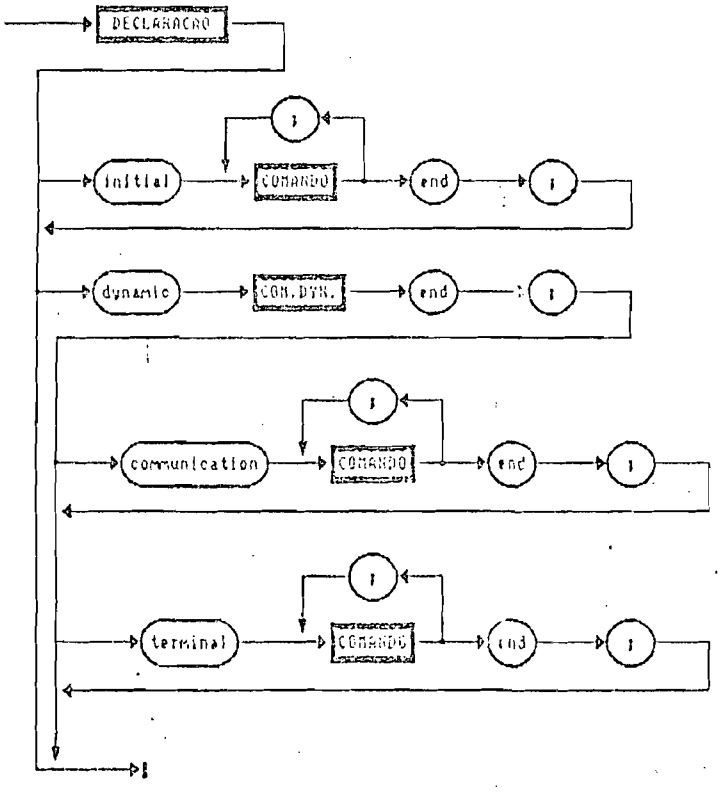
FASE



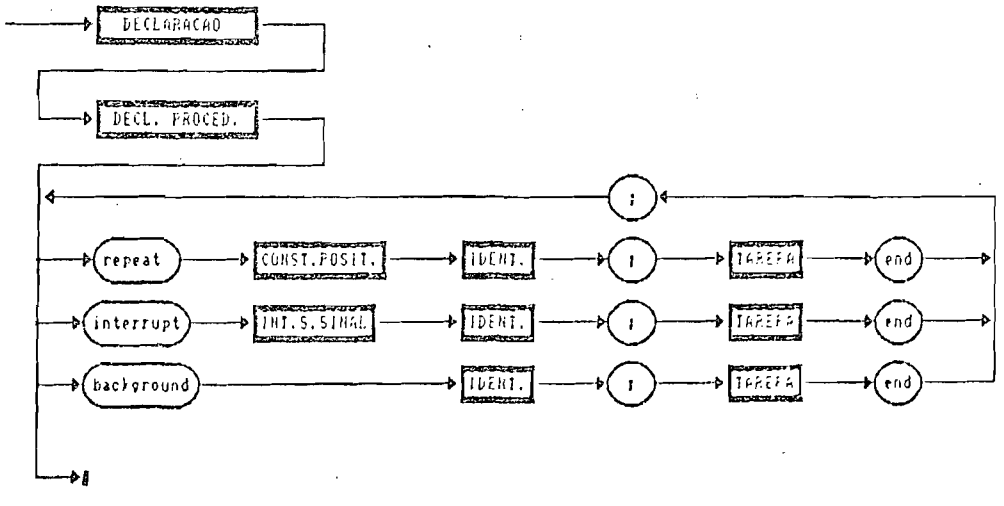
COM.DYN.



TAREFA



PROGRAMA



APÊNDICE B

EXEMPLO DE TRADUÇÃO DE PROGRAMA EM L E DE SIMULAÇÃO DE EXECUÇÃO EM AMBIENTE COM MONOPROCESSAMENTO

O exemplo a ser apresentado tem por objetivo esclarecer e completar a discussão realizada no capítulo 6. Embora seja um exemplo simples, ilustra todo o mecanismo envolvido na tradução de um programa escrito em linguagem L para programas parciais em linguagem C.

B.1 - PROBLEMA A SER RESOLVIDO

O problema a ser resolvido é o sistema de equações lineares descrito a seguir:

```
dy1/dt = a * y1 + b * y2
dy2/dt = c * y1 + d * y2
```

Este problema é ideal para ser dividido entre dois processadores do Computador Incremental, já que as equações, conforme os parâmetros utilizados, apresentam carga de processamento equilibrada.

Os parâmetros a serem utilizados são:

```
a = 0;   b = 1
c = -1;  d = 0
```

A utilização desses parâmetros faz com que a solução

analítica do sistema seja:

$$y1 = sen(t)$$
$$y2 = cos(t)$$

A solução analítica vai permitir que os resultados obtidos sejam facilmente conferidos.

B.2 - PROGRAMA EM L

O programa em L que descreve o problema apresentado no item anterior é descrito na Figura B.1.

```
{ Sistema Linear Bidimensional }
const a = 0;    b = 1;
      c = -1;   d = 0;
background SistLinear;
  state y1,y2: double;
  var x: double;
  initial
    y1 := 0;
    x  := y1;
    y2 := 1;
    dt := 0.05;
    print(t,x,y1,y2);
  end;
  dynamic
    ad 1: y1 := integ(a * y1 + b * y2);
    ad 2: y2 := integ(c * y1 + d * y2);
          x  := y1;
  end;
  communication
    print(t,x,y1,y2);
  end;
  terminal
    print(t);
  end;
end;
```

Fig. B.1 - Programa em L.

Os parâmetros das equações (parâmetros a, b, c e d) foram declarados como constantes. Eles foram declarados a nível de programa, mas poderiam ter sido declarados internamente ao processo Background.

As variáveis de estado (variáveis y1 e y2) foram declaradas internamente ao processo, para ilustrar o tratamento dessas variáveis pelo Tradutor.

Neste programa, foi inserido um comando ilustrativo, que não faz parte da solução do problema; por isso foi declarada a variável x.

Na região Initial são atribuídas as condições iniciais das variáveis de estado, bem como é atribuído um valor inicial à variável x e à variável de controle dt, que determina o incremento da variável de controle t (variável independente do sistema de equações), a cada Iteração Funcional Completa (neste caso, a cada passo de integração). Outras variáveis de controle, tais como cinterval, que determina o intervalo de ativação da região Communication ou method, que determina o Método Funcional (neste caso, o método de integração) que vai ser utilizado para solução do problema também podem ser inicializadas nesta região. Quando não são inicializadas, essas variáveis de controle assumem o valor padrão. No caso, o valor padrão para a variável de controle method determina que o método a ser utilizado é o método Runge Kutta clássico e o valor padrão para a variável de controle cinterval determina o intervalo de ativação da região Communication igual a 0,1, ou seja, a cada duas execuções de uma Iteração Funcional Completa (dois passos de integração), a região Communication é ativada.

Na região Dynamic é descrito o sistema de equações a ser resolvido. Neste caso, cada AD vai resolver uma equação. O comando **"x := y1"**, desnecessário para a solução do problema, foi colocado para ilustrar o fato de que o CT pode executar comandos em paralelo com os

ADs e para ilustrar a utilização das duas cópias das variáveis do tipo State.

Na região Communication foi colocado o comando para listar as variáveis atualizadas na região Dynamic. Note que a variável de controle t é atualizada automaticamente na região Dynamic.

Na região Terminal foi colocado apenas o comando para listar o valor final da variável de controle t.

B.3 - PROGRAMAS PARCIAIS

Um arquivo de programa em L deve ter a extensão "l". Neste caso, o arquivo que contém o programa da Figura B.1 foi chamado de "sl.l".

Colocando o arquivo "sl.l" como entrada do Tradutor L, os seguintes arquivos são gerados:

a) Arquivo "sl_ct.c":

O arquivo "sl_ct.c" contém o programa parcial do CT em linguagem C. A Figura B.2 descreve esse arquivo.

As declarações de constantes e variáveis do tipo State e Global que são globais aos processos são colocadas no arquivo "sl.dec", que é anexado ao programa parcial do CT no momento da compilação.

O processo Background é transformado em uma função da linguagem C (função "sistlinear").

As variáveis locais ao processo, que não são visíveis para os programas parciais dos ADs, são declaradas internamente a essa

```
#include "sl.dec"
void sistllnear()
{
static double cont_L=0;
static int pg=0;
static double cinterval=0.1;
double x;
/* INITIAL */
y1_01[!pg] = 0;
x = y1_01[!pg];
y2_01[!pg] = 1;
dt_01 =  5.0000000000E-02;
printf("\n");
printf("    t= %f",t_01);
printf("    x= %f",x);
printf("    y1= %f",y1_01[!pg]);
printf("    y2= %f",y2_01[!pg]);
printf("\n");
/* DYNAMIC */
ini_L: while (1)
{
suspende();
if (!(t_01>10))
{ tfc_01 = 0;
while (!tfc_01)
{
pg = !pg;
ADs(  1,pg);
x = y1_01[pg];
Esp_F_Fase();
update(&t_01, dt_01, method_01, &fm_01, &tfc_01, &iflag_01);
}
/* COMMUNICATION */
if (comm(cinterval, dt_01, &cont_L))
{
printf("\n");
printf("    t= %f",t_01);
printf("    x= %f",x);
printf("    y1= %f",y1_01[!pg]);
printf("    y2= %f",y2_01[!pg]);
printf("\n");
}
}
/* TERMINAL */
else
{
printf("\n");
printf("    t= %f",t_01);
printf("\n");
termina();
}
}
}
```

Fig. B.2 - Arquivo "sl_ct.c".

(continua)

```
main()
{
start(sistlinear);
suspende();
}

void ADs(int palavra_controle, int pg)
{
AD01(palavra_controle, pg);
AD02(palavra_controle, pg);
}
```

Fig. B.2 - conclusão.

função. Essas variáveis incluem algumas variáveis de controle pré-declaradas e as variáveis comuns (declaradas com o uso da palavra reservada 'var'). As constantes, as variáveis de controle que devem ser visualizadas pelos programas parciais dos ADs (tais como t e dt), e as variáveis do tipo State e Global declaradas internamente ao processo, são colocadas no arquivo "sl.dec", com um sufixo que indica à qual processo pertencem.

O código da região Initial é inserido no início da função "sistlinear". Em seguida, é inserido o código da região Dynamic, com as estruturas de controle discutidas no capítulo 5.

A cada início de iteração da região Dynamic, é invocado o monitor, através da primitiva Suspende, que vai permitir que outros processos, se existirem, sejam também ativados.

Neste exemplo, não foi utilizado o comando de controle "Terminate", portanto, a condição de término padrão ($t \triangleright 10$) é utilizada, como refletido no comando "if (!(t_01 ▷ 10))".

Note que todas as variáveis locais a esse processo, mas que devem ser visíveis para os programas parciais dos ADs são utilizadas com o sufixo "_01" (a declaração dessas variáveis está no arquivo "s1.dec").

Os comandos da região Dynamic descrevem uma Iteração Funcional. O número de Iterações Funcionais que serão executadas para completar uma Iteração Funcional Completa é controlado pelo comando "while (!tfc_01)". A variável de controle tfc é atualizada através da função "update", de acordo com o Método Funcional que está sendo utilizado, para refletir o término de uma Iteração Funcional Completa.

No início da execução de uma Iteração Funcional, são trocadas as cópias das variáveis do tipo State que serão utilizadas para armazenar os resultados gerados por essa Iteração Funcional, através do comando "pg = !pg" ("pg := not pg"). Em seguida, é disparada a execução de uma Iteração Incremental nos ADs, através da rotina "ADs(1,pg)". Enquanto os ADs realizam seus cálculos, o CT pode realizar os cálculos a ele atribuídos nesta Iteração Incremental. Depois disso, o CT aciona a rotina "Esp_F_Fase", para aguardar que todos os ADs terminem os cálculos. Se houver outras Iterações Incrementais, o ciclo se repete: disparo da execução dos comandos dos ADs; cálculo das tarefas do CT; espera do término do cálculo nos ADs.

Finalmente, as variáveis de controle são atualizadas com a invocação da rotina "update".

As variáveis de controle atualizadas são:

1) t: atualizada de acordo com o Método Funcional utilizado, para refletir a mudança incremental da variável independente.

2) fm: atualizada de acordo com o Método Funcional utilizado, para indicar a próxima fase do Método Funcional a ser realizada.

3) **tfc:** atualizada de acordo com o Método Funcional utilizado, para indicar o fim de uma Iteração Funcional Completa.

4) **iflag:** variável que indica se é a primeira vez que o Método Funcional está sendo chamado. É atualizada com valor "false" a primeira vez que a rotina "update" é chamada e permanece com esse valor até o término da região Dynamic.

Depois de terminada uma Iteração Funcional Completa, é verificado se este é um ponto de comunicação, conforme definido pela variável "cinterval", através da rotina "comm", e, em caso positivo, os comandos da região Communication são executados.

Quando a condição de término da região Dynamic for verdadeira, serão executados os comandos da região Terminal, e o processo será desativado através da invocação da primitiva Termina do monitor.

Além da rotina que contém o processo Background, é gerado o programa de ativação dessa rotina, que, através da invocação da primitiva Start do monitor, coloca o processo na tabela de processos do monitor com estado de "pronto". Em seguida, o programa de ativação é suspenso, através da invocação da primitiva Suspende, e o monitor dispara a execução do processo Background em questão.

Neste arquivo é colocada também a rotina ADs(palavra_controle, pg), que ativa a execução dos comandos dos ADs.

b) **Arquivos "sl_01.c" e "sl_02.c":**

Os arquivos "sl_01.c" e "sl_02.c" contêm os programas parciais gerados para o AD 1 e para o AD 2, respectivamente. A figura B.3 mostra o arquivo "sl_01.c".

As variáveis declaradas em "sl.dec" são referenciadas nos programas parciais dos ADs através do arquivo "sl.ext", que contém as mesmas declarações do arquivo "sl.dec", precedidas da palavra reservada "extern".

Os programas parciais de cada AD são ativados através da rotina ADs(palavra_controle,página). O parâmetro "palavra_controle" vai ser utilizado para selecionar os comandos que devem ser executados conforme a Iteração Incremental que o CT estiver executando. O parâmetro "página" vai determinar qual cópia das variáveis do tipo State será utilizada para armazenar os resultados gerados.

```
#include "sl.ext"
void AD01(int palavra_controle, int pg) {
switch (palavra_controle) {
case  1:
y1_01[!pg] = integ(a*y1_01[pg]+b*y2_01[pg], y1_01[pg], dt_01,
                   method_01, fm_01,  0, iflag_01);
Termina_Fase();
break;
}}
```

Fig. B.3 - Arquivo "sl_01.c".

A chamada "integ(expressão)", feita pelo programador é transformada pelo tradutor em:

integ(expressão,condição inicial, dt, método, fm, número, iflag)

O parâmetro "dt" é o incremento do passo de integração. Os parâmetros "método" e "fm" servem para determinar qual a fase do método de integração que está sendo calculada no momento. O parâmetro "número" serve para determinar a posição, em um vetor de trabalho, onde os resultados parciais do processo de cálculo são armazenados. O

parâmetro "iflag" indica se é a primeira Iteração Funcional em que a rotina "Integ" está sendo invocada.

c) **Arquivos "sl.dec" e "sl.ext":**

O arquivo "sl.dec" reúne as declarações que devem ser visíveis aos programas parciais do CT e dos ADs. Neste arquivo são colocadas as constantes e variáveis do tipo State e Global declaradas à nível de programa e as constantes e variáveis do tipo State e Global declaradas internamente aos processos e acrescidas de um sufixo que as relaciona aos processos no qual foram declaradas. Neste arquivo também são colocadas as variáveis de controle pré-declaradas que devem ser visíveis aos programas parciais do CT e dos ADs. O arquivo "sl.dec" é mostrado na Figura B.4

```
#include <math.h>
double integ(double f, double y, double dt, int method,
             int fm, int n, int iflag);
void update(double *t, double dt, int method, int *fm,
            int *tfc, int *iflag);
int comm(double cinterval, double dt, double *cont);
void ADs(int palavra_controle, int pg);
void Esp_F_Fase();
double c_1[100];
double const a=0;
double const b=1;
double const c=-1;
double const d=0;
double dt_01=0.1;
double t_01=0;
int tfc_01;
int method_01=0;
int fm_01=0;
int iflag_01=1;
double y1_01[2],y2_01[2];
double x_01;
```

Fig. B.4 - Arquivo "sl.dec".

O arquivo "s1.ext" contém as mesmas declarações do arquivo "s1.dec", acrescidas da palavra reservada "extern". Esse arquivo é utilizado para que os programas parciais dos ADs possam fazer referência às variáveis declaradas no CT.

d) **Arquivo "s1.bat":**

O arquivo "s1.bat" contém os comandos necessários para ativação do processo de compilação dos programas parciais e de conexão do código objeto destes com as rotinas de simulação do ambiente paralelo, com as rotinas da biblioteca L e com as rotinas do monitor para simulação da execução concorrente dos processos L.

e) **Arquivo "s1.lst":**

O arquivo "s1.lst" contém a listagem do programa fonte em L (Fig. B.1), acusando os erros de compilação que foram detectados.

B.4 - EXECUÇÃO

Depois que o programa em L é submetido ao tradutor, basta digitar o comando "s1", para que seja gerado o arquivo de código executável "s1.exe". (O comando "s1" ativa a execução dos comandos do arquivo "s1.bat").

A execução desse arquivo gera os resultados mostrados na Figura B.5.

Note que a variável x não tem o mesmo valor da variável y1, como se poderia supor à primeira vista. Na realidade, o valor atribuído à variável x é o valor da variável y1 na Iteração Funcional

precedente àquela que provocou a execução dos comandos da região Communication.

```
t= 0.000000    x= 0.000000    y1= 0.000000    y2= 1.000000
t= 0.100000    x= 0.099833    y1= 0.099833    y2= 0.995004
t= 0.200000    x= 0.198669    y1= 0.198669    y2= 0.980067
t= 0.300000    x= 0.295510    y1= 0.295520    y2= 0.955336
t= 0.400000    x= 0.389408    y1= 0.389418    y2= 0.921061
t= 0.500000    x= 0.479416    y1= 0.479426    y2= 0.877583
t= 0.600000    x= 0.564633    y1= 0.564642    y2= 0.825336
t= 0.700000    x= 0.644209    y1= 0.644218    y2= 0.764842
t= 0.800000    x= 0.717348    y1= 0.717356    y2= 0.696707
t= 0.900000    x= 0.783320    y1= 0.783327    y2= 0.621610
t= 1.000000    x= 0.841465    y1= 0.841471    y2= 0.540302
t= 1.100000    x= 0.891202    y1= 0.891207    y2= 0.453596
                  ---         ...= 0.932039   y2= 0.362358
t= 9.700000    x= -0.271750   y1=
t= 9.800000    x= -0.366469    y1= -0.366479    y2= -0.930426
t= 9.900000    x= -0.457526    y1= -0.457535    y2= -0.889191
t= 10.000000    x= -0.544012    y1= -0.544021    y2= -0.839072
t= 10.000000
```

Fig. B.5 - Resultado da execução do programa em L.