UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA
Dept.Engenharia Electrotécnica e de Computadores

Compilador de Grafcet para os Autómatos SIEMENS S100U e S115

Armando Jorge Miranda de Sousa

Jorge Miguel Oliveira Freitas

José Miguel Basto Machado

Julho de 1993

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Departamento de Engenharia Electrotécnica e de Computadores
Rua dos Bragas, 4099 Porto Codex, PORTUGAL
Telef. 351-2-317105/107/412/457 · Telex 27323 FEUP P · Telefax 351-2-319280

Direcção Geral do Ensino Superior
Subprograma 4 - PRODEP
Medida 4.3 - Estágios Profissionais para Bacharelato, Licenciatura e Pós-gradução

Parecer

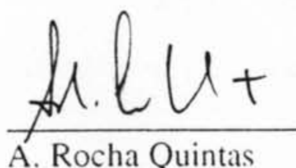
Assunto: Estágios de:
Armando Jorge Miranda de Sousa
Jorge Miguel Oliveira Freitas
José Miguel Basto Machado

Na qualidade de supervisores dos estagiários mencionados em epígrafe cumpre-nos informar:

A dedicação e empenho demonstrados durante o decorrer do estágio foi muito satisfatória.
Efectuando uma análise do trabalho desenvolvido verifica-se que os estagiários não só cumpriram os objectivos inicialmente propostos, para o projecto em que se envolveram, como ainda mostraram em termos científicos alguma capacidade de inovação.
Assim somos do parecer que foram cumpridos os objectivos que se pretendiam alcançar com os estágios propostos.

Porto, 3 de Maio de 1993

Os orientadores


A. Rocha Quintas

Raul Filipe T. Oliveira

Trabalho Desenvolvido no Âmbito do PRODEP

Compilador de Grafcet para os Autómatos SIEMENS S100U e S115

Sob a orientação do Eng. Raúl Oliveira

Autores:

Armando Jorge M. Sousa

Jorge Miguel O. Freitas

José Miguel B. Machado

Universidade do Porto
Faculdade de Engenharia
Biblioteca 4
Nº
CDU 621.3(047.3)/LEEC/1992/SOUSA vol.1
Data 02/10/2009

Índice

1. Sumário 1

2. Definição do trabalho 2

3. Concepção Ideológica 3

4. Modelização 4

4.1. Etapas 4

4.2. Acções 5

4.3. Transições 5

4.4. Macro-Etapa 6

5. Compilador 7

6. Algoritmos de compilação 8

6.1. Algoritmo geral 8

6.1.1. Algoritmo da geração do código das condições de evolução 9

6.1.2. Algoritmo da activação/desactivação das etapas 10

6.1.2.1. Algoritmo para as acções normais 11

6.1.3. Algoritmo de geração do código das acções especiais 12

6.1.3.1. Algoritmo para as acções impulsionais 13

6.1.3.2. Algoritmo para as acções tipo L e tipo D 14

6.1.3.3. Algoritmo para as acções condicionais 15

7. Síntaxe do Ficheiro de Texto 16

7.1. Notações: 16

7.2. Comandos disponíveis: 16

7.2.1. Definição do Grafcet 16

7.2.2. Definição das linhas de E/S do A.P. 17

7.2.3. Definição de Etapas e seus Atributos 17

7.2.4. Definição das Acções, seus tipos 17
7.2.5. Definição das Acções de Forçagem 18
7.2.6. Definição Receptividades 18
7.2.7. Facilidades não implementadas 18
7.3. Comentários: 19
7.4. Exemplos: 20
8. Implementação 22
8.1. Dados: 22
8.1.1. grafcet.h 22
8.1.1.1. Transições 22
8.1.1.2. Etapas 24
8.1.1.3. Acções 25
8.1.1.4. Variáveis Auxiliares 26
8.1.1.5. Próximo endereço 26
8.1.2. grafglob.h 27
8.1.2.1. Listas Ligadas 27
8.1.2.2. Recursos usados no AP 27
8.1.2.3. Anti-estouro dos FBs 27
8.1.2.4. Ficheiros 28
8.2. Funções de suporte 28
8.2.1. suporte.c 28
8.2.1.1. Alocação das variáveis globais 28
8.2.1.2. Ficheiros 28
8.2.1.3. Recursos ocupados no AP 29
8.2.1.4. Anti-estouro dos FBs 29
8.2.1.5. Criar tuplos de etapas 29
8.2.1.6. Criar tuplos de transições 30

8.2.1.7. Criar tuplos de acções 30

8.2.1.8. Criar tuplos de variáveis auxiliares 31

8.2.1.9. Procura acção pela etapa 31

8.2.1.10. Procura transição pelo seu número 32

8.2.1.11. Procura etapa pelo seu número 32

8.2.1.12. Procura etapa pelo seu endereço 32

8.2.1.13. Procura variável pelo seu texto 32

8.2.1.14. Procura etapa pelo seu endereço 33

8.2.1.15. Libertar tuplos 33

8.2.1.16. Escrever etiquetas e anti-estouro dos FBs 34

8.3. Leitura do ficheiro 34

8.3.1. Lefich.c 34

8.3.1.1. Precompilação de forçagens 35

8.3.1.2. Precompilação de Acções 36

8.3.1.3. Precompilação de transições 39

8.3.1.4. Leitura genérica do ficheiro 41

8.4. Cálculo das condições de Evolução 45

8.4.1. pilha.h 45

8.4.1.1. Definição de operações e números da pilha 45

8.4.1.2. Definição de uma pilha 46

8.4.1.3. Funções que acedem à pilha 46

8.4.1.4. Funções que produzem a NPI 46

8.4.1.5. Funções trabalho com os elentos da pilha 46

8.4.2. comptran.c 47

8.4.2.1. Compilação geral de transições 47

8.4.3. pol_inv.c 50

8.4.3.1. Push de uma variável para a pilha 50

8.4.3.2. push de uma operação para a pilha ... 51

8.4.3.3. prioridade de uma certa operação ... 51

8.5. Activação / Desactivação de etapas ... 52

8.5.1. des_acti.c ... 53

8.5.2. ajuda.c ... 55

8.5.2.1. susceptibilização das transições de saída ... 55

8.5.2.2. desusceptibilização das transições de saída ... 56

8.5.2.3. disparo dos temporizadores associados às transições ... 56

8.5.2.4. acções do tipo set/reset (memorizadas) ... 57

8.5.2.5. reset das acções normais ... 58

8.5.2.6. reset das acções ... 59

8.6. Acções ... 59

8.6.1. Acções Normais ... 60

8.6.2. Acções Especiais ... 64

8.6.2.1. Acções Impulsionais ... 64

8.6.2.2. Acções Memorizadas ... 67

8.6.2.3. Acções Tipo L ... 67

8.6.2.4. Acções Tipo D ... 69

8.6.2.5. Acções Condicionais ... 70

8.7. Escrita de OBs e PBs ... 72

8.8. Programa Principal ... 75

9. Grafcets de teste ... 76

9.1. Metodologia ... 76

9.2. Grafcets testados ... 76

9.2.1. Grafcet com acções simples ... 76

9.2.2. ... 78

9.2.2. Grafcet com acções de duração limitada ... 79

9.2.3. Grafcet com forçagens ... 81

9.2.4. Grafcet com acções set/reset ... 85

9.2.5. grafcet com acções condicionais ... 87

9.2.6. grafcet com acções condicionais temporizadas ... 89

10. Listagem dos Erros ... 92

10.1. Erros do ficheiro lefich.c ... 92

10.2. Erros do ficheiro comptran.c ... 93

10.3. Erros do ficheiro pol_inv.c ... 93

11. Bibliografia ... 95

1. *Sumário*

- Concepção e idealização geral do trabalho
- Modelização
- Definição de dados
- Estudo preliminar da implementação: Definição das estruturas de dados

2. *Definição do trabalho*

Este trabalho destina-se a construir uma ferramenta que permita meios de implementar optimizadamente a linguagem Grafcet nos Autómatos Programáveis da Siemens, CPUs S100U e S115.

A metodologia a seguir foi preconizada na Tese de Mestrado do Eng. Raul Oliveira [1], professor na FEUP, devidamente adaptada pelos autores.

A linguagem grafcet implementada é o mais próxima possivel da que vem referenciada em [2] "Le Grafcet, de nouveau concepts", da GREPA, edição Cepadues, presente na biblioteca do DEEC com a ref. A.C 1.7:11.

Como principais características desta moderna definição do grafcet destaca-se:

- Macro-Etapas
- Receptividades temporizadas
- Acções temporizadas
- Hierarquia entre grafcets

Como plataforma de implementação escolheu-se os computadores IBM-PC compatíveis, pelo sua disponibilidade, pelo seu baixo preço e suficiente performance. A linguagem para o compilador é a linguagem C e o software utilizado o Turbo C++ na sua versão 1.0 .

3. *Concepção Ideológica*

Desde cedo se decidiu que o conjunto de ferramentas deveria ser o seguinte:

- Um editor gráfico
- Um compilador
- Uma ferramenta de transmissão para o A.P.

Decidiu-se deixar editor gráfico para uma segunda fase, visto uma interface em ficheiro de texto ser suficientemente fácil de utilizar, permitindo que um utilizador escreva sem dificuldade, a descrição do grafcet directamente para o compilador.

O editor gráfico deveria então implementar uma entrada gráfica, de acordo com as especificações do problema e deixar um ficheiro de saída de acordo com a síntaxe apresentada na capítulo 7.

O compilador deve aceitar como entrada o ficheiro de texto, e escrever um outro ficheiro, de saída, no formato próprio para ser transmitido para o Autómato Programável.

Devido a problemas inesperados, o interface de comunicações não foi implementado. Muitas tentativas se fizeram mas a arquitectura dos A.P. Siemenes revelou-se fechada, impedindo a transmissão de programas para dentro do autómato. Seria apenas necessário um compilador de ASCII para linguagem STEP 5 mas tal software não existe em Portugal. Decorreram vários esforços para que as ferramentas necessárias fossem disponibilizadas pela Siemens, sem qualquer resultado.

4. Modelização

O grafcet é uma linguagem de programação/modelização gráfica de sistemas complexos.

A versatilidade do grafcet resulta de se poder implementar facilmente as noções de partilha de recursos e paralelismo na execução.

Os elementos básicos desta linguagem são as etapas e as transições.

Uma etapa representa um estado elementar do sistema. Uma etapa activa faz com ocorram certas acções, inerentes ao estado elementar que se reproduz nesta etapa.

O conjunto das etapas activas representa o estado de todo o processo, pois cada etapa representa cada elemento do sistema global.

A evolução do estado do grafcet, equivalente ao estado global do processo, é permitida através das transições que, quando transpostas (satisfeita uma condição de evolução), fazem avançar o processo em paralelo.

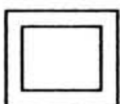
As regras de comportamento do grafcet são exaustivamente descritas em [2] e não serão aqui repetidas.

Neste capítulo faz-se a recolha de todos elementos da linguagem grafcet:

- *Etapas* (suas variantes)
- Dentro de cada etapa, sintaxe das suas *acções*
- Dentro das acções, sintaxe de cada *ordem*
- *Transições*
- Síntaxe da *receptividade* de cada transição (condição de evolução)
- Síntaxe de cada elemento da receptividade

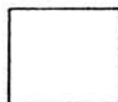
Descreve-se a seguir cada um dos elementos que fazem parte do grafcet, junto com a sua representação gráfica. Para pormenores acerca do significado consulte, p. f., a bib. [2]; para a sua síntaxe de programação, refira-se ao capítulo 7.

4.1. Etapas

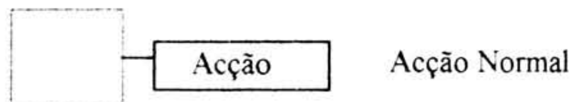


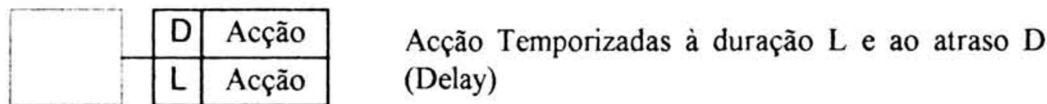
Etapa inicial

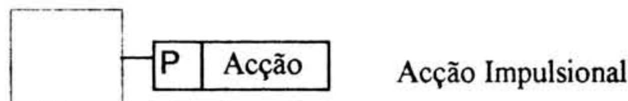
Etapa normal

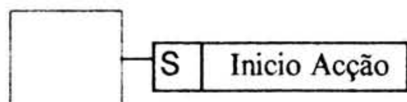


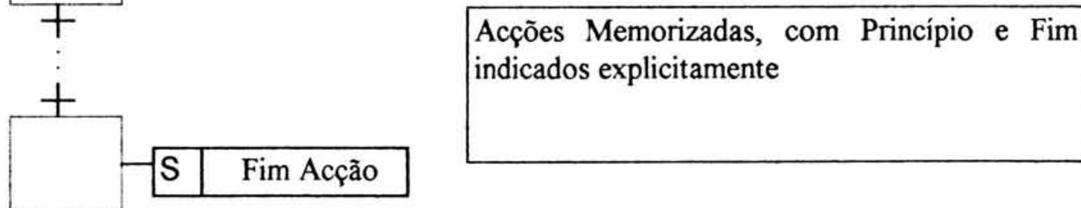
4.2. Acções

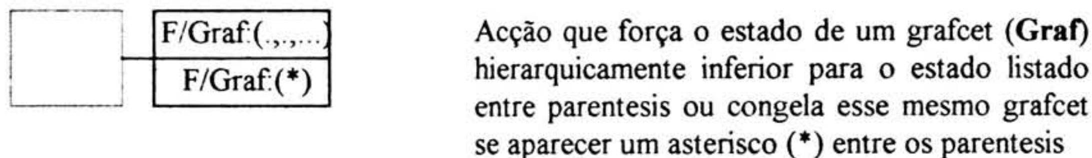


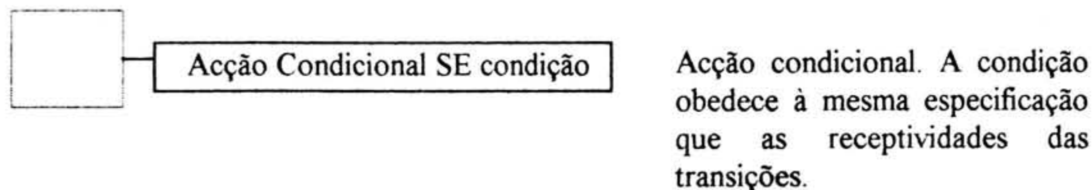




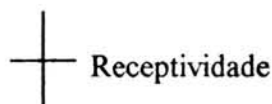








4.3. Transições



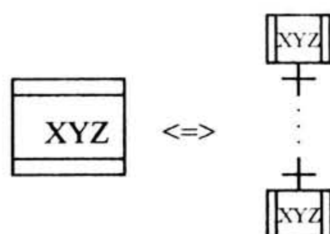
Nas receptividades aceitam-se os seguinte símbolos:

-Xn que representa o estado actual da etapa nº **n**

-t/etapa_disparo/tempo representa um temporizador, atrasado ao disparo **tempo**, numa contagem iniciada na activação da etapa **etapa_disparo**

-↗var e **↘var** representam as transições de flanco ascendentes e descendentes (edge trigger) da variável **var**

4.4. Macro-Etapa



Macro Etapa e respectivo grafcet equivalente, mostrando as etapas iniciais e finais e a respectiva notação gráfica

5. *Compilador*

Este módulo do programa deve aceitar como uma entrada um ficheiro de texto e apresentar na saída um outro em STEP 5 (ASCII), que deverá ser transmitido posteriormente para o AP.

O primeiro passo a definir é procurar uma representação apropriada para o ficheiro de entrada, que representa uma estrutura gráfica grafcet.

Um grafcet é um diagrama constituído por etapas e receptividades, obrigatoriamente numa sequência alternada e com ligações múltiplas tanto do lado das transições como do lado das etapas. Para cada transição existe então um conjunto arbitrário de etapas de entrada e outro conjunto arbitrário de etapas de saída; Para cada etapa existe um conjunto arbitrário de transições de entrada e outro conjunto arbitrário de transições de saída.

Existem então várias maneiras de representar o grafcet:

1.Para cada etapa listar as transições de entrada e as transições de saída

2.Para todas as transições listar as etapas de entrada e as etapas de saída

3.Para todas as etapas e transições listar respectivamente as transições de entrada e as etapas de entrada

4.Para todas as etapas e transições listar respectivamente as transições de saída e as etapas de saída

5.Para todas as etapas e transições listar respectivamente as transições de saída e as etapas de entrada

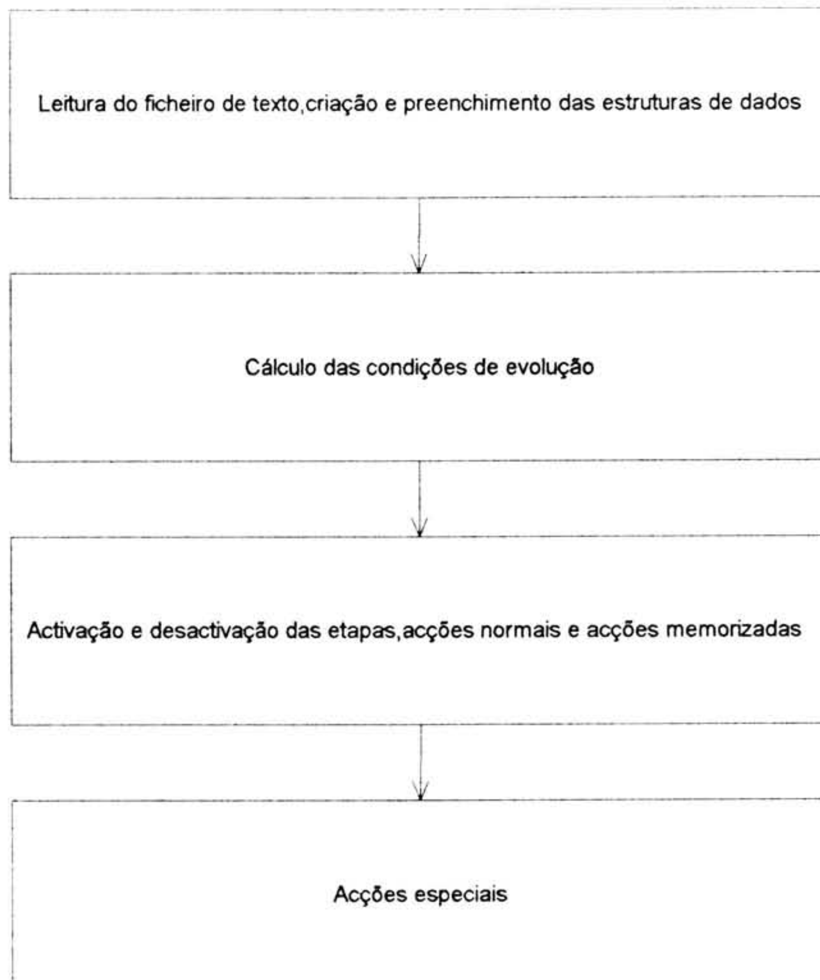
6.Para todas as etapas e transições listar respectivamente as transições de entrada e as etapas de saída

Verificou-se que seria mais intuitivo usar a representação 1, que se pode designar como sendo orientada às etapas. Chegou-se a considerar a hipótese de implementar mais que um tipo de entrada mas tal ideia foi abandonada pois conduziria a uma certa confusão.

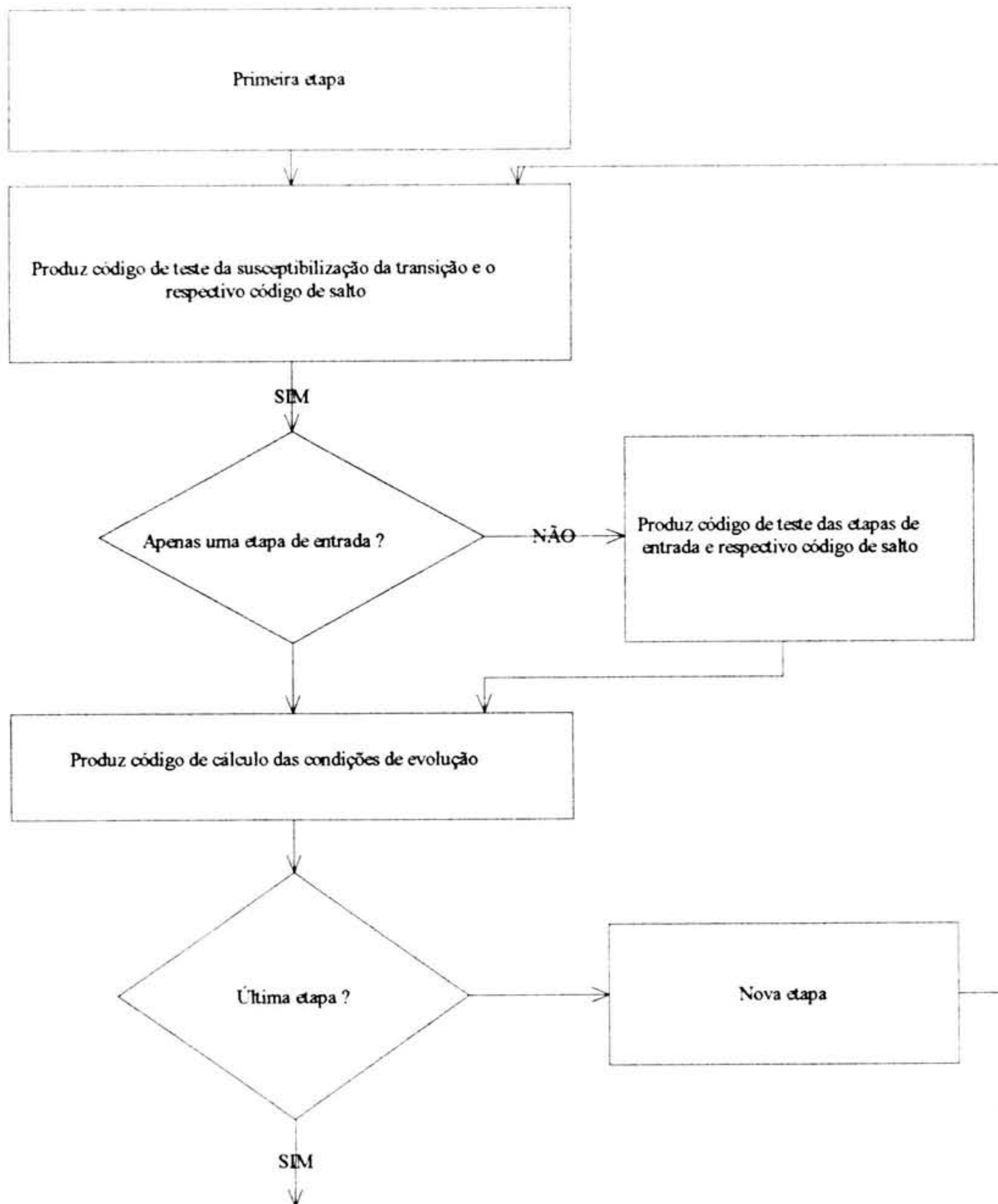
O ficheiro de texto de entrada deve então conter a descrição do grafcet orientada às etapas, quer dizer, o ficheiro descreve para cada etapa o conjunto de transições de entrada e de saída, definindo assim univocamente (e sem redundância) o grafcet a representar. Este ficheiro pode ser gerado por um processo automático ou o utilizador pode escreve-lo com um qualquer processador de texto.

6. Algoritmos de compilação

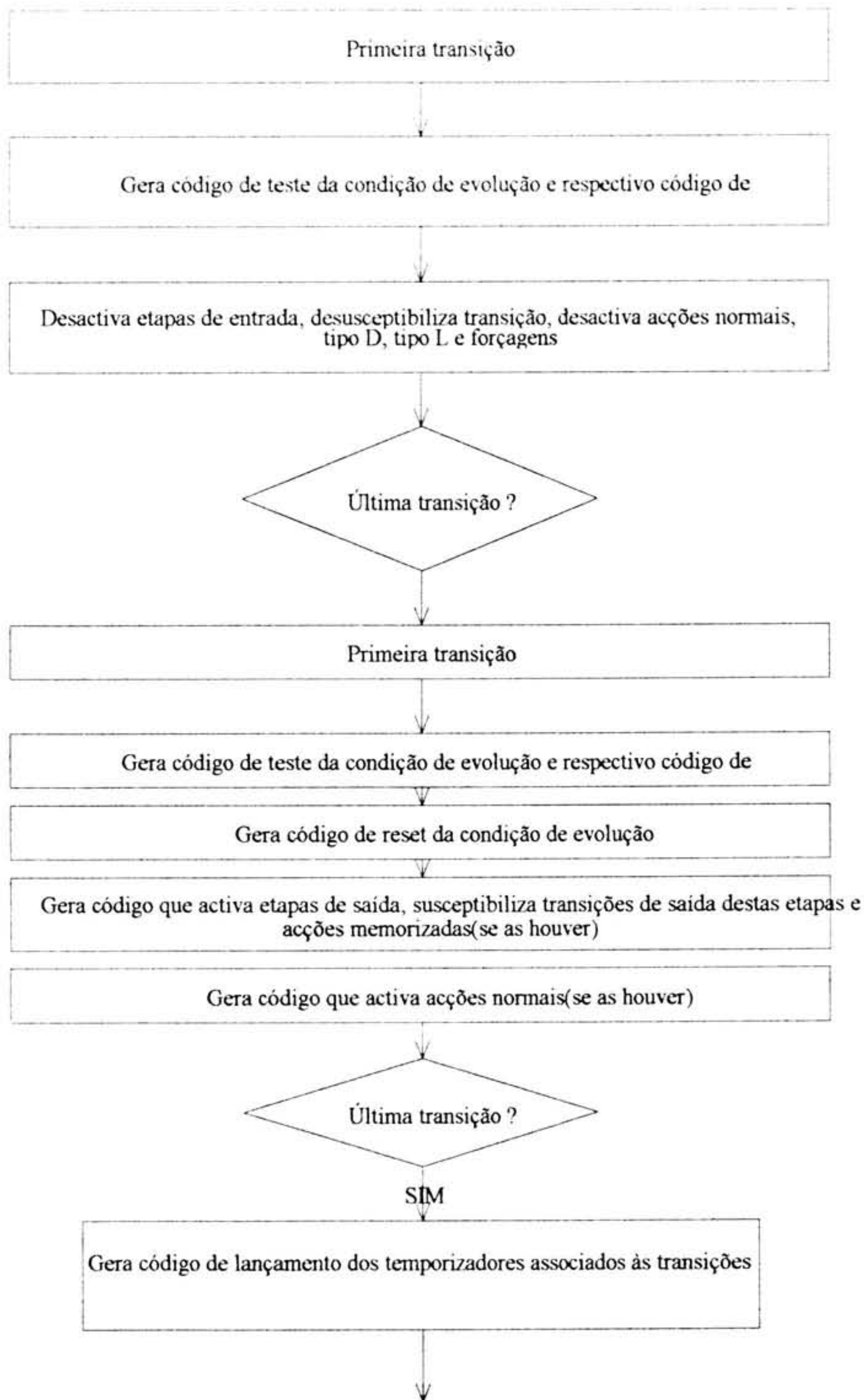
6.1. Algoritmo geral



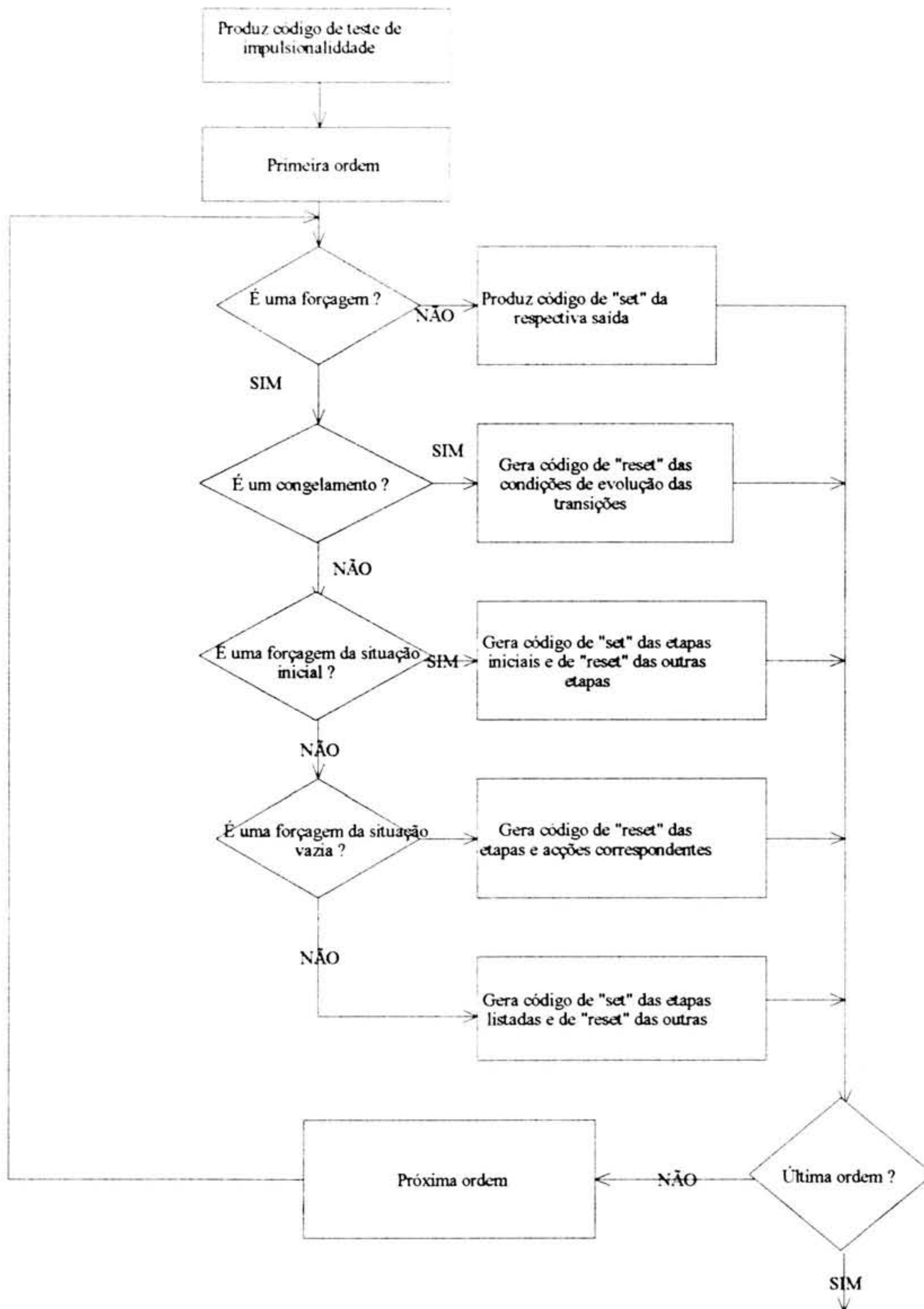
6.1.1. Algoritmo da geração do código das condições de evolução



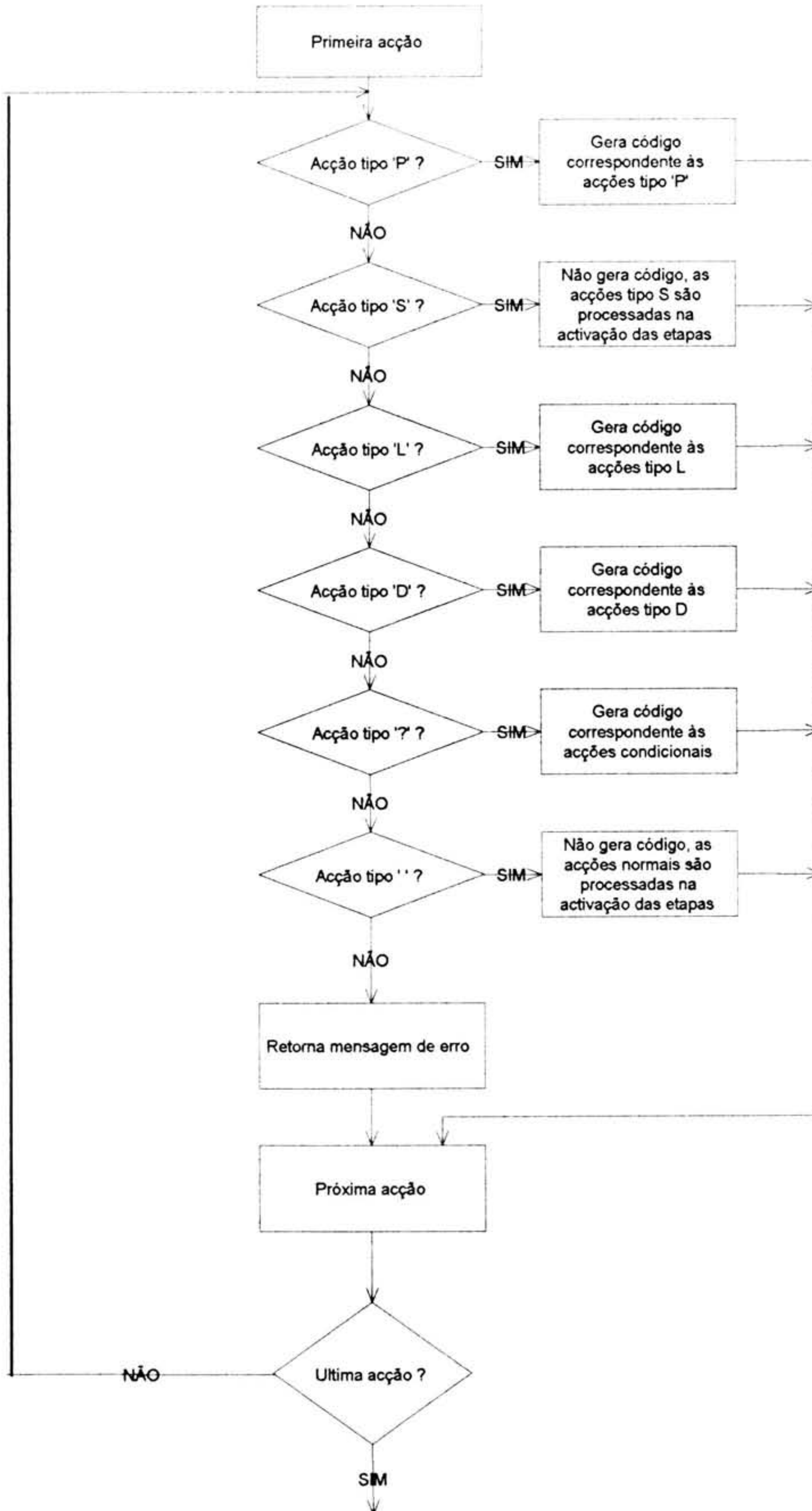
6.1.2. Algoritmo da activação/desactivação das etapas



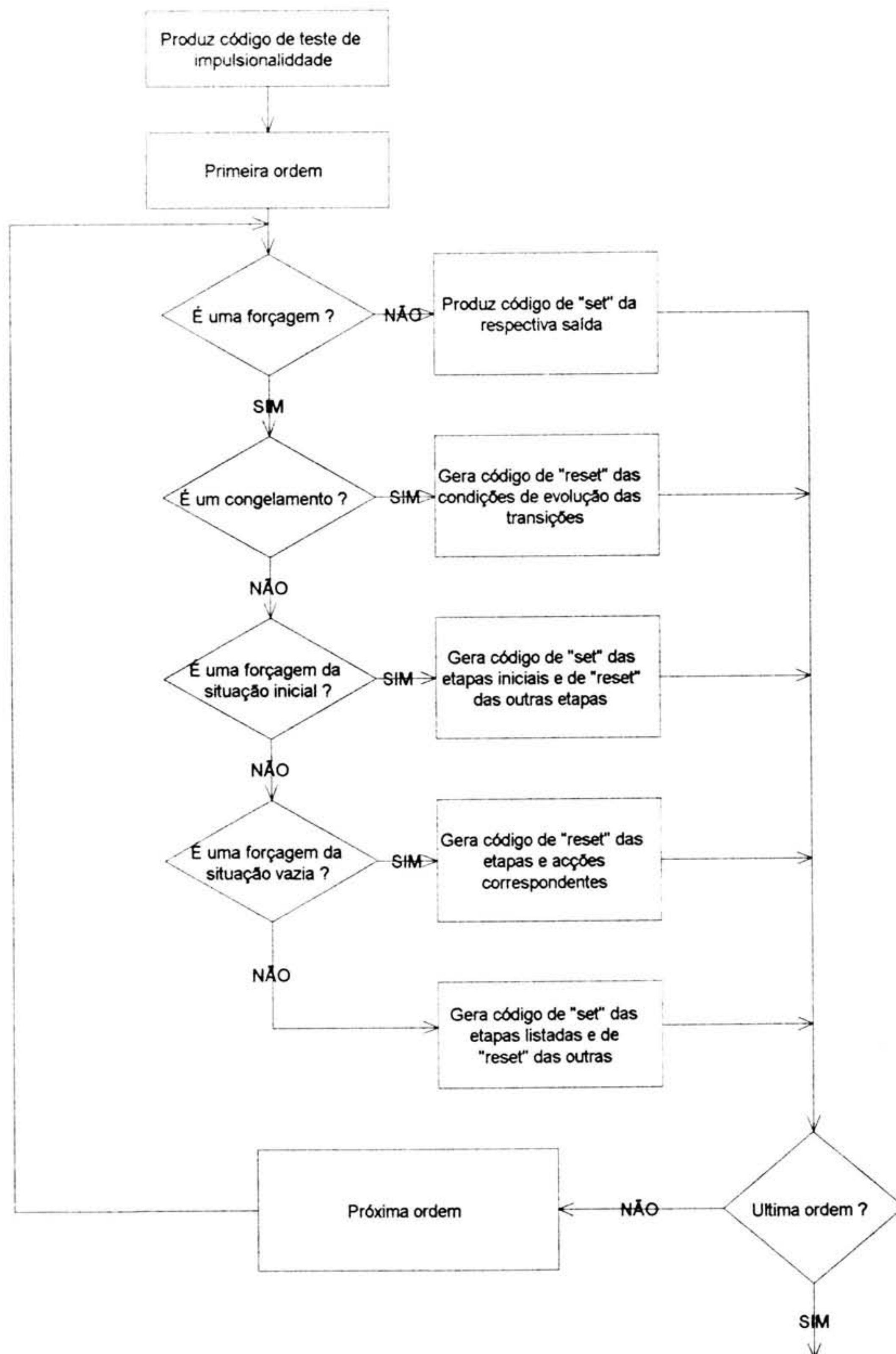
6.1.2.1.Algoritmo para as acções normais



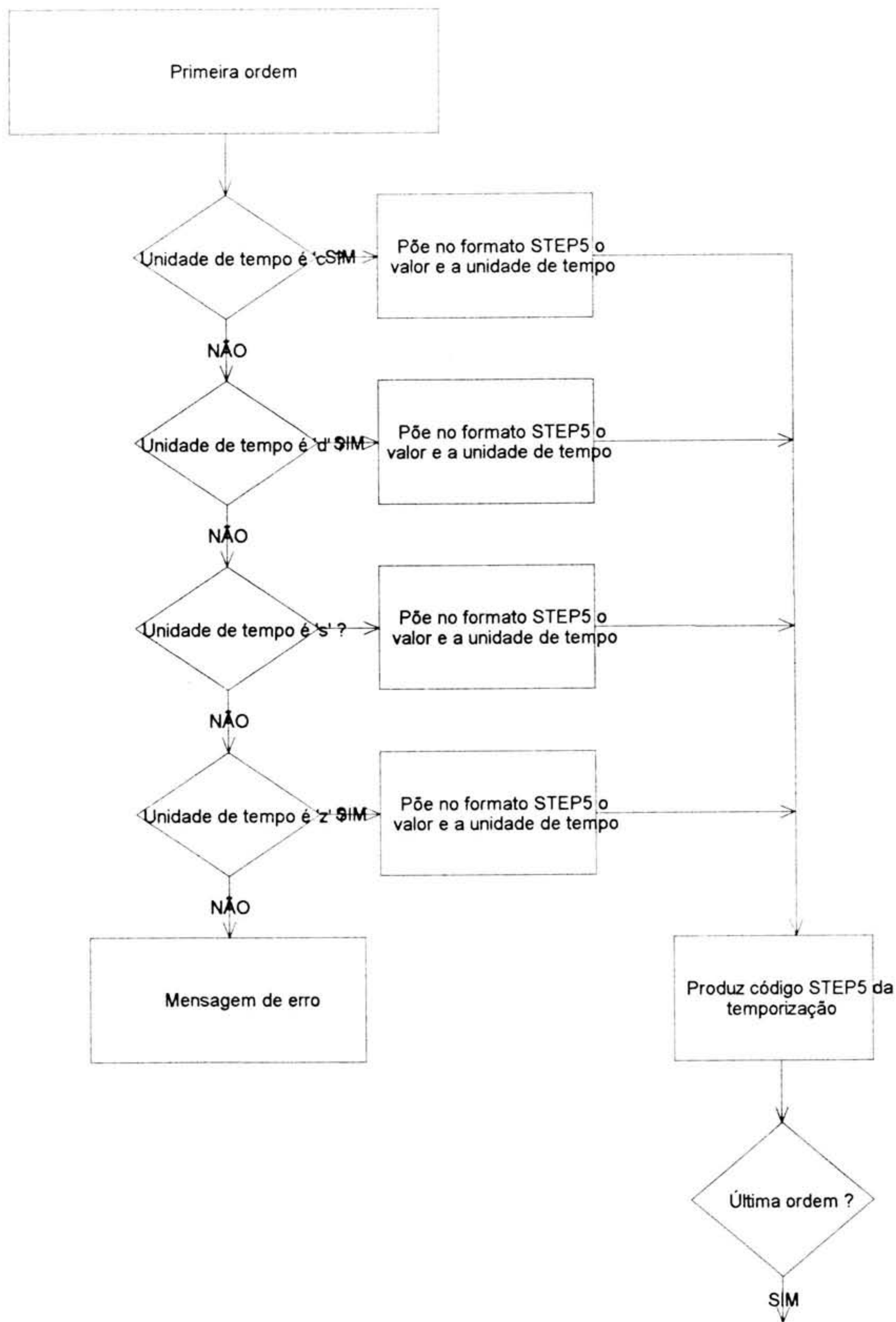
6.1.3. Algoritmo de geração do código das acções especiais



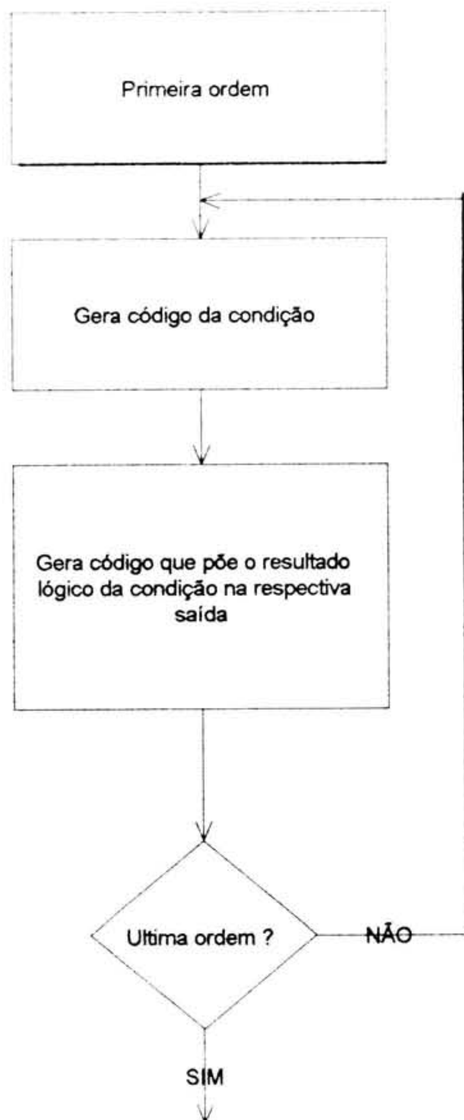
6.1.3.1.Algoritmo para as acções impulsionais



6.1.3.2.Algoritmo para as acções tipo L e tipo D



6.1.3.3.Algoritmo para as acções condicionais



7. Síntaxe do Ficheiro de Texto

O problema tratado nesta secção é o de representar numa linguagem intuitiva mas não ambígua todas os elementos de síntaxe do grafcet.

O precompilador funciona numa estrutura flexível onde a ordem em que aparecem definidas as etapas dentro de um grafcet, as transições de entrada ou saída, as definições das variáveis, etc. pode ser qualquer. A única obrigação está na ordem dos grafcets: o grafcet de hierarquia mais baixa deve aparecer em primeiro lugar. Todas as forçagens devem-se referir a grafcets já completamente definidos. A descrição de um grafcet não pode ser interrompida e depois retomada.

Como se disse atrás, o ficheiro de entrada é orientado às etapas, o que quer dizer que para cada etapa se devem declarar todas a transições de saída e de entrada.

7.1. Notações:

A notação utilizada para a descrição do ficheiro é:

'_' equivale a um espaço obrigatório. A existência de outros espaços que não estes (obrigatórios) não está documentada e é desaconselhada.

{param1,param2,param3} significa escolha multipla de um só dos elementos listados

param1, param2,... significa que a acção a tomar recairá sobre todos os parâmetros, de número qualquer, separados por vírgula ',' e/ou ponto-e-vírgula ';'

7.2. Comandos disponíveis:

Os comandos devem aparecer um por linha.

Quando existe um número indeterminado de parâmetros nessa linha, os caracteres de comando são separados do corpo de parâmetros por um espaço obrigatório. É obrigatório aparecer pelo menos um parâmetro.

Não devem existir espaços no principio nem no fim das linhas.

Podem existir linhas em branco no ficheiro.

Os comandos para o ficheiro de texto são:

7.2.1. Definição do Grafcet

GNome- Declara o inicio da definição do grafcet **Nome**, passa a ser este o grafcet corrente. Este comando pode não existir. A descrição de um grafcet não pode ser interrompida e depois retomada.

7.2.2. Definição das linhas de E/S do A.P.

:Qx.y=Variavel - Atribui a saída física **Qx.y** à variável lógica **Variavel**. Uma variável é uma sequência de quaisquer ASCII, excepto espaços, vírgulas e ponto e vírgulas. As variáveis não poderão começar pelos caracteres 't/' e 'X' pois estes símbolos têm significados especiais. As variáveis tem o tamanho máximo **MAXVARAUX** (ver grafcet.h-variáveis auxiliares). Para efeitos de comparações, as letras maiúsculas são diferentes das maiúsculas (estas variáveis são "case sensitive").

:Ix.y=Variavel - Atribui entrada física **Ix.y** à variável lógica **Variavel**. Apilcam-se os mesmos comentários e restrições do comando anterior.

7.2.3. Definição de Etapas e seus Atributos

#n - Cria a etapa nº **n**, se ela não existir ainda, define esta etapa como sendo a etapa corrente. Esta etapa pertence ao grafcet corrente e o seu número da etapa define-a univocamente para todos os grafcets. Não existe nenhuma relação obrigatória entre grafcets e etapas. A definição de uma etapa pode ser interrompida e depois retomada mas tal prática comprometeria a legibilidade do ficheiro de texto. O número das etapa é um inteiro n, em que 0<=n<=2^16-3.

I - Atribui a característica de "inicial" à etapa corrente. Esta etapa será activada nas inicializações do grafcet corrente

E_n1,n2,... - Define as transições de entrada da etapa corrente como sendo as transições **n1** e **n2**; Este comando pode aparecer repetidas vezes, acrescentado sempre mais transições de entrada; Não pode haver etapas nem transições com o mesmo número, mesmo em grafcets diferentes. A ordem individual das transições não é importante. A numeração das transições é fornecida pelo comando **Tn_receptividade**

S_n1,n2,... - Define as transições de saída da etapa corrente como sendo as transições **n1** e **n2**; Este comando pode aparecer repetidas vezes, acrescentado sempre mais transições de saída.

7.2.4. Definição das Acções, seus tipos

A_Ordem1,Ordem2,... - Define uma acção normal, com duas ordens: **ordem1** e **ordem2**. Estas ordens estão activas sempre que a etapa correspondente o estiver. Quaisquer ordens declaradas na mesma acção têm exactamente as mesmas propriedades, sendo ligadas e desligadas simultâneamente.

AD_OrdemEspecial1,OrdemEspecial2...=tempo {c,d,s,z} - acção com 2 ordens **"OrdemEspecial1"** e **"OrdemEspecial2"** sujeitas a um atraso de **"tempo"**, na unidade de **c**=centésimos de segundo, **d**=décimos, **s**=segundos, **z**=dezenas de segundos. Só serão permitidos tempos inteiros. Qualquer acção pode ser transformada em acção especial.

AL_OrdemEspecial1,OrdemEspecial2...=tempo {c,d,s,z} - o mesmo mas as ordens têm, agora a duração é de **"tempo"**. Só serão permitidos tempos inteiros. Qualquer acção pode ser transformada em acção especial.

AP_OrdemEspecial1,OrdemEspecial2... - a(s) ordens referidas serão executadas durante um único ciclo do AP: tratam-se de Acções Impulsionais. Qualquer acção pode ser transformada em acção especial.

AS_OrdemEspecial1,OrdemEspecial2... = {0,1} - Liga ou Desliga "**OrdemEspecial**" conforme aparece '=**1**' ou '=**0**' respectivamente

A?_OrdemCondicional=Condição,... - "**OrdemCondicional**" será executada sempre que o resultado da avaliação da condição lógica "Condicao" resultar verdadeira e a etapa associada estiver activa

7.2.5. Definição das Acções de Forçagem

(numa ordem normal ou impulsional)

F/nomegrafcet:(et1,et2,et3,...) - Força o estado do grafcet **nomegrafcet** (definido através da instrucção **G**) para o estado dado. O estado do grafcet é constituido pelas etapas listadas entre virgulas, que serão as únicas activas. Ver a secção de comentários.

(numa ordem normal ou impulsional)

F/nomegrafcet:(*) - Congela a evolução do grafcet **nomegrafcet**. As etapas e respectivas saídas continuam activas, o tempo continua a contar, apenas as transições não se podem transpôr. Ver a secção de comentários.

(numa ordem normal ou impulsional)

F/nomegrafcet:(I) - Força a que o grafcet **nomegrafcet** seja colocado no seu estado inicial. Ver a secção de comentários.

7.2.6. Definição Receptividades

Tn_receptividade - Atribui a (condição lógica) **receptividade** à transição nº **n**; Este comando pode aparecer em qualquer altura do ficheiro de texto. A condição da receptividade pode conter o símbolo '**Xn**' que simboliza o estado (activo ou inactivo) da etapa número **n**. São também permitidos os temporizadores sob a forma '**t/etapadisparo/tempo {z,s,d,c}**'. Este temporizador demorará **tempo** (**z**-dezenas de segundos, **s**-segundos, **d**-décimos de segundo, **c**-centésimos de segundo) até se tornar verdadeiro, desde o primeiro instante em que a etapa **etapadisparo** ficou activa. Dá-se novo disparo do temporizador sempre que a etapa é activada ou reactivada. É ainda possivel ter variaveis activadas aos flancos (edge-triggered) com os sinais **^var e /^var**, respectivamente para activações aos flancos ascendentes e descendentes da variável var. Ver a secção de comentários.

7.2.7. Facilidades não implementadas

MMacro - Qualifica esta como sendo uma macro etapa, equivale a dizer que todo o grafcet do nome **Macro** está situado nesta posição. O grafcet **Macro** deverá ter uma etapa de entrada (sem transições de entrada) e uma etapa de saída (sem transições de saída) que serão reconhecidas automaticamente. *NÃO IMPLEMENTADO.*

7.3. Comentários:

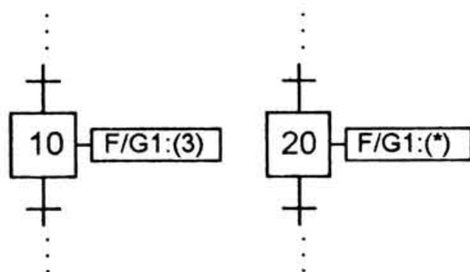
1.A forçagem de grafcets desliga todas as acções das etapas que estejam activas, excepto acções do tipo S.

2.Depois das forçagens pode haver temporizadores já disparados, activos, que não o estariam normalmente. Deve-se prever esta situação ao construir o grafcet.

3.Não se implementa a forçagem de situações caracterizadas através de um nome (exceptuando a situação inicial). Por favor represente sempre a totalidade do estado que pretende obter.

4.Após o congelamento de um grafcet, ele não poderá evoluir. Apenas se modificarão (em devida altura) as acções temporizadas e/ou impulsionais. O grafcet evoluirá quando a accção de forçagem deixar de estar activa ou se se forçar um outro estado no grafcet.

5.Uma possivel situação de ocorrência de erro é a seguinte:



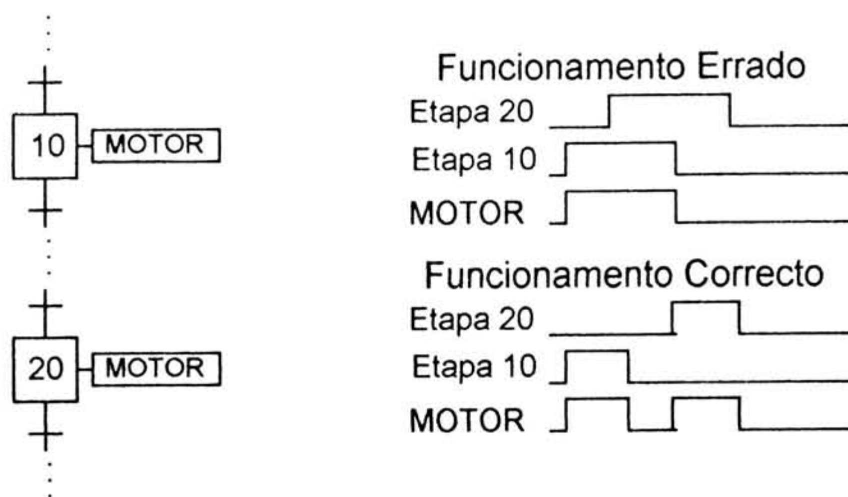
O estado final de G1 dependerá da ordem de activação/desactivação das etapas 10 e 20; Cada uma destas etapas funcionará bem se as acções forem mutuamente exclusivas.

6.O ciclo de um congelamento poderá demorar substancialmente mais que o tempo de ciclo normal, devido ao facto de todas as etapas e transições terem sido obrigadas a tomarem um certo estado. Este tempo não tem qualquer relação com a complexidade das receptividades mas sim com o nº de etapas do grafcet a forçar.

7.Os grafcet's devem aparecer no ficheiro de entrada por ordem crescente de hierarquia, i.e. só se pode forçar o estado de um grafcet já totalmente definido.

8.Pode-se definir etapas e transições por qualquer ordem; as etapas podem ser definidas duas vezes, em duas declarações **#n** diferentes mas não as transições: não pode haver mais que uma declaração **Tn** para cada transição número **n** no ficheiro.

9.Uma situção de erro possivel é quando duas etapas comandam (ao mesmo tempo) a mesma ordem:



A acção é comandada durante a activação/desactivação da etapa, pelo que aparecerá quando a 1ª etapa for activada, desaparecendo quando a 1ª etapa for desactivada.

10.Não está prevista a compilação de estruturas activadas aos flancos, p. ex. não é possivel uma receptividade ser ^(a+b) nem /^(a+b).

7.4. *Exemplos:*

Segue-se um exemplo simples e incompleto, onde se pretende ver apenas como definir as relações entre etapas e transições.

Aspecto gráfico	Codificação	Comentários
10, T1, 11, T2	#10	Cria Etapa 10
	I	Diz que é a etapa inicial
	E 2	Transição de entrada nº2
	S 1	Transição de saída nº 1
	#11	Cria Etapa 11
	E 1	Transição de entrada nº1
	S 2	Transição de saída nº 2

Segue-se um gracet completo:

Aspecto gráfico	Codificação	Comentários
	:I0.0=inicio	Define as variáveis de entrada, atribuindo-lhes as ligações respectivas
	:I0.1=cheio	
	:I0.2=vazio	
	:Q0.0=ENCHE	Declara a relação entre ordem e ligação física (saída)
	:Q0.1=ABRE	
	:Q0.2=ESVAZIA	
10 T1 inicio 11 ENCHE T2 cheio 12 P ABRE ESVAZIA T3 vazio	G Único	Define o nome do grafcet (opcional)
	#10	Cria Etapa nº 10
	I	É uma etapa Inicial
	E 2,3	A etapa 10 tem T2 e T3 como entradas
	S 1	Como saída apenas a transição T1
	#11	
	E 1	
	S 2	
	A ENCHE	Define a acção normal da etapa 11
	#12	
	E 2	
	S 3	
	A ESVAZIA	
	AP ABRE	Define a acção impulsional "ABRE"
	T1 inicio	Receptividade de T1
	T2 cheio	Receptividade de T2
	T3 vazio	Receptividade de T3

8. *Implementação*

Decidiu-se tornar a informação redundante, para um manuseamento mais fácil.

Criou-se na memória 4 listas ligadas, que contêm toda a informação relativa ao grafcet; o pré-compilador deve gerar toda esta base de dados para que o código a poder utilizar de seguida.

8.1. Dados:

A estrutura de dados foi arrajada de tal maneira que se crie redundância na representação interna de dados. Assim, para facilidade de escrita no programa e melhoria da performance do compilador, é criada internamente uma representação do grafcet orientada às etapas e uma outra orientada às transições. As duas representações são redundantes mas interessantes para facilitar a programação e melhorar a velocidade de execução.

Para albergar os dados, criou-se uma estrutura baseada em listas ligadas, com identificadores tipicos de campos chave, criando assim um estrutura do tipo de uma base de dados.

8.1.1. grafcet.h

Os dados relativos ao grafcet, tipos de variáveis e outras definições estão agrupados no ficheiro **grafcet.h**. Todas as listagens são agrupadas no fim deste documento, em apêndice.

Serão Repetidos os trechos de código considerados interessantes.

8.1.1.1. Transições

As transições são guardadas na seguinte estrutura:

```
#define N_MAX_ET 10
typedef struct t_transic {
        id_transic n;
        id_etapa e_entr          [N_MAX_ET];
        t_etapa_s_ptr * e_entr [N_MAX_ET];
        id_etapa e_saida         [N_MAX_ET];
        t_etapa_s_ptr * e_saida[N_MAX_ET];
        t_grafcet grafcet;
        char * texto;
        t_endereco sensib;
        struct t_transic * prox;
} t_transic;
```

Para facilidade de leitura do programa, todos os tipos de variáveis são definidas através de **typedef**s com nomes sugestivos precedidos do elemento "t_" que siginfica tipo de dados.

id_transic é um tipo de dados de todas as variáveis que se referem a números de transições. O campo **t_transic.n** é tal como um campo chave numa B.D., define univocamente a transição através de um número que é dado no ficheiro de entrada.

Tendo em conta que o ficheiro de entrada é orientado às etapas, a informação acerca das transições vai aparecendo a um ritmo irregular e por isso após a criação da transição, os campos são preenchidos de maneira irregular.

id_etapa é o tipo de dados que identifica etapas. Corresponde a um campo chave para as etapas na B.D. de etapas. O campo **t_transic.e_entr[]** é um array que contém todas as etapas de entrada de cada transição. Todos os tamanhos máximos de arrays estão definidos com **#define**'s, e estes são localizados nos ficheiros "Header File" respectivos (neste caso é o **grafcet.h**). Este array é preenchido até que um dos elementos seja marcado com sendo o elemento final: esse elemento é **NADA** e representa o fim do conjunto de etapas de entrada. Este array tem o tamanho de **N_MAX_ET**, o que lhe permite conter **N_MAX_ET-1** (desde 0 até **N_MAX_ET-2)** etapas de entrada (para cada transição).

As mesmas considerações se aplicam ao conjunto de etapas de saída **t_transic.e_saida[]**.

A par destes vectores de campos-chave, existem vectores de ponteiros para as estruturas do tipo **t_etapa_s_ptr,** que é basicamente o mesmo que **t_etapa.** Este vector é equivalente ao anterior mas contém directamente os ponteiros de memória das estruturas correspondentes, evitando assim estar permanentemente a percorrer a lista ligada à procura da estrutura da etapa em questão.

```
typedef struct t_etapa_s_ptr {
        id_etapa n;
        id_transic t_entr  [N_MAX_TR];
        void    * p_t_entr  [N_MAX_TR];
        id_transic t_saida [N_MAX_TR];
        void    * p_t_saida [N_MAX_TR];
        t_grafcet grafcet;
        struct t_etapa * prox;
        t_endereco endereco;
        t_endereco anterior;
        t_classif classif;
        int disparatemp : 1;
        int setreset    : 1;
} t_etapa_s_ptr;
```

A estrutura chama-se: tipo-etapa-sem-os-ponteiros-para-as-transições. Esta estrutura tem os mesmos campos que a estrutura das etapas, exceptuando que os ponteiros que seriam para a estrura (ainda não completamente definida) de etapas são do tipo **void**. Esta é a única solução para resolver o problema que é a estrutura de etapas aponta para transições e vice-versa, pelo que se tem de criar um *fac-simile* para se poder fazer a definição. Assim as transições apontam para (quase) etapas e as etapas para transições (verdadeiras). Isto funciona porque todos os campos de **t_etapa** têm campos gémeos (com o mesmo nome e que ocupam o mesmo espaço) em **t_etapa_s_ptr**, permitindo assim utilizar os campos da segunda quando se pensa utilizar os da primeira.

Continuando a descrição da estrutura das transições, aparece **t_grafcet**, que é o tipo de dados do campo **t_transic.grafcet**, onde se diz a que grafcet esta transição pertence. Não existe uma lista ligada de descrição de grafcets, essa informação deve ser procurada junto de cada etapa e de cada transição.

t_transic.texto é uma string que contém a descrição da receptividade, depois da precompilação, onde as variáveis são substituídas pela sua representação em termos de entrada/saída do A.P. .

O tipo de dados **t_endereço** contém dentro de si um endereço de uma entrada ou saída do AP com a diferença que o último algarismo está entre 0 e 7 e representa o bit a endereçar, dentro do byte representado pelos algarismos mais significativos seguintes. Neste caso, o campo **t_transic.sensib** é o endereço da flag que representa a sensibilização (ou não) da transição em causa. O tipo de dados **t_endereco** permite apenas guardar um número; já se sabe que será sempre uma flag a executar este trabalho. A flag seguinte a esta **t_transic.sensib** é a flag que diz se esta transição foi transposta ou não. Por exempo: se o campo **t_transic.sensib** tiver o valor 120, a F12.0 estaria destinada a guaradar a informação da sensibilização da transição e a flag seguinte F12.1 seria destinada a saber se a transição foi transposta ou não. Se em vez de ser 120 fosse 127, então a sensibilização estaria guarada em F12.7 e a transposição em F13.0 . Para trabalhar facilmente com este tipo de dados, desenvolveram-se uma série de macro-funções, guardadas no ficheiro de cabeçalhos (header file) **grafcet.h** . Dirija-se ao parágrafo "proximo endereço" para mais informações.

O campo seguinte **t_transic.prox** é um ponteiro para uma outra estrutura to tipo **t_transic** e serve para implementar a lista ligada em blocos de memória não contínguos, obtidos a partir de alocação dinânmica de memória (malloc()). Este campo terá o ponteiro nulo **(t_transic *) NULL** no caso de ser a última estrutura da B.D. .

8.1.1.2.Etapas

As etapas são guardadas na seguinte estrutura:

```
#define N_MAX_ET 10
#define N_MAX_TR N_MAX_ET
typedef struct t_etapa {
        id_etapa n;
        id_transic  t_entr   [N_MAX_TR];
        t_transic * p_t_entr [N_MAX_TR];
        id_transic  t_saida  [N_MAX_TR];
        t_transic * p_t_saida [N_MAX_TR];
        t_grafcet grafcet;
        struct t_etapa * prox;
        t_endereco endereco;
        t_endereco anterior;
        t_classif classif;
        int disparatemp : 1;
        int setreset    : 1;
} t_etapa;
```

O número da etapa na B.D. de etapas é guardado no campo **t_etapa.n**, do tipo **id_etapa** (já discutido nas transições).

Da mesma forma que para as transições, guarda-se para cada etapa o seu conjunto de transições de entrada e transições de saída, repectivamente guardados nos campos **t_etapa.t_entr[]** e **t_etapa.t_saida[]**. Estes campos são preenchidos com dados do tipo **id_transic**, que identificam transições através do seu número (para mais detalhes ver secção de transições). Aplicam-se as consideraçõs já referidas no conjunto de etapas de entrada das transições de maneira dual. Existem também os vectores **t_etapa.p_t_entr[]** e **t_etapa.p_t_saida[]**, que apontam para as estruturas de memória das transições (**t_transic ***) de entrada e saída, respectivamente.

Guarda-se também o grafcet a que pertence esta etapa, com o tipo de dados **t_grafcet**, no campo **t_etapa.grafcet**.

O campo **t_etapa.endereco** guarda o endereço da flag disponibilizada apara representar esta etapa, sob o formato do tipo de dados **t_endereco**, já discutido a secção das transições.

No campo **t_etapa.classif** são guardados algumas características exclusivas acerca das propriedades da etapa. Este campo é do tipo de dados t_classif, codificado da seguinte maneira:

```
typedef enum t_classif {NORMAL=0,INICIAL,MACRO,ERRADO} t_classif;
```

Neste campo guarda-se então a informação de que esta etapa é uma etapa inicial, é uma macro-etapa ou uma etapa sem qualquer atributo fora do normal.

Seguem-se alguns campos de bits, que procuram também optimizar o trabalho com as etapas e que guardar informação acerca das características da etapa. O bit **disparatemp** diz se esta etapa trabalha com temporizadores; se este bit for verdadeiro, o programa vai à B.D. de acções procurar as informações necessárias para escrever o código correctamente. O bit **setreset** informa se esta etapa contém uma (ou mais) acção do tipo **S**. O bit **anterior** informa que é necessário guardar o estado anterior desta etapa, devido a uma condição disparada ao flanco (edge trigger).

Finalmente aparce o campo **t_etap.prox** que se destina a apontar na memória a localização da estrutura **t_etapa** seguinte na B.D. . Ver o campo homónimo nas transições, para mais informações.

8.1.1.3.Acções

```
typedef struct t_accao {
        char *ordens;
        id_etapa etapa;
        t_grafcet grafcet;
        char tipo;
        int temporizada : 1;
        struct t_accao *prox;
} t_accao;
```

Esta estrutura serve para implementar a relação de 1 para muitos, que existe entre etapas e acções.

Tal como está implementado, uma etapa pode ter várias acções e cada uma dessas acções pode conter um número variável de ordens.

O campo **t_accao.ordens** é uma uma string que contém as várias ordens já pré-compiladas e separadas pelo caracter ponto-e-vírgula (';'). Por ter um comprimento muito imprevisivel, a área de memória reservada para este efeito é alocada dinâmicamente e libertada no fim do programa.

Os campo **t_accao.etapa** e **t_accao.grafcet** são do tipo **id_etapa** e **t_grafcet**, dizendo que esta accao pertence à etapa e grafcet indicados, respectivamente.

O campos **t_accao.tipo** é uma letra que diz qual o tipo desta acção. Pode conter as seguintes letras:'?','S','P','D','L' ou ' ' (espaço). Estas letras têm os mesmos significados que os comandos respectivos; o espaço significa acção normal.

O bit **t_accao.temporizada** diz se esta acção é temporizada ou não. Se for veradeiro, então deve lançar o temporizador necessário ou procurar na lista ligada de variáveis e encontrar o temporizador apropriado.

8.1.1.4. Variáveis Auxiliares

Para guardar a correspondência entre nomes e respectivos endereços de entrad/saída do AP, existe uma B.D. que guarda todas as variáveis definidas até ao momento.

Cada elemento será do tipo t_variável.

```
typedef struct t_variavel {
        char texto[MAXVARAUX];
        t_endereco endereco;
        char letra;
        struct t_variavel * prox;
} t_variavel;
```

Existe uma string disponivel para guardar o nome da variável (campo **t_variavel.texto**) que corresponderá à ligação fisica do AP do tipo **t_variavel.letra**, no endereço **t_variavel.endereco** (guardado sob o formato **t_endereco**, discutido na seção das transições).

Esta estrutura pode também é utilizada para conter variáveis auxiliares e para facilitar o rastreio e despistagem de erros ao conter a correspodência entre cada bit do AP e uma representação interna do programa.

8.1.1.5. Próximo endereço

Para facilitar o trabalho com o tipo de dados **t_endereco**, utilizam-se macro-funções que garantem a legalidade dos dados, lembremo-nos que o algarismo menos significativo só pode estar entre 0 e 7 pois representa o bit dentro byte a aceder, representado pelos algarismos restantes.

Existem Macro-Funções, definidas no ficheiro "grafcet.h" que se destinam a trabalhar com este tipo de dados. Uma destas macro é a função prox:

```
#define prox(x) ( ((x)%10) >=7 ? ((x)+3) : ((x)+1) )
```

Esta função devolve o endereço seguinte ao da entrada, repare-se que se a entrada for 127 (representado 12.7 - último bit de um byte) então **prox(127)** devolverá 130 (representado o inicio do byte seguinte). Neste caso soma-se 3 para grantir a legalidade dos dados. Normalmente será **prox(10)=11** (10 representa 1.0 e 11 representa 1.1).

Para implementar a impressão deste tipo de dados existem também:

```
#define txt(dest,letra,x) sprintf (dest,"%c%0.1f",letra,x/10.0)
#define txtvar(dest,variav) txt(dest,variav->letra,variav->endereco)
```

Estas funções escrevem num buffer previamente alocado para o efeito, a representação do AP dos dados, num caso uma variável auxiliar, noutro a letra e endereço dados directamente.

8.1.2. grafglob.h

Este ficheiro contém algumas varáveis globais de interesse na trabalho com grafcets e sua implementalção no AP.

8.1.2.1.Listas Ligadas

Existem também variáveis globais, que são alocadas no módulo "suporte.c" mas definidas como externas:

```
extern t_transic * ll_transic;
extern t_etapa * ll_etapas;
extern t_variavel * ll_variaveis;
extern t_accao * ll_accoes;
```

Ponteiros globais para o início de cada uma das listas ligadas que suportam as B.D's. respectivamente lista ligada (ll) de transições, inicio da ll de etapas, inicio da ll variáveis auxiliares, inicio da ll de acções.

8.1.2.2.Recursos usados no AP

```
extern U ultima_flag;
extern U ultimo_temp;
```

Guardam respectivamente a ultima flag usada e o ultimo temporizador usado. São globais para se poderem ir alocando à medida das necessidades, em qualquer parte do programa.

8.1.2.3.Anti-estouro dos FBs

Os Function Blocks têm um tamanho máximo. Para que esse tamanho máximo não seja excedido, permitindo porém fazer um grande aproveitamento dentro de cada FB (visto que o seu nº também está limitado) utilizaram-se as seguintes variáveis:

```
#define MAXLINHAFB 500
extern U ultimo_fb;
extern U n_linhas;
extern U prox_label;
```

Estas variáveis destinam-se a permitir a mudança de Function Block, sempre que se escreve mais de **MAXLINHAFB** linhas para o ficheiro de saída, dentro do mesmo FB. De cada vez que se escreve uma linha no ficheiro de saída incrementa-se a variável n_linhas; sempre que se escrever uma etiqueta (label) no ficheiro de saída, verifica-se se é altura de mudar de FB (se se escreveram mais de **MAXLINHAFB** linhas para o ficheiro de saída). Consegue-se assim (através de uma estimativa) ocupar grande percentagem dos F.B., sem correr o perigo se ultrapassar o limite máximo de código por F.B. . Para mais informações veja nas funções de suporte, o procedimento **poe_label()**.

8.1.2.4.Ficheiros

```
extern FILE *out;
extern FILE *relat;
```

São os ficheiros de saída e de relatório, respectivamente. O ficheiro de saída conterá o código ASCII, depois de compilado; o ficheiro de relatório descreve pormenorizadamente a pré-compilação e seus resultados: pode ser utilizado para despistar erros (debugging) ou para confirmar a correcção do ficheiro de entrada; os erros de pre-compilação serão enviados para este ficheiro.

No ficheiro **grafcet.h** estão também os cabeçalhos (headers) para todas as funções globais, definidas no módulo **suporte.c** ou outro, mas sempre com interesse para a gneralidade dos módulos.

8.2. Funções de suporte

8.2.1. suporte.c

No módulo "suporte.c" existem uma série de funções que implementam criar variáveis, procurar tuplos na B.D., etc. Tentou-se fazer aqui um pouco de encapsulamento. Os cabeçalhos destas funções são conhecidos pelos módulos que fazem o include do "grafcet.h".

As funções que criam tuplos tem por objectivo facilitar a manutenção da B.D., ao centralizar tarefas de criação e por dar a todos os campos um valor pré-definido como sendo vazio ou inválido.

8.2.1.1.Alocação das variáveis globais

É neste módulo que se aloca o espaço destinado às variáveis globais, bem assim com se lhes atribui o seu valor inicial, que representa a que a lista ligada está vazia.

```
/* definição e alocação das var globais */
t_transic  * ll_transic=NULL;
t_etapa    * ll_etapas=NULL;
t_variavel * ll_variaveis=NULL;
t_accao    * ll_accoes=NULL;
```

8.2.1.2.Ficheiros

```
FILE *out;
FILE *relat;
```

Os ficheiros são inicializados na função **main()**, módulo "compila.c".

8.2.1.3.Recursos ocupados no AP

```
U int ultima_flag=0;
U int ultimo_temp=0;
```

8.2.1.4.Anti-estouro dos FBs

```
U int ultimo_fb=1;
U int n_linhas=0;
U int prox_label=1;
```

8.2.1.5.Criar tuplos de etapas

Para criar uma nova etapa chama-se esta função. Como parâmetros aparecem os elementos caracterizadores obrigatórios das etapas, um campo chave que é o número e o grafcet que é obrigatório ser conhecido no momento em que se cria a etapa. Devolve-se um ponteiro para a etapa recém-criada.

Existe uma variável local estática para optimizar a performance, que guarda o endereco da ultima estrutura criada, evitando-se assim percorrer toda a lista ligada.

Todos os campos que não são passados nos parâmetros são inicializados como vazios.

```
t_etapa * criaetapa(id_etapa num,t_grafcet grafcet)
{
static t_etapa * e_anterior;

    if (ll_etapas) {
        e_anterior->prox = (t_etapa *) malloc (sizeof(t_etapa));
        e_anterior=e_anterior->prox;
    } else {
        ll_etapas = e_anterior = (t_etapa *) malloc (sizeof(t_etapa));
        fprintf (relat, "'#'LL de etapas criada\n\r");
    }
    fprintf (relat, "'#'Etapa criada : id=%3d\n\r",num);
    e_anterior->n=(id_etapa) num;
    e_anterior->t_saida[0]=NADA;
    e_anterior->t_entr[0]=NADA;
    e_anterior->p_t_saida[0]=NULL;
    e_anterior->p_t_entr[0]=NULL;
    e_anterior->prox=(t_etapa *) NULL;
    e_anterior->grafcet=grafcet;
    e_anterior->classif=NORMAL;
    ultima_flag=prox(ultima_flag);
    e_anterior->endereco=ultima_flag;
    return e_anterior;
}
```

8.2.1.6.Criar tuplos de transições

Dual da criação de etapas. Aplicam-se os mesmos comentários.

```
t_transic * criatransic (id_transic num,t_grafcet grafcet)
{
static t_transic * t_anterior;

    if (ll_transic) { /* se não é o 1º nó */
        t_anterior->prox = (t_transic *) malloc (sizeof(t_transic));
        t_anterior=t_anterior->prox;
        fprintf (relat, "'T' : Transição criada : id=%3d\n\r",num);
    } else {/* se é o 1º nó */
        ll_transic=t_anterior=(t_transic *) malloc (sizeof(t_transic));
        fprintf (relat, "LL de transições criada\n\rTransição criada : id=%3d\n\r",num);
    }
    t_anterior->n=(id_transic) num;
    t_anterior->e_saida[0]=NADA;
    t_anterior->e_entr[0]=NADA;
    t_anterior->p_e_saida[0]=NULL;
    t_anterior->p_e_entr[0]=NULL;
    t_anterior->prox=NULL;
    t_anterior->grafcet=grafcet;
    ultima_flag=prox(ultima_flag);
    t_anterior->sensib=ultima_flag; /* primeiro a flag sensibilização */
    ultima_flag=prox(ultima_flag); /* avanca duas flags para 1 trasic */
    return t_anterior;
}
```

8.2.1.7.Criar tuplos de acções

Dual da criação de etapas. Aplicam-se os mesmos comentários.

```
t_accao * criaaccao(id_etapa n,t_grafcet grafcet)
{
static t_accao * a_anterior;
    if (ll_accoes) { /* se não é o 1º nó */
        a_anterior->prox = (t_accao *) malloc (sizeof(t_accao));
        a_anterior=a_anterior->prox;
        fprintf (relat, "Acção da etapa id=%3d criada\n\r",n);
    } else {/* se é o 1º nó */
        ll_accoes=a_anterior=(t_accao *) malloc (sizeof(t_accao));
        fprintf (relat, "LL de acções criada\n\rAcção da etapa id=%3d criada\n\r",n);
    }
    a_anterior->etapa=n;
    a_anterior->grafcet=grafcet;
```

```
    a_anterior->ordens=(char *) NULL;
    a_anterior->prox=(t_accao *) NULL;
    a_anterior->temporizada=0;
    a_anterior->disparatemp=0;
    a_anterior->anterior=0;
    return a_anterior;
}
```

8.2.1.8.Criar tuplos de variáveis auxiliares

Dual da criação de etapas. Aplicam-se os mesmos comentários.

```
t_variavel * criavariavel(char letra,t_endereco end)
{
static t_variavel * v_anterior;

    if (ll_variaveis) { /* se não é o 1º nó */
        v_anterior->prox = (t_variavel *) malloc (sizeof(t_variavel));
        v_anterior=v_anterior->prox;
        fprintf (relat, "Variavel %c%.1f criada\n\r",letra,(float) end/10);
    } else {/* se é o 1º nó */
        ll_variaveis=v_anterior=(t_variavel *) malloc (sizeof(t_variavel));
         fprintf (relat, "LL de variaveis criada\n\rVariavel %c%.1f criada\n\r",letra,(float)
end/10);
    }
    strcpy(v_anterior->texto,".Vazio.");
    v_anterior->endereco=end;
    v_anterior->letra=letra;
    v_anterior->prox=(t_variavel *) NULL;
    return v_anterior;
}
```

8.2.1.9.Procura acção pela etapa

Por causa do formato do ficheiro de entrada, todas as acções de uma etapa estão juntas, sem qualquer esforço especial. Ao devolver o ponteiro da primeira acção da etapa pretendida está-se na realidade a devolver uma lista ligada de acções, que *não* é porém terminada por *t_accao.prox==NULL*.

Devolve (t_accao *) NULL no caso de não existir.

```
t_accao * procura_a (id_etapa n)
{
    t_accao *accao=ll_accoes;
    while (accao!=NULL && accao->etapa!=n) accao=accao->prox;
    return (accao);
} /* na realidade devolve o 1º elemento de uma ll de acções */
```

8.2.1.10. Procura transição pelo seu número

Tal como todas as outras devolve um ponteiro para a estrutura, desta vez para um tipo de dados **t_transic.**

```
t_transic * procura_t (id_transic n)
{
    t_transic *transic=ll_transic;
    while (transic!=NULL && transic->n!=n) transic=transic->prox;
    return (transic);
}
```

8.2.1.11. Procura etapa pelo seu número

Devolve um **t_etapa**

```
t_etapa * procura_e (id_etapa n)
{
    t_etapa *etapa=ll_etapas;
    while (etapa!=NULL && etapa->n!=n) etapa=etapa->prox;
    return (etapa);
}
```

8.2.1.12. Procura etapa pelo seu endereço

Devolve um **t_etapa**

```
t_etapa *e_procura(t_endereco ender)
{
    t_etapa *etapa=ll_etapas;
    while (etapa!=NULL && etapa->endereco!=ender) etapa=etapa->prox;
    return(etapa);
}
```

8.2.1.13. Procura variável pelo seu texto

Utilizado na pré-compilação, para fazer a busca de variáveis.

```
t_variavel * procura_v(char *string)
{
    t_variavel *var=ll_variaveis;
    while (var!=NULL && strcmp(var->texto,string)) var=var->prox;
    return var;
}
```

8.2.1.14.Procura etapa pelo seu endereço

```
t_etapa *e_procura(t_endereco ender)
{
    t_etapa *etapa=ll_etapas;
    while (etapa!=NULL && etapa->endereco!=ender) etapa=etapa->prox;
    return(etapa);
}
```

8.2.1.15.Libertar tuplos

Existem funções que libertam o espaço alocado para as várias listas ligadas.

As várias funções **liberta_x(...)** libertam cada uma das várias listas ligadas, tendo como parâmetro o inicio da mesma.

A função **libertatudo()** não recebe qualquer parâmetro. Tal requisito é necessário para que ela possa ser transformada numa função em tempo de saída (at exit time function), que é chamada antes de o programa acabar normal ou anormalmente. Para fazer isto basta chamar a função **atexit(...)** - ver módulo **compila.c**.

```
void liberta_e (t_etapa * inicio)
{
    if (inicio->prox) liberta_e (inicio->prox);
    free (inicio);
}

void liberta_t (t_transic * inicio)
{
    if (inicio->prox) liberta_t (inicio->prox);
    if (inicio->texto) free (inicio->texto);
    free (inicio);
}

void liberta_v (t_variavel * inicio)
{
    if (inicio->prox) liberta_v (inicio->prox);
    free (inicio);
}

void liberta_a (t_accao * inicio)
{
    if (inicio->prox) liberta_a (inicio->prox);
    if (inicio->ordens) free (inicio->ordens);
    free (inicio);
}
```

```
void libertatudo(void)
{
    liberta_e(ll_etapas);
    liberta_t(ll_transic);
    liberta_a(ll_accoes);
    liberta_v(ll_variaveis);
}
```

8.2.1.16.Escrever etiquetas e anti-estouro dos FBs

Para evitar que se ultrapasse o limitado número de Function Block's disponiveis, recorreu-se à seguinte estratégia:

-Sempre que se escreve um linha no ficheiro de saída incrementa-se a varável **n_linhas**

-Para escrever uma etiqueta deve-se chamar a função **poe_label()**

-Sempre que se chama a função **poe_label()** o número de linhas é comparado com **MAXLINHAFB** e se for maior então escreve a label e muda o código para o FB seguinte, assinalando este facto na varável **ultimo_fb**. Note-se que os saltos são sempre para a próxima etiqueta, o que implica que sempre que seja escrita uma etiqueta pode-se mudar de FB sem problemas.

-As etiquetas têm a letra L, seguinda de um número sempre diferente por FB.

O código que implementa este algorítmos é o seguinte:

```
void poe_label(void)
{
        if (n_linhas>MAXLINHAFB) {
                fprintf(out,"L%u:\tBE\n<FB%u>\n",prox_label,++ultimo_fb);
                prox_label=1;
                n_linhas=0;
        } else fprintf(out,"L%u:",prox_label++);
}
```

8.3. Leitura do ficheiro

8.3.1. Lefich.c

Este módulo implementa toda a precompilação, verificação rudimentar de síntaxe e cria as B.D. (em variaveis globais) para uso das outras funções.

A leitura genérica é feita pelo procedimento lefich(...). Para melhorar a legibilidade, introduziram-se várias funções se especializaram na precompilação de vários comandos.

```
void precompilatransic(char * tudo, t_transic *transic);
```

```
void precompilaaccao  (char * texto, t_accao *accao);
void forcagem(char *texto,t_accao *accao);
```

8.3.1.1.Precompilação de forçagens

Recebe o texto da ordem de forçagem e o elemento da lista ligada de acções onde deverá ser colocado o resultado.

Para trabalhar sem problemas com a função **strtok()**, cria-se um duplicado na variavel **copia**.

A saída desta função é um texto ASCII, nome do grafcet mudado pelo seu número, etapas mudadas pelos seus endereços. Faz-se uma verificação de síntaxe e produz-se erros se necessário.

```
void forcagem(char *texto,t_accao *accao)
{
char *copia=strdup(texto),*ini=copia,*fim=copia,temp[10];
t_variavel *var;
t_etapa *etapa;

    while (*ini=='F' || *ini=='/' || *ini==' ') ini++;
    for (fim=ini;*fim!=')' && *fim;fim++); /* não utilizar strtok () */
    fim[1]=0;                    /* corta a cópia pelo char depois do ')' */
    for (fim=ini;*fim!=':' && *fim;fim++);
    *fim=0;                                  /* corta a cópia pelo ':' */
    fim++; /* fim="( . . . )" */
    if ((var=procura_v(ini))==NULL) {
        fprintf (relat, "Grafcet \"%s\" desconhecido\n\r",ini);
        /* não criar aqui a var correspondente ao grafcet
        ** inexistente exige a existência de hierarquia na
        \* escrita do ficheiro de texto */
        exit (ERRO);
    }
    if (var->letra!='G') {
       fprintf (relat, "Erro: Grafcet não encontrado\n\r0 nome do grafcet não pode coincidir
com o nome de variáveis\n\r");
       exit (ERRO);
    }
    itoa(var->endereco,temp,10);
    strcat(accao->ordens,"F/");
    strcat(accao->ordens,temp);
    strcat(accao->ordens,":");
    if (strpbrk(fim,"*")) /* assinala força congelamento do grafcet e sai */
       strcat(accao->ordens,fim);
    else {
```

```
        ini=++fim; /* passa '(' à frente */
        strcat(accao->ordens,"(");
        while (*fim && *fim==' ') fim++;
        while (*fim && *fim!=')')
        {
            while (*fim && *fim!=',' && *fim!=')')
                fim++;
                *fim=0;
            etapa=procura_e(atoi(ini));
                if (etapa==NULL) { fprintf (relat, "Forçar etapa %s (não declarada) é
ilegal\n\r",ini);exit (ERRO);}
            do fim++; while (*fim && *fim==' ');
            txt(temp,'F',etapa->endereco);
            strcat(accao->ordens,temp);
            strcat(accao->ordens,",");
            ini=fim;
        }
        accao->ordens[strlen(accao->ordens)-1]=')';/* muda ultima ',' por ')' */
    }
    free (copia);
}
```

8.3.1.2.Precompilação de Acções

Recebe o texto da acção, com várias ordens (do mesmo tipo) e o elemento da lista ligada de acções onde deverá ser colocado o resultado.

Para trabalhar sem problemas com a função **strtok()**, cria-se um duplicado na variavel **dup**.

A saída desta função é um texto ASCII, em que as ordens já substtuídas pelas respectivas representações do AP, são separadas por vírgulas.

Trata-se diferentemente o caso normal ou o caso das acções impulsionais ou temporizadas.

Durante o tratamento de qualquer tipo de acção, se aparecer uma forçagem será chamada a função correspondente.

```
void precompilaaccao  (char * texto,t_accao *accao)
{
    char * dup=strdup(texto),*dupini,*dupfim;
    char ender[10],op[2]={"_"};
    t_variavel *var;

    accao->ordens=(char *) malloc(MAJORA_A); /* majorante do espaço */
    *accao->ordens=0; /* prepara para strcat */
    sscanf (texto,"%c",&accao->tipo);
```

```
switch (toupper(accao->tipo)) {
    case ' ' :
        dupfim=strtok(dup,".: \n\r");
        dupini=dup;
        while (dupini && (*dupini)) {
            if (*dupini==' ') {dupini++;continue;}
            if ((var=procura_v(dupini))==NULL) {
                if (dupini[1]=='/') { /* Forçagem */
                    if (dupini[3]!='*') dupini[strlen(dupini)]=*dupfim; /* limpa o 0 a +
*/
                    forcagem(dupini,accao);
                    dupfim=strpbrk(dupfim,")")+1;
                    dupini=strpbrk(dupini+1,")");
                    dupini=strtok(dupini+1,".:\n\r");
                    if (dupfim) strcat (accao->ordens,":");
                    continue;
                } /* se não é um forçagem dá erro! */
                fprintf (relat, "Variavel \"%s\" desconhecida\n\r",dupini);
                exit (ERRO);
            }
            txtvar (ender,var);
            strcat (accao->ordens,ender);
            strcat (accao->ordens,":"); /* acrescenta-se TOKEN de separação */

            dupfim=strpbrk(++dupfim,".:");
            dupini=strtok(NULL,".: \n\r");
        } /* fim das ordens de uma accao normal */
    break;
    case 'D' : case 'L' : case 'S': case 'P' :
        dupfim=strpbrk(texto,".:=\n\r");
        dupini=strtok(dup+1,".: =\n\r");
        while (dupini && (*dupini)) {
            if (*dupini==' ') {dupini++;continue;}
            if (!isalnum(*dupini)) { /* tratar de '(' ')' '/' */
                *op=*dupini;
                strcat(accao->ordens,op);
                dupini++;
                continue;
            }
            if ((var=procura_v(dupini))==NULL) {
                if (dupini[0]=='F' && dupini[1]=='/') { /*Forçagem*/
                    dupini[strlen(dupini)]=*dupfim;/*limpa o 0 a +*/
                    forcagem(dupini,accao);
                    dupfim=strpbrk(dupfim,")")+1;
```

```
                        dupini=strpbrk(dupini+1,")");
                        dupini=strtok(dupini+1,".:\n\r");
                        if (dupfim) strcat (accao->ordens,":");
                        continue;
                    } /* se não é um forçagem dá erro! */
                    fprintf (relat, "Variavel \"%s\" desconhecida \n\r" , dupini);
                    exit (ERRO);
                }
                txtvar (ender,var);
                strcat (accao->ordens,ender);
                if (*dupfim=='=' || !dupfim) break;
                dupfim=strpbrk(++dupfim,".:=\n\r");
                dupini=strtok((dupfim),".: =\n\r");
                strcat (accao->ordens,":"); /* acrescenta-se TOKEN de separação */
            }
            if (toupper(accao->tipo)!='P' && *dupfim!='=') {
                fprintf (relat, "Erro de sintaxe: Falta '=' em accao temporizada");
                exit (ERROACCAO);
            }
            if (toupper(accao->tipo)!='P') {
                sscanf (dupfim,"=%s",ender); /* ler o texto depois do '=' */
                dupini=ender;   /* dupini funciona como uma var. temp */
                while (*dupini==' ') dupini++; /* tirar espaços à esq */
                strtok (ender," ");            /* tirar espaços à dir */
                strcat (accao->ordens,"=");    /* acrescentar token */
                strcat (accao->ordens,ender);
            }
        break; /* fim das ordens de uma accao temporizada ou impulsional */
        case '?' :
            fprintf (relat, "Acção condicional não implementada");
            exit (ERRO);
            break;
        default : fprintf (relat, "Tipo de acção desconhecida:\"%c\"\n\r",accao->tipo);
            exit (ERRO);
            break;
    }
    dupfim=accao->ordens+strlen(accao->ordens)-1; /* var. temp. */
    if (*dupfim==':') *dupfim=0; /* limpa um ':' eventualmente a mais */
    accao->ordens=(char *) realloc(accao->ordens,strlen(accao->ordens)+2); /* tamanho real */
    fprintf (relat, "'A' : Accão Tipo %c acrescentada à etapa nº%3d:%s\n\r",accao->tipo,accao-
>etapa,accao->ordens);
}
```

8.3.1.3.Precompilação de transições

```
#define TOKSTR " +*)\n"

void precompilatransic(char * tudo,t_transic *transic)
{
    char *dup,*dupini,*dupfim;
    char ender[7]={"X999.9"},cp[2]={"_"},fim;
    t_etapa *et;
    id_etapa n_et;
    t_variavel *var;

    transic->texto=(char *) malloc(MAJORA_T); /* majorante do espaço  */
    transic->texto[0]=0;  /* prepara para fazer strcat */

    dupfim=tudo;

    dup=(char*) malloc(strlen(tudo)*sizeof(char)+3);
    strcpy(dup," »");     /* acrescentar um token para iniciar o strtok */
    strcat (dup,tudo);
    dupini=strtok(dup,"»"); /* depois faz-se strtok(NULL,"...") */

    for (;;) {
        while (!isalnum(*dupfim) && *dupfim!='\n')
            if (*dupfim!=' ') { /* tratar de '(' e '/' e operações também */
                *cp=*dupfim;
                strcat(transic->texto,cp);
                dupfim++;
            } else dupfim++;

        dupfim=strpbrk(dupfim,TOKSTR);
        dupini=strtok(NULL,TOKSTR);

        if (!dupini) break; /* já não há mais tokens */

        while (!isalnum(*dupini)) dupini++;

        if (dupini[0]=='t' && dupini[1]=='/') { /* temporizadores */
            ultimo_temp=prox(ultimo_temp);
            var=criavariavel('T',ultimo_temp);
            /* devia-se verificar se já existe, antes de criar */
            strcpy(var->texto,dupini);
          if (sscanf(dupini+2,"%d",&n_et)!=1) {fprintf (relat,"Erro na leitura de uma temporiz.
:'%s' ", dupini); exit (ERRO);}
            et=procura_e(n_et); /* procurar a et que dispara temporiz. */
```

```
            if (et==NULL) et=criaetapa(n_et,transic->grafcet);
            et->disparatemp=1;      /* busca grafcet à transic */
            sprintf (ender,"T%d",ultimo_temp);
            strcat(transic->texto,ender);
        } else if (dupini[0]=='X') {
            fim=sscanf(dupini+1,"%d",&n_et);
            if ((et=procura_e(n_et))==NULL || fim!=1) {
                        fprintf(relat,"Erro na leitura/procura da etapa referida com 'Xn' em
%s",dupini);
                exit (ERRO);
            }
            txt(ender,'F',et->endereco);
            strcat(transic->texto,ender);
        } else {  /* não é um temporizador */
            if ((var=procura_v(dupini))==NULL) {
                fprintf (relat, "Variavel \"%s\" desconhecida\n\r",dupini);
                exit (ERRO);
            }
            txtvar(ender,var);
            strcat (transic->texto,ender);
        }
    }

    /* nesta altura var->texto contem uma expressao   *\
    ** com os nomes substituidos por símbolos do AP   **
    ** a seguir passa-se essa expesssão para Notação **
    \* Polaca Inversa, que resolve qq parentesis      */

      fprintf (relat, "'T': Precompilação t. nº%3d resulta em: %s\n\r",transic->n,transic-
>texto);
    free(dup);
    return;
}
```

8.3.1.4.Leitura genérica do ficheiro

Nesta rotina faz-se a leitura do ficheiro e preenche-se os campo das etapas de entrada/saída e transições de entrada/saída. Para tratar das acçoões e das receptividades das transições são chamadas as funções respectivas.

Esta rotina recebe como parâmetro o ficheiro de entrada.

```
void lefich (FILE *in)
{
    char texto[MAXLINHAFICH],comando,fim,letra_ap,*txt;
    t_etapa * etapa=NULL;
    t_transic * transic=NULL;
    t_accao * accao;
    t_variavel * var;
    t_endereco byte,bit;
    U int num;
    id_etapa n_e;
    id_transic n_t;
    t_grafcet grafcet=0;
    float end_fich;

    clrscr();

    fscanf(in,"%c",&comando);
    while (!feof(in))
    {
        if (isspace(comando)) { fscanf(in,"%c",&comando); continue; }

        switch (comando) {
            case '#' :
                    fscanf (in," %d ",&num);
                    etapa=procura_e(num); /* cria etapa normal se não existe */
                    if (etapa==NULL) etapa=criaetapa(num,grafcet);
            break;

            case 'T' : case 't' :
                     fscanf (in," %d",&num);
                     fgets(texto,MAXLINHAFICH,in);
                     transic=procura_t(num);
                    if (transic==NULL) transic=criatransic(num,grafcet);
                    transic->texto=(char *) malloc(MAJORA_T*sizeof(char));
                    transic->texto[0]=0;  /* prepara para strcat */
                    precompilacondicao(texto,transic->texto,transic->grafcet);
```

```
        break;

        case 'E' : case 'e' :
            n_t=0;
            while (etapa->t_entr[n_t]!=NADA) n_t++;
            fscanf (in,"%c",&fim);
            do {
                fscanf (in,"%d%c",&num,&fim);
                etapa->t_entr[n_t]=num;
                etapa->t_entr[++n_t]=NADA;
                fprintf (relat, "'E': Etapa nº%3d: Transição de entrada nº%3d=%3d\n\r",
etapa->n,n_t,num);
                transic=procura_t(num);
                if (transic) {
                    n_e=0; /* falta prever estouros */
                    while (transic->e_saida[n_e]!=NADA) n_e++;
                    transic->e_saida[n_e]=etapa->n;
                    transic->e_saida[++n_e]=NADA;
                    fprintf (relat, "'E': Transição:%3d Actualizada: Etapa de saida
nº%3d=%3d\n\r", num,n_e,etapa->n);
                } else { /* trata de criar a transição que não existe */
                    transic=criatransic(num,grafcet);
                    transic->e_saida[0]=etapa->n;
                    transic->e_saida[1]=NADA;
                    n_e=1;
                }
                transic->p_e_saida[n_e-1]=(t_etapa_s_ptr *) etapa;
                transic->p_e_saida[n_e]=NULL;
                etapa->p_t_entr[n_t-1]=transic;
                etapa->p_t_entr[n_t]=NULL;
            } while (fim=='.');
        break;

        case 'S' : case 's' :
            n_t=0;
            while (etapa->t_saida[n_t]!=NADA) n_t++;
            fscanf (in," ",&fim); /* apaga espaço */
            do {
                fscanf (in,"%d%c",&num,&fim); /* apaga espaço */
                etapa->t_saida[n_t]=num;
                etapa->t_saida[++n_t]=NADA;
                fprintf (relat, "'S': Etapa nº%3d: Transição de saida nº%3d=%3d\n\r",
etapa->n,n_t,num);
                transic=procura_t(num);
                if (transic) {
                    n_e=0;
```

```
                        while (transic->e_entr[n_e]!=NADA) n_e++;
                        transic->e_entr[n_e]=etapa->n;
                        transic->e_entr[++n_e]=NADA;
                         fprintf (relat, "'S': Transição:%3d Actualizada: Etapa de saida
n°%3d=%3d\n\r", transic->n,n_e,etapa->n);
                    } else {   /* trata de criar a transição que não existe */
                        transic=criatransic(num,grafcet);
                        transic->e_entr[0]=etapa->n;
                        transic->e_entr[1]=NADA;
                        n_e=1;
                    }
                    transic->p_e_entr[n_e-1]=(t_etapa_s_ptr *) etapa;
                    transic->p_e_entr[n_e]=NULL;
                    etapa->p_t_saida[n_t-1]=transic;
                    etapa->p_t_saida[n_t]=NULL;
                } while (fim==',');
            break;

            case 'A' : case 'a' :
                fgets (texto, MAXLINHAFICH, in);
                accao=criaaccao (etapa->n,grafcet);
                precompilaaccao(texto,accao);
            break;

            case 'G' : case 'g' :
                var=criavariavel('G',++grafcet);
                fgets (var->texto, MAXVARAUX, in);
                strtok(var->texto,"\n\r");
                 fprintf (relat, "Inicio da definição do grafcet %s=n°%d\n\r",var-
>texto,grafcet);
            break;

            case 'I' : case 'i' :
                etapa->classif=INICIAL;
                fprintf (relat, "Etapa %3d é inicial\n\r",etapa->n);
            break;

            case 'M' : case 'm' :
                    fprintf (relat, "Comando 'M' não implementado. Por favor reformule o
grafcet\n\r");
                exit(ERRO);
            break;

            case ':' :
                fscanf (in,"%c%f=%s",&letra_ap,&end_fich,texto);
                var=criavariavel(letra_ap,(t_endereco) (end_fich*10+.5)); /*letra=I ou Q*/
```

```
                strcpy(var->texto,texto);
                fprintf (relat, "':' %c%.1f=\"%s\"\n\r",letra_ap,end_fich,var->texto);
            break;

            default :
                fgets (texto, MAXLINHAFICH, in);
                        fprintf (relat, "Comando '%c' desconhecido em:'%c%s'\n\r", comando,
comando,texto);
            break;
        }
        fscanf(in,"%c",&comando);
    }
}
```

8.4. Cálculo das condições de Evolução

O algorítmo genérico foi descrito em secção própria.

O procedimento que trata de escrever o código relativo a esta secção é o **compilatransic()**.

O que este procedimento faz é percorrer a lista ligada de etapas e para cada uma delas escreve o código relativo a:

- Verifica se a transição está susceptibilizada, se não o estiver salta para o inicio do código seguinte.

- Sabe-se que a transição está susceptibilizada. Se existir uma só etapa de entrada, a susceptibilização é suficiente para se saber que a única etapa de entrada está activa. Se houver mais que uma etapa de entrada, elas são testadas, se alguma não estiver activa então dá-se um salto para o código seguinte.

-Sabe-se que todas as etapas de entrada estão activas, a transição pode ser transposta. Calcula-se o valor da condição de evolução e para isso:

  - Calcula-se a Notação Polaca Inversa da condição (função **iniciaNPI**)
  - Testa-se se a pilha contém uma única variável, senão chama a função **escreve1op**, que é recursiva e escreve o código relativo a qualquer pilha de entrada (em NPI).

- O código seguinte pode ser o tratamento de outra transição ou o inicio da activação/desactivação de etapas

8.4.1. pilha.h

8.4.1.1.Definição de operações e números da pilha

```
typedef enum t_bool {AFIRMADO=0,NEGADO} t_bool;

#define nega(x) ((x==AFIRMADO) ? NEGADO : AFIRMADO)

typedef unsigned char t_prio;

typedef struct t_op {char op;t_bool nega;} t_op;

typedef struct t_num {char letra;float n;t_bool nega;} t_num;

typedef struct {
        enum tipo {NUM=1,OP}tipo;
        union {
                t_op op;
                t_num num;
```

```
        } info;
} t_elem;                          /* definição de cada elemento da pilha */
```

8.4.1.2.Definição de uma pilha

```
typedef struct {
        int topo;
        t_elem items[MAXPILHA];
        } t_pilha;
/* definição da estrutura da pilha */
/* o tamanho deste array pode ser controlado através de um mallloc  */
```

8.4.1.3.Funções que acedem à pilha

Por este código ser pouco interessante não será aqui reproduzido. Para pormenores consulte o apêndice, ficheiro **stack_ops.c**.

```
int      pop (t_pilha *p_pilha,t_elem *elem );
int      push (t_pilha *p_pilha,t_elem *p_elem);
void     mostrapilha(t_pilha *p_pilha);
```

8.4.1.4.Funções que produzem a NPI

Por este código ser pouco interessante não será aqui reproduzido. Para pormenores consulte o apêndice, ficheiro **pol_inv.c**.

```
t_pilha * iniciapilha(void);
t_pilha * iniciaNPI(char *); /* transforma string em pilha NPI */
```

8.4.1.5.Funções trabalho com os elentos da pilha

Estas funções estão no ficheiro pol_inv.c e serão discutidas na próxima secção.

```
void pushnum(char * num,t_bool negado);
void pushop(char * num,t_bool negado);
t_prio priorid(char chr);
char * polacainversa(char *string,t_bool nega,t_prio prio);
```

8.4.2. comptran.c

É neste ficheiro que se trata a generalidade das situações, são utilizadas definições várias das funções que suportam a pilha, que definem a pilha, etc. . São chamdas várias funções dos ficheiros **pol_inv.c** e **stack_ops.c**, por favor veja a secção acima para mais pormenores.

8.4.2.1.Compilação geral de transições

```
void compilatransic(void)
{
t_elem primeiro;
t_transic *t_corrente;
id_etapa n_et_entr;
t_etapa *etapa;
t_pilha *pilha;

/* para cada transic testa se a transição está sensibilizada, senão salta.  *\
** Se está, testa etapas de entrada e a condição guardando o resultado como **
\* sendo a transposição da receptividade                                     */

    for (t_corrente=ll_transic;t_corrente;t_corrente=t_corrente->prox)
    {
        pilha=iniciaNPI(t_corrente->texto); /* faz malloc pilha e faz NPI */
#ifdef DEBUG
        fprintf (relat,"Transic. %d:\n\r%s\n\r",t_corrente->n,t_corrente->texto);
        mostrastack(pilha);
#endif
        poe_label();
        fprintf (out,"\tAN F%.1f\n\r\tJC =L%d\n\r",t_corrente->sensib/10.0,prox_label);
        n_linhas+=2;
        /* testa a susceptibilização da transição e salta se não está sensib. */

        if (t_corrente->e_entr[0]!=NADA && t_corrente->e_entr[1]!=NADA) {
        /* Se houver uma etapa de entrada, a susceptibilização da transição **
        ** equivale à verificação da etapa de entrada                        */
            for (n_et_entr=0;t_corrente->e_entr[n_et_entr]!=NADA;n_et_entr++)
            {
                if ((etapa=procura_e(t_corrente->e_entr[n_et_entr]))==NULL) {
                    fprintf (relat,"Etapa nº%3d não encontrada",n_et_entr);exit (ERRO);}
                fprintf (out,"\tON F%.1f\n\r",etapa->endereco/10.0);
            }
            fprintf (out,"\tJC =L%d\n\r",prox_label);
            n_linhas+=n_et_entr+1;
        }
        /* este salto é necessário para evitar problemas de associações **
```

```
        ** (erradas) na avaliação (c/ prioridades) das operações do AP. **
        ** Assim separa-se totalmente o teste das etapas de entrada  do **
        \* código da receptividade (a seguir)                            */

        if (pop(pilha,&primeiro)==UNDERFLOW) {fprintf (relat,"Erro de lógica na escrita da
transic nº%d",t_corrente->n);mostrastack(pilha);exit (ERRO);}
        if (primeiro.tipo==NUM) {
            if (primeiro.info.num.nega==NEGADO) fprintf (out,"\tAN"); else fprintf (out,"\tA");
            fprintf (out," %c%.*f\n\r",
                        primeiro.info.num.letra,
                        primeiro.info.num.letra!='T' ? 1 : 0,
                        primeiro.info.num.n);
         n_linhas+=2;
        } else {
            escrevelop(primeiro.info.op.op,pilha);
        }
        fprintf (out,"\t=F%.1f\n\r",prox(t_corrente->sensib)/10.0);
        n_linhas++;
        /* a flag seguinte à da sensibilização representa a transposição da transic */
        free(pilha); /* ok */
    }

}
```

A função escreve-uma-operação recebe uma pilha e uma operação (nivel de prioridade) e a apartir daí escreve o código STEP5 de qualquer pilha pois é uma função recursiva.

```
void escreve1op(char op,t_pilha *pilha)
{
t_elem elem1,elem2;

    if (pop(pilha,&elem1)==UNDERFLOW) {fprintf (relat,"Erro interno:Falta parâmetro para uma operação");exit (ERRO);}

   if (op=='*') fprintf (out,"\tA"); else fprintf (out,"\tO");
   if (elem1.tipo==NUM) {
      if (elem1.info.num.nega==NEGADO) fprintf (out,"N");
      fprintf (out," %c%.*f\n\r",
                elem1.info.num.letra,
                elem1.info.num.letra!='T' ? 1 : 0,
                elem1.info.num.n);
      n_linhas++;
   } else {
      fprintf (out,"(\n\r");
      escreve1op(elem1.info.op.op,pilha);
      fprintf (out,"\t)\n\r");
      n_linhas+=2;
   }

    if (pop(pilha,&elem2)==UNDERFLOW) {fprintf (relat,"Erro interno:Falta parâmetro para uma operação");exit (ERRO);}

   if (op=='*') fprintf (out,"\tA"); else fprintf (out,"\tO");
   if (elem2.tipo==NUM) {
      if (elem2.info.num.nega==NEGADO) fprintf (out,"N");
      fprintf (out," %c%.*f\n\r",
                elem2.info.num.letra,
                elem2.info.num.letra!='T' ? 1 : 0,
                elem2.info.num.n);
      n_linhas++;
   } else {
      fprintf (out,"(\n\r");
      escreve1op(elem2.info.op.op,pilha);
      fprintf (out,"\t)\n\r");
      n_linhas+=2;
   }
}
```

8.4.3. *pol_inv.c*

8.4.3.1.Push de uma variável para a pilha

A função anterior recorre à função pushnum, que se encarrega de tirar um "número" (variável) da string e introduzi-lo na pilha, que é (neste ficheiro) uma var global **stack**. Por causa da função escrevelop ser recursiva é possivel estarmos a escrever código que deva ser premanentemente negado, daí que este seja também um parâmetro.

```
void pushnum(char * num,t_bool negado)
{
t_elem elem;
t_variavel *p_var;

    sscanf(num++,"%c",&elem.info.num.letra);
    if (elem.info.num.letra=='/') {
        sscanf(num++,"%c",&elem.info.num.letra);
        negado=nega(negado); /* negado é uma var. local! */
    }
    elem.tipo=NUM;
    elem.info.num.nega=negado;

/* cuidado c/ temporizadores! */

    if (elem.info.num.letra!='^') {
       if (sscanf(num,"%f",&elem.info.num.n)!=1) {
          fprintf (relat, "Erro na leitura de um número:Síntaxe errada (rotina NPI)\n\r");
          exit(ERROLEIT);
       }
                   if  (push(stack,&elem)!=OK)  {fprintf  (relat,  "Falta  de  espaço  na
pilha");exit(ERROPUSH);}
    }

    if (elem.info.num.letra=='^') {
    /* em vez do '^' escreve (var)*(not v_anterior_var) na pilha */
       if (sscanf(num,"%c%f",&elem.info.num.letra,&elem.info.num.n)!=2) {
           fprintf (relat, "Erro na leitura de uma estrutura 'Xm.n', rot NPI,pushnum\n\r Erro
em %s :\n\r",num);
           exit(ERROLEIT);
       }
                   if  (push(stack,&elem)!=OK)  {fprintf  (relat,  "Falta  de  espaço  na
pilha");exit(ERROPUSH);}

       /* criar var temp para guardar valor anterior */
       /* aqui devia-se procurar a ver se já existe (antes de criar) */
       ultima_flag=prox(ultima_flag);
```

```
        p_var=criavariavel('<',ultima_flag); /* '<'='F' mas a guardar no fim do ciclo */
        if (negado==AFIRMADO) *p_var->texto=0; else strcpy(p_var->texto,"/");
        strncat(p_var->texto,num,6*sizeof(char));
        *(num=strchr(p_var->texto,'.')+2)=0; /* corta chrs a mais */

        elem.tipo=NUM;
        elem.info.num.letra='F';
        elem.info.num.n=ultima_flag/10.0;
        if (negado==NEGADO) elem.info.num.nega=AFIRMADO; else elem.info.num.nega=NEGADO;
                if (push(stack,&elem)!=OK) {fprintf (relat, "Falta de espaço na
pilha");exit(ERROPUSH);}

        pushop ("*",negado);
#ifdef DEBUG
        mostrastack(stack);
#endif
        return;
    }
}
```

8.4.3.2.push de uma operação para a pilha

O mesmo que **pushnum** mas para operações. Também as operações podem ser ter que ser negadas, veja-se por exemplo '/(I0.0+Q2.0)', resultaria no stack '(*,I0.0,Q2.0)'.

```
void pushop(char * op,t_bool negado)
{
t_elem elem;

    elem.tipo=OP;
    sscanf(op++,"%c",&elem.info.op.op);

    if (elem.info.op.op=='.') elem.info.op.op='*';
    if (negado==NEGADO) elem.info.op.op=elem.info.op.op=='+' ? '*' : '+';
    elem.info.op.nega=AFIRMADO;
    if (push(stack,&elem)!=OK) {fprintf (relat, "Falta de espaço na pilha");exit(ERROPUSH);}
}
```

8.4.3.3.prioridade de uma certa operação

A função **escreve1op** precisa de saber a prioridade da operação seguinte para decidir acerca de um novo salto recursivo; serve também para haver uma verificção de síntaxe mais apertada.

```
t_prio priorid(char chr)
{
    switch (chr) {
        case ')' : case '»' : return 0;
        case '+' : return 1;
```

```
        case '*' : case ':' : return 2;
    }
    fprintf (relat, "Erro: avalição de uma operação desconhecida:'%c'\n\r",chr);
    exit (ERROOPERAC);
    return 0; /* só para deixar de dar aqui um aviso na compilação */
}
```

8.5. Activação / Desactivação de etapas

O procedimento responsável por gerar o código nesta secção é o **desactiva_activa_etapas()**.

Esta função está dividida em duas partes distintas que correspondem à desactivação das etapas de entrada de uma dada transição e à activação das etapas de saída dessa mesma transição.

Em ambas as partes do procedimento é percorrida a lista ligada de transições sendo gerado para cada uma delas o código respectivo.

Na parte correspondente à desactivação das etapas de entrada é escrito o código referente a:

- Teste da condição de evolução (CE) da transição.
- Salto para o início do código seguinte se a CE=0.
- Para todas as etapas de entrada da transição é:
    - Desactivada a etapa.
    - Feito o *reset* de todas as acções excepto as memorizadas e condicionais. O procedimento responsável por esta parte é o **reset_accoes()**.
- Dessusceptibilização da transição, para que a sua CE não seja avaliada no ciclo seguinte.

Na parte correspondente à activação das etapas de saída é escrito o código referente a:

- Teste da condição de evolução da transição.
- Salto para o início do código seguinte se a CE=0.
- *Reset* da condição de evolução da transição.
- Para todas as etapas de saída da etapa transposta é:
    - Activada essa etapa (*set* da *flag* correspondente).
    - Susceptibilizadas as transições de saída da etapa activada pela função **suscept_transic_saida()**.

- Processadas as acções do tipo set/reset (memorizadas) pela função **set_reset_accao()**.

- Processadas as acções do tipo normal pela função **proc_accao_normal()**.

Na parte final da função é gerado o código correspondente ao lançamento dos temporizadores associados às transições. Este código é gerado pela função **dispara_temporizadores()** e ao contrário do restante fica numa zona sem *labels* para que seja corrido em todos os ciclos do autómato.

As funções até agora referidas encontram-se no módulo "ajuda.c" que está descrito numa das próximas secções. A única excepção é a função **proc_accao_normal()** que se encontra no módulo "accoes.c".

8.5.1. des_acti.c

Este ficheiro tem o programa que gera o código descrito na secção anterior. Para além das funções anteriormente referidas, são usadas outras que se encontram no módulo "suporte.c".

```
#include "grafcet.h"
#include "grafglob.h"
#include <string.h>

void desactiva_activa_etapas(void)
{

    t_transic * ptr_transic=ll_transic;
    t_etapa * ptr_etapa;
    t_accao * ptr_accao;
    char dest[10];
    int i;

    /* para todos os grafcets, desactiva etapas de entrada,  */
    /*  dessusceptibiliza transições se CE=1, desactiva     */
    /*  acções normais, ao atraso (D), duração limitada (L)  */
    /*   e desactiva as forçagens feitas por essa etapa      */

       while(ptr_transic)
       {
           poe_label();
           txt(dest,'F',prox(ptr_transic->sensib));
           /* testa C.E. da transição */
           fprintf(out,"\tAN %s\n\r",dest);
           n_linhas++;
           fprintf(out,"\tJC=L%d\n\r",prox_label);
```

```
        n_linhas++;
        i=0;
        while(ptr_transic->e_entr[i]!=NADA)
        {
                ptr_etapa=procura_e(ptr_transic->e_entr[i]);
                txt(dest,'F',ptr_etapa->endereco);
                /* desactiva etapa */
                fprintf(out,"\tR %s\n\r",dest);
                n_linhas++;
                /* reset de todas as acções excepto as memorizadas e condicionais */
                reset_accoes(ptr_transic->e_entr[i]);
                i++;
        }
        txt(dest,'F',ptr_transic->sensib);
        /* dessusceptibiliza transição */
        fprintf(out,"\tR %s\n\r",dest);
        n_linhas++;
        ptr_transic=ptr_transic->prox;
}



/* para todos os grafcets, põe CE=0 das transições transpostas */
/*  activa etapas de saída, susceptibiliza as respectivas      */
/*  transições de saída, faz o set ou reset das acções         */
/*  memorizadas e processa acções normais                      */

ptr_transic=ll_transic;
while(ptr_transic)
{
        poe_label();
        txt(dest,'F',prox(ptr_transic->sensib));
        fprintf(out,"\tAN %s\n\r",dest);
        n_linhas++;
        fprintf(out,"\tJC=L%d\n\r",prox_label);
        n_linhas++;
        fprintf(out,"\tR %s\n\r",dest);
        n_linhas++;
        i=0;
        /* activa etapas de saída, susceptbiliza transições */
        /*   e processa algumas acções                      */
        while(ptr_transic->e_saida[i]!=NADA)
        {
                ptr_etapa=procura_e(ptr_transic->e_saida[i]);
                /* activa etapa */
```

```
                    txt(dest,'F',ptr_etapa->endereco);
                    fprintf(out,"\tS %s\n\r",dest);
                    n_linhas++;
                    /* susceptibiliza transições de saída */
                    suscept_transic_saida(ptr_etapa);
                    /* processa acções do tipo set/reset */
                    if (ptr_etapa->setreset==-1) set_reset_accao(ptr_etapa->n);
                    /* processa acções normais */
                    ptr_accao=ll_accoes;
                    while (ptr_accao) {
                        if (ptr_accao->etapa==ptr_transic->e_saida[i] && ptr_accao-
>tipo==' ')
                        proc_accao_normal(ptr_accao->ordens);
                        ptr_accao=ptr_accao->prox;
                    }
                    i++;
                }
                ptr_transic=ptr_transic->prox;
        }
        dispara_temporizadores();
}
```

8.5.2. ajuda.c

Este módulo contém as funções que são usadas (principalmente) na função **desactiva_activa_etapas()**. Os módulos que fazem o include do "grafcet.h" podem usá-las iguamente.

8.5.2.1.susceptibilização das transições de saída

Esta função é usada para susceptibilizar as transições de saída de uma etapa aquando da sua activação. Tem como parâmetro o apontador para a etapa e faz uso das funções **procura_t()** e **txt()** que estão no módulo "suporte.c" e "grafcet.h" respectivamente.

```
void suscept_transic_saida(t_etapa * ptr_etapa)
{
        t_transic * ptr_transic_saida;
        int i=0;
        char dest[10];

        while(ptr_etapa->t_saida[i]!=NADA)
        {
                ptr_transic_saida=procura_t(ptr_etapa->t_saida[i]);
```

```
                txt(dest,'F',ptr_transic_saida->sensib);
                fprintf(out,"\tS %s\n",dest);
                n_linhas++;
                i++;
        }
}
```

8.5.2.2.desusceptibilização das transições de saída

Dual da anterior. Aplicam-se os mesmos comentários.

```
void desuscept_transic_saida(t_etapa * ptr_etapa)
{
        t_transic * ptr_transic_saida;
        int i=0;
        char dest[10];

        while(ptr_etapa->t_saida[i]!=NADA)
        {
                ptr_transic_saida=procura_t(ptr_etapa->t_saida[i]);
                txt(dest,'F',ptr_transic_saida->sensib);
                fprintf(out,"\tR %s\n",dest);
                n_linhas++;
                i++;
        }
}
```

8.5.2.3.disparo dos temporizadores associados às transições

Esta função percorre a lista ligada de etapas e verifica se essas são responsáveis pelo lançamento de temporizadores (associados às transições) aquando da sua activação. Isto é verificado pelo teste do bit **disparatemp** presente na lista ligada de etapas. O conhecimento do temporizador a lançar, assim como do respectivo tempo é feito percorrendo a lista ligada de variáveis testando-se, para cada variável do tipo temporizador, se a etapa que o lança corresponde à etapa pretendida. Para tal usa-se a função **strtok()**, criando-se sempre um duplicado na variável **dup**.

```
void dispara_temporizadores(void)
{
    char dup[20], * etapa, * unidade, * tempo, n_unidade, dest[10];
    t_variavel * ptr_variavel;
    t_etapa * ptr_etapa=ll_etapas;
    int n_etapa,flag=0;
```

```
while(ptr_etapa)
{
    if (ptr_etapa->disparatemp==1) {
        ptr_variavel=ll_variaveis;
        while(ptr_variavel)
        {
            if (ptr_variavel->letra=='T') {
                strcpy(dup,ptr_variavel->texto);
                strtok(dup,"/");
                etapa=strtok(NULL,"/");
                n_etapa=atoi(etapa);
                if (n_etapa==ptr_etapa->n) {
                    unidade=strpbrk(ptr_variavel->texto,"cdsz");
                    tempo=strtok(NULL,"cdsz");
                    switch (* unidade) {
                        case 'c': n_unidade='0';
                                  break;
                        case 'd': n_unidade='1';
                                  break;
                        case 's': n_unidade='2';
                                  break;
                        case 'z': n_unidade='3';
                    } /* fim do case */
                    txt(dest,'F',ptr_etapa->endereco);
                    if (flag==0) poe_label(); /* só põe label uma vez */
                    flag=1;
                    fprintf(out,"\tA %s\n\r",dest);
                    fprintf(out,"\tL KT %s.%c\n\r",tempo,n_unidade);
                    fprintf(out,"\tSR T%u\n\r",ptr_variavel->endereco);
                    n_linhas+=3;
                } /* fim do if(n_etapa...) */
            } /* fim do if(ptr_variavel->...) */
            ptr_variavel=ptr_variavel->prox;
        } /* fim do while(ptr_variavel) */
    } /* fim do if(ptr_etapa->disparatemp...) */
    ptr_etapa=ptr_etapa->prox;
} /* fim do while(ptr_etapa) */
} /* fim da função */
```

8.5.2.4.acções do tipo set/reset (memorizadas)

Esta função recebe o número da etapa e gera o código correspondente ao *set* ou *reset* de(as) acção(s) memorizada(s) associada(s) a essa etapa. Isto é feito percorrendo a lista ligada de acções até ser encontrada a acção correspondente. O texto da acção é lido, compilado e é então gerado o código.

```
void set_reset_accao(id_etapa num_etapa)
{
    t_accao * ptr_accao=ll_accoes;
    char * ordem, letra, * dup, set_res;
    int sr;
    float enderec;

    while(ptr_accao->etapa!=num_etapa || ptr_accao->tipo!='S')
        ptr_accao=ptr_accao->prox;
    dup=strdup(ptr_accao->ordens);
    ordem=strtok(dup,";");
    while(ordem) {
        sscanf(ordem,"%c%f=%d",&letra,&enderec,&sr);
        if (sr==0) set_res='R';
        else set_res='S';
        fprintf(out,"\t%c %c%.1f\n\r",set_res,letra,enderec);
        n_linhas++;
        ordem=strtok(NULL,";");
    }
    free(dup);
}
```

8.5.2.5.reset das acções normais

Esta função recebe as ordens associadas a uma acção do tipo normal (excepto forçagens) e faz o *reset* dessas acções. Basicamente, faz o *reset* da saída física **Qx.y** associada à acção.

```
void reset_accao_normal(char * ordens)
{
    char * destin, * dup;

    dup=strdup(ordens);
    destin=strtok(dup,";");
    while (destin) {
        if (strpbrk(destin,"/")==NULL) {
            fprintf(out,"\tR %s\n\r",destin);
            n_linhas++;
        }
    destin=strtok(NULL,";");
    }
    free(dup);
}
```

8.5.2.6.reset das acções

Esta função recebe o número da etapa e gera o código correspondente ao *reset* de todas as acções associadas à etapa. Usa as funções **reset_accao_normal()** (descrita anteriormente) assim como a **reset_forcagens()** que se encontra no módulo "accoes.c".

```
void reset_accoes(id_etapa num_etapa)
{
    t_accao * ptr_accao=ll_accoes;

    while (ptr_accao) {
        if (ptr_accao->etapa==num_etapa)
            if (ptr_accao->tipo==' ') {
                /* faz reset acções das acções normais (excepto forçagens) */
                reset_accao_normal(ptr_accao->ordens);
                /* faz reset das forçagens */
                reset_forcagens(ptr_accao->ordens);
            }
        ptr_accao=ptr_accao->prox;
    }
}
```

8.6. Acções

A implementação do código (em STEP5) das accões é feita com base na informação contida na lista ligada das acções. As acções são implementadas uma a uma á medida que se percorre a respectiva lista ligada. Para cada tipo de acção existe uma função que gera o respectivo código.

Por razões de eficiência do código, as **Acções Normais** e as **Acções tipo S**(memorizadas) não são tratadas juntamente com as outras, mas sim quando se gera o código de activação e desactivação das etapas (que é executado pelo autómato nas transposições das transições). As **Acções tipo P** (impulsionais), as **Acções tipo ?**(condicionais), as **Acções tipo L** (limitadas no tempo) e as **Acções tipo D**(atrasadas no tempo) são tratadas em **accoes()**. A diferença está no facto das segundas serem executadas em todos os ciclos do autómato e as primeiras serem activadas e desactivadas em paralelo com etapas a que estão ligadas, não havendo necessidade do autómato executar o respectivo código em todos os ciclos de programa.

As acções que envolvem o uso temporizadores têm que ser executadas em todos os ciclos devido á estrutura interna do autómato. É por essa razão que as **Acções tipo ?**, as **Acções tipo L** e as **Acções tipo D** são tratadas em **accoes()**. Quanto ás **Acções tipo P** verificou-se que para serem tratadas juntamente com as **Acções Normais** e as **Acções tipo S** seria necessário criar código mais complexo que absorveria eventuais ganhos a obter com este tipo de implementação.

A estrutura base da implementação do códogo das acções pode ser vista abaixo em **accoes()** que se encontra no ficheiro compac.c.

```
void accoes(void)
{
 char booleana;
 ptr_accoes=ll_accoes;

 while (ptr_accoes){
    switch (ptr_accoes->tipo){
      case 'P': proc_accao_impulsional(ptr_accoes,ptr_accoes->ordens);
                break;
      case 'S': /*  As acções memorizadas são implementadas em des_acti.c  */
                break;
      case 'L': accao_temp_L(ptr_accoes,ptr_accoes->ordens);
                break;
      case 'D': accao_temp_D(ptr_accoes,ptr_accoes->ordens);
                break;
      case '?': accao_condicional(ptr_accoes,ptr_accoes->ordens);
                break;
      case ' ': /* As acções normais são  implementadas em des_acti.c */
                break;
      default:  printf("erro de compilação ou de síntese\n");
    }
    ptr_accoes=ptr_accoes->prox;
 }
}
```

8.6.1. Acções Normais

O código(STEP5) das acções normais é tratado pela função **proc_accao_normal**. Esta função tem como parâmetro de entrada um apontador para uma cadeia de caracteres constituida pelas ordens da acção separadas por ponto e vírgula.

Dentro da função, a primeira coisa que se faz é criar um duplicado de **ordens** na variável **dup_str** para trabalhar sem problemas com a função **strtok()**.

As ordens encontram-se separadas por ";" sendo processadas uma a uma através de um ciclo *while ( )* e do uso de **strtok**'s. Se a ordem é uma forçagem, consoante o tipo de forçagem assim será gerado o respectivo código.Senão será gerado apenas o"set" da respectiva saída por:

fprintf(out,"\tS %s\n\r",ordem)

Estão previstos vários tipos de forçagem: congelamento, forçagem da situação inicial, forçagem da situação vazia e forçagem normal (forçagem em que o projectistado grafcet enumera as etapas a forçar).

Congelamento: é implementado fazendo o "*reset*" de todas as etapas de saída do grafcet forçado.Para que o autómato não esteja a calcular as condições de evolução durante o congelamento faz-se também o "*reset*" da sensibilidade da transição

Forçagem da situação inicial: é implementada percorrendo a lista ligada de etapas fazendo o "*set*" das que pertencem ao grafcet forçado e são do tipo "INICIAL". Se houver acções **tipo S** ou **tipo Normal** nas etapas forçadas é necessário gerar código adequado para activa-las usando a função **setreset** primeiro caso e **proc_accao_normal** no segundo. As etapas do grafcet forçado que não são do tipo "INICIAL" sofrem um "*reset*", sendo desactivadas as respectivas acções por **reset_accoes()**. Para que a forçagem se mantenha enquanto a etapa está activa, é necessário produzir código de congelamento pois este código de forçagem é executado apenas na activação das etapas.

Forçagem normal e da situação vazia: situação é análoga á anterior estando neste caso a informação das etapas a forçar não na lista ligada das etapas, mas na própria ordem.

```
void proc_accao_normal(char *ordens)
{
    char *ordem,*corrente,*existeforcagem,*dup_str,letra,buffer[]={"X123.4"};
    t_grafcet n_grafcet;
    t_endereco ender,int_ender;
    t_transic *p_transic;
    t_etapa *p_etapa,*sets[MAXETFORC];
    t_accao *p_accao,*ptr_accao;
    float fl_ender;
    int i,x;

    dup_str = strdup(ordens);
    ordem=strtok(dup_str,";");
    while(ordem) {
        if ((existeforcagem=strchr(ordem,'/'))!=NULL) {             /* vê se a ordem é uma
forçagem,se não for salta */
           sscanf(ordem,"F/%d",&n_grafcet);              /*extrai para n_grafcet a identificação
do grafcet, que  é um inteiro */
           existeforcagem=strchr(ordem,'(');
           switch (existeforcagem[1]) {
              case '*': /* congelar */
                 for (p_transic=ll_transic;p_transic;p_transic=p_transic->prox)
                    if (n_grafcet==p_transic->grafcet){
                               txt(buffer,'F',p_transic->sensib);
                               fprintf(out,"\tR %s\n\r",buffer);
                               txt(buffer,'F',prox(p_transic->sensib));
                               fprintf(out,"\tR %s\n\r",buffer);
                               n_linhas+=2;
                    }
                 break;
              case 'I': /* força situação inicial */
                    for (p_etapa=ll_etapas;p_etapa;p_etapa=p_etapa->prox)
```

```
if (n_grafcet==p_etapa->grafcet){
 if (p_etapa->classif==INICIAL){
          txt(buffer,'F',p_etapa->endereco);
          fprintf(out,"\tS %s\n\r",buffer);
          n_linhas++;
          /* processa acções do tipo set/reset */
          if (p_etapa->setreset==1) set_reset_accao(p_etapa->n);
          /* processa acções normais */
          ptr_accao=ll_accoes;
          while (ptr_accao) {
                if (ptr_accao->etapa==p_etapa->n && ptr_accao-
>tipo==' ')
                                     proc_accao_normal(ptr_accao-
>ordens);
                ptr_accao=ptr_accao->prox;
          }
 }
 else {
          txt(buffer,'F',p_etapa->endereco);
          fprintf(out,"\tR %s\n\r",buffer);
          n_linhas++;
          reset_accoes(p_etapa->n);
 }
}
/* produz código de congelamento para que a forçagem */
/* se mantenha enquanto a etapa est  activa, pois este   */
/* código □ executado apenas na activação das etapas    */
for (p_transic=ll_transic;p_transic;p_transic=p_transic->prox)
    if (n_grafcet==p_transic->grafcet){
          txt(buffer,'F',p_transic->sensib);
          fprintf(out,"\tR %s\n\r",buffer);
          txt(buffer,'F',prox(p_transic->sensib));
          fprintf(out,"\tR %s\n\r",buffer);
          n_linhas+=2;
    }
break;
default:        /*forçagem  normal e da situação vazia*/
    /* faz set das flags listadas */
    corrente=strchr(ordem,'(')+1;
    if (*corrente==' '||*corrente==')') *corrente=NULL;/* forçagem da situação
vazia */
    else{
         i=0;
         while(*corrente){
                sscanf(corrente,"%c%f",&letra,&fl_ender);
                fprintf(out,"\tS %c%.1f\n\r",letra,fl_ender);
```

```
                        n_linhas++;
                        fl_ender=fl_ender*10;
                        int_ender=fl_ender;
                        p_etapa=e_procura(int_ender);
                        sets[i]=p_etapa;
                        i++;
                              /* processa acções do tipo set/reset(memorizadas) */
                        if (p_etapa->setreset==-1) set_reset_accao(p_etapa->n);
                              /* processa acções normais */
                        ptr_accao=ll_accoes;
                        while (ptr_accao) {
                                    if (ptr_accao->etapa==p_etapa->n && ptr_accao-
>tipo==' ')
                                          proc_accao_normal(ptr_accao->ordens);
                                    ptr_accao=ptr_accao->prox;
                        }
                        corrente=strchr(corrente+1,',')+1;
                  }
            }
            i--;
            /* faz reset das as etapas de grafcet */
            for (p_etapa=ll_etapas;p_etapa;p_etapa=p_etapa->prox)
                if (n_grafcet==p_etapa->grafcet){
                        for(x=0;x<=i;x++){
                          if (sets[i]!=p_etapa){
                            txt(buffer,'F',p_etapa->endereco);
                            fprintf(out,"\tR %s\n\r",buffer);
                            reset_accoes(p_etapa->n);
                            n_linhas++;
                          }
                        }
                }
            /* produz código de congelamento para que a forçagem  */
            /* se mantenha enquanto a etapa est  activa, pois este*/
            /* código □ executado apenas na activação das etapas  */
                for (p_transic=ll_transic;p_transic;p_transic=p_transic->prox)
                if (n_grafcet==p_transic->grafcet){
                        txt(buffer,'F',p_transic->sensib);
                        fprintf(out,"\tR %s\n\r",buffer);
                        txt(buffer,'F',prox(p_transic->sensib));
                        fprintf(out,"\tR %s\n\r",buffer);
                        n_linhas+=2;
                }
      } /* fim do switch do tipo de forçagem */
    } /* else do if (existeforçagem...) - forçagem */
```

```
        else  {
                fprintf(out,"\tS %s\n\r",ordem);
                n_linhas++;
        }
        ordem=strtok(NULL,";");
    } /* fim do while (ordem) ,tratamento das ordens        */
    free(dup_str);
}
```

8.6.2. Acções Especiais

8.6.2.1.Acções Impulsionais

A implementação das acções impulsionais é muito semelhante à das acções normais sendo a principal diferença o facto das forçagens não necessiterem de código de congelmento já que se pretende que elas sejam de facto impulsionais. Por outro lado como este código é executado em todos os ciclos de programa do autómato não é necessário desactivar as acções das etapas do grafcet que sofreram um "*reset*". Outra diferença é a necessidade de haver um teste de impulsionalidade no programa do autómato para que a acção só esteja activa um ciclo de programa (do autómato) após activação da respectiva etapa. Para isso existe uma "*flag*" extra para esta etapa que indica o seu estado anterior, sendo o valor lógico do teste impulsionalidade dado pela operação lógica: **/(estado_anterior)*(estado_actual)**. O código de actualização dos estados actual e anterior da etapa é executado no fim do ciclo de programa do autómato.

```
void proc_accao_impulsional(t_accao *accao,char *ordens)
{
    char *destin,*corrente,*existeforcagem,*dup_str,letra,buffer[]={"X123.4"};
    t_grafcet n_grafcet;
    t_endereco ender,int_ender;
    t_transic *p_transic;
    t_etapa *p_etapa;
    float fl_ender;
    t_elem primeiro;
    t_etapa *etapa,*sets[MAXETFORC];
    t_pilha *pilha;
    char buff[10];
    int i,x;

    dup_str = strdup(ordens);

    /*************************************************************************************/
        /* ESTA PARTE PRODUZ O CODIGO DE TESTE DE IMPULSIONALIDADE */
/*************************************************************************************/
    buff[0]='^';
    txt(&(buff[1]),'F',(procura_e(accao->etapa))->endereco);
    pilha=iniciaNPI(buff); /* faz malloc pilha e faz NPI */
```

```
if (pop(pilha,&primeiro)==UNDERFLOW)
{
   printf ("Erro de lógica na acção impulsional\n");exit (ERRO);
}
if (primeiro.tipo==NUM)
{
   if (primeiro.info.num.nega==NEGADO) fprintf (out,"\tAN"); else fprintf (out,"\tAN");
   fprintf (out," %c%.1f\n\r",primeiro.info.num.letra,primeiro.info.num.n);
   n_linhas+=2;
}
else
{
   escrevelop(primeiro.info.op.op,pilha);
}


/************************************************************************/



destin=strtok(dup_str,";");

while(destin)
{
    if ((existeforcagem=strchr(destin,'/'))!=NULL){
    sscanf(destin,"F/%d",&n_grafcet);
    existeforcagem=strchr(destin,'(');
    switch (existeforcagem[1]) {
       case '*': /* congelar */
           for (p_transic=ll_transic;p_transic;p_transic=p_transic->prox)
               if (n_grafcet==p_transic->grafcet) {
                           txt(buffer,'F',p_transic->sensib);
                           fprintf(out,"\tR %s\n\r",buffer);
                           n_linhas++;
               }
       break;

       case 'I': /* força situação inicial */
           for (p_etapa=ll_etapas;p_etapa;p_etapa=p_etapa->prox)
               if (n_grafcet==p_etapa->grafcet){
                   if (p_etapa->classif==INICIAL){
                           txt(buffer,'F',p_etapa->endereco);
                           fprintf(out,"\tS %s\n\r",buffer);
                           n_linhas++;
                   }
```

```
                else {
                            txt(buffer,'F',p_etapa->endereco);
                            fprintf(out,"\tR %s\n\r",buffer);
                            n_linhas++;
                }
                  }
        break;

        default:
             /* faz set das flags listadas */
            corrente=strchr(destin,'(')+1;
             if (*corrente==' '||*corrente==')') *corrente=NULL;/* forçagem da situação
vazia */
            else{
                    i=0;
                    while(*corrente){
                            sscanf(corrente,"%c%f",&letra,&fl_ender);
                            fprintf(out,"\tS %c%.1f\n\r",letra,fl_ender);
                            n_linhas++;
                            fl_ender=fl_ender*10;
                            int_ender=fl_ender;
                            p_etapa=e_procura(int_ender);
                            sets[i]=p_etapa;
                            i++;
                            corrente=strchr(corrente+1,',')+1;
                    }
            }
            i--;
            /* faz reset das as etapas de grafcet */
            for (p_etapa=ll_etapas;p_etapa;p_etapa=p_etapa->prox)
                if (n_grafcet==p_etapa->grafcet)
                {
                            for(x=0;x<=i;x++){
                              if ((sets[i]->n)!=(p_etapa->n)){
                                txt(buffer,'F',p_etapa->endereco);
                                fprintf(out,"\tR %s\n\r",buffer);
                                n_linhas++;
                              }
                            }
                }
    } /* fim do switch do tipo de forçagem */
      } /* else do if (existeforcagem...) - forçagem */
      else  {
            fprintf(out,"\t=%s\n\r",destin);
            n_linhas++;
```

```
        }
        destin=strtok(NULL,":");
    } /* fim do while (destin) */
    free(dup_str);
}
```

8.6.2.2.Acções Memorizadas

O código das acções memorizadas é gerado juntamente com o código de activação e desactivação das etapas (ver 9.5. Activação / Desactivação de etapas).

8.6.2.3.Acções Tipo L

As acções tipo L são tratadas pela função **accao_temp_L**. Nesta função a primeira coisa que se faz é tirar uma cópia da cadeia de caracteres das ordens para dup_str. Também aqui se processa a acção ordem a ordem. Para tal extrai-se a informação de dup_str procurando os caracteres separadores das ordens (que são ';').

O código STEP5 duma acção tipo L tem a seguinte estrutura:

A {endereço da etapa}

L KT {(valor).(unidade de tempo)}

SP Tx

= Q{endereço da saída}

em que x é o número do timer. Para construir este código é portanto necessário saber o endereço da etapa, o valor e a unidade de tempo, o endereço da saída e que timer usar. O endereço da etapa é colocado na posição apontada por estado_etapa por:

txt(estado_etapa,'F',procura_e(accao->etapa)->endereco)

A informação relativa ao valor e a unidade de tempo é retirada da ordem (que tem a sintaxe: **Qx.y=valor{c,d,s,z}**) por:

p=strtok(dup_str,"=");

p=strtok(NULL,"=");

switch (p[(strlen(p))-1])

```
{
 case 'c':               /*unidade centésimos de segundos*/
   p=strtok(p,"c");
   sprintf(dest,"%s.0",p);
   break;
        .
        {...}
        .
 default:printf("erro na acção temp L\n\r");
}
```

O timer é "*alocado*" por:

timer=ultimo_temp+1

e é incrementado o número de timers usados:

ultimo_temp++

AS últimas linhas da função " fprintf(out,... " produzem o código STEP5 usando estas informações.

```
void accao_temp_L(t_accao *accao,char *ordens)
{
 char dest[25],*p,estado_etapa[25],*dup_str,*destin;
 U timer;

      dup_str=strdup(ordens);
      destin=strtok(dup_str,":");
      while (destin)
      {
       txt(estado_etapa,'F',procura_e(accao->etapa)->endereco);
       p=strtok(dup_str,"=");
       p=strtok(NULL,"=");
       switch (p[(strlen(p))-1])
         {
          case 'c':                 /*unidade centésimos de segundos*/
             p=strtok(p,"c");
```

```
            sprintf(dest,"%s.0",p);
            break;
         case 'd':            /*unidade décimos de segundos*/
            p=strtok(p,"d");
            sprintf(dest,"%s.1",p);
            break;
         case 's':            /*unidade segundos*/
            p=strtok(p,"s");
            sprintf(dest,"%s.2",p);
            break;
         case 'z':            /*unidade dezenas de segundos*/
            p=strtok(p,"z");
            sprintf(dest,"%s.3",p);
            break;
         default:printf("erro na acção temp L\n\r");
        }
      fprintf(out,"\tA %s\n\r",estado_etapa);
      fprintf(out,"\tL KT %s\n\r",dest);
      n_linhas+=2;
      timer=ultimo_temp+1;
      ultimo_temp++;
      fprintf(out,"\tSP T%d\n\r",timer);
      fprintf(out,"\tA T%d\n\r",timer);
      fprintf(out,"\t=%s\n\r",dup_str);
      n_linhas=n_linhas+3;
      destin=strtok(NULL,":");
     }/*fim do while (destin) */
     free(dup_str);

}
```

8.6.2.4.Acções Tipo D

As acções tipo D são implementadas de modo perfeitamente análogo ás acções tipo L não se justificando uma repetição do que foi dito na secção anterior.A função que produz o código destas acções é **accao_temp_D**.

```
void accao_temp_D(t_accao *accao,char *ordens)
{
 char *dup_str,dest[25],*p,estado_etapa[25],*destin;
 U timer;


     dup_str=strdup(ordens);
```

```
    destin=strtok(dup_str,":");
    while (destin)
    {
     txt(estado_etapa,'F',procura_e(accao->etapa)->endereco);
     p=strtok(dup_str,"=");
     p=strtok(NULL,"=");
     switch (p[(strlen(p))-1])
       {
        case 'c':
           p=strtok(p,"c");
           sprintf(dest,"%s.0",p);
           break;
        case 'd':
           p=strtok(p,"d");
           sprintf(dest,"%s.1",p);
           break;
        case 's':
           p=strtok(p,"s");
           sprintf(dest,"%s.2",p);
           break;
        case 'z':
           p=strtok(p,"z");
           sprintf(dest,"%s.3",p);
           break;
        default:printf("erro na ac‡„o temp D\n\r");
       }
     fprintf(out,"\tA %s\n\r",estado_etapa);
     fprintf(out,"\tL KT %s\n\r",dest);
     n_linhas+=2;
     timer=ultimo_temp+1;
     ultimo_temp++;
     fprintf(out,"\tSR T%d\n\r",timer);
     fprintf(out,"\tA T%d\n\r",timer);
     fprintf(out,"\t=%s\n\r",dup_str);
     n_linhas=n_linhas+3;
     destin=strtok(NULL,":");
    }/* fim do while destin */
    free(dup_str);
}
```

8.6.2.5.Acções Condicionais

As acções condicionais são tratadas pela função **accao_condicional**. Nesta função as ordens são processadas uma a uma através de um ciclo *while( )*. Para gerar o código STEP5 das

condições contidas na ordem recorre-se às funções **iniciaNPI** e **escrevelop** definidas anteriormente.

```
void accao_condicional(t_accao *accao,char *ordens)
{
    char *dup_str,*destin,*ptr,letra,estado_etapa[25];
    t_etapa *p_etapa;
    t_elem primeiro;
    t_pilha *pilha;

    dup_str=strdup(ordens);
    destin=strtok(dup_str,";");
    while (destin)
    {
      txt(estado_etapa,'F',procura_e(accao->etapa)->endereco);
      fprintf(out,"\tA %s\n\r",estado_etapa);
      n_linhas++;
      pilha=iniciaNPI(&(ptr[1])); /* faz malloc pilha e faz NPI */
      if (pop(pilha,&primeiro)==UNDERFLOW)
      {
         printf ("Erro de lógica na acção condicional\n");exit (ERRO);
      }
      if (primeiro.tipo==NUM)
      {
         if (primeiro.info.num.nega==NEGADO) fprintf (out,"\tAN"); else fprintf (out,"\tA");
         fprintf (out," %c%.1f\n\r",primeiro.info.num.letra,primeiro.info.num.n);
         n_linhas+=2;
      }
      else
      {
         escrevelop(primeiro.info.op.op.pilha);
      }
      ptr=strtok(destin,"=");
      fprintf(out,"\t=%s\n\r",ptr);
      destin=strtok(NULL,";");
    }/* fim do while (destin) */
    free(dup_str);
```

8.7. Escrita de OBs e PBs

Neste ficheiro, escrito no fim do fim da compilação, trata-se de escrever os ficheiros relativos à inicialização do trabalho do A.P. .

No bloco OB21, trata-se de inicilizar todas as etapas do grafcet no seu devido estado.

No bloco OB1, apenas se chama em sequência todos os FBs atrás definidos.

No PB1 calculam-se as variáveis auxiliares e no PB2 trata-se o problema de guardar em locais próprios os valores anteriores de algumas variáveis.

```
void ob21(void)  /* organização inicial */
{
    t_etapa * ptr_etapa=ll_etapas;
    char dest[10];

    fprintf(out,"<OB21>\n\r");
    while (ptr_etapa) {
       if (ptr_etapa->classif==INICIAL) {
          txt(dest,'F',ptr_etapa->endereco);
          fprintf(out,"\tS %s\n\r",dest);
          suscept_transic_saida(ptr_etapa);
          if (ptr_etapa->setreset==-1) set_reset_accao(ptr_etapa->n);
       } else {
          txt(dest,'F',ptr_etapa->endereco);
          fprintf(out,"\tR %s\n\r",dest);
          desuscept_transic_saida(ptr_etapa);
       }

       ptr_etapa=ptr_etapa->prox;
    }
    fprintf(out,"\tBE\n\r<FB1>\n\r");
}


void ob1(void)      /* organização geral */
{
U char i;

    fprintf (out,"<OB1>\n\r");

    fprintf (out,"\tJU PB1\n\r\tBE\n\r");

    for (i=1;i<=ultimo_fb;++i)
       fprintf (out,"\tJU FB%d\n\r",i);
```

```
    fprintf (out,"\tJU PB2\n\r\tBE\n\r");

}

void pb2(void)  /* a executar no fim dos ciclos */
{
t_variavel *var;

    fprintf (out,"<PB2>\n\r");

    for (var=procura_letra_v(ll_variaveis,'<'); var;
         var=procura_letra_v(var->prox,'<'))
    {
        fprintf (out,"\tA %s\n\r",var->texto);
        fprintf (out,"\t=F%.1f\n\r",var->endereco/10.0);
    }

    fprintf (out,"\tBE\n\r");

}

void pb1(void)    /* a executar no principio dos ciclos */
{
t_variavel *var;
t_elem primeiro;
t_pilha *pilha;

    fprintf (out,"<PB1>\n\r");

    for (var=procura_letra_v(ll_variaveis,'*'); var;
         var=procura_letra_v(var->prox,'*'))
    {

        pilha=iniciaNPI(var->texto); /* faz malloc pilha e faz NPI */

        if (pop(pilha,&primeiro)==UNDERFLOW) {printf ("Erro de lógica na escrita da v. aux.:
%s",var->texto);mostrastack(pilha);exit (ERRO);}
        if (primeiro.tipo==NUM) {
            if (primeiro.info.num.nega==NEGADO) fprintf (out,"\tAN"); else fprintf (out,"\tA");
            fprintf (out," %c%.*f\n\r",
                          primeiro.info.num.letra,
                          primeiro.info.num.letra!='I' ? 1 : 0,
                          primeiro.info.num.n);
            n_linhas++;
        } else escrevelop(primeiro.info.op.op,pilha);
```

```
        fprintf (out,"\t=F%.1f\n\r",var->endereco/10.0);
        n_linhas++;

        free (pilha); /* liberta pilha temporária */
    }

    fprintf (out,"\tBE\n\r");

}
```

8.8. Programa Principal

O programa principal encontra-se no módulo "compila.c".

Este programa contém instruções de compilação condicional que só são executadas aquando do teste do programa (#if DEBUG).

A primeira parte do programa principal faz a leitura dos argumentos da linha de comando por forma a determinar qual o ficheiro de entrada e o ficheiro de saída do compilador. Se o nome desses ficheiros for omitido ou dado de forma incorrecta será gerada uma mensagem de erro.

Na segunda parte do programa, é lido o ficheiro de entrada do compilador (pela função **lefich()**) e são chamadas ordenadamente as funções que geram o código STEP5 presente no ficheiro de saída. Essas funções são : **ob21()**, **compilatransic()**, **desactiva_activa_etapas()**, **accoes()**, **pb1()**, **pb2()** e **ob1()**.

Após a chamada a cada uma das funções é corrida a função **flushall()** por forma a limpar todos os *buffers* e escrever o seu conteúdo no ficheiro de saída.

Por fim é chamada a função **fcloseall()** que já foi descrita numa das secções anteriores.

9. *Grafcets de teste*

9.1. Metodologia

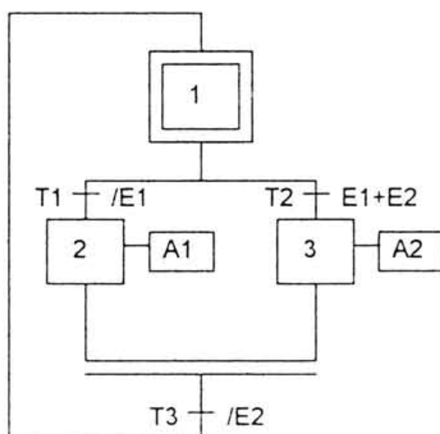
Para o teste do programa foram usados uma Programadora e um Autómato da SIEMENS fornecidos para o efeito. Procurou-se testar o mair número de grafcets por forma a verificar a total operacionalidade do compilador de grafcet. Assim, do programa mais simples até ao mais complexo, tentou-se cobrir todas as situações possíveis que possam ocorer num dado grafcet.

Os grafcets a testar foram convertidos (manualmente) para o formato do ficheiro de entrada do compilador sendo posteriormente (após compilação) passados também manualmente para a Programadora do Autómato. Estas fases, como seriam de esperar, eram muito morosas, pelo que o número de grafcets testados foi reduzido. Por fim, após a transmissão do código da Programadora para o Autómato, era então efectuado o teste através do conjunto de entradas e saídas fisicas presentes no Autómato. Nesse teste eram varridas todas as situações possíveis do grafcet, verificando-se se os resultados obtidos correspondiam aos esperados.

9.2. Grafcets testados

Seguidamente apresentam-se os grafcets usados para o para o teste do programa. Juntamente com a representação gráfica dos grafcets estão os ficheiros de entrada e saída do compilador de grafcet.

9.2.1. Grafcet com acções simples



FICHEIRO DE ENTRADA:

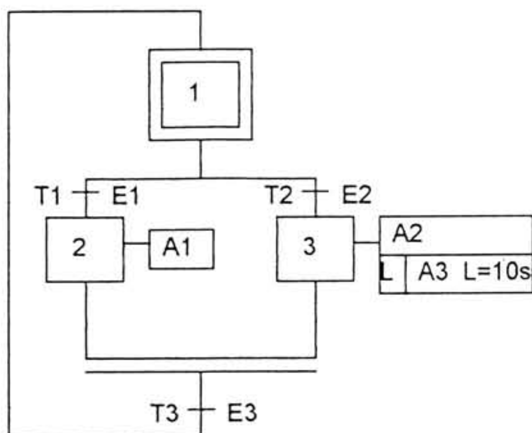
```
:I0.1=E1
:I0.2=E2
:I0.3=E3

:Q0.0=A1
:Q0.1=A2

GTeste

#1
E 3
S 1,2
I

#2
E 1
S 3
A A1

#3
E 2
S 3
A A2

T1 /E1
T2 E1+E2
T3 /E2
```

FICHEIRO DE SAÍDA:

```
<OB21>

        S F0.1
        S F0.4
        S F0.6
        R F1.0
        R F0.2
        R F1.1
        R F0.2
        BE
<FB1>
L1:     AN F0.2
        JC =L2
        ON F1.0
        ON F1.1
        JC =L2
        AN I0.2
        =F0.3

L2:     AN F0.4
        JC =L3
        ON F0.1
        JC =L3
        AN I0.1
        =F0.5
L3:     AN F0.6
        JC =L4
        ON F0.1
        JC =L4
        O I0.2
        O I0.1
        =F0.7
L4:     AN F0.3
        JC=L5
        R F1.0
        R Q0.0
        R F1.1
        R Q0.1
        R F0.2
L5:     AN F0.5
        JC=L6
        R F0.1
        R F0.4
L6:     AN F0.7
        JC=L7
        R F0.1
```

```
        R F0.6
L7:     AN F0.3
        JC=L8
        R F0.3
        S F0.1
        S F0.4
        S F0.6
L8:     AN F0.5
        JC=L9
        R F0.5
        S F1.0
        S F0.2
        S Q0.0

L9:     AN F0.7

        JC=L10

        R F0.7

        S F1.1

        S F0.2
        S Q0.1

<PB1>

        BE

<PB2>

        BE

<OB1>

        JU PB1

        BE

        JU FB1

        JU PB2

        BE
```

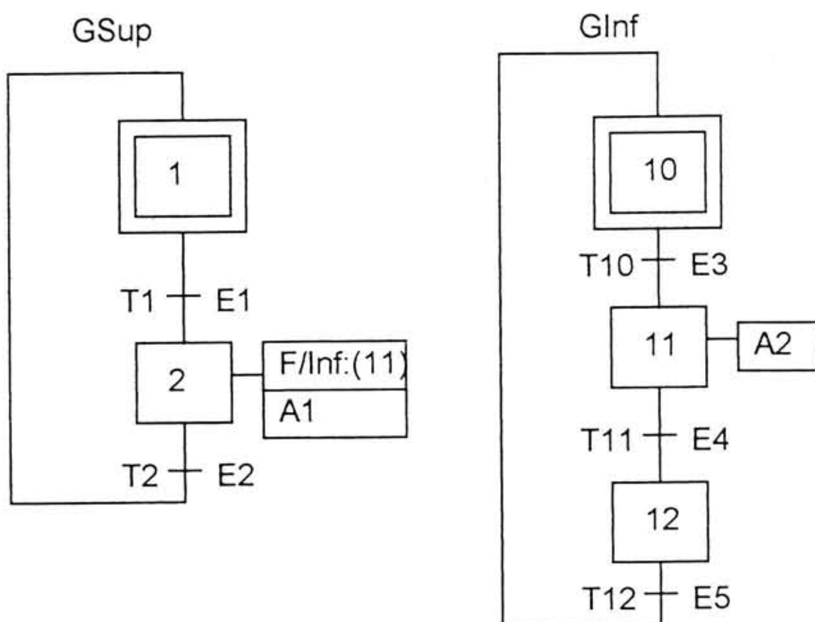
9.2.2. Grafcet com acções de duração limitada




```
FICHEIRO DE ENTRADA:


:I0.1=E1
:I0.2=E2
:I0.3=E3

:Q0.0=A1
:Q0.1=A2
:Q0.2=A3

GTeste

#1
E 3
S 1,2
I

#2
E 1
S 3
A A1

#3
E 2
S 3
A A2
AL A3=10s

T1 E1
T2 E2
T3 E3

FICHEIRO DE SAÍDA:

<OB21>
        S F0.1
        S F0.4
        S F0.6
        R F1.0
        R F0.2
        R F1.1
        R F0.2
        BF

<FB1>
L1:     AN F0.2
        JC =L2
        ON F1.0
        ON F1.1
        JC =L2
        A I0.3
        =F0.3
L2:     AN F0.4
        JC =L3
        ON F0.1
        JC =L3
```

```
        A I0.1
        =F0.5
L3:     AN F0.6
        JC =L4
        ON F0.1
        JC =L4
        A I0.2
        =F0.7
L4:     AN F0.3
        JC=L5
        R F1.0
        R Q0.0
        R F1.1
        R Q0.1
        R F0.2
L5:     AN F0.5
        JC=L6
        R F0.1
        R F0.4
L6:     AN F0.7
        JC=L7
        R F0.1
        R F0.6
L7:     AN F0.3
        JC=L8
        R F0.3
        S F0.1
        S F0.4
        S F0.6
L8:     AN F0.5
        JC=L9
        R F0.5
        S F1.0
        S F0.2
        S Q0.0

L9:     AN F0.7
        JC=L10

        R F0.7

        S F1.1

        S F0.2
        S Q0.1

L10:    A F1.1

        L KT 10.2

        SP T1

        A T1

        =Q0.2

<PB1>

        BE

<PB2>

        BE

<OB1>

        JU PB1

        BE

        JU FB1

        JU PB2

        BE
```

9.2.3. Grafcet com forçagens




```
FICHEIRO DE ENTRADA:
:I0.0-E1
:I0.1-E2
:I0.2 E3
:I0.3 E4
:I0.4 E5

:Q3.0-A1
:Q3.1 A2


GInf

#10
E 12
S 10
I

#11
E 10
S 11
A A2

#12
E 11
S 12


GSup

#1
E 2
S 1
I

#2
E 1
S 2
A F/Inf:(11)
A A1


T1 E1
```

```
T2 E2
T10 E3
T11 E4
T12 E5
```

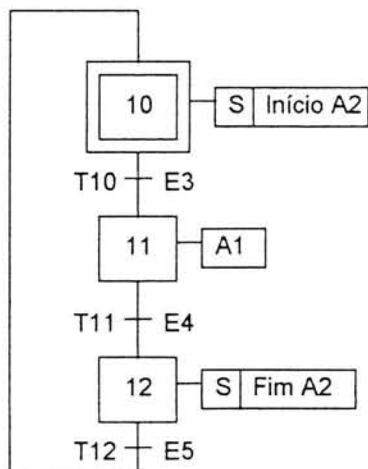
FICHEIRO DE SAÍDA:

```
<OB21>

        S F0.1

        S F0.4
        R F0.6

        R F0.7
        R F1.1

        R F0.2
        S F1.2

        S F1.5
        R F1.7

        R F1.3
        BE

<FB1>

L1:     AN F0.2

        JC =L2

        ON F1.1

        JC =L2

        A I0.4

        =F0.3

L2:     AN F0.4

        JC =L3

        ON F0.1

        JC =L3

        A I0.2

        =F0.5

L3:     AN F0.7

        JC =L4

        ON F0.6

        JC =L4

        A I0.3

        =F1.0

L4:     AN F1.3

        JC =L5

        ON F1.7

        JC =L5

        A I0.1

        =F1.4

L5:     AN F1.5

        JC =L6

        ON F1.2

        JC =L6

        A I0.0

        =F1.6

L6:     AN F0.3
```

```
        JC=L7
        R F1.1
        R F0.2
L7:     AN F0.5
        JC=L8
        R F0.1
        R F0.4
L8:     AN F1.0
        JC=L9
        R F0.6
        R Q3.1
        R F0.7
L9:     AN F1.4
        JC=L10
        R F1.7
        S F0.7
        R Q3.0
        R F1.3
L10:    AN F1.6
        JC=L11
        R F1.2
        R F1.5
L11:    AN F0.3
        JC=L12
        R F0.3
        S F0.1
        S F0.4
L12:    AN F0.5
        JC=L13
        R F0.5
        S F0.6
        S F0.7
        S Q3.1
L13:    AN F1.0
        JC=L14
        R F1.0
        S F1.1
        S F0.2
L14:    AN F1.4
        JC=L15
        R F1.4
        S F1.2
        S F1.5
L15:    AN F1.6
        JC=L16
        R F1.6
        S F1.7
        S F1.3
        R F0.1
        R F0.6
        R F1.1
        S F0.6
```

```
S Q3.1
R F0.2
R F0.3
R F0.4
R F0.5
R F0.7
R F1.0
S Q3.0
<PB1>
BE
<PB2>
BE
<OB1>
JU PB1
BE
JU FB1
JU PB2
BE
```

9.2.4. Grafcet com acções set/reset



FICHEIRO DE ENTRADA:

```
:I2.0=E3
:I2.1=E4
:I2.2=E5

:Q3.0=A1
:Q3.1=A2

GInf

#10
E 12
S 10
AS A2=1

I

#11
E 10
S 11
A A1

#12
E 11
S 12
AS A2=0

T10 E3
T11 E4
T12 E5
```

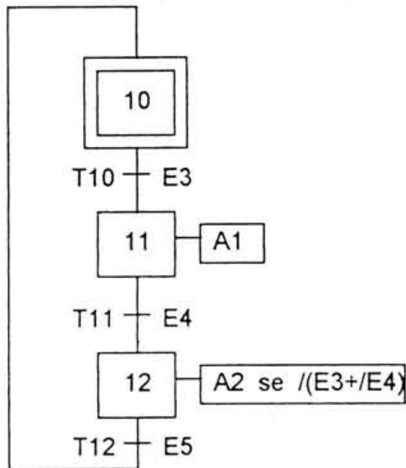
FICHEIRO DE SAÍDA:

```
[illegible]

    S F0.1

    S F0.4
    S Q3.1

R F0.6

R F0.7
R F1.1

R F0.2
```

```
        BE

<FB1>

L1:     AN F0.2
        JC =L2
        ON F1.1
        JC =L2
        A I2.2
        =F0.3
L2:     AN F0.4
        JC =L3
        ON F0.1
        JC =L3
        A I2.0
        =F0.5
L3:     AN F0.7
        JC =L4
        ON F0.6
        JC =L4
        A I2.1
        =F1.0
L4:     AN F0.3
        JC=L5
        R F1.1
        R F0.2
L5:     AN F0.5
        JC=L6
        R F0.1
        R F0.4
L6:     AN F1.0
        JC=L7
        R F0.6
        R Q3.0
        R F0.7
L7:     AN F0.3
        JC=L8
        R F0.3
        S F0.1
        S F0.4
        S Q3.1
L8:     AN F0.5
        JC=L9
        R F0.5
        S F0.6
        S F0.7
        S Q3.0
L9:     AN F1.0
        JC=L10
        R F1.0
        S F1.1
        S F0.2
        R Q3.1
```

```
<PB1>

        BE

<PB2>

        BE

<OB1>

JU PB1

BE

JU FB1

JU PB2

BE
```

9.2.5. grafcet com acções condicionais




```
FICHEIRO DE ENTRADA:

:I2.0 E3
:I2.1 E4
:I2.2 E5

:Q3.0 A1
:Q3.1-A2

[illegible]

#10
E 12
S 10

I

#11
E 10
S 11
A A1

#12
E 11
S 12
A? A2=/(E3+/E4)

T10 E3
T11 E4
```

T12 E5

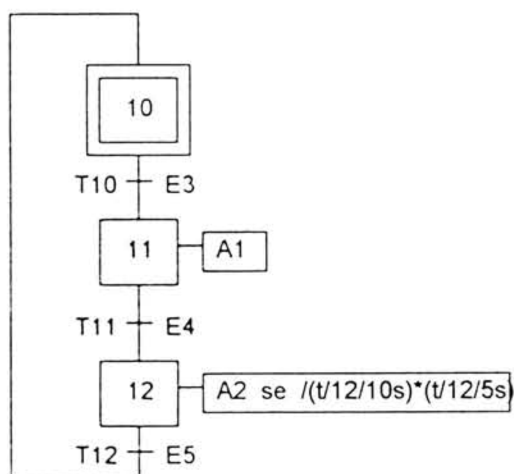
FICHEIRO DE SAÍDA:

```
<OB21>

        S F0.1

        S F0.4
        R F0.6

        R F0.7
        R F1.1

        R F0.2
        BE

<FB1>

L1:     AN F0.2

        JC =L2

        ON F1.1

        JC =L2

        A I2.2

        =F0.3

L2:     AN F0.4

        JC =L3

        ON F0.1

        JC =L3

        A I2.0

        =F0.5

L3:     AN F0.7

        JC =L4

        ON F0.6

        JC =L4

        A I2.1

        =F1.0

L4:     AN F0.3

        JC=L5

        R F1.1

        R F0.2

L5:     AN F0.5

        JC=L6

        R F0.1

        R F0.4

L6:     AN F1.0

        JC=L7

        R F0.6

        R Q3.0

        R F0.7

L7:     AN F0.3

        JC=L8

        R F0.3

        S F0.1

        S F0.4
L8:     AN F0.5
```

```
        JC=L9

        R F0.5

        S F0.6

        S F0.7
        S Q3.0

L9:     AN F1.0

        JC=L10

        R F1.0

        S F1.1

        S F0.2
L10:    A F1.1

        A I2.1

        AN I2.0
```

```
        =Q3.1

<PB1>

        BE

<PB2>

        BE

<OB1>

        JU PB1

        BE

        JU FB1

        JU PB2

        BE
```

9.2.6. grafcet com acções condicionais temporizadas




```
FICHEIRO DE ENTRADA:

:I2.0=E3
:I2.1=E4
```

```
:I2.2=E5
```

```
:Q3.0=A1
:Q3.1=A2


GTeste

#10
E 12
S 10
I

#11
E 10
S 11
A A1

#12
E 11
S 12
A? A2=/(t/12/10s)*(t/12/5s)

T10 E3
T11 E4
T12 E5
```

FICHEIRO DE SAÍDA:

```
<OB21>

        S F0.1

        S F0.4
        R F0.6

        R F0.7
        R F1.1

        R F0.2
        BE

<FB1>

L1:     AN F0.2

        JC =L2

        ON F1.1

        JC =L2

        A I2.2

        =F0.3

L2:     AN F0.4

        JC =L3

        ON F0.1

        JC =L3

        A I2.0

        =F0.5

L3:     AN F0.7

        JC =L4

        ON F0.6

        JC =L4

        A I2.1

        =F1.0

L4:     AN F0.3

        JC=L5

        R F1.1

        R F0.2

L5:     AN F0.5

        JC=L6

        R F0.1
```

```
      R F0.4
L6:   AN F1.0
      JC=L7
      R F0.6
      R Q3.0
      R F0.7
L7:   AN F0.3
      JC=L8
      R F0.3
      S F0.1
      S F0.4
L8:   AN F0.5
      JC=L9
      R F0.5
      S F0.6
      S F0.7
      S Q3.0
L9:   AN F1.0
      JC=L10
      R F1.0
      S F1.1
      S F0.2
L10:  A F1.1
      L KT 10.2
      SR T1
      A F1.1
      L KT 5.2
      SR T2
L11:  A F1.1
      A T2
      AN T1
      =Q3.1
<PB1>
      BE
<PB2>
      BE
<OB1>
      JU PB1
      BE
      JU FB1
      JU PB2
      BE
```

10. Listagem dos Erros

10.1. Erros do ficheiro lefich.c

•"Grafcet "..." desconhecido"

O grafcet "..." foi procurado para uma forçagem e depois não foi encontrado.

•"Erro: Grafcet não encontrado: O nome do grafcet não pode coincidir com o nome de variáveis"

O nome do grafcet ou não foi encontrado ou está a ser confundido com uma varável.

•"Forçar etapa ... não declarada é contra as regras de hierarquia"

A etapa número ... não foi ainda definida; Só se podem forçar etapas anteriormente definidas, visto que só estas pertencerem a um grafcet hierarquicamente inferior

•"Erro na leitura de uma temporiz.: '...' "

A temporizção não está completa ou têm algum erro de síntaxe

•"Erro na leitura/procura da etapa referida com 'Xn' em ...\n\r"

As etapa referidas com 'X..." devem já estar definidas. Uma maneira de resover o problema deste erro é utilizar o comando #... imeditamente antes do comando que inicia a definição da etapa que originou o erro

•"Variavel "..." desconhecida"

Não existe um comando :I nem :Q com esta variável.

•"Erro de síntaxe: Falta '=' em accao especial não impulsional"

Numa accção temporizada, condicional ou de set/reset não foi encontrado o separador '=', pelo que a síntaxe está errada.

•"Tipo de acção desconhecida:"..." "

Os tipos de acção permitidos são: A_= Acção normal, AP=Impulsional, AL=duração temporizada, AD=temporização do inicio da acção.

•"Comando 'M' não implementado. Por favor reformule o grafcet"

Por favor utilize os recursos do seu processador de texto para repetir sempre o texto que seria a Macro. Não esqueça que as transições e etapas devem ter números sempre diferentes entre si.

10.2. Erros do ficheiro comptran.c

•"Etapa nº... não encontrada"

Este é um erro interno

•"Erro de lógica na escrita da transic nº..."

Erro na receptividade desta transição

• "Erro interno:Falta parâmetro para uma operação"

Erro na receptividade desta transição, pode ser causado externamente.

10.3. Erros do ficheiro pol_inv.c

•"Erro interno:Faltou ')' "

Erro na receptividade, causa externa

•" '^(' e '/^(' não implementado"

Por favor reformule esta receptividade

•"Erro na leitura de um número:Síntaxe errada (rotina NPI)"

Recetividade errada.

•"Falta de espaço na pilha"

Falta de memória para formar o heap, uma tentativa de alocação dinâmica de memória falhou.

•"Erro na leitura de uma estrutura 'Xm.n', rot NPI,pushnum Erro em ..."

Recetividade errada

•"Erro avalição de uma operação desconhecida:'%c'\n\r"

Recetividade errada

Erros do ficheiro pilha.c

•"Falta de espaço para formar a pilha"

Falta de memória para formar o heap, uma tentativa de alocação dinâmica de memória falhou.

```
extern t_transic * ll_transic;
extern t_etapa * ll_etapas;

extern t_variavel * ll_variaveis;
extern t_accao * ll_accoes;

extern U ultima_flag;
extern U ultimo_temp;

extern FILE *out;
extern FILE *relat;
extern U ultimo_fb;
extern U n_linhas;
extern U prox_label;
```

11. Bibliografia

[1] Tese de mestrado do Eng. Raúl Oliveira, presente no DEEC

[2] Le GRAFCET: de Nouveaux Concept, GREPA - Éditions Cepadues

ref A.C 1.1:11 da bib. do DEEC

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA

0000101616