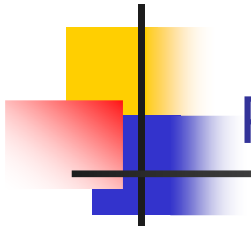
UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

AMBIENTE PARA AUXILIAR O DESENVOLVIMENTO DE PROGRAMAS MONOLÍTICOS

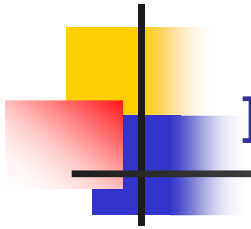
Orientando: Oliver Mário da Silva
Orientador: José R. V. da Silva

2004/2



Roteiro

- Introdução
- Fundamentação teórica
- Contexto básico
- Desenvolvimento do trabalho
- Resultados e discussões
- Conclusão
- Extensões

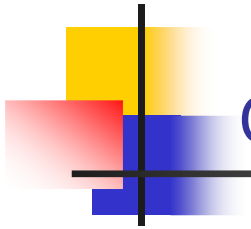


Introdução

- Este trabalho consiste no desenvolvimento de um ambiente para auxiliar o desenvolvimento de programas monolíticos, utilizando somente registradores que comportem números naturais (mesma estrutura de dados da máquina NORMA);

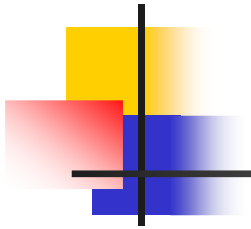
- Aplicação de um novo algoritmo para transformação de um programa monolítico na forma de instruções rotuladas para instruções rotuladas compostas proposto em Silva, 2004;

- Verificação de propriedades em cima da estrutura estática dos programas monolíticos.



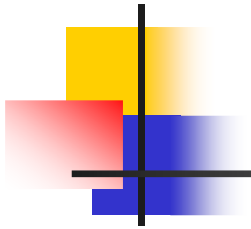
Objetivos do trabalho

- O objetivo deste trabalho é desenvolver um ambiente para auxiliar o desenvolvimento de programas monolíticos.
  - Os objetivos específicos são:
    - disponibilizar um analisador léxico e sintático para programas monolíticos;
    - disponibilizar um módulo para conversão de programas monolíticos na forma de instruções rotuladas para instruções rotuladas compostas;
    - disponibilizar um módulo para verificar as propriedades:
      - existência de ciclos infinitos,
      - identificação de instruções mortas;
    - disponibilizar uma opção para verificar a equivalência entre dois programas monolíticos;
    - disponibilizar um módulo para interpretar programas monolíticos.



1.1 – Máquina

- O objetivo de uma máquina é suprir todas as informações necessárias para que a computação de um programa possa ser descrita (DIVERIO; MENEZES, 2003, p. 20-21);
- Cabe a máquina suprir o significado aos identificadores das operações e testes;
- Os identificadores de operação e teste interpretados pela máquina devem ser associados a uma transformação na estrutura de memória e a uma função verdade, respectivamente;
- A máquina deve descrever o armazenamento ou recuperação de informações na estrutura de memória.



1.2 – Programa

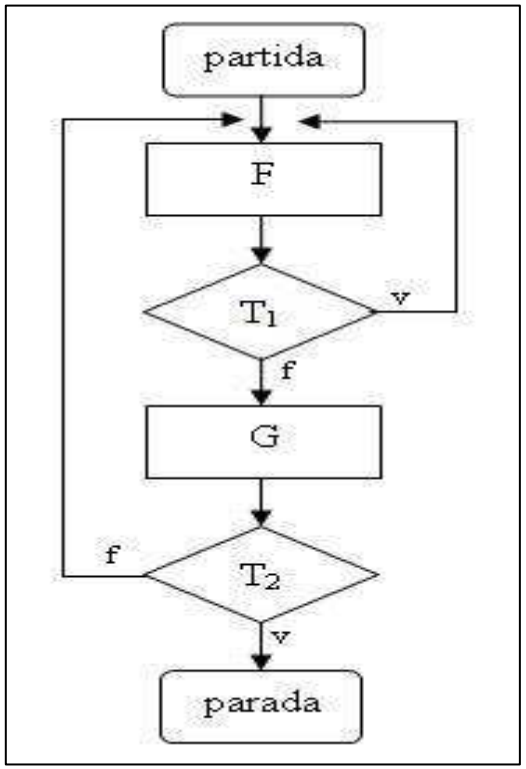
- Um programa pode ser descrito como um conjunto estruturado de instruções que capacitam uma máquina a aplicar sucessivamente certas operações básicas e testes em uma parte determinada dos dados iniciais fornecidos, até que esses dados tenham se transformado numa forma desejável.

- Um programa deve explicitar como as operações ou testes devem ser compostos, ou seja, deve possuir uma estrutura de controle para operações e testes.

- Existem três formas de estruturação do controle, as quais serão expostas como tipos de programas.

1.2.1 – Tipos de programa

- programa monolítico: baseado em desvios condicionais e incondicionais, não possuindo mecanismos explícitos de programação, sub-divisão ou recursão;



Fluxograma

```
1: faça F vá_para 2
2: se T1 então vá_para 1 senão vá_para 3
3: faça G vá_para 4
4: se T2 então vá_para 5 senão vá_para 1
```

Instruções rotuladas

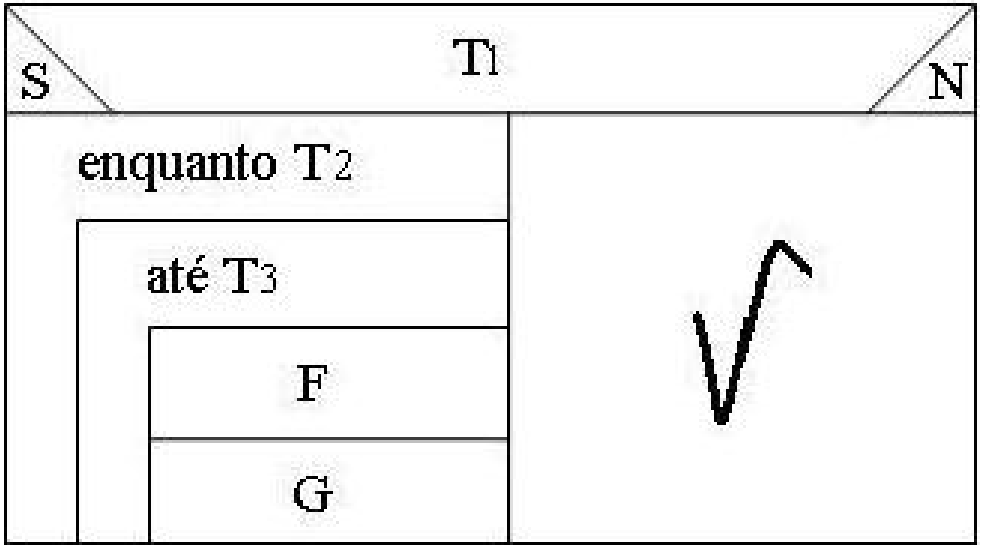
Na instrução rotulada 4, existe um desvio para o rótulo 5. Porem o rótulo 5 não existe no conjunto de instruções rotuladas. Sempre que houver um desvio para um rótulo que não exista no conjunto de instruções rotuladas, subentende-se que é um rótulo final.

1.2.1 – Tipos de programa (Cont.)

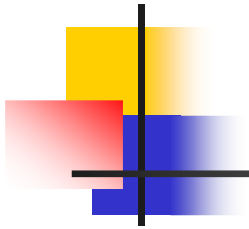
- Programa iterativo: possui mecanismos de controle de iterações de trechos de programas, não permitindo o uso de desvios incondicionais

```
(se T1
então enquanto T2
      faça  (até T3
            faça (F;G))
senão √)
```

Programa iterativo



Programa iterativo representado através do diagrama de Nassi-Schneidermann

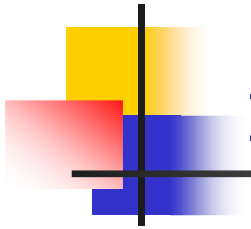


1.2.1 – Tipos de programa (Cont.)

- Programa recursivo: é uma forma indutiva de definir programas.
  - Possui mecanismos de estruturação em sub-rotinas recursivas.

```
P é R;S onde
      R def F;(se T então G;S),
      S def (se T então √ senão F;R)
```

Programa recursivo



1.3 – Computação / Função computada

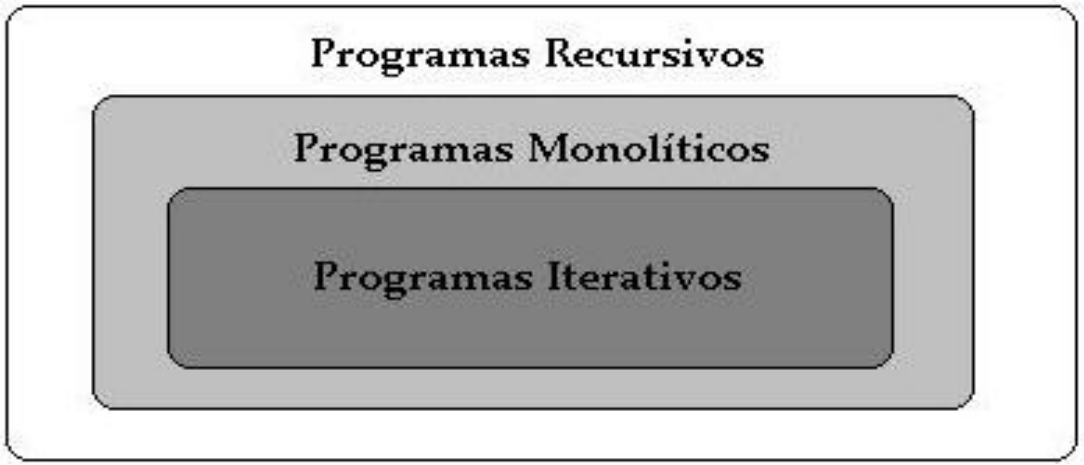
- A computação de um programa monolítico é um histórico das instruções executadas e o correspondente valor de memória ao final da execução do programa

- A função computada é o valor resultante da aplicação da função de saída ao valor da memória quando o programa atinge o rótulo final.
  - Se um programa não atingir um rótulo final, ele entrará num estado de ciclo infinito ou seja, ele jamais irá terminar e a função computada é indefinida.

1.4 – Equivalência forte de programas

- Um par de programas pertence à relação de equivalência se as correspondentes funções computadas coincidem para qualquer máquina.

- Todo programa iterativo pode ser transformado em monolítico, assim como todo monolítico pode ser transformado em recursivo, para qualquer máquina. A inversa não necessariamente é verdadeira, pois existe uma hierarquia de classes de programas



Hierarquia induzida pela relação equivalência forte de programas

1.5 – Instruções rotuladas compostas

- Uma instrução rotulada composta é uma seqüência de símbolos
- Instruções rotuladas compostas possuem uma única forma, diferentemente das instruções rotuladas que podem ser de duas formas: operação ou teste;
  - Uma instrução rotulada composta combina ambas em uma única forma.
- Não existem duas instruções diferentes com um mesmo rótulo (determinismo) e, um rótulo referenciado por alguma instrução o qual não é associado a qualquer instrução rotulada é dito um rótulo final.

r1: se T então faça F vá_para r2 senão faça G vá_para r3

Instruções rotuladas compostas

r1: (F, r2), (G, r3)

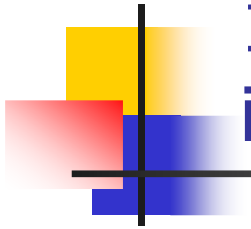
Instruções rotuladas compostas abreviada

1.5 – Instruções rotuladas compostas (Cont.)

- Todo programa monolítico representado através de instruções rotuladas pode ser convertido para instruções rotuladas compostas;

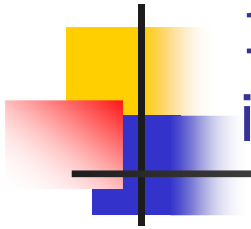
```
1: faca G vá_para 2
2: se T então vá_para 3 senão vá_para 5
3: faca F vá_para 4
4: se T então vá_para 10 senão vá_para 7
5: faca G vá_para 6
6: se T então vá_para 7 senão vá_para 8
7: faca H vá_para 10
8: faca F vá_para 9
9: faca G vá_para 1
```

1:	(G, 2),	(F, 3)
2:	(G, 2),	(F, 3)
3:	(F, 4),	(G, 5)
4:	(F, 4),	(G, 5)
5:	(F, 6),	(ciclo, ω)
6:	(parada, ϵ),	(G, 7)
7:	(G, 7),	(G, 7)
ω:	(ciclo,	ω), (ciclo, ω)



1.6 – Conversão de programas monolíticos para instruções rotuladas compostas

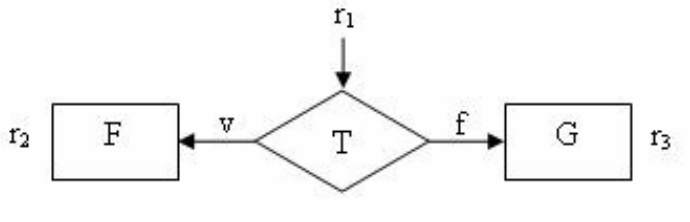
- Algoritmos para conversão de instruções rotuladas em instruções rotuladas compostas:
  - Algoritmo descrito em Diverio e Menezes (2003, p. 42):
    - Instruções rotuladas -> Fluxograma -> Instruções Rotuladas Compostas
  - Algoritmo proposto em Silva (2004):
    - Instruções rotuladas -> Instruções Rotuladas Compostas.



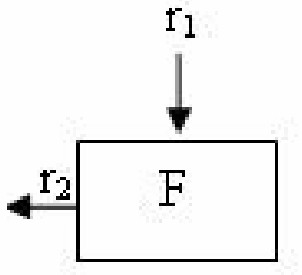
1.6.1 – Conversão de programas monolíticos para instruções rotuladas compostas (DIVÉRIO; MENEZES, 2003, P. 42)

- Os componentes elementares de partida, parada e operação são chamados de nós.
  - rotulação de nós:
    - rotula-se cada nó do fluxograma, supondo que só exista um único nó de parada,
    - o rótulo correspondente ao nó de partida é o rótulo inicial do programa na forma de instruções rotuladas,
    - a rotulação dos nós de um fluxograma usa números naturais, fixando o número 1 para o nó de partida;
  - instruções rotuladas compostas:
    - a construção de uma instrução rotulada composta parte do nó de partida e segue o caminho do fluxograma,
    - Dependendo do próximo componente tem-se que:

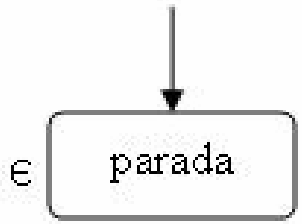
1.6.1 – Conversão de programas monolíticos para instruções rotuladas compostas (Cont.)



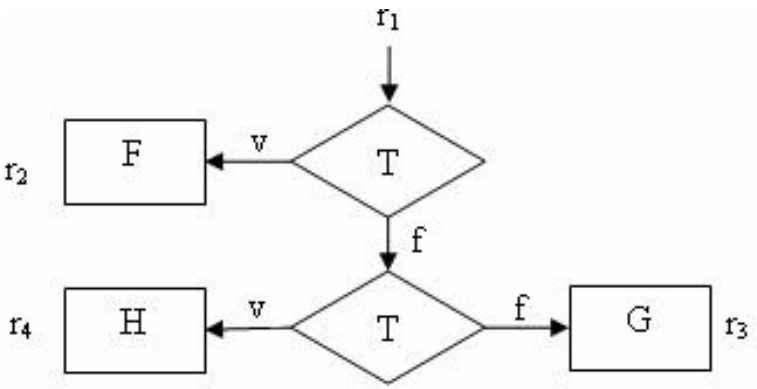
r1: (F, r2), (G, r3)



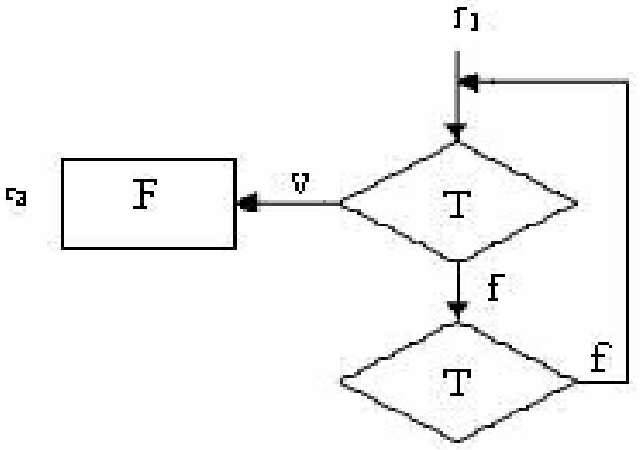
r1: (F, r2), (F, r2)



r: (parada, ε), (parada, ε)

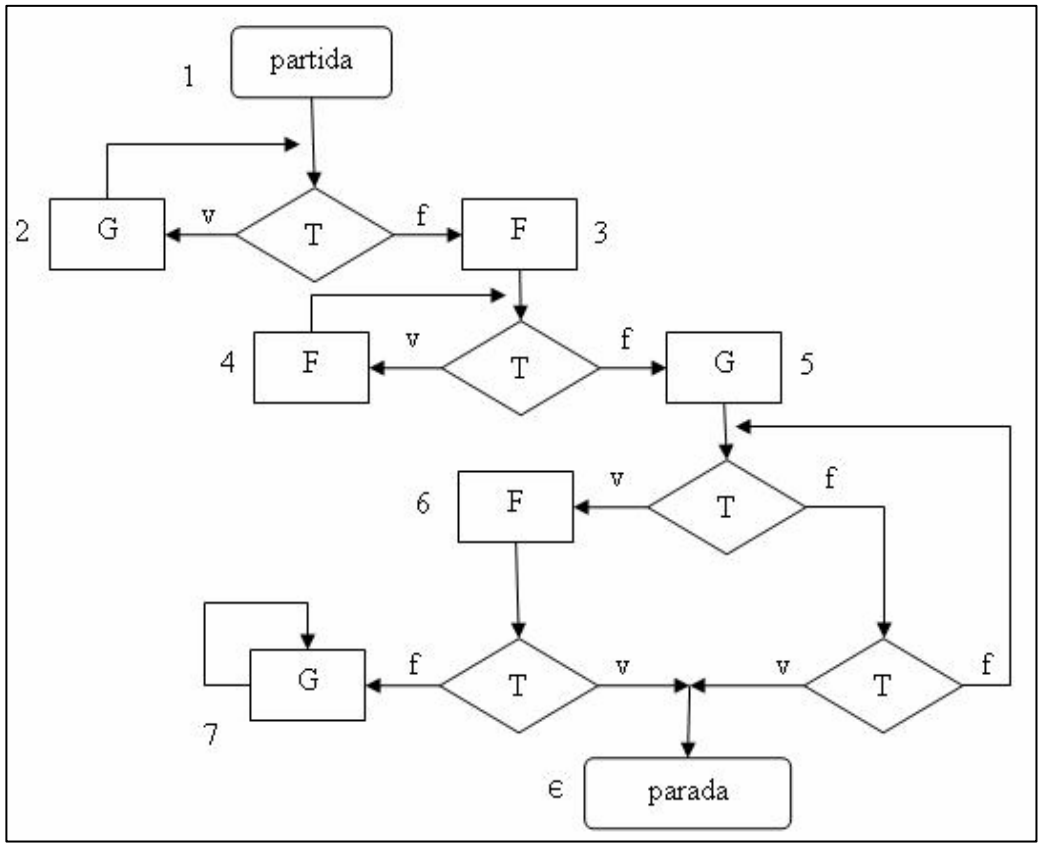


r1: (F, r2), (G, r3)

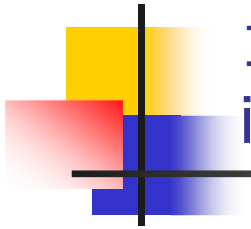


r1: (F, r2), (ciclo, ω)

1.6.1 – Conversão de programas monolíticos para instruções rotuladas compostas (Exemplo)



1:	(G, 2),	(F, 3)
2:	(G, 2),	(F, 3)
3:	(F, 4),	(G, 5)
4:	(F, 4),	(G, 5)
5:	(F, 6),	(ciclo,ω)
6:	(parada, ε),	(G, 7)
7:	(G, 7),	(G, 7)
ω:	(ciclo,ω),	(ciclo,ω)



1.6.2 – Conversão de programas monolíticos para instruções rotuladas compostas (SILVA, 2004)

- O algoritmo proposto em Silva (2004):
  - Instruções Rotuladas (PMIR) -> Instruções Rotuladas Compostas (PMIRC) obedece dois passos:
    - No primeiro passo cria-se uma "tabela de transformação";
    - No segundo passo cria-se o programa monolítico na forma de instruções rotuladas compostas a partir da "tabela de transformação".

1.6.2 – Conversão de programas monolíticos para instruções rotuladas compostas (Cont.)

- Primeiro passo:
  - criar uma tabela com 4 colunas por *x* linhas (*x* é o número de linhas do PMIR mais uma);
  - inicializar a tabela.

Tabela 1 – Tabela de transformação

Linhas C11	rótulos do PMIRC C12	Rótulos do PMIR C13	Instruções do PMIR C14
1	1		
2		1	`Se T então vá_para 2 senão vá_para 3`
3	2	2	`Faça G vá_para 1`
4	3	3	`Faça F vá_para 4`
5		4	`Se T então vá_para 5 senão vá_para 4`
6	4	5	`Faça G vá_para 6`
7	5	6	`Faça F vá_para 6`

1.6.2 – Conversão de programas monolíticos para instruções rotuladas compostas (Cont.)

- Segundo passo:
  - Cria-se uma nova tabela com 5 colunas por 1 linha;

Tabela 2 – Instruções Rotuladas Compostas

Rótulos do prog. (rc)	Separador Literal ":"	instrução PMIRC opção 'verdadeira' (então) (Oper, rc')	Separador Literal ","	instrução PMIRC opção 'falsa' (senão) (Oper,rc'')
C21	**C22**	**C23**	**C24**	**C25**

- Durante o processo de transformação, serão inseridas novas linhas na tabela, tantas quantas necessárias;

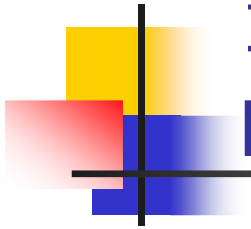
- As colunas C22 e C24 a cada linha inserida na tabela deverão ser preenchidas com ":" e "," respectivamente.

Tabela 1 – Tabela de transformação

C11	C12	C13	C14
1	1		
2		1	Se T então vá_para 2 senão vá_para 3
3	2	2	Faça G vá_para 1
4	3	3	Faça F vá_para 4
5		4	Se T então vá_para 5 senão vá_para 4
6	4	5	Faça G vá_para 6
7	5	6	Faça F vá_para 6

Tabela 2 – Instruções rotuladas compostas

C21	C22	C23	C24	C25
1	:	(G, 2)	,	(F, 3)
2	:	(G, 2)	,	(F, 3)
3	:	(G, 4)	,	(ciclo, ω)
4	:	(F, 5)	,	(F, 5)
5	:	(F, 5)	,	(F, 5)
ω	:	(ciclo, ω)	,	(ciclo, ω)



1.7 – Propriedades identificáveis em um programa monolítico

- Em cima da estrutura estática de programas monolíticos, pode-se verificar propriedades importantes, as quais são:
  - estados mortos ou inatingíveis;
    - Esta propriedade é verificada no programa monolítico na forma de instruções rotuladas;
  - ciclos infinitos;
    - Esta propriedade é verificada no programa monolítico na forma de instruções rotuladas compostas.

1.7.1 – Instruções mortas em um programa monolítico (instruções rotuladas)

- Instruções mortas são aquelas que jamais serão atingidas a partir do estado inicial;
- A identificação de uma instrução morta inicia-se a partir da instrução de "inicio", determinando os seus sucessores. Por exclusão, uma instrução que não possui antecessor é considerada uma instrução morta.

```
1: faça F vá_para 2
2: se T1 então vá_para 1 senão vá_para 3
3: faça G vá_para 6
4: faça F vá_para 5
5: faça G vá_para 6
6: se T2 então vá_para 7 senão vá_para 1
```

Programa monolítico com instruções mortas

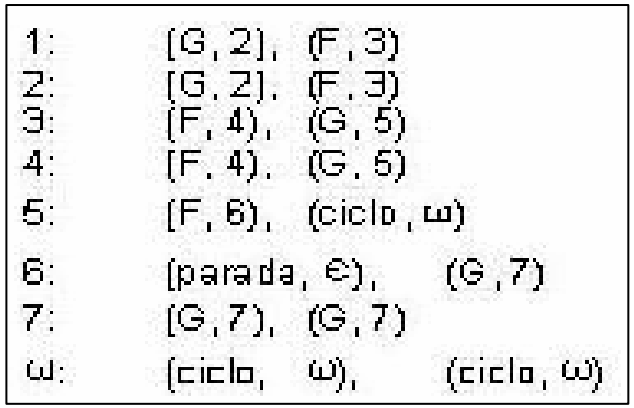
A0 = { 1 }
A1 = { 1, 2 }
A2 = { 1, 2, 3 }
A3 = { 1, 2, 3, 6 }
A4 = { 1, 2, 3, 6 }

```
1: faça F vá_para 2
2: se T1 então vá_para 1 senão vá_para 3
3: faça G vá_para 6
6: se T2 então vá_para 7 senão vá_para 1
```

Programa monolítico sem instruções mortas

1.7.2 – Ciclos infinitos em um programa monolítico (instruções rotuladas compostas simplificadas)

- Ciclos infinitos são instruções que jamais alcançarão o estado final.
- A identificação dos ciclos infinitos inicia-se a partir da instrução de "parada", determinando os seus antecessores. Por exclusão, uma instrução que não é antecessora de "parada" determina um ciclo infinito.



Instruções rotuladas compostas

A0 = {ε}

A1 = {6, ε}

A2 = {5, 6, ε}

A3 = {3, 4, 5, 6, ε}

A4 = {1, 2, 3, 4, 5, 6, ε}

A5 = {1, 2, 3, 4, 5, 6, ε}

Instruções rotuladas compostas simplificadas

1.8 – Equivalência forte de programas monolíticos (instruções rotuladas compostas simplificadas)

- Pode-se verificar sua equivalência entre dois programas monolíticos.
  - Ambos os programas primeiramente devem ser convertidos para instruções rotuladas compostas
  - O segundo programa deve ser re-rotulado;
    - O primeiro rótulo do segundo programa deve ser o último rótulo do primeiro programa mais um

```
1:  (G, 2), (F, 3)
2:  (G, 2), (F, 3)
3:  (F, 4), (G, 5)
4:  (F, 4), (G, 5)
5:  (F, 6), (ciclo, w)
6:  (parada, ε), (ciclo, w)
w:  (ciclo, w), (ciclo, w)
```

Programa monolítico na forma de instruções rotuladas compostas simplificadas (P1)

```
8:   (G, 9), (F, 10)
9:   (G, 9), (F, 10)
10:  (F, 10), (G, 11)
11:  (F, 12), (ciclo, w)
12:  (parada, ε), (ciclo, w)
w:   (ciclo, w), (ciclo, w)
```

Programa monolítico na forma de instruções rotuladas compostas simplificadas (P2)

1.8 – Equivalência forte de programas monolíticos (Cont.)

```
1:   (G, 2), (F, 3)
2:   (G, 2), (F, 3)
3:   (F, 4), (G, 5)
4:   (F, 4), (G, 5)
5:   (F, 6), (ciclo, w)
6:   (parada, ε), (ciclo, w)
8:   (G, 9), (F, 10)
9:   (G, 9), (F, 10)
10:  (F, 10), (G, 11)
11:  (F, 12), (ciclo, w)
12:  (parada, ε), (ciclo, w)
w:   (ciclo, w), (ciclo, w)
```

União disjunta de P1 e P2
(instruções rotuladas compostas simplificadas)

- Para verificar se P1 ≡ P2, é necessário verificar se (P1, 1) ≡ (P2, 8).
- Como as instruções dos rótulos 1 e 8 são equivalentes fortemente então $B_0 = \{(1, 8)\}$.
- Verificando-se a equivalência entre os sucessores dos rótulos 1 e 8, têm-se a cadeia de conjuntos apresentado no quadro abaixo, onde o conjunto B5 = ∅.
  - Logo, (P1, 1) ≡ (P2, 8) e portanto, P1 ≡ P2.

$B_0 = \{(1, 8)\}$	pares de rótulos equivalentes fortemente
$B_1 = \{(2, 9), (3, 10)\}$	pares de rótulos equivalentes fortemente
$B_2 = \{(4, 10), (5, 11)\}$	pares de rótulos equivalentes fortemente
$B_3 = \{(6, 12), (w, w)\}$	pares de rótulos equivalentes fortemente
$B_4 = \{(\varepsilon, \varepsilon)\}$	pares de rótulos equivalentes fortemente
$B_5 = \varnothing$	

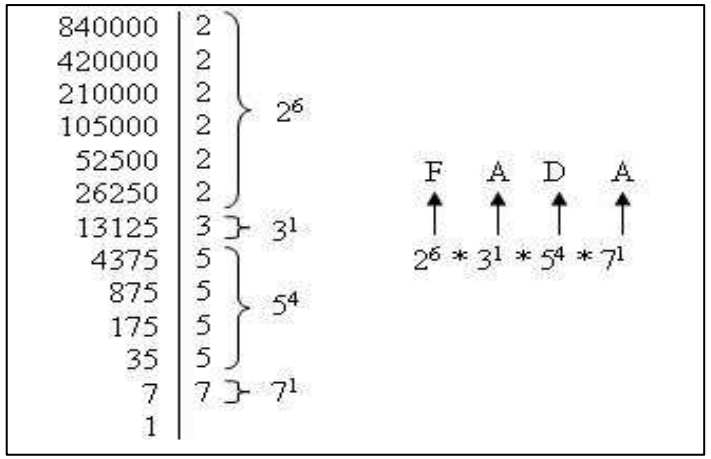
1.9 – Codificação de conjuntos estruturados

- Elementos de tipos de dados estruturados podem ser representados como números naturais, através do teorema fundamental da aritmética;
- Programas monolíticos na forma de instruções rotuladas também podem ser representados através da codificação de conjuntos estruturados;
- Cadeias de caracteres podem ser representados através da codificação de n-uplas naturais.

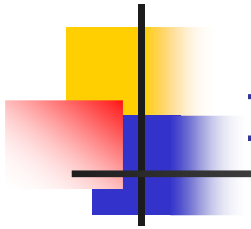
$F = 6$
$A = 1$
$D = 4$
$A = 1$

$$\text{“FADA”} = 2^6 * 3^1 * 5^4 * 7^1$$
$$\text{“FADA”} = 64 * 3 * 625 * 7$$
$$\text{“FADA”} = 840000$$

Codificação da palavra "FADA"



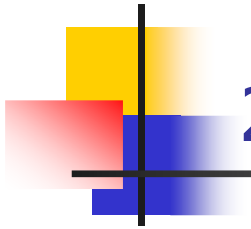
Decodificação da palavra "FADA"



1.10 – Máquina NORMA

- A máquina NORMA, proposta em Bird (1976, p. 50) é uma máquina de registradores. Possui um conjunto infinito de registradores naturais como memória e somente três operações sobre cada registrador:
  - adição do valor um;
  - subtração do valor um;
  - teste se o valor armazenado é zero.

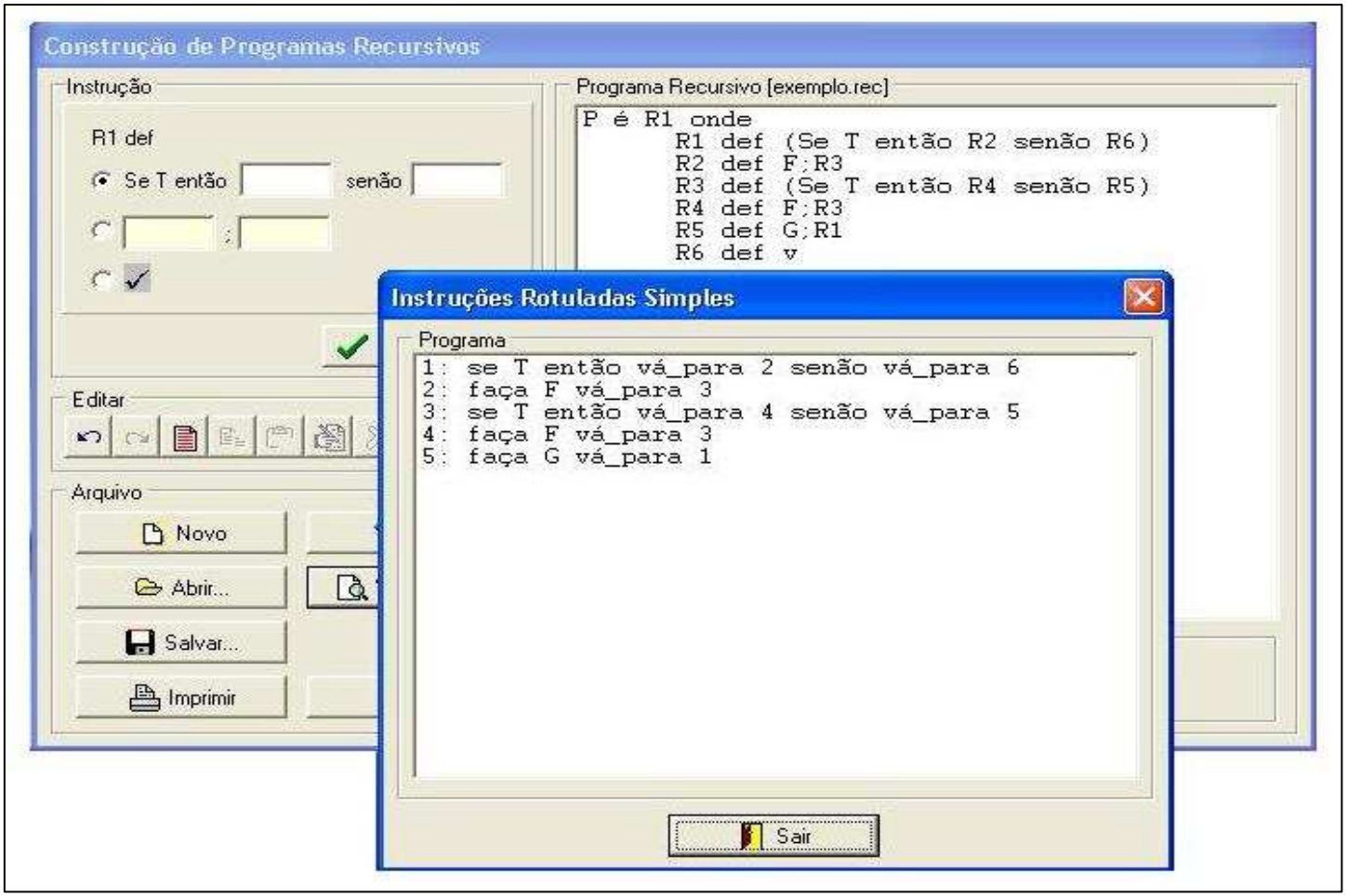
- A máquina NORMA é universal e extremamente simples, com poder computacional de qualquer computador moderno (DIVERIO; MENEZES, 2003, p. 72).



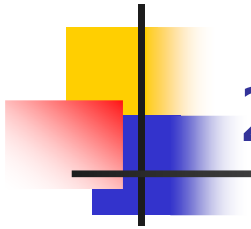
2.1 – Trabalho correlato

- O software *Programas Recursivos* é um protótipo para converter programas monolíticos na forma de instruções rotuladas em programa recursivo; (FERNANDES et al, 2004). O software tem as seguintes opções principais:
  - permite criar ou editar um programa recursivo;
    - simplificação do programa recursivo,
  - Permite verificar a equivalência entre dois programas monolíticos;
- O software tem as seguintes limitações:
  - não permite a interpretação dos programas,
  - não mostra quais instruções não são equivalentes durante a verificação da equivalência.
  - Não verifica os estados mortos no programa monolítico

2.1 – Trabalho correlato



Software Programas Recursivos

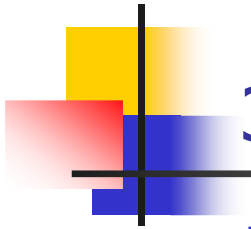


2 – Conceitos básicos

- O ambiente foi especificado utilizando técnicas de orientação a objetos com a UML, através dos diagramas de casos de uso, de classe e de seqüência. A especificação foi feita através da ferramenta Rational Rose;

- A especificação da linguagem monolítica para o ambiente foi feita utilizando-se a notação (BNF);

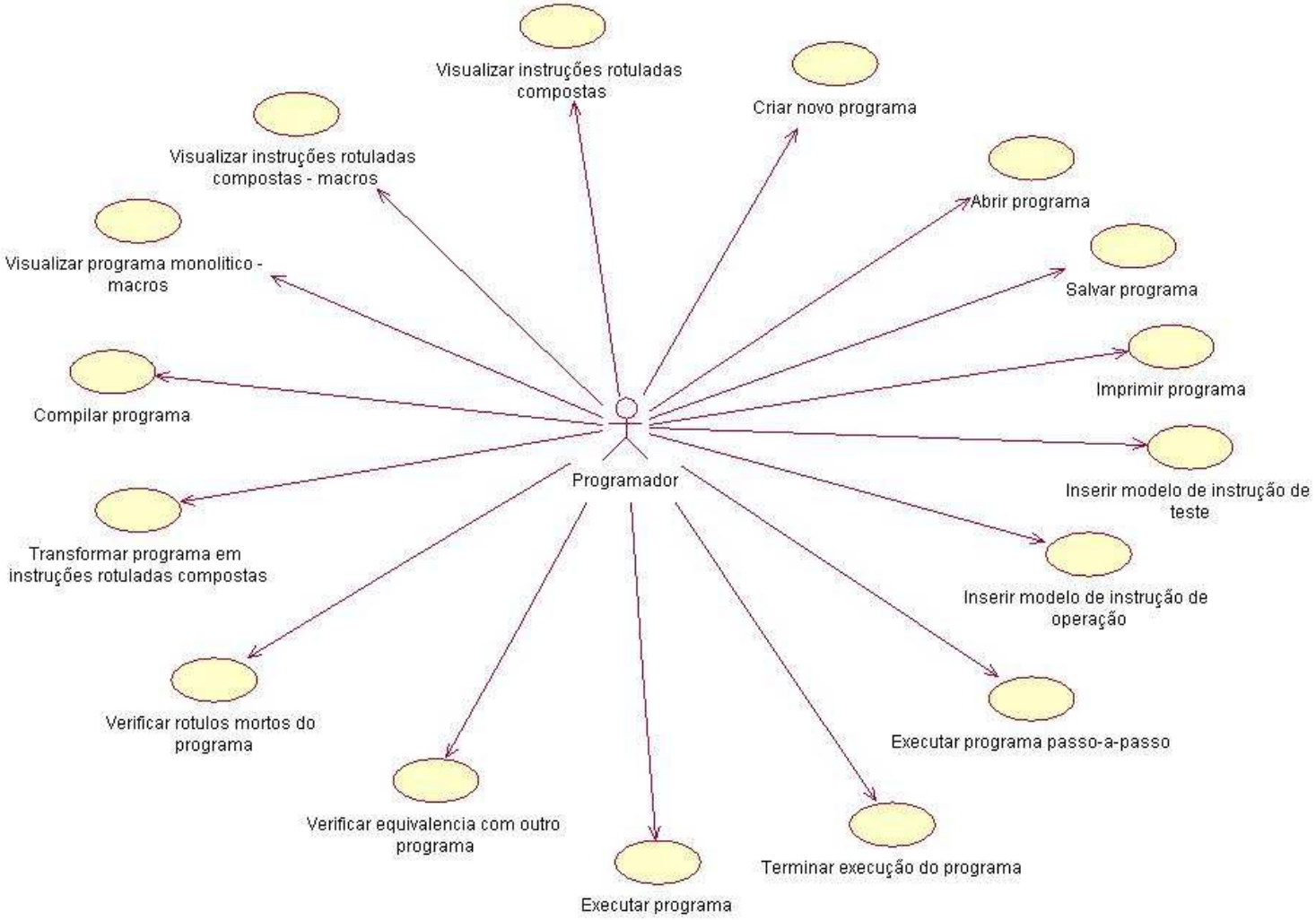
- A implementação do ambiente foi feita no ambiente de programação Delphi 7.0, utilizando classes criadas pelo Gerador de Analisadores Léxicos e Sintáticos (GALS).



3.1 – Requisitos principais

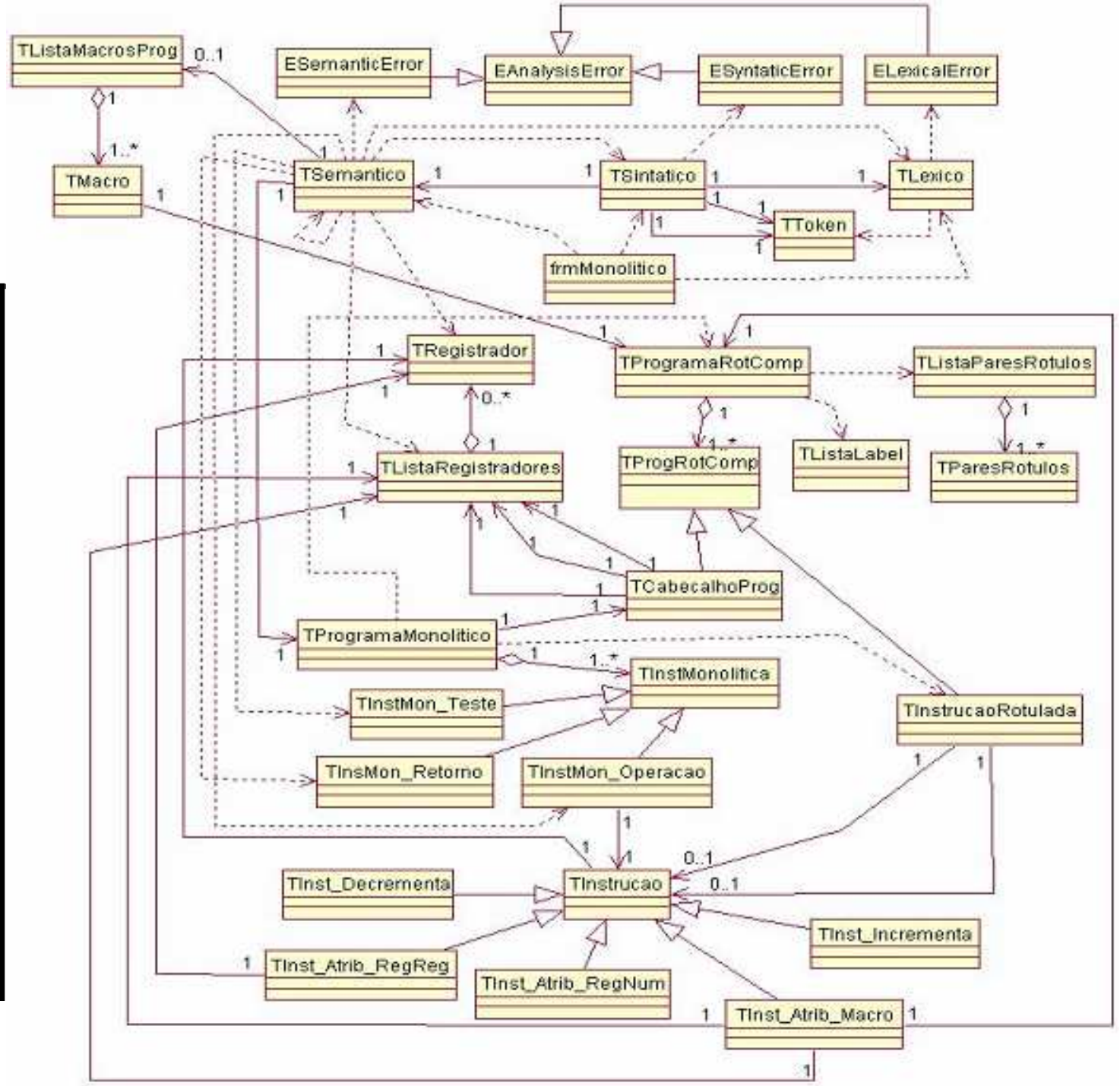
- Editor: para a linguagem monolítica
- Compilador: (léxico, sintático e semântico)
- Conversão:
  - programa monolítico na forma de instruções rotuladas -> instruções rotuladas compostas;
- Simplificação: simplificar as instruções rotuladas compostas (ciclos infinitos);
- instruções mortas: no programa monolítico (instruções rotuladas);
- Equivalência: entre dois programas monolíticos (instruções rotuladas compostas);
- Interpretação: interpretar o programa na forma de instruções rotuladas compostas (não gera código objeto);
- Interpretação passo-a-passo: programa na forma de instruções rotuladas compostas;
- Registradores: visualização dos valores dos registradores durante a execução;
- Terminar execução: possibilitar que a execução de um programa seja interrompida;

3.2 – Especificação do ambiente (Diag. Casos de Uso)



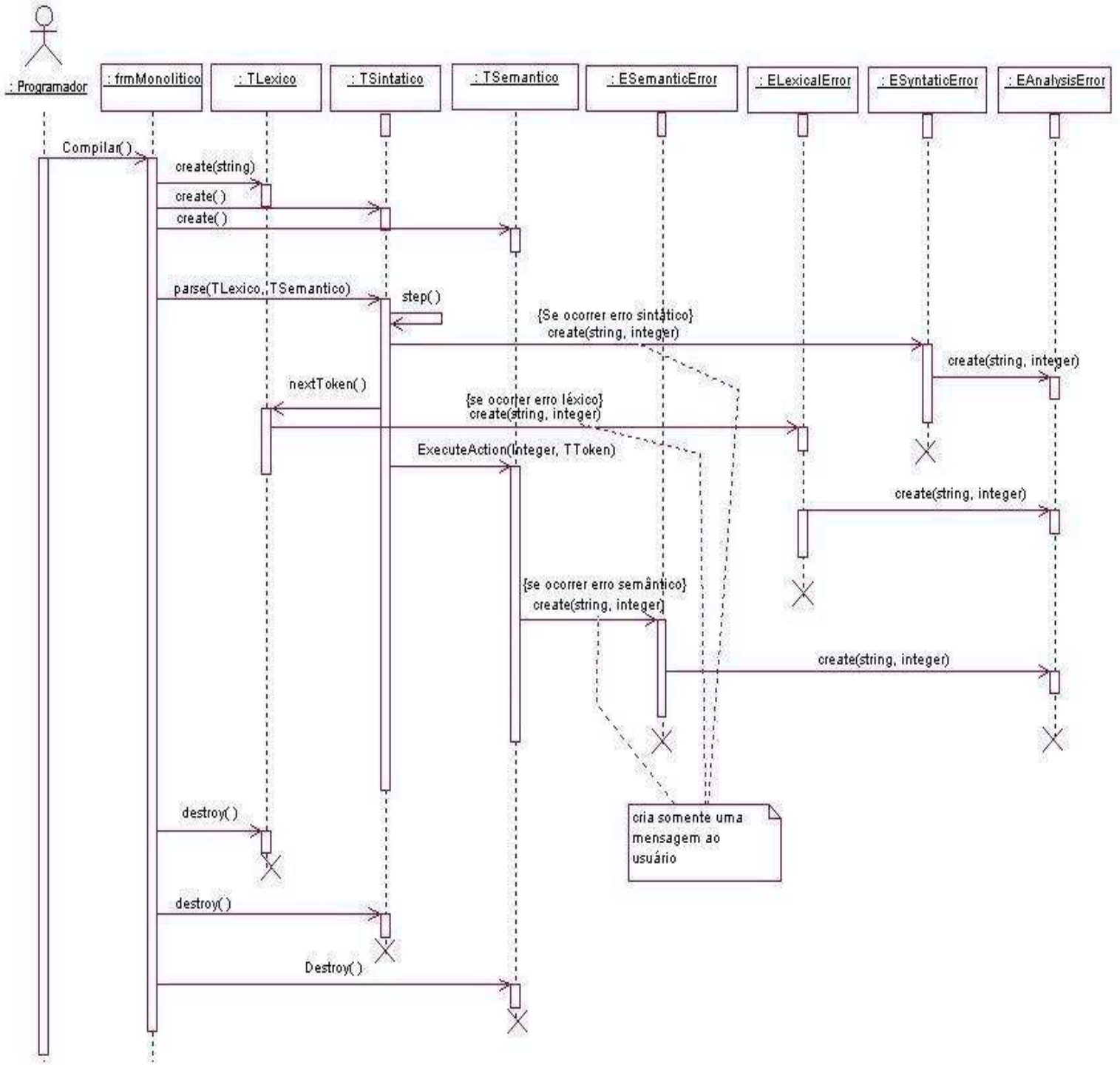
3.3 Especificação do Ambiente

Diagrama de Classes



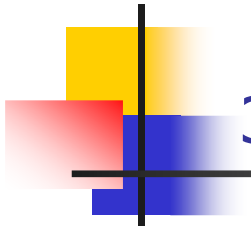
3.4 Especificação do Ambiente

Diagrama de Seqüência



3.5 – Especificação da linguagem (gramática – notação BNF)

```
<Cabecalho>        ::= Programa Pr#1 <Parametros> "->"#2 <Regs_Retorno>#3 <Monolitico> ;

<Parametros>       ::= "(" <Registradores> ")" | î;

<Regs_Retorno>     ::= <Registradores>;

<Registradores>    ::= Registrador#4 <R_Registrador> ;

<R_Registrador>    ::= "," Registrador#4 <R_Registrador> | î ;

<Monolitico>       ::= NumeroNatural#5 ":" <Mon_Ret> ;

<Mon_Ret>          ::= <Monolitico_L> | <Retorno> ;

<Monolitico_L>     ::= <Operacao><Monolitico> | <Teste><Monolitico> ;

<Operacao>         ::= Faca#6 <Operacoes> Va_Para NumeroNatural#16 ;

<Operacoes>        ::= <Inc_Dec>#7 "(" Registrador#8 ")" |
                       Registrador#9  <R_Regist> "=" <Macro_Reg_Num> ;
<R_Regist>         ::= "," Registrador#10 <R_Regist> | î ;

<Inc_Dec>          ::= Inc | Dec;

<Macro_Reg_Num>    ::= #11Registrador#13 |
                       #12NumeroNatural#13 |
                       Pr#14 <Parametros>#15 ;

<Teste>            ::= Se#6 T Entao Va_Para NumeroNatural#17 Senao Va_Para NumeroNatural#18;

<Retorno>          ::= Retorna#6;
```



3.6 – Linguagem

- A linguagem criada foi baseada na estrutura dos programas monolíticos na forma de instruções rotuladas, com a mesma estrutura de dados da máquina NORMA.
- Possui um número infinito de registradores como memória, e as seguintes operações sobre cada registrador:
  - atribuição de um valor;
  - atribuição do valor de um registrador para outro registrador;
  - adição do valor um;
  - subtração do valor um;
  - teste se o valor armazenado é zero.

3.6 – Linguagem (Cont.) – Exemplos

```
programa Exemplo_Cabecalho (r1, r2) -> r3, r4
```

Cabeçalho de um programa

```
1: faca inc(r1) va_para 2
```

Instrução de adição do valor um

```
1: faca dec(r2) va_para 2
```

Instrução de subtração do valor um

```
1: faca r1 = r2 va_para 2
```

Instrução de atribuição do valor de
um registrador para outro registrador

```
1: faca r1 = 0 va_para 2
```

Instrução de atribuição de um valor natural
a um registrador

```
1: faca r1 = igual(r1, r2) va_para 2
```

Atribuição com chamada de uma
macro com parâmetros

```
1: faca r1, r2 = constante_PI va_para 2
```

Atribuição com chamada de uma
macro constante

```
1: se T entao va_para 2 senao va_para 1
```

Instrução de teste

```
1: retorna
```

Instrução de retorno

3.6 – Linguagem (Cont.) - Exemplos

```
programa Comp_Dois_Num_Iguais(R1, R2)-> rT
1: faca rt= r1 va_para 2
2: se T entao va_para 3 senao va_para 5
3: faca rt = r2 va_para 4
4: se T entao va_para 10 senao va_para 7
5: faca rt= r2 va_para 6
6: se T entao va_para 7 senao va_para 8
7: faca rt=1 va_para 10
8: faca dec(r1) va_para 9
9: faca dec(r2) va_para 1
10: retorna
```

Programa que compara se dois números naturais são iguais

```
-- este programa retorna o fatorial de um número natural...
-- r1 = função de entrada
-- r1 = função de saída
programa fatorial(r1) -> r1
1: faca rt = r1 va_para 2                          -- atribui o valor de r1 a rt
2: se T entao va_para 3 senao va_para 4            -- testa se rt = 0
3: faca r1 = 1 va_para 9                           -- atribui o valor 1 a rt
4: faca r2 = r1 va_para 5                          -- atribui o valor de r1 a r2
5: faca dec(r2) va_para 6                          -- decrementa o valor de r2
6: faca rt = r2 va_para 7                          -- atribui o valor de r2 a rt
7: se T entao va_para 9 senao va_para 8            -- testa se rt = 0
8: faca r1 = Mult_Int_SemSinal(r1, r2) va_para 5   -- r1 = r1 x r2
9: retorna                                         -- programa terminou
```

Programa que retorna o fatorial de um número (com comentários de linha)

```
programa Comp_Tres_Num_Iguais(r1, r2, r3)-> rt
1: faca  rt = Comp_Dois_Num_Iguais(r1, r2) va_para 2
2: se T entao va_para 3 senao va_para 4
3: faca  rt = Comp_Dois_Num_Iguais(r1, r3) va_para 4
4: retorna
```

Programa que compara se três números naturais são iguais utilizando uma macro macro

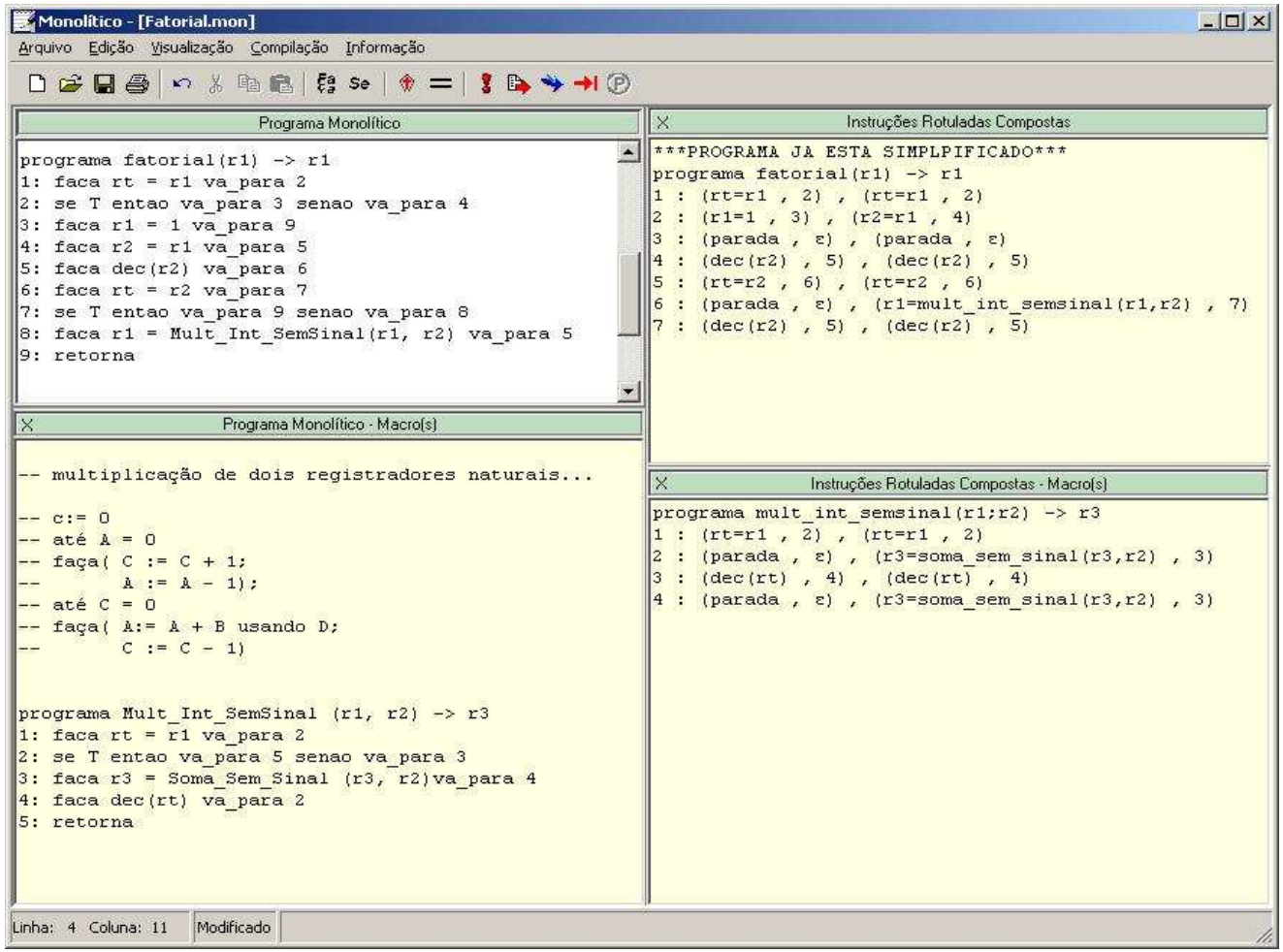
3.7 – Implementação: interação entre código gerado pelo GALS e código do ambiente

```
procedure TfrmMonolitico.acCompilarExecute(Sender: TObject);
var
  vLexico : TLexico;
  vSintatico : TSintatico;
  vSemantico : TSemantico;
begin
  // limpa area de mensagens
  StatusBar.Panels[2].Text:=EmptyStr;
  // cria uma instancia para cada analisador
  vLexico := TLexico.create(reEditor.Text); //o parametro eh o programa mon. fonte
  vSintatico := TSintatico.create;
  vSemantico := TSemantico.Create;
  try
    // executa a analise lexica / sintatica e semantica
    vSintatico.parse(vLexico, vSemantico);
    ShowMessage(sCompiladoSucesso);
  except // tratamento de excecoes
    on e : ELexicalError do // MOSTRA ERRO LEXICO
      StatusBar.Panels[2].Text:=Format(sErroLinha, [getLinha(e.getPosition), e.getMessage]);
    on e : ESyntaticError do // MOSTRA ERRO SINTATICO
      StatusBar.Panels[2].Text:=Format(sErroLinha, [getLinha(e.getPosition), e.getMessage]);
    on e : ESemanticError do // MOSTRA ERRO SEMANTICO
      StatusBar.Panels[2].Text:=Format(sErroLinha, [getLinha(e.getPosition), e.getMessage]);
  end;
  vLexico.Free;
  vSintatico.Free;
  vSemantico.Free;
end;
```

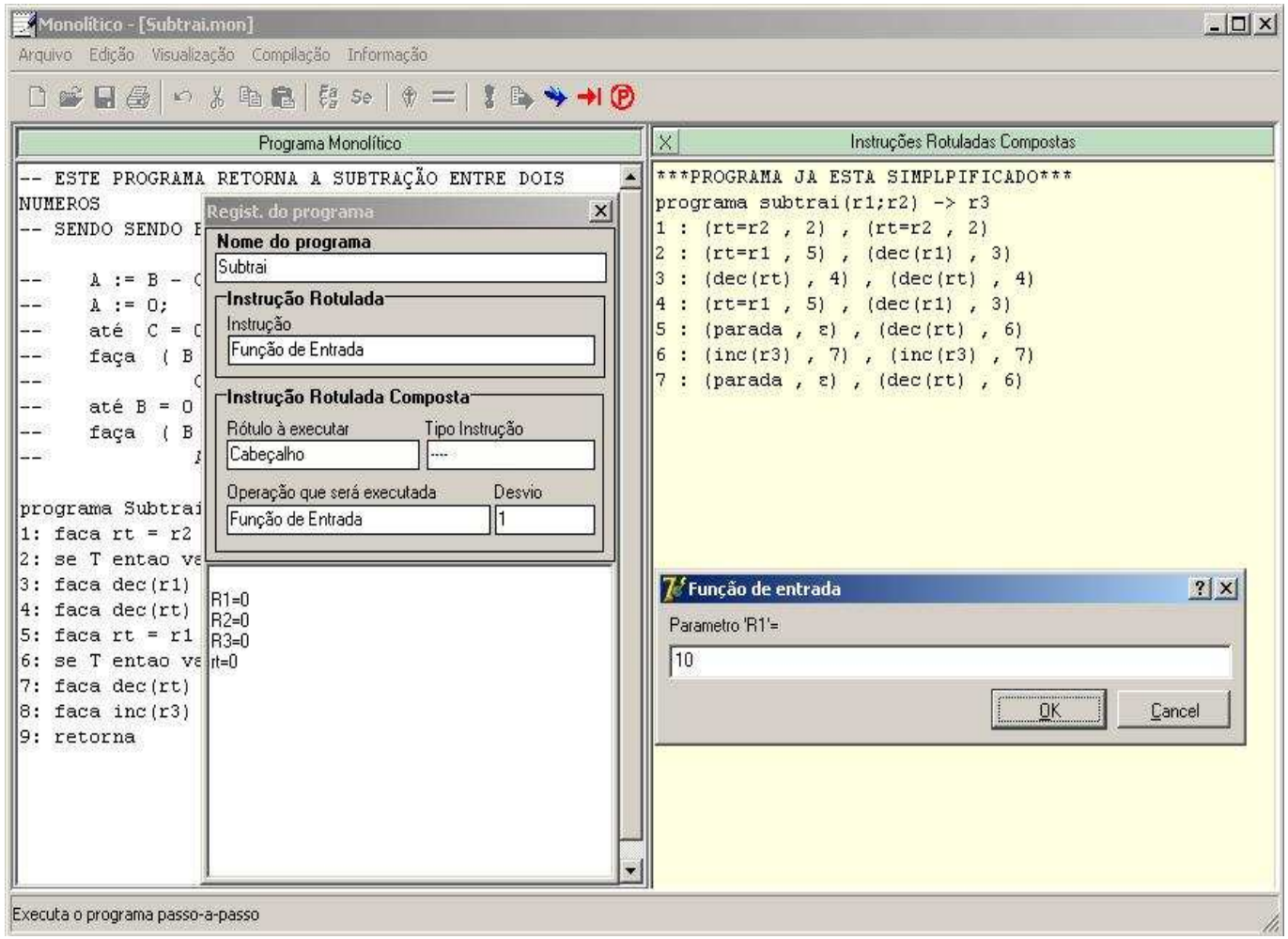
3.7 – Implementação: Método para simplificação das instruções rotuladas compostas

```
function TProgramaRotComp.SimpRotComp : TProgramaRotComp;
var
  vRotulos,
  vRotulosExec : TListaLabel;
  vCabecalho : TCabecalhoProg;
begin
  vRotulos := TListaLabel.Create;
  vRotulosExec := TListaLabel.Create;
  getRotulos(vRotulos, vRotulosExec,0, Parada);
  if (vRotulos.ListaLabels.Count > 0) then
    begin
      if (VerifSimplifRotComp(vRotulos, vRotulosExec)) then
        ExecSimplifRotComp(vRotulos, vRotulosExec)
    end
  else // programa nunca para, fica em loop
    begin
      vCabecalho := fListaInstrucoes[0];
      fListaInstrucoes.Clear;
      fListaInstrucoes.Add(vCabecalho);
      AdicionaInstCicloW;
    end;
  vRotulos.Destroy;
  vRotulosExec.Destroy;
  Result := self;
end;
```

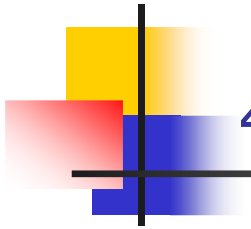
3.8 – Operacionalidade da implementação



3.8 – Operacionalidade da implementação

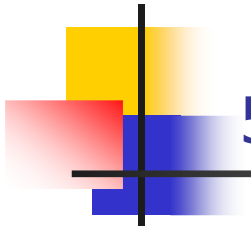


3.8 – Operacionalidade da implementação (arquivo de ajuda)



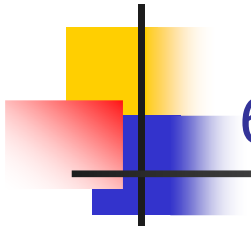
4 – Resultados e Discussão

- Os resultados obtidos após o término do ambiente foram satisfatórios.
- Alguns módulos que não estavam como requisito para o ambiente foram implementados.
  - chamadas de macros no programa,
  - execução passo-a-passo do programa na forma de instruções rotuladas compostas
  - visualização durante a execução de um programa, dos valores dos registradores utilizados pelo mesmo.
- O ambiente possui algumas limitações.
  - não poder ser feito o endereçamento indireto, utilizando registradores.
  - para verificação da equivalência entre dois programas, os programas devem utilizar os mesmos registradores.
  - identificação de ciclos infinitos em tempo de execução:
    - O ambiente detecta os ciclos infinitos na estrutura estática do programa
    - Se o usuário definir uma instrução que, quando executada, entre em um ciclo infinito.
      - o ambiente irá executar infinitamente o programa.
        - Para amenizar este problema, foram implementados dois módulos:
        - execução “passo-a-passo”
        - visualização dos valores dos registradores do programa
        - Interrupção da execução em qualquer instante.



5 – Conclusão

- O objetivo principal deste trabalho, construir um ambiente para auxiliar o desenvolvimento de programas monolíticos foi alcançado.
- A ferramenta GALS utilizada para a construção dos analisadores léxico e sintático foi importante para o desenvolvimento do trabalho, pois ela gerou todas as classes para os analisadores léxico e sintático;
- A principal contribuição deste trabalho é a aplicação e validação do algoritmo proposto em Silva (2004), que possibilita a transformação de programas monolíticos na forma de instruções rotuladas para instruções rotuladas compostas;
- Este trabalho poderá ser utilizado como ferramenta de apoio na disciplina de Teoria da Computação;
- Foram encontradas poucas referências bibliográficas sobre o assunto;
  - O livro de Diverio e Menezes (2003) é fortemente baseado no livro de Bird (1976);



6 – Extensões

- Como possíveis extensões para o trabalho, destacam-se:
  - implementação do endereçamento indireto para programas monolíticos;
  - implementação de um módulo para conversão de programas iterativos para programas monolíticos na forma de instruções rotuladas.