Verificação de Equivalência Forte entre Programas

Teoria da Computação

INF05501

Equivalência Forte entre Programas

- Vimos que é possível estabelecer-se uma Relação de Equivalência Forte entre dois programas quaisquer

- A determinação de que dois programas são fortemente equivalentes nos permite **substituir um pelo outro, em qualquer máquina, e continuarmos a obter a mesma função computada**

- Mas, para isto, precisamos de uma maneira de, **dados dois programas quaisquer, verificarmos se eles são de fato fortemente equivalentes**

Verificação de Equivalência Forte entre Programas Recursivos

- Vimos que **todo programa monolítico ou iterativo possui um programa recursivo fortemente equivalente**
- Com isto, uma solução simples seria termos um **método para determinar se dois programas recursivos são fortemente equivalentes**
- Tal método deveria ser **genérico**, a fim de ser aplicado a quaisquer programas recursivos, e **decidível**, de forma que obtivéssemos uma resposta em um número finito de passos
- Entretanto, até o momento, não se sabe se o problema de decisão sobre a equivalência forte entre dois programas recursivos é **solucionável**

Verificação de Equivalência Forte entre Programas Monolíticos

- Na impossibilidade de termos um método genérico para programas recursivos, passamos o nosso foco para a **classe dos programas monolíticos**

- Neste caso, **se tivermos uma solução para programas monolíticos, também a teremos para programas iterativos**

- A verificação de que dois programas monolíticos são fortemente equivalentes pode seguir **duas abordagens**:

  - Máquina de Traços
  - Instruções Rotuladas Compostas

Máquina de Traços

- Uma máquina de traços é uma máquina que **não executa efetivamente as operações**, mas apenas produz um **histórico (ou rastro) da ocorrência destas**

- Este histórico é chamado de traço (de execução)

- Desta forma, para computações finitas, um traço nada mais é do que **uma palavra sobre o alfabeto determinado pelo conjunto de identificadores de operações**

- Veremos que a determinação de que **dois programas são equivalentes em qualquer máquina de traços** nos permite concluir que eles são **fortemente equivalentes**

Definição Formal de Máquina de Traços

*Uma **máquina de traços** é uma máquina*

$$M = (Op^*, Op^*, Op^*, id_{Op^*}, id_{Op^*}, \Pi_F, \Pi_T)$$

onde:

- Op^* *é o conjunto de palavras de operações, tal que* $Op = \{F, G, ...\}$
- id_{Op^*} *é a função identidade em* Op^*
- Π_F *é o conjunto de interpretações de operações, onde, para cada identificador de operação* $F \in Op$, *a interpretação* $\pi_F : Op^* \to Op^*$ *é tal que, para qualquer* $w \in Op^*$, $\pi_F(w) = wF$
- Π_T *é o conjunto de interpretações de testes, tal que, para cada identificador de teste* T, $\pi_T : Op^* \to \{verdadeiro, falso\}$ *é função de* Π_T

Definição Formal de Máquina de Traços (cont.)

- Portanto, o **efeito de cada operação** interpretada por uma máquina de traços é simplesmente o de **acrescentar o identificador da nova operação à direita do valor atual da memória**

- O **valor atual da memória**, desta forma, corresponde à **sequência de identificadores de operações efetuadas**

- A **função computada** consiste em um **histórico das operações executadas durante a computação**

- Logo, a **definição** de uma máquina de traços requer **apenas a especificação das interpretações dos testes**, pois o efeito das operações é predeterminado

Função Induzida por um Traço em um Máquina

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ *uma máquina,* $Op = \{F, G, H, ...\}$ *o conjunto de operações interpretadas em* Π_F *e* $w = FG...H$ *um possível traço de* M*, tal que* $w \in Op^*$*.*

A ***função induzida pelo traço*** w ***na máquina*** M*, denotada por*

$$[w, M] : X \to V$$

é a função (total)

$$[w, M] = \pi_H \circ ... \circ \pi_G \circ \pi_F \circ \pi_X$$

de forma que a aplicação da função $[w, M]$ *a uma entrada* $x \in X$ *é denotada por*

$$[w, M] = \pi_H \circ ... \circ \pi_G \circ \pi_F \circ \pi_X(x)$$

Teorema 1: Equivalência Forte $\leftrightarrow$ Equivalência em Máquina de Traços

Teorema 1. *"Sejam P e Q dois programas quaisquer, não necessariamente do mesmo tipo. Então $P \equiv Q$ sss, para qualquer máquina de traços M, $P \equiv_M Q$."*

- Para provar este teorema, precisamos mostrar que **equivalência forte implica equivalência em máquina de traços e que o contrário também é verdadeiro**:
  - A prova da **primeira parte é trivial** e se baseia na definição de equivalência
  - A **segunda parte** da prova é dada pela demonstração de que é **absurdo** supor-se que $P \equiv_M Q$ mas $P \not\equiv Q$

Corolário 1: Equivalência Forte $\leftrightarrow$ Equivalência em Máquina de Traços

Corolário 1. *“Sejam P e Q dois programas quaisquer, não necessariamente do mesmo tipo. Então $P \equiv Q$ sss, para qualquer máquina de traços M, $\langle P, M\rangle(\varepsilon) = \langle Q, M\rangle(\varepsilon)$.”*

Instruções Rotuladas Compostas

- Instruções rotuladas compostas possibilitam uma outra maneira de **verificarmos equivalência forte entre programas**

- Elas possuem **um único formato**, ao contrário das instruções rotuladas, as quais podem ter dois formatos (operação ou teste)

Definição de Instrução Rotulada Composta

*Uma **instrução rotulada composta** é uma sequência de símbolos da seguinte forma (suponha que F e G são identificadores de operação e que T é um identificador de teste):*

r_1: se T então faça F vá_para r_2 senão faça G vá_para r_3

Assim, ela combina operações e teste em uma única forma.

Diz-se que r_2 e r_3 são rótulos sucessores de r_1, o qual, por sua vez, é chamado de rótulo antecessor de r_2 e r_3

Definição de Programa Monolítico com Instruções Rotuladas Compostas

Um programa monolítico com instruções rotuladas compostas P *é um par ordenado* (I, r_i) *onde:*

- I *é um conjunto finito de instruções rotuladas compostas*
- r_i *é o rótulo inicial*

Relativamente ao conjunto I*:*

- *Não existem duas instruções diferentes associadas ao mesmo rótulo*
- *Um rótulo referenciado em uma instrução que não possui instrução associada é dito um rótulo final*

Definição de Programa Monolítico com Instruções Rotuladas Compostas (cont.)

- Para facilitar a verificação de equivalência, sem perda de generalidade, **consideraremos somente o caso particular de programas com um único identificador de teste** (T)

- A partir desta definição, uma instrução rotulada composta

  r_1: se T então faça F vá_para r_2 senão faça G vá_para r_3

  pode ser abreviada usando-se o formato:

$$r_1 : \quad (F, r_2), (G, r_3)$$

Fluxograma $\Rightarrow$ Instruções Rotuladas Compostas

- Para utilizarmos instruções rotuladas compostas na verificação de equivalência forte entre programas monolíticos, precisamos utilizar um **algoritmo que transforme fluxogramas em instruções rotuladas compostas**

- Para este algoritmo, considere que **componentes elementares de fluxogramas** (partida, parada e operação) são genericamente chamados de nós

Fluxograma $\Rightarrow$ IRC: Algoritmo

Dado um **programa monolítico** P, **descrito por um fluxograma**, aplica-se o seguinte algoritmo para obtermos um **programa monolítico** P' **constituído de instruções rotuladas compostas**, o qual é descrito em duas partes:

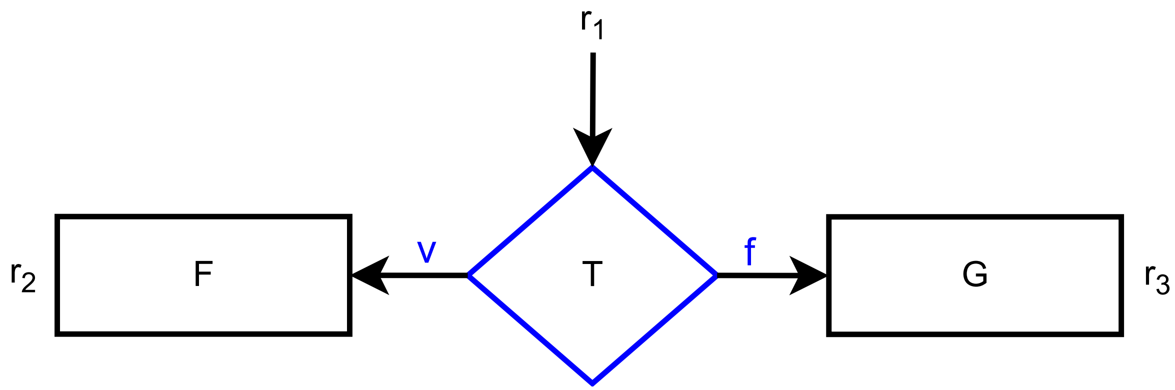
- *Parte 1: Rotulação de Nós*
  - **Rotula-se cada nó do fluxograma**
  - Supondo-se que existe um único componente elementar de **parada**, a este é associado o **identificador** ε **(palavra vazia)**
  - O rótulo correspondente ao nó **partida** é o **rótulo inicial**

Fluxograma $\Rightarrow$ IRC: Algoritmo

- *Parte 2: Construção de Instruções Rotuladas Compostas*
  - **Parte-se do nó partida e segue-se o fluxograma**
  - Constroi-se **uma instrução rotulada composta para cada componente elementar** na sequência do fluxograma
  - Para isto, aplicam-se as **regras** a seguir

Fluxograma $\Rightarrow$ IRC: Algoritmo

- Teste: Para um teste da forma

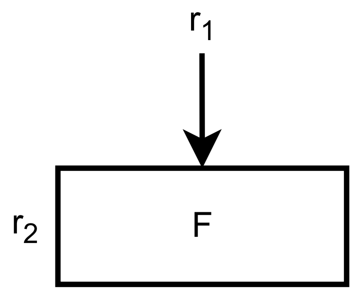


a correspondente instrução rotulada composta é

$$r_1 : (F, r_2), (G, r_3)$$

Fluxograma $\Rightarrow$ IRC: Algoritmo

- Operação: Para uma operação da forma

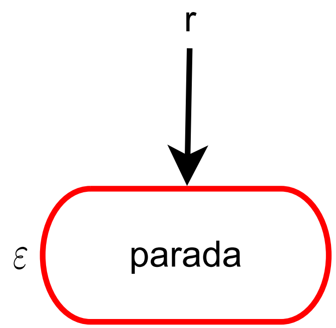


a correspondente instrução rotulada composta é

$$r_1 : (F, r_2), (F, r_2)$$

Fluxograma $\Rightarrow$ IRC: Algoritmo

- Parada: Para uma parada da forma

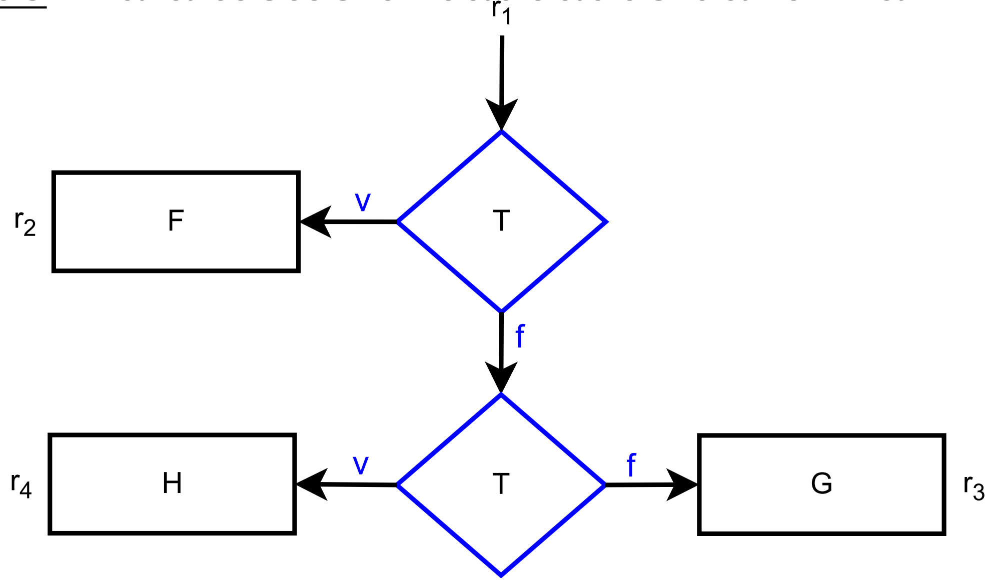


a correspondente instrução rotulada composta é

$$r : (parada, \varepsilon), (parada, \varepsilon)$$

Fluxograma $\Rightarrow$ IRC: Algoritmo

- Teste encadeados: Para testes encadeados da forma

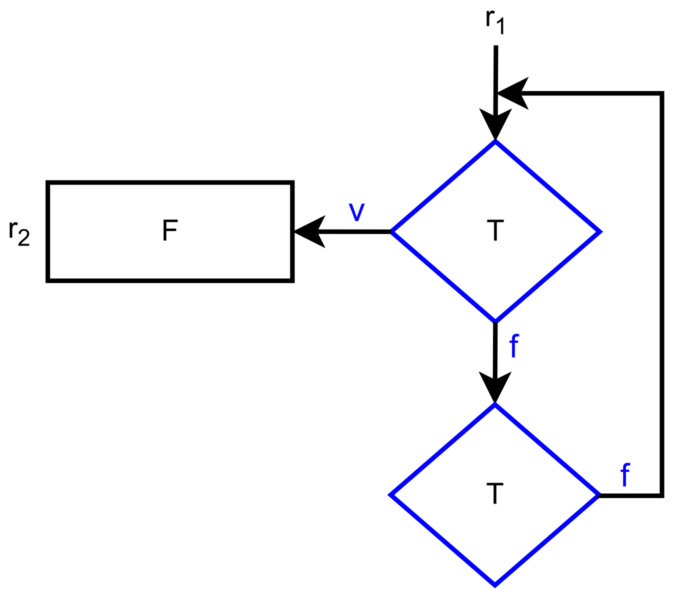


segue-se o fluxo até que seja encontrado um nó, sendo a correspondente instrução rotulada composta como segue

$$r_1 : (F, r_2), (G, r_3)$$

Fluxograma ⇒ IRC: Algoritmo

- Teste encadeados com ciclo infinito: Para testes encadeados com ciclo infinito da forma



a correspondente instrução rotulada composta é

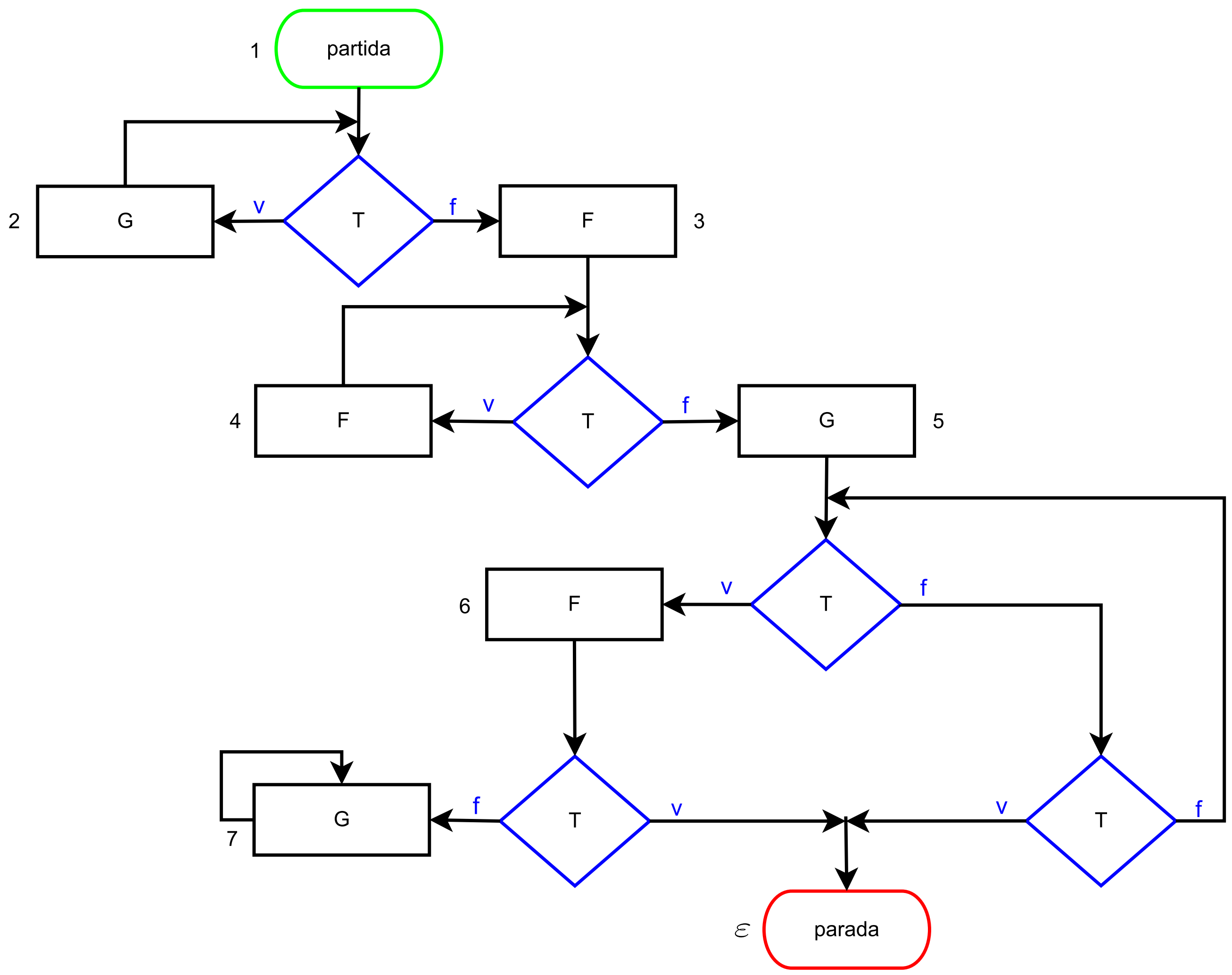
$$r_1 : (F, r_2), (ciclo, \omega)$$

Fluxograma $\Rightarrow$ IRC: Algoritmo

- Teste encadeados com ciclo infinito: (continuação)

  Neste caso, deve ser incluída uma instrução rotulada composta adicional que corresponde ao **ciclo infinito**:

$$\omega : (ciclo, \omega), (ciclo, \omega)$$



1
partida
2
G
v
T
f
F
3
4
F
v
T
f
G
5
6
F
v
T
f
7
G
f
T
v
v
T
f
ε
parada

Exemplo (cont.)

- O programa correspondente com instruções rotuladas compostas é o seguinte, supondo-se que o rótulo 1 é o rótulo inicial:

$$
\begin{array}{l}
1 : (G, 2), (F, 3) \\
2 : (G, 2), (F, 3) \\
3 : (F, 4), (G, 5) \\
4 : (F, 4), (G, 5) \\
5 : (F, 6), (ciclo, \omega) \\
6 : (parada, \varepsilon), (G, 7) \\
7 : (G, 7), (G, 7) \\
\omega : (ciclo, \omega), (ciclo, \omega)
\end{array}
$$

Exemplo (cont.)

- Note que:
  - O rótulo 2 é **sucessor dele mesmo**, assim como ocorre com os rótulos 4, 7 e ω
  - Existem dois caminhos no fluxograma que atingem o nó parada, mas só um é representado no conjunto de instruções rotuladas compostas
  - Na instrução rotulada por 7, ocorre um **ciclo infinito**

Exercícios

- Crie um programa iterativo com pelo menos 2 estruturas de repetição
- Traduza o programa criado em um programa monolítico equivalente
- Mostre, usando uma máquina de traços, que os dois programas anteriores são fortemente equivalentes
- Traduza seu programa do exercício 1 em um programa recursivo equivalente
- Apresente o programa monolítico do exercício 2 na forma de instruções rotuladas compostas

Definição Auxiliar: União Disjunta

- A união disjunta de conjuntos garante que **todos os elementos dos conjuntos** componentes constituem o conjunto resultante, **mesmo que possuam a mesma identificação**

- Considera-se que os **elementos são distintos**, mesmo que possuam a mesma identificação

- **Exemplo:** para os conjuntos $A = \{a, x\}$ e $B = \{b, x\}$, o conjunto resultante da união disjunta é: $\{a_A, x_A, b_B, x_B\}$, ou, simplificando, $\{a, x_A, b, x_B\}$

Equivalência Forte entre Programas Monolíticos

- A verificação de equivalência forte entre programas monolíticos baseia-se no seguinte corolário sobre a união disjunta:

  Corolário 2. *"Sejam $Q = (I_Q, q_0)$ e $R = (I_R, r_0)$ dois programas monolíticos descritos através de instruções rotuladas compostas e sejam $P_q = (I, q_0)$ e $P_r = (I, r_0)$ programas monolíticos onde I é o conjunto resultante da união disjunta de I_Q e I_R. Então, $P_q \equiv P_r$ sss $Q \equiv R$."*

- Portanto, o algoritmo para **verificação da equivalência forte de Q e R** resume-se à **verificação se P_q e P_r são fortemente equivalentes**

Equivalência Forte entre Programas Monolíticos (cont.)

- Contudo, para este algoritmo, devem-se levar em consideração **três conceitos**:
  - Cadeia de conjuntos: sequência de conjuntos ordenada pela relação de inclusão
  - Programa monolítico simplificado: instruções rotuladas compostas que determinam ciclos infinitos são excluídas (excetuando-se a instrução rotulada por ω, se existir), sendo que tal simplificação baseia-se em cadeia de conjuntos
  - Rótulos fortemente equivalentes: o algoritmo de verificação se P_q e P_r são fortemente equivalentes baseia-se em rótulos fortemente equivalentes de programas simplificados

Definição Formal de Cadeia de Conjuntos

*Uma sequência de conjuntos $A_0 A_1 ...$ é dita uma **cadeia de conjuntos** se, para todo $k \geq 0$, $A_k \subseteq A_{k+1}$.*

*Uma **cadeia de conjuntos finita** é uma cadeia de conjuntos onde existe um n tal que, para todo $k \geq 0$, $A_n = A_{n+k}$. Neste caso, define-se o **limite da cadeia de conjuntos finita** como $\lim A_k = A_n$.*

Lema 1: Identificação de Ciclos Infinitos em Programa Monolítico

- Utilizando a definição de cadeia de conjuntos e cadeia de conjuntos finita, temos a base para a ideia de **simplificação de programas monolíticos**:

  Lema 1. *"Seja I um conjunto de n instruções rotuladas compostas e $A_0A_1...$ uma sequência de conjuntos de rótulos indutivamente definida da seguinte maneira:*

  $A_0 = \{\varepsilon\}$

  $A_{k+1} = A_k \cup \{r | r$ *é rótulo de instrução antecessora de alguma instrução rotulada por* $A_k\}$

  Então, $A_0A_1...$ é uma cadeia de conjuntos finita e, para todo rótulo r de I, tem-se que $(I, r) \equiv (I, \omega)$ sss $r \notin \lim A_k$."

Lema 1: Identificação de Ciclos Infinitos em Programa Monolítico (cont.)

- Este lema fornece um **algoritmo para determinar se existem ciclos infinitos em um conjunto de instruções rotuladas compostas**

- A ideia básica é **partir da instrução** $parada$, rotulada por ε, e **determinar os seus antecessores**

- Por exclusão, **uma instrução que não é antecessora de** $parada$ **determina um ciclo infinito**

Exemplo

- Dado o programa abaixo:

$$
\begin{array}{l}
1 : (G, 2), (F, 3) \\
2 : (G, 2), (F, 3) \\
3 : (F, 4), (G, 5) \\
4 : (F, 4), (G, 5) \\
5 : (F, 6), (ciclo, \omega) \\
6 : (parada, \varepsilon), (G, 7) \\
7 : (G, 7), (G, 7) \\
\omega : (ciclo, \omega), (ciclo, \omega)
\end{array}
$$

É possível simplificá-lo?

Exemplo (cont.)

$A_0 = \{\varepsilon\}$

$A_1 = \{6, \varepsilon\}$

$A_2 = \{5, 6, \varepsilon\}$

$A_3 = \{3, 4, 5, 6, \varepsilon\}$

$A_4 = \{1, 2, 3, 4, 5, 6, \varepsilon\}$

$A_5 = \{1, 2, 3, 4, 5, 6, \varepsilon\}$

Logo, $limA_k = \{1, 2, 3, 4, 5, 6, \varepsilon\}$ e $(I, 7) \equiv (I, \omega)$, pois $7 \notin \lim A_k$

Portanto, podemos simplificar o programa eliminando a instrução de rótulo 7

Algoritmo de Simplificação de Ciclos Infinitos

Seja I um conjunto finito de instruções rotuladas compostas. O algoritmo de simplificação de ciclos infinitos é definido pelos seguintes passos:

1. Determina-se a correspondente cadeia de conjuntos finita $A_0 A_1 ...$, como visto no lema anterior

2. Para todo rótulo r de instrução de I, tal que $r \notin \lim A_k$

   (a) Exclui-se a instrução rotulada por r

   (b) Substituem-se todas as referências a pares da forma (F, r) em I, para uma operação F qualquer, por $(ciclo, \omega)$

   (c) $I = I \cup \{\omega : (ciclo, \omega), (ciclo, \omega)\}$

Exemplo

Aplicando-se o algoritmo ao exemplo anterior, temos:

$$
\begin{aligned}
1 &: (G, 2), (F, 3) \\
2 &: (G, 2), (F, 3) \\
3 &: (F, 4), (G, 5) \\
4 &: (F, 4), (G, 5) \\
5 &: (F, 6), (ciclo, \omega) \\
6 &: (parada, \varepsilon), (ciclo, \omega) \\
\omega &: (ciclo, \omega), (ciclo, \omega)
\end{aligned}
$$

Lema 2: Rótulos Consistentes

Lema 2. *"Seja I um conjunto finito de instruções rotuladas compostas e simplificadas. Sejam r e s dois rótulos de instruções de I, ambos diferentes de ε. Suponha que as instruções rotuladas por r e s são da seguinte forma, respectivamente:*

$$\begin{array}{l} r : (F_1, r_1), (F_2, r_2) \\ s : (G_1, s_1), (G_2, s_2) \end{array}$$

*Então, r e s são **consistentes** sss $F_1 = G_1$ e $F_2 = G_2$."*

Definição de Rótulos Fortemente Equivalentes

"Seja I um conjunto finito de instruções rotuladas compostas e simplificadas. Sejam r e s dois rótulos de instruções de I. Suponha que as instruções rotuladas por r e s são da seguinte forma, respectivamente:

$$
\boxed{\begin{array}{l} r : (F_1, r_1), (F_2, r_2) \\ s : (G_1, s_1), (G_2, s_2) \end{array}}
$$

Então, r e s são fortemente equivalentes sss

- *ou $r = s = \varepsilon$;*
- *ou ambos são diferentes de ε e são consistentes."*

Teorema 2: Determinação de Rótulos Fortemente Equivalentes

Teorema 2. Seja I um conjunto de n instruções compostas e simplificadas. Sejam r e s dois rótulos de instruções de I. Define-se, indutivamente, a sequência de conjuntos B_0B_1... por:

$B_0 = \{(r, s)\}$

$B_{k+1} = \{(r'', s'') | r''$ e s'' são rótulos sucessores de r' e s', respectivamente, $(r', s') \in B_k$ e $(r'', s'') \notin B_i$, para $0 \leq i \leq k\}$

Então B_0B_1... é uma sequência que converge para o conjunto vazio, e r e s são **rótulos fortemente equivalentes** sss todo par de B_k é constituído por rótulos consistentes

Algoritmo de Verificação de Equivalência Forte entre Programas Monolíticos

Sejam $Q = (I_Q, q)$ e $R = (I_R, r)$ dois programas monolíticos simplificados especificados usando instruções rotuladas compostas.

Verifica-se a equivalência forte entre Q e R através dos seguintes passos:

Passo 1. Sejam $P_q = (I, q)$ e $P_r = (I, r)$ programas monolíticos onde I é o conjunto resultante da união disjunta de I_Q e I_R, excetuando-se a instrução rotulada ω, se existir, a qual ocorre, no máximo, uma vez em I

Passo 2. Se q e r são rótulos fortemente equivalentes, então $B_0 = \{(q, r)\}$. Caso contrário, Q e R não são fortemente equivalentes, e o algoritmo termina

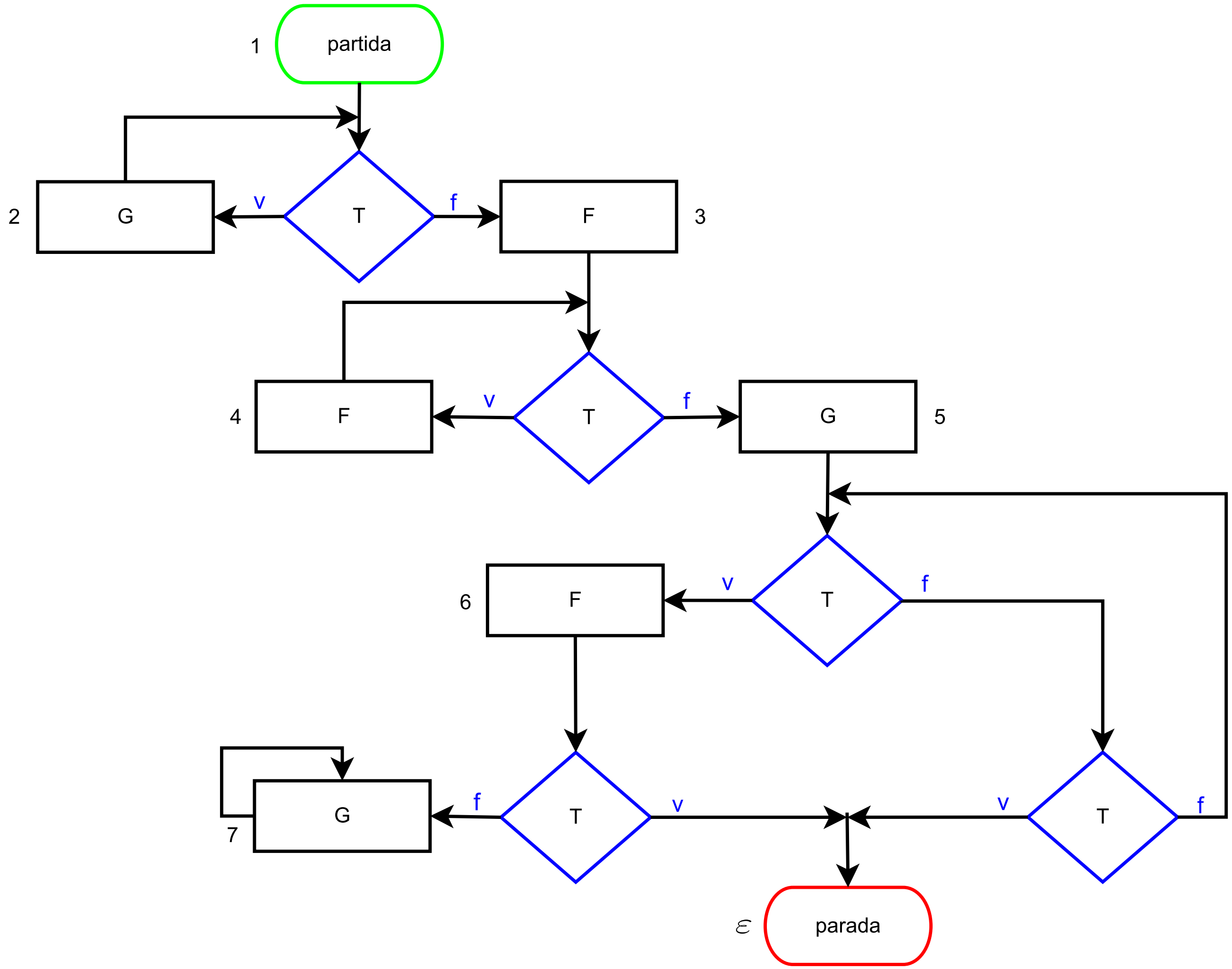
Algoritmo de Verificação de Equivalência Forte entre Programas Monolíticos (cont.)

Passo 3. Para $k \geq 0$, define-se o conjunto B_{k+1}, contendo somente os pares (q'', r'') de rótulos sucessores de cada $(q', r') \in B_k$, tais que:

- $q' \neq r'$
- q' e r' são ambos diferentes de ε
- Os pares sucessores (q'', r'') não são elementos de $B_0, B_1, ..., B_k$

Passo 4. Dependendo de B_{k+1}, tem-se que:

- $B_{k+1} = \emptyset$: Q e R são fortemente equivalentes e o algoritmo termina
- $B_{k+1} \neq \emptyset$: se todos os pares de rótulos de B_{k+1} são fortemente equivalentes, então vá para o **Passo 3**; caso contrário, Q e R não são fortemente equivalentes e o algoritmo termina

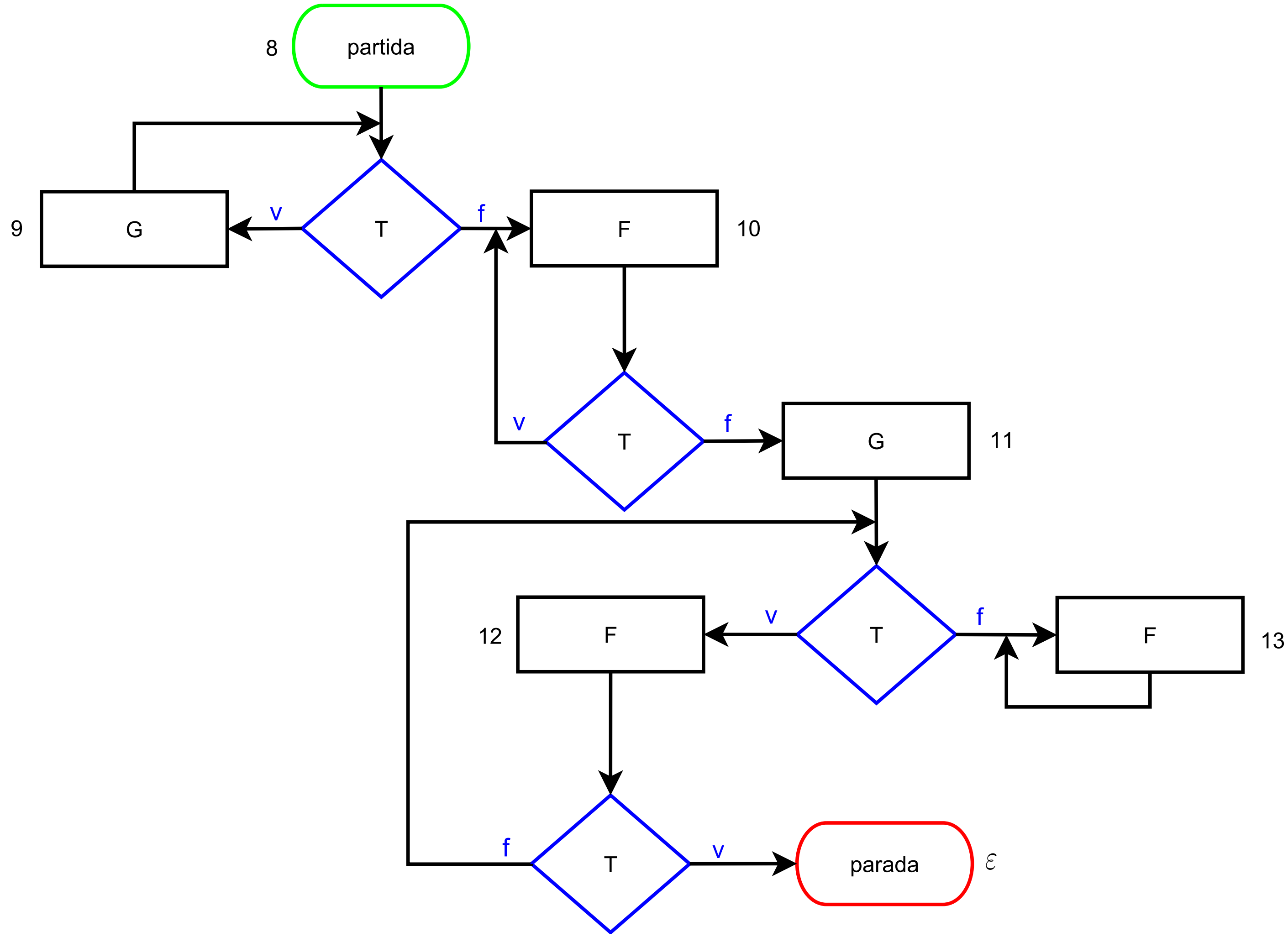


1
partida
2
G
v
T
f
F
3
4
F
v
T
f
G
5
6
F
v
T
f
7
G
f
T
v
v
T
f
ε
parada

Exemplo

Programa Q - Já tínhamos as IRC simplificadas:

$$
\begin{aligned}
&1 : (G, 2), (F, 3)\\
&2 : (G, 2), (F, 3)\\
&3 : (F, 4), (G, 5)\\
&4 : (F, 4), (G, 5)\\
&5 : (F, 6), (ciclo, \omega)\\
&6 : (parada, \varepsilon), (ciclo, \omega)\\
&\omega : (ciclo, \omega), (ciclo, \omega)
\end{aligned}
$$



8
partida
9
G
v
T
f
F
10
v
T
f
G
11
12
F
v
T
f
F
13
f
T
v
parada
ε

Exemplo

Programa R - IRC

$$
\begin{array}{l}
8 : (G, 9), (F, 10) \\
9 : (G, 9), (F, 10) \\
10 : (F, 10), (G, 11) \\
11 : (F, 12), (F, 13) \\
12 : (parada, \varepsilon), (F, 13) \\
13 : (F, 13), (F, 13)
\end{array}
$$

Exemplo

Programa R - Identificação de ciclos infinitos

$A_0 = \{\varepsilon\}$

$A_1 = \{12, \varepsilon\}$

$A_2 = \{11, 12, \varepsilon\}$

$A_3 = \{10, 11, 12, \varepsilon\}$

$A_4 = \{8, 9, 10, 11, 12, \varepsilon\}$

$A_5 = \{8, 9, 10, 11, 12, \varepsilon\}$

Logo, $limA_k = \{8, 9, 10, 11, 12, \varepsilon\}$ e $(I_R, 13) \equiv (I, \omega)$, pois $13 \notin \lim A_k$

Exemplo

Programa R - Simplificação de ciclos infinitos

$$
\begin{aligned}
&8 : (G, 9), (F, 10) \\
&9 : (G, 9), (F, 10) \\
&10 : (F, 10), (G, 11) \\
&11 : (F, 12), (F, 13) \\
&12 : (parada, \varepsilon), (ciclo, \omega) \\
&\omega : (ciclo, \omega), (ciclo, \omega)
\end{aligned}
$$

Exemplo

Algoritmo - Passo 1: União disjunta de I_Q e I_R

$$
\begin{array}{l}
\mathbf{1} : (G, 2), (F, 3) \\
2 : (G, 2), (F, 3) \\
3 : (F, 4), (G, 5) \\
4 : (F, 4), (G, 5) \\
5 : (F, 6), (ciclo, \omega) \\
6 : (parada, \varepsilon), (ciclo, \omega) \\
\mathbf{8} : (G, 9), (F, 10) \\
9 : (G, 9), (F, 10) \\
10 : (F, 10), (G, 11) \\
11 : (F, 12), (F, 13) \\
12 : (parada, \varepsilon), (ciclo, \omega) \\
\omega : (ciclo, \omega), (ciclo, \omega)
\end{array}
$$

Exemplo

Algoritmo - Passo 2: Verificar se 1 e 8 são fortemente equivalentes

Como 1 e 8 são fortemente equivalentes, então temos:

$B_0 = \{(1, 8)\}$

Exemplo

Algoritmo - Passo 3: Para $k \geq 0$, construir B_{k+1}

$B_1 = \{(2, 9), (3, 10)\}$

$B_2 = \{(4, 10), (5, 11)\}$

$B_3 = \{(6, 12), (\omega, \omega)\}$

$B_4 = \{(\varepsilon, \varepsilon)\}$

$B_5 = \emptyset$

Exemplo

Algoritmo - Passo 4: Decidir baseado em B_{k+1}

Como $B_5 = \emptyset$, $(I, 1) \equiv (I, 8)$ e, portanto, $Q \equiv R$