Compiladores

Aula 24: Análise de Fluxo de Dados

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Junho de 2026

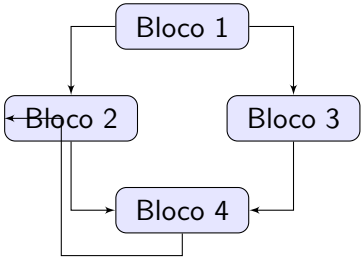
Sumário

Objetivos da Aula

- Revisar os conceitos de Grafos de Fluxo de Controle (CFG).
- Compreender o propósito e a importância da Análise de Fluxo de Dados.
- Estudar os princípios matemáticos e lógicos das Equações de Fluxo.
- Analisar em detalhes os três principais problemas clássicos:
  - *Reaching Definitions* (Definições Alcançáveis)
  - *Liveness Analysis* (Variáveis Vivas)
  - *Available Expressions* (Expressões Disponíveis)
- Entender o algoritmo iterativo para resolução das equações de fluxo.

O que é Análise de Fluxo de Dados?

- É uma técnica para coletar informações sobre como os dados fluem através do programa em tempo de execução, mas realizada estaticamente (em tempo de compilação).
- **Objetivo:** Permitir que o compilador tome decisões seguras sobre otimizações globais.
- Analisa-se o programa modelado como um **Grafo de Fluxo de Controle (CFG)**, onde:
  - Nós = Blocos Básicos.
  - Arestas = Possíveis transições.



Análise Local vs. Global

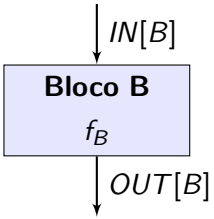
- **Análise Local (Intrabloco):**
  - Foca em um único Bloco Básico.
  - Simples, de cima para baixo, sem se preocupar com saltos.

- **Análise Global (Interbloco):**
  - Considera a interação entre múltiplos blocos.
  - Precisa lidar com ramificações (`if/else`) e laços (`while`, `for`).
  - Aqui entra a **Análise de Fluxo de Dados**.

Pontos de Programa e Caminhos

Para raciocinar sobre o fluxo, definimos "pontos" antes e depois de cada instrução ou bloco.

- **IN[B]:** Informação disponível *imediatamente antes* de entrar no bloco B.
- **OUT[B]:** Informação disponível *imediatamente após* a saída do bloco B.



A análise de fluxo estabelece um sistema de **equações de fluxo de dados** baseadas nestes conjuntos para cada bloco do CFG.

Funções de Transferência (f_B)

A transformação da informação de IN para OUT (numa análise forward) dentro de um bloco é dada por uma **função de transferência** f_B:

$$OUT[B] = f_B(IN[B])$$

Na maioria das análises clássicas, f_B assume a forma:

Equação Geral (para análises forward)

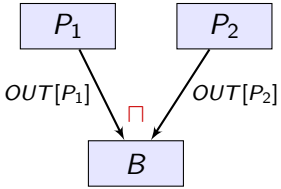
$$OUT[B] = GEN[B] \cup (IN[B] - KILL[B])$$

- **GEN[B]**: Informação **gerada** localmente pelo bloco B.
- **KILL[B]**: Informação **morta (invalidada)** pelo bloco B.

Arcabouço Teórico (Data-Flow Frameworks)

Uma análise de fluxo de dados é definida formalmente por um arcabouço $\langle V, \sqcap, F \rangle$:

- V: Um domínio de valores com uma ordem parcial $\sqsubseteq$, formando um **semirreticulado** (*semilattice*).
- $\sqcap$: Operador de **confluência** ou *meet* (ex: $\cup$ ou $\cap$). Extrai o limite inferior na junção de caminhos.
- F: Uma família de **funções de transferência** $f : V \to V$.



Equação de Confluência em Bifurcações:

$$IN[B] = \sqcap_{P \in pred(B)} OUT[P]$$

Propriedades das Funções de Transferência

Para garantir corretude e convergência do algoritmo iterativo, F deve ter propriedades:

- **Monotonicidade:** $x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$.
  Garante que a informação no reticulado caminhe monotonicamente em uma direção, atingindo um **Ponto Fixo** em um número finito de passos (se a altura do reticulado for finita).
- **Distributividade:** $f(x \sqcap y) = f(x) \sqcap f(y)$.
  Uma propriedade mais forte. Se as funções forem distributivas, a solução iterativa converge para a solução exata/ideal do fluxo.

Reaching Definitions (Definições Alcançáveis)

- **Definição:** Uma atribuição `d: x = ...` alcança um ponto p se existe um caminho de d até p onde x não é redefinida.
- **Domínio** V**:** Conjunto das partes de todas as definições. **Operador** $\sqcap$**:** União ($\cup$).

Cálculo de GEN e KILL para o bloco com instrução `d: x = y + z`

- $gen[B] = \{d\}$
- $kill[B] = Defs(x) - \{d\}$ (todas as outras definições da variável x).

Inicialização: $OUT[Entry] = \emptyset$; $OUT[B] = \emptyset$ (Elemento top do reticulado).

Equações para Reaching Definitions

- **Direção:** *Forward*.
- **Aplicações:** Propagação de constantes, Uso de variáveis não inicializadas.

Sistema de Equações

$$IN[B] = \bigcup_{P \in pred(B)} OUT[P]$$

$$OUT[B] = GEN[B] \cup (IN[B] - KILL[B])$$

Liveness Analysis (Análise de Variáveis Vivas)

- Uma variável está **viva** em p se seu valor será lido em algum caminho futuro antes de ser sobrescrita.
- **Domínio** V**:** Conjunto das partes de variáveis. **Operador** $\sqcap$**:** União ($\cup$).

Cálculo de GEN e KILL para instrução `x = y + z`

- $gen[B] = \{y, z\}$ (variáveis lidas/usadas antes de serem escritas no bloco).
- $kill[B] = \{x\}$ (variáveis escritas/sobrescritas).

Inicialização: $IN[Exit] = \emptyset$; $IN[B] = \emptyset$.

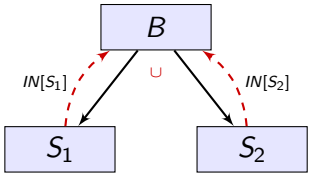
Equações para Liveness Analysis

- **Direção:** *Backward* (A informação flui do fim para o início).
- **Aplicações:** Alocação de registradores, Eliminação de código morto.

Sistema de Equações Backward

$$OUT[B] = \bigcup_{S \in succ(B)} IN[S]$$

$$IN[B] = GEN[B] \cup (OUT[B] - KILL[B])$$



Fluxo Backward

Available Expressions (Expressões Disponíveis)

- `x op y` está **disponível** em p se foi computada em **todos os caminhos** até p, e nem x nem y sofreram alteração.
- **Domínio** V**:** Conjunto das partes de expressões. **Operador** $\sqcap$**:** Interseção ($\cap$).

Cálculo de GEN e KILL para instrução `x = y + z`

- $gen[B] = \{y + z\}$ (desde que x não seja y nem z).
- $kill[B]$ = todas as expressões do programa que dependem da variável x.

Inicialização: $OUT[Entry] = \emptyset$; $OUT[B] = U$ (Conjunto Universo, elemento top).

Equações para Available Expressions

- **Direção:** *Forward*.
- **Aplicações:** Eliminação Global de Subexpressões Comuns (Global CSE).

Sistema de Equações com Interseção

$$IN[B] = \bigcap_{P \in pred(B)} OUT[P]$$

$$OUT[B] = GEN[B] \cup (IN[B] - KILL[B])$$

O Algoritmo Iterativo de Worklist

A presença de ciclos no CFG exige uma solução iterativa de ponto fixo.

1. **Inicialização:** IN/OUT baseados no elemento Top ($\top$) do reticulado ($\emptyset$ para $\cup$, U para $\cap$).
2. Insere todos os blocos na *Worklist*.
3. Enquanto a *Worklist* não estiver vazia:
   - Retira um bloco B.
   - Recalcula $IN[B]$ aplicando o operador $\sqcap$ nos predecessores.
   - Recalcula $OUT[B]$ aplicando $f_B(IN[B])$.
   - Se $OUT[B]$ mudou, reinsere os sucessores de B na *Worklist*.

Qualidade da Solução: MOP vs MFP

- **MOP (Meet-Over-All-Paths):** Solução teórica ideal. Computa as funções ao longo de todos os caminhos do CFG da entrada até o bloco e então aplica o *meet*. (Pode ser indecidível).
- **MFP (Maximal Fixed Point):** Solução prática alcançada pelo Algoritmo Iterativo. Aplica o *meet* em cada bifurcação.

O Teorema de Kam-Ullman

- O algoritmo itetativo é conservador: $MFP \sqsubseteq MOP$.
- Se as funções de transferência de fluxo forem **Distributivas**, então $MFP = MOP$. (Ex: Os problemas de bit-vector vistos hoje são distributivos!).

Referencias

- Aho, et al. *Compilers: Principles, Techniques, and Tools* (Livro do Dragao).
- Cooper & Torczon. *Engineering a Compiler*.
- Appel, Andrew W. *Modern Compiler Implementation*.

Duvidas?

Obrigado!

Email: alessio@cefetmg.br