# SNU 4541.664A Program Analysis
# Note 12

Prof. Kwangkeun Yi

## 요약해석 구현 알고리즘

- 초보적인 고정점 반복(*fixpoint iterations*) 알고리즘
- 할일만 하는 고정점 계산 알고리즘(worklist algorithm)
- 요약의미 공간 $\hat{D}$이 함수공간일 경우

# 초보적인 고정점 반복(*fixpoint iterations*) 알고리즘

프로그램 $C$의 분석은

$$Tabulate(\hat{\mathcal{F}}, C)$$

의 계산

- $i$-번째 반복은 $\hat{\mathcal{F}}_C^i(\bot_{\hat{D}}, \cdots, \bot_{\hat{D}})$를 계산
- 방정식을 세우고 푸는 것이 따로 있지않고 프로그램 부품 $C_k$의 방정식 오른편 계산이 $\hat{\mathcal{F}}\,(\lambda x.T(x))\,C_k$를 호출하는 것으로
- $\hat{D}$의 높이가 유한하다면, 유한번 반복으로 멈추게 된다.
- 테이블 $T$가 분석결과(방정식의 해)가 된다.

$Tabulate(\hat{\mathcal{F}} : (Code \to \hat{D}) \to (Code \to \hat{D}),\ C : Code)$
$T, T' : Code \to \hat{D};$
**begin**
  $\forall C_i$ of $C : T(C_i) := T'(C_i) := \bot_{\hat{D}};$
  **repeat**
    $T' := T;$
    $\forall C_i$ of $C : T(C_i) := \hat{\mathcal{F}}\ (\lambda x.T(x))\ C_i;$
  **until** $T \sqsubseteq T'$ (* no more increase *)
**end**

Figure: 초보적인 프로그램 분석 알고리즘

# 알고리즘의 복잡도

방정식의 갯수(분석할 프로그램의 부품 수, 프로그램의 크기)가 $N$이라고 하면, 매 반복마다 드는 최대 시간은

$$N \times \mathcal{O}(\hat{\mathcal{F}}) + N \times \mathcal{O}(\sqsubseteq)$$

이고, 반복의 최대 횟수는 $\hat{D}$의 높이가 된다.

- $\hat{D}$의 높이가 무한하던가, 유한하더라도 빨리 알고리즘이 끝나게 하려면, 축지법(widening)을 사용해야 (다음 페이지 알고리즘)
- 축지법(*widening operator*) $\triangledown$을 써서 안전한 방정식의 해를 신속히 계산하고, 좁히기(*narrowing operator*) $\triangle$로 그 결과를 다듬게 된다.

$Tabulate^{\triangle}_{\triangledown}(\hat{\mathcal{F}} : (Code \to \hat{D}) \to (Code \to \hat{D}),\ C : Code)$
$T, T' : Code \to \hat{D};$
$d : \hat{D};$
**begin**
  $\forall C_i$ of $C : T(C_i) := T'(C_i) := \bot_{\hat{D}};$
  **repeat**
    $T' := T;$
    $\forall C_i$ of $C$ :
      $d := \hat{\mathcal{F}}\ (\lambda x.T(x))\ C_i;$
      if $d \not\sqsubseteq T'(C_i)$ then $T(C_i) := T'(C_i) \triangledown d$
  **until** $T \sqsubseteq T'$ (* no more increase *)
  **repeat**
    $T' := T;$
    $\forall C_i$ of $C : T(C_i) := T'(C_i) \triangle \hat{\mathcal{F}}\ (\lambda x.T(x))\ C_i;$
  **until** $T' \sqsubseteq T$ (* no more decrease *)
**end**

Figure: 초보적인 프로그램 분석 알고리즘: 축지법과 좁히기

위의 알고리즘들이 초보적인 이유:

- 불필요한 계산이 많다; 매 반복마다 모든 방정식의 오른쪽을 다시 계산:

$$\forall C_i \in C : T(C_i) := \hat{\mathcal{F}}\,(\lambda x.T(x))\,C_i$$

# 할일만 하는 고정점 계산 알고리즘(worklist algorithm)

- 좀더 효율적이기 위해서는, 매 반복마다 다시 계산할 필요가 있는 방정식만 다시 계산하면 된다.
- 다시 계산할 필요가 있는 방정식은 이전 반복에서 변화가 일어난 $T(C_i)$에 영향받을 수 있는 방정식들 만이다.

이렇게 할일만하기 방식으로 알고리즘(*worklist algorithm*)을 정의하면 다음과 같다.

$Tabulate(\hat{\mathcal{F}}\colon (Code \to \hat{D}) \to (Code \to \hat{D}),\ C\colon Code)$
$T\colon Code \to \hat{D}, \quad y\colon \hat{D}, \quad W\colon 2^{Code}, \quad w\colon Code$

$f(c\colon Code)\colon \hat{D}$
**begin**
    record that evaluation of $w$ requires that of $c$;
    **return** $T(c)$

**begin**
    $\forall C_i$ of $C : T(C_i) := T'(C_i) := \bot_{\hat{D}}$;
    $W := \{C_i \mid C_i \in C\}$
    **repeat**
        $w := \textbf{Select}(W)$
        $y := \hat{\mathcal{F}}\ f\ w$
        **if** $y \not\sqsubseteq T_Y(w)$ **then**
            $T(w) := y$
            $\forall w'$ whose evaluation needs that of $w$ :
                $W := \textbf{Add}(W, w')$
    **until** $W = \{\}$
**end**

# 요약의미 공간 $\hat{D}$이 함수공간일 경우

의미를 결정하는 함수 $\hat{\mathcal{F}}$는 대게

$$\hat{\mathcal{F}} \in (Code \to \hat{A} \to \hat{B}) \to (Code \to \hat{A} \to \hat{B})$$

의 꼴

- 방정식의 재구성 (단수를 낮추어)

$$\begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix} = \hat{\mathcal{F}}_C \begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix}$$

보다는,

$$\begin{pmatrix} \hat{X}_0^{\downarrow}, \cdots, \hat{X}_n^{\downarrow} \\ \hat{X}_0^{\uparrow}, \cdots, \hat{X}_n^{\uparrow} \end{pmatrix} = \hat{\mathcal{G}}_C \begin{pmatrix} \hat{X}_0^{\downarrow}, \cdots, \hat{X}_n^{\downarrow} \\ \hat{X}_0^{\uparrow}, \cdots, \hat{X}_n^{\uparrow} \end{pmatrix}$$

를 푼다.

# 초보적인 분석 알고리즘: 프로그램 의미($\in \hat{A} \to \hat{B}$)가 입출력 함수일 때

$Tabulate(\hat{\mathcal{F}}\colon (Code \to \hat{A} \to \hat{B}) \to Code \to \hat{A} \to \hat{B},\ C\colon Code,\ \hat{a}_0\colon \hat{A})$
$T_A, T'_A\colon Code \to \hat{A}$;
$T_B, T'_B\colon Code \to \hat{B}$;

$f\ (c\colon Code)\ (a\colon \hat{A}) \quad : \hat{B}$
**begin**
  $T_A(c) := T_A(c) \sqcup a$;
  **return** $T_B(c)$
**end**

**begin**
  $\forall C_i$ of $C : T_A(C_i) := \bot_{\hat{A}}, \quad T_B(C_i) := \bot_{\hat{B}}$;
  $T_A(C) = \hat{a}_0$;
  **repeat**
    $\langle T'_A, T'_B \rangle = \langle T_A, T_B \rangle$;
    $\forall C_i \in C : T_B(C_i) := \hat{\mathcal{F}}\ f\ C_i\ (T_A(C_i))$;
  **until** $(T_A \sqsubseteq T'_A) \wedge (T_B \sqsubseteq T'_B)$
**end**

# 할일만 하기 방식의 프로그램 분석 알고리즘: 프로그램 의미($\in \hat{A} \to \hat{B}$)가 입출력 함수일 때

$Tabulate(\hat{\mathcal{F}}\colon (Code \to \hat{A} \to \hat{B}) \to (Code \to \hat{A} \to \hat{B}),\ C\colon Code,\ \hat{a}_0\colon \hat{A})$
$T_A\colon Code \to \hat{A}, \quad T_B\colon Code \to \hat{B}, \quad W\colon 2^{Code}, \quad w\colon Code, \quad y\colon \hat{B}$

$f\ (c\colon Code)\ (a\colon \hat{A}) \quad : \hat{B}$
begin
  record that evaluation of $w$ requires that of $c$;
  if $(a \not\sqsubseteq T_A(c))$ then
    $T_A(c) := T_A(c) \sqcup a$;
    $W := \textbf{Add}(W, c)$;
  return $T_B(c)$
end
begin
  $\forall C_i \in C : T_A(C_i) := \bot_{\hat{A}}, T_B(C_i) := \bot_{\hat{B}}$;
  $T_A(C) := \hat{a}_0; \quad W := \{C_i \mid C_i \in C\}$
  repeat
    $w := \textbf{Select}(W); \quad y := \hat{\mathcal{F}}\ f\ w\ (T_A(w))$
    if $y \not\sqsubseteq T_B(w)$ then
      $T_B(w) := y$;
      $\forall w'$ whose evaluation needs that of $w$ :
        $W := \textbf{Add}(W, w')$
  until $W = \{\}$
end