

SNU 4541.664A Program Analysis

Spring 2006

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

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

의 계산

- i -번째 반복은 $\hat{\mathcal{F}}_C^i(\perp_{\hat{D}}, \dots, \perp_{\hat{D}})$ 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( $\hat{\mathcal{F}}: (Code \rightarrow \hat{D}) \rightarrow (Code \rightarrow \hat{D}), C: Code$ )
 $T, T': C \rightarrow \hat{D}$ ;
begin
   $\forall C_i \text{ of } C : T(C_i) := T'(C_i) := \perp_{\hat{D}}$ ;
  repeat
     $T' := T$ ;
     $\forall C_i \text{ of } C : T(C_i) := \hat{\mathcal{F}} (\lambda x.T(x)) C_i$ ;
  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, 그림 2의 알고리즘을 사용.
- 축지법(*widening operator*) ∇ 을 써서 안전한 방정식의 해를 신속히 계산하고, 좁히기(*narrowing operator*) \triangle 로 그 결과를 다듬게 된다.


```

Tabulate $\triangleleft_{\nabla}$ ( $\hat{\mathcal{F}} : (Code \rightarrow \hat{D}) \rightarrow (Code \rightarrow \hat{D}), C : Code$ )
 $T, T' : C \rightarrow \hat{D}$ ;
 $d : \hat{D}$ ;
begin
   $\forall C_i \text{ of } C : T(C_i) := T'(C_i) := \perp_{\hat{D}}$ ;
  repeat
     $T' := T$ ;
     $\forall C_i \text{ of } C :$ 
       $d := \hat{\mathcal{F}} (\lambda x. T(x)) C_i$ ;
      if  $d \not\sqsubseteq T'(C_i)$  then  $T(C_i) := T'(C_i) \nabla d$ 
  until  $T \sqsubseteq T'$  (* no more increase *)
  repeat
     $T' := T$ ;
     $\forall C_i \text{ of } C : T(C_i) := T'(C_i) \triangleleft \hat{\mathcal{F}} (\lambda x. T(x)) C_i$ ;
  until  $T' \sqsubseteq T$  (* no more decrease *)
end

```


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 그림 4과 같다.

$W: 2^{Code} (* \text{worklist} *)$ $w: Code$ $f(c: Code): \hat{D}$

begin

 record that evaluation of w requires that of c ; $W := \mathbf{Add}(W, c)$; return $T(c)$

end

Figure: 할일만 하기 방식의 프로그램 분석 알고리즘
(1/2)(worklist algorithm)

```

Tabulate( $\hat{\mathcal{F}}: (Code \rightarrow \hat{D}) \rightarrow (Code \rightarrow \hat{D}), C: Code$ )
 $T_X: Pgm \rightarrow \hat{X}; \quad T_Y: Pgm \rightarrow \hat{Y}; \quad y: \hat{Y};$ 
begin
   $\forall C_i \text{ of } C : T(C_i) := T'(C_i) := \perp_{\hat{D}};$ 
   $W := \{C_i \mid C_i \in C\}$ 
  repeat
     $w := \mathbf{Select}(W)$ 
     $y := \hat{\mathcal{F}} f w$ 
    if  $y \not\sqsubseteq T_Y(w)$  then
       $T_Y(w) := y$ 
       $\forall w'$  whose evaluation needs that of  $w :$ 
         $W := \mathbf{Add}(W, w')$ 
    until  $W = \{\}$ 
end

```

Figure: 할일만 하기 방식의 프로그램 분석 알고리즘
(2/2)(worklist algorithm)

요약의미 공간 \hat{D} 이 함수공간일 경우

의미를 결정하는 함수 $\hat{\mathcal{F}}$ 는 대개

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

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,

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

$$f(c: Code)(a: \hat{A}) : \hat{B}$$

begin

$$T_X(c) := T_X(c) \sqcup a;$$

$$\text{return } T_Y(c)$$

end

$$\text{Tabulate}(\hat{\mathcal{F}}: (Code \rightarrow \hat{A} \rightarrow \hat{B}) \rightarrow Code \rightarrow \hat{A} \rightarrow \hat{B}, C: Code, a: \hat{A})$$

$$T_A, T'_A: Code \rightarrow \hat{A};$$

$$T_B, T'_B: Code \rightarrow \hat{B};$$

begin

$$\forall C_i \text{ of } C : T_A(C_i) := \perp_{\hat{A}}, \quad T_B(C_i) := \perp_{\hat{B}};$$

$$T_A(C) = a;$$

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

$W : 2^{Code}$ $w : Code$ $f(c : Code)(a : \hat{A}) : \hat{B}$

begin

 record that evaluation of w requires that of c ; if $(x \not\sqsubseteq T_A(c))$ then $T_A(c) := T_A(c) \sqcup a$; $W := \mathbf{Add}(W, c)$; return $T_B(c)$

end

Figure: 할일만 하기 방식의 프로그램 분석 알고리즘 (worklist algorithm): 프로그램 의미 ($\in \hat{A} \rightarrow \hat{B}$)가 입출력 함수일 때 (1/2)


```

Tabulate( $\hat{\mathcal{F}}: (Code \rightarrow \hat{A} \rightarrow \hat{B}) \rightarrow (Code \rightarrow \hat{A} \rightarrow \hat{B}), C: Code, \hat{a}: \hat{A}$ )
 $T_A: C \rightarrow \hat{A}; \quad T_B: C \rightarrow \hat{B}; \quad y: \hat{B};$ 
begin
   $\forall C_i \in C: T_A(C_i) := \perp_{\hat{A}}, T_B(C_i) := \perp_{\hat{B}};$ 
   $T_A(C) := \hat{a};$ 
   $W := \{C_i \mid C_i \in C\}$ 
  repeat
     $w := \mathbf{Select}(W)$ 
     $y := \hat{\mathcal{F}} f w T_A(w)$ 
    if  $y \not\sqsubseteq T_B(w)$  then
       $T_B(w) := y$ 
       $\forall w'$  whose evaluation needs that of  $w$  :
         $W := \mathbf{Add}(W, w')$ 
    until  $W = \{\}$ 
end

```

Figure: 할일만 하기 방식의 프로그램 분석 알고리즘 (worklist algorithm)