

1.5

SNU 4541.664A Program Analysis Note 2

Prof. Kwangkeun Yi

계획

1 문법구조

- 요약된 혹은 구체적인

2 의미구조

- 궁극을 드러내는 의미구조(*denotational semantics*)

프로그램 집합 = 귀납적인 정의

프로그램의 생김새

- 나무구조를 갖춘 2차원 모습

정수식

$$\begin{array}{l} E \rightarrow n \quad (n \in \mathbb{Z}) \\ | \\ | \quad E + E \\ | \\ | \quad -E \end{array}$$

명령형 언어 프로그램

$$\begin{array}{l} C \rightarrow \text{skip} \\ | x := E \\ | \text{if } E C C \\ | C ; C \end{array}$$

군더더기 없이

$$\begin{array}{l}
 C \rightarrow \& \\
 \quad | \quad = x E \\
 \quad | \quad ? E C C \\
 \quad | \quad ; C C
 \end{array}$$

$$\begin{array}{l}
 E \rightarrow n \quad (n \in \mathbb{Z}) \\
 \quad | \quad + E E \\
 \quad | \quad - E
 \end{array}$$

$$\begin{array}{l}
 C \rightarrow \& \\
 \quad | \quad x E \\
 \quad | \quad E C C \\
 \quad | \quad C C
 \end{array}$$

$$\begin{array}{l}
 E \rightarrow n \quad (n \in \mathbb{Z}) \\
 \quad | \quad E E \\
 \quad | \quad - E
 \end{array}$$

이렇게 까지 최대한 요약해서?

요약된 v.s. 구체적인 문법구조

- 요약된 문법구조(*abstract syntax*)
 - 프로그램을 만들 때 사용하는 규칙
 - 나무구조를 가진 2차원의 구조물
- 구체적인 문법구조(*concrete syntax*)
 - 읽을 때 사용하는 규칙
 - 프로그램의 표현: 1차원의 실
 - 1차원의 실에서 2차원의 구조물을 복구하는 규칙

구체적인 문법(*concrete syntax*)

$-1+2$ 는 $((-1) + 2)$? $-(1 + 2)$?

■ 답변 불가

$$\begin{array}{l}
 E \rightarrow n \quad (n \in \mathbb{Z}) \\
 | \quad E + E \\
 | \quad -E
 \end{array}$$

■ 답변 가능

$$\begin{array}{l}
 E \rightarrow n \quad (n \in \mathbb{Z}) \\
 | \quad E + E \\
 | \quad -F \\
 F \rightarrow n \\
 | \quad (E)
 \end{array}$$

구체적인 문법(*concrete syntax*)

- 1차원의 실에서 2차원의 구조물을 혼동없이 복구시키는 규칙
 - 방향성(*associativity*)과 우선순위(*precedence*)
- 프로그램 복원(*parsing*) 혹은 문법검증(*parsing*) 과정의 설계도

이제부터 “프로그램” 하면

요약된 문법(*abstract syntax*)으로 만들어진 2차원의 나무 구조물

- 편의를 위해서 1차원 실로 표현
- 적절히 괄호를 이용해서 그 구조를 명시

의미구조(*semantics*)

- 프로그램이 뜻하는 바를 정의
- 프로그램이 뜻 하는 바?

“1+2” 라는 프로그램의 뜻?

- 의미 = 궁극: “3”
궁극을 드러내는 의미구조(*denotational semantics*)
- 의미 = 계산과정: “1과 2을 더해서 3을 계산한다.”
과정을 드러내는 의미구조(*operational semantics*)

다양하다

프로그램의 의미를 혼동없이 표현하는 스타일들

- 모두 충분히 엄밀
- 각 스타일마다 다양한 증명 기술이 존재
- 각 스타일이 어울리는 경우가 있음
- 우리 의도에 맞는 의미구조 방식을 선택

공극을 드러내는 의미구조(*denotational semantics*)

- 프로그램의 의미: 전통적인 수학의 세계에서, 의미하는 바 그 공극을 드러냄
- “denotational semantics” = “지칭하는 바를 드러내는 의미구조”
- 조립식 의미구조(*compositional semantics*): 프로그램 부품들의 의미들을 가지고 정의
- 고정점 의미구조(*fixpoint semantics*): 어느함수의 고정점으로 프로그램의 의미를 정의

$$\begin{array}{l}
 C \rightarrow \text{skip} \\
 \quad | \quad x := E \\
 \quad | \quad \text{if } E \ C \ C \\
 \quad | \quad C ; C \\
 E \rightarrow n \quad (n \in \mathbb{Z}) \\
 \quad | \quad x \\
 \quad | \quad E + E \\
 \quad | \quad - E
 \end{array}$$

- 명령어 C 의 의미 = 함수: 메모리에서 메모리로
- 메모리 = 함수: 주소에서 값으로

의미공간(*semantic domain*)

의미를 정의할 때 사용하는 물건들의 집합

$$M \in \text{Memory} = \text{Var} \rightarrow \text{Value}$$

$$z \in \text{Value} = \mathbb{Z}$$

$$x \in \text{Var} = \text{Program Variable}$$

$$\text{명령문 } C \text{의 의미 } \llbracket C \rrbracket \in \text{Memory} \rightarrow \text{Memory}$$

$$\text{계산식 } E \text{의 의미 } \llbracket E \rrbracket \in \text{Memory} \rightarrow \mathbb{Z}$$

$$\begin{aligned}
\llbracket \text{skip} \rrbracket M &= M \\
\llbracket x := E \rrbracket M &= M\{x \mapsto \llbracket E \rrbracket M\} \\
\llbracket \text{if } E \ C_1 \ C_2 \rrbracket M &= \text{if } \llbracket E \rrbracket M \neq 0 \text{ then } \llbracket C_1 \rrbracket M \text{ else } \llbracket C_2 \rrbracket M \\
\llbracket C_1 ; C_2 \rrbracket M &= \llbracket C_2 \rrbracket (\llbracket C_1 \rrbracket M) \\
\llbracket n \rrbracket M &= n \\
\llbracket E_1 + E_2 \rrbracket M &= (\llbracket E_1 \rrbracket M) + (\llbracket E_2 \rrbracket M) \\
\llbracket - E \rrbracket M &= -(\llbracket E \rrbracket M)
\end{aligned}$$

잘 정의되었는가? 조립식인가?

조립식이 안되는 경우

`while E C`

의 의미

$$\begin{aligned} & \llbracket \text{while } E \ C \rrbracket M \\ & = \text{if } \llbracket E \rrbracket M \neq 0 \text{ then } \llbracket \text{while } E \ C \rrbracket (\llbracket C \rrbracket M) \text{ else } M \end{aligned}$$

조립식인가?

$$\begin{aligned} & \llbracket \text{while } E \ C \rrbracket M \\ & = \text{if } \llbracket E \rrbracket M \neq 0 \text{ then } \llbracket \text{while } E \ C \rrbracket (\llbracket C \rrbracket M) \text{ else } M \end{aligned}$$

- 정의가 아님; 방정식일 뿐
- 그 해는 무엇일까? 해는 과연 있을까? 항상 있을까?
있다면 유일하게 있을까?

- 그 질문들에 대한 답은 모두 “예”
- 의미방정식에서 사용하는 물건들이 소속된 공간 (*semantic domain*)이 특별하기 때문
- 읽기: [Sco89,Sco72,Sco70]

모든 컴퓨터 프로그램(모든 계산 가능한 함수들)의 의미 (의미방정식의 해)는 의미공간 이론에서 규정하는 성질의 집합안에서 유일하게 존재하고 그것은 이러이러하다.

의미공간 이론(*domain theory*)

- 의미공간 = CPO(*complete partial order set*)
- 프로그램의 의미방정식들에 쓰이는 것들은 모두 어떤 CPO의 원소들
- 연산자들은 모두 CPO에서 CPO로 가는 연속함수(*continuous function*)
 - “Computability is continuity.”
 - “프로그램은 연속함수이다.”

프로그램 C 의 의미 $\llbracket C \rrbracket$ 에 대한 의미방정식은 항상 다음과 같고

$$\llbracket C \rrbracket = \mathcal{F}(\llbracket C \rrbracket)$$

여기서 $\llbracket C \rrbracket \in D$ (어떤 CPO)

그리고 $\mathcal{F} \in D \rightarrow D$ (D 에서 D 로의 연속함수들의 CPO)

- 이 방정식의 해는 항상 연속함수 \mathcal{F} 의 최소고정점(*least fixpoint*)으로 정의.
- 고정점 의미구조(*fixpoint semantics*)

CPO (의미공간)

프로그램의 수학적(공극적인) 의미는 CPO(*complete partial order*)라는 공간의 한 원소

- CPO 는 집합
- 집합의 원소들 간에 어떤 순서가 있고(*partial order*)
- 모든 원소보다작은 밑바닥 원소(\perp)가 있고
- 그 순서로 일렬로 줄 세울 수 있는 원소들(*chain*)의 최소윗뚜껑(*least upper bound, LUB*)이 항상 그 집합안에 있다.

연속함수(*continuous function*)

CPO D_1 에서 CPO D_2 로 가는 함수 $f : D_1 \rightarrow D_2$ 가 연속 함수란, 체인(*chain*)을 체인으로 보내면서, D_1 의 체인(*chain*)의 LUB를 보전해 주는 함수:

$$\forall \text{chain } X \subseteq D_1. f(\bigsqcup X) = \bigsqcup_{x \in X} f(x).$$

CPO에서 CPO로 가는 연속함수 f 는 항상 최소 고정점 $\text{fix } f$ 이 유일하게 있고, 그것은

$$\perp \sqcup f(\perp) \sqcup f(f(\perp)) \sqcup \dots = \bigsqcup_{i \in \mathbb{N}} f^i(\perp)$$

이다.(왜?)

CPO 만들기

$$\begin{array}{l}
 S \\
 CPO \rightarrow S_{\perp} \\
 \quad | \quad CPO \times CPO \\
 \quad | \quad CPO + CPO \quad | \quad CPO \oplus CPO \\
 \quad | \quad CPO \rightarrow CPO
 \end{array}$$

- 올려붙인 집합(*lifted set*) S_{\perp} 은 CPO
- CPO와 CPO의 데카르트 곱(*Cartesian product*)도 CPO
- CPO와 CPO의 출신을 기억하는 합(*separated sum*)도 CPO
- CPO에서 CPO로 가는 연속함수들의 집합도 CPO

충분히 풍부하다.

심지어는 CPO사이에서 위의 방법으로 정의된 CPO 방정식 (domain equation)의 해도 항상 존재한다.

예를들어,

$$\begin{aligned} Store &= Loc \rightarrow (Int + Loc + Cmd) \\ Cmd &= Store \rightarrow Store \end{aligned}$$

를 만족하는 CPO $Store$ 는 항상 존재한다.

또,

$$D = D \rightarrow D$$

를 만족하는 CPO D 도 존재. 이 사실이, 왜 CPO가 모든 프로그램의 의미를 담을 수 있는 그릇인지를 말해준다.

$$\mathbb{Z}_{\perp} = \mathbb{Z} \cup \{\perp\}$$

- \mathbb{Z} 원소들 사이의 순서는 없고
- 순서는 오직 \perp 과 \mathbb{Z} 사이에만: $\forall x \in \mathbb{Z}. \perp \sqsubseteq x$.
- 모든 체인은 유한. 따라서,

CPO D_1 과 D_2 의 데카르트 곱(*Cartesian product*)

$$D_1 \times D_2 = \{\langle x, y \rangle \mid x \in D_1, y \in D_2\}$$

- 원소들의 순서는 조립식(*component-wise*)

$$\langle x, y \rangle \sqsubseteq \langle x', y' \rangle \text{ iff } x \sqsubseteq_{D_1} x' \wedge y \sqsubseteq_{D_2} y'.$$

- 왜 CPO?

CPO D_1 과 D_2 의 합

$$D_1 + D_2 = \{\langle x, 1 \rangle \mid x \in D_1\} \cup \{\langle x, 2 \rangle \mid x \in D_2\} \cup \{\perp\}$$

- 원소들의 순서는 출신별로

$$\begin{aligned} \langle x, 1 \rangle \sqsubseteq \langle x', 1 \rangle & \text{ iff } x \sqsubseteq_{D_1} x' \\ \langle x, 2 \rangle \sqsubseteq \langle x', 2 \rangle & \text{ iff } x \sqsubseteq_{D_2} x' \end{aligned}$$

- 왜 CPO?

함수를 표기하는 방법

“ λx .함수 내부” 로 (x 는 함수의 인자)

- 1을 더하는 연속함수: “ $\lambda x.x + 1$ ”
- 함수 f 를 2에 적용: “ $f2$ ”, “ $f(2)$ ”

CPO D_1 에서 D_2 로 가는 모든 연속함수의 집합 $D_1 \rightarrow D_2$ 은
CPO

- 연속함수들 순서는 조립식(*point-wise*):

$$f \sqsubseteq g \text{ iff } \forall x \in D_1. f(x) \sqsubseteq_{D_2} g(x).$$

- 왜 CPO?

- 의미방정식에서 사용하는 모든 물건들은 CPO의 원소
- 연산자들은 모두 CPO에서 CPO로 가는 연속함수들
- 따라서 모든 의미방정식의 해는 항상 어떤 연속함수 \mathcal{F} 의 고정점:

$$X = \mathcal{F}(X)$$

- 위의 방정식의 해는 \mathcal{F} 의 최소 고정점 $fix\mathcal{F}$ 로 정의:

$$fix\mathcal{F} = \sqcup_{i \in \mathbb{N}} \mathcal{F}^i(\perp)$$

그래서 while-문의 의미는?

$M \in Memory$	$= Var \rightarrow Value$	연속함수 CPO
$z \in Value$	$= \mathbb{Z}_\perp$	올려붙인 CPO
$x \in Var$	$= ProgramVariable_\perp$	올려붙인 CPO

명령문 C 의 의미

연속함수 $\llbracket C \rrbracket \in Memory \rightarrow Memory$ 연속함수 CPO

계산식 E 의 의미

연속함수 $\llbracket E \rrbracket \in Memory \rightarrow \mathbb{Z}_\perp$ 연속함수 CPO

while-문의 의미방정식은

$$\begin{aligned} & \llbracket \text{while } E \ C \rrbracket M \\ & = \text{if } \llbracket E \rrbracket M \neq 0 \text{ then } \llbracket \text{while } E \ C \rrbracket (\llbracket C \rrbracket M) \text{ else } M \end{aligned}$$

다시 쓰면,

$$\begin{aligned} & \llbracket \text{while } E \ C \rrbracket = \\ & \lambda M. \text{if } \llbracket E \rrbracket M \neq 0 \text{ then } \llbracket \text{while } E \ C \rrbracket (\llbracket C \rrbracket M) \text{ else } M. \end{aligned}$$

즉, while-문의 의미 $\llbracket \text{while } E \ C \rrbracket$ 는 연속 함수

$$\begin{aligned} & \lambda X. (\lambda M. \text{if } \llbracket E \rrbracket M \neq 0 \text{ then } X(\llbracket C \rrbracket M) \text{ else } M) \\ & \in (\text{Memory} \rightarrow \text{Memory}) \rightarrow (\text{Memory} \rightarrow \text{Memory}) \end{aligned}$$

의 최소 고정점

$$\begin{aligned}
& \llbracket \text{while } E \ C \rrbracket \\
&= \text{fix } \mathcal{F} \in \text{Memory} \rightarrow \text{Memory} \\
&= \text{fix}(\lambda X. (\lambda M. \text{if } \llbracket E \rrbracket M \neq 0 \text{ then } X(\llbracket C \rrbracket M) \text{ else } M))
\end{aligned}$$

정의되었는가? $\llbracket \text{while } E \ C \rrbracket$ 는 $\llbracket E \rrbracket$ 와 $\llbracket C \rrbracket$ 를 가지고 조립.