SNU 4541.664A Program Analysis

Note 2

Prof. Kwangkeun Yi

Seoul National University

계획

문법구조

요약된 혹은 구체적인

의미구조

무엇 의미구조(*denotational semantics*)

프로그램 집합 = 귀납적인 정의

프로그램의 생김새

- 나무구조를 갖춘 2차원 모습

정수식

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

명령형 언어 프로그램

$$
\begin{array}{rcl}
C & \rightarrow & \texttt{skip} \\
& | & x \texttt{ := } E \\
& | & \texttt{if } E\ C\ C \\
& | & C \texttt{ ; } C
\end{array}
$$

군더더기 없이

$$
\begin{array}{rcll}
C & \to & \& & \\
 & | & = x\ E & \\
 & | & ?\ E\ C\ C & \\
 & | & ;\ C\ C & \\
 & & & \\
E & \to & n & (n \in \mathbb{Z}) \\
 & | & +\ E\ E & \\
 & | & -\ E &
\end{array}
$$

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

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

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

$$\begin{array}{rcll} E & \rightarrow & n & (n \in \mathbb{Z}) \\ & | & E \texttt{ + } E & \\ & | & \texttt{ - } E & \end{array}$$

- 답변 가능

$$\begin{array}{rcll} E & \rightarrow & n & (n \in \mathbb{Z}) \\ & | & E \texttt{ + } E & \\ & | & \texttt{ - } F & \\ F & \rightarrow & n & \\ & | & \texttt{(} E \texttt{)} & \end{array}$$

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 있슴
- 우리 의도에 맞는 의미구조 방식을 선택

무엇 의미구조(*denotational semantics*)

- 프로그램의 의미: 전통적인 수학의 세계에서, 의미하는 바 무엇인지를 드러냄
- 단도직입, 결국 무엇인지를 드러내는 의미구조
- (별명) 조립식 의미구조(*compositional semantics*): 전체의 의미 $=$ 부품들의 의미로 조립
- (별명) 고정점 의미구조(*fixpoint semantics*): 함수의 고정점으로 정의

$$
\begin{array}{rcll}
C & \to & \texttt{skip} & \\
 & | & x \texttt{ := } E & \\
 & | & \texttt{if } E\ C\ C & \\
 & | & C \texttt{ ; } C & \\
E & \to & n & (n \in \mathbb{Z}) \\
 & | & x & \\
 & | & E \texttt{ + } E & \\
 & | & \texttt{- } E &
\end{array}
$$

- 명령어 C의 의미 $=$ 함수: 메모리에서 메모리로
- 메모리 $=$ 함수: 주소에서 값으로

의미공간(*semantic domain*)

의미를 정의할 때 사용하는 물건들의 집합

$$
\begin{array}{rcl}
M \in Memory & = & Var \to Value \\
z \in Value & = & \mathbb{Z} \\
x \in Var & = & Program\,Variable \\
\text{명령문 } C\text{의 의미 } [\![C]\!] & \in & Memory \to Memory \\
\text{계산식 } E\text{의 의미 } [\![E]\!] & \in & Memory \to \mathbb{Z}
\end{array}
$$

$$
\begin{aligned}
[\![\texttt{skip}]\!]\, M &= M \\
[\![x \texttt{ := } E]\!]\, M &= M\{x \mapsto [\![E]\!]\, M\} \\
[\![\texttt{if } E\ C_1\ C_2]\!]\, M &= \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![C]\!]M\ \mathit{else}\ [\![C_2]\!]M \\
[\![C_1 \texttt{ ; } C_2]\!]\, M &= [\![C_2]\!]\,([\![C_1]\!]\, M) \\
[\![n]\!]\, M &= n \\
[\![E_1 \texttt{ + } E_2]\!]\, M &= ([\![E_1]\!]\, M) + ([\![E_2]\!]\, M) \\
[\![\texttt{- } E]\!]\, M &= -([\![E]\!]\, M)
\end{aligned}
$$

잘 정의되었는가? 조립식인가?

조립식이 안되는 경우

$$\texttt{while}\ E\ C$$

의 의미

$$\begin{aligned}&[\![\texttt{while}\ E\ C]\!]M\\ &= \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M\end{aligned}$$

조립식인가?

$$[\![\texttt{while}\ E\ C]\!]M$$
$$= \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M$$

- 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 C의 의미 $[\![C]\!]$에 대한 의미방정식은 항상 다음과 같고

$$
\begin{array}{rcll}
[\![C]\!] & = & \mathcal{F}([\![C]\!]) & \\
\text{여기서 } [\![C]\!] & \in & D & \text{(어떤 CPO)} \\
\text{그리고 } \mathcal{F} & \in & D \to D & (D\text{에서 } D\text{로의 연속함수들의 CPO})
\end{array}
$$

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

CPO (의미공간)

프로그램의 수학적인(궁극적인) 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*), 최소고정점(*least fixpoint*)

- CPO A, CPO B. 연속함수 $f : A \to B$란, A의 체인(*chain*)의 최소윗뚜껑(*least upper bound*)를 보전해주는 함수:

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

- 연속함수 $f : A \to A$의 최소고정점 $\mathit{fix} f$은

$$\mathit{fix} f = \bigsqcup_{i \in \mathbb{N}} f^i(\bot) = \bigsqcap \{x \mid f(x) \sqsubseteq x\}.$$

  - 연속함수는 단조함수
  - 단조함수 f: $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$
  - 팽창함수 f: $x \sqsubseteq f(x)$

소풍: 일반적으로, 고정점(*fixpoint*) 예

"Computer science is full of fixpoints."

소풍: 일반적으로, 고정점(*fixpoint*) 예

"Computer science is full of fixpoints."

- `N = {0} ∪ {n+1|n ∈ N}`
  $= lfp\lambda X.\{0\} \cup \{n+1 | n \in X\}$

소풍: 일반적으로, 고정점(*fixpoint*) 예

"Computer science is full of fixpoints."

- `N = {0} ∪ {n+1|n ∈ N}`
  $= lfp\lambda X.\{0\} \cup \{n+1 | n \in X\}$
- `list = {nil} ∪ {(0,l)|l∈ list}`
  $= lfp\lambda X.\{\texttt{nil}\} \cup \{(0, l) | l \in X\}$

소풍: 일반적으로, 고정점(*fixpoint*) 예

"Computer science is full of fixpoints."

- `N = {0} ∪ {n+1|n ∈ N}`
  $= lfp\lambda X.\{0\} \cup \{n+1 | n \in X\}$
- `list = {nil} ∪ {(0,l)|l∈ list}`
  $= lfp\lambda X.\{\texttt{nil}\} \cup \{(0, l) | l \in X\}$
- `reach(N) = N ∪ reach(next(N))`
  $= lfp\lambda f.(\lambda N.N \cup f(next(N)))$

소풍: 일반적으로, 고정점(*fixpoint*) 예

"Computer science is full of fixpoints."

- `N = {0} ∪ {n+1|n ∈ N}`
  $= lfp\lambda X.\{0\} \cup \{n+1 | n \in X\}$
- `list = {nil} ∪ {(0,l)|l∈ list}`
  $= lfp\lambda X.\{\texttt{nil}\} \cup \{(0, l) | l \in X\}$
- `reach(N) = N ∪ reach(next(N))`
  $= lfp\lambda f.(\lambda N.N \cup f(next(N)))$
- `sort(A) = if sorted(A) then A else sort(exch(A))`
  $= lfp\lambda f.(\lambda A.sorted(A)?\ A :\ f(exch(A)))$

소풍: 일반적으로, 고정점(*fixpoint*) 예

"Computer science is full of fixpoints."

- `N = {0} ∪ {n+1|n ∈ N}`

  $= lfp\lambda X.\{0\} \cup \{n+1 | n \in X\}$
- `list = {nil} ∪ {(0,l)|l∈ list}`

  $= lfp\lambda X.\{\texttt{nil}\} \cup \{(0, l) | l \in X\}$
- `reach(N) = N ∪ reach(next(N))`

  $= lfp\lambda f.(\lambda N.N \cup f(next(N)))$
- `sort(A) = if sorted(A) then A else sort(exch(A))`

  $= lfp\lambda f.(\lambda A.sorted(A)?\ A :\ f(exch(A)))$
- `fac(n) = if n=0 then 1 else n*fac(n-1)`

  $= lfp\lambda f.(\lambda n.n = 0?\ 1 :\ n \times f(n-1))$

CPO 만들기

$$
\begin{array}{rcl}
S & & \text{set} \\
CPO & \rightarrow & S_{\perp} \\
 & | & CPO \times CPO \\
 & | & CPO + CPO \mid CPO \oplus CPO \\
 & | & CPO \rightarrow CPO
\end{array}
$$

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

$S_\perp = S \cup \{\perp\}$

- S 원소들 사이의 순서는 없고
- 순서는 오직 $\perp$과 S 사이에만: $\forall x \in S. \perp \sqsubseteq x$.
- 모든 체인은 유한. 따라서 CPO.

CPO D_1과 D_2의 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*)

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

- 따라서 CPO. (왜?)

CPO D_1과 D_2의 합

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

- 원소들의 순서는 출신별로

$$\langle x, 1\rangle \sqsubseteq \langle x', 1\rangle \quad \text{iff} \quad x \sqsubseteq_{D_1} x'$$
$$\langle x, 2\rangle \sqsubseteq \langle x', 2\rangle \quad \text{iff} \quad x \sqsubseteq_{D_2} x'$$

- 따라서 CPO. (왜?)

CPO D_1에서 D_2로 가는 모든 연속함수의 집합 $D_1 \to D_2$ 은 CPO

- 연속함수들 순서는 조립식(*point-wise*):

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

- 따라서 CPO. (왜?)

의미구조에서 연속함수 표기법

$$\lambda x. \cdots x \cdots$$

(x는 함수의 인자).

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

CPO는 모든 프로그래밍언어의 의미공간으로 충분

CPO사이에서 위의 방법으로 정의된 CPO 방정식(domain equation)의 해가 항상 존재.

- 예:

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

  를 만족하는 CPO $Store$는 항상 존재

- 참고:

$$D = D \to D$$

  를 만족하는 CPO D도 존재(Dana Scott's domain theory). 이 사실이, 왜 CPO가 모든 프로그램의 의미를 담을 수 있는 그릇인지를 말해준다. (왜?)

- 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 최소고정점 $lfp\mathcal{F}$로 정의:

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

그래서 while-문의 의미는?

$$
\begin{array}{rcll}
M \in Memory & = & Var \to Value & \text{연속함수 CPO} \\
z \in Value & = & \mathbb{Z}_\perp & \text{올려붙인 CPO} \\
x \in Var & = & ProgramVariable_\perp & \text{올려붙인 CPO}
\end{array}
$$

명령문 C의 의미

연속함수 $[\![C]\!] \in Memory \to Memory$ 연속함수 CPO

계산식 E의 의미

연속함수 $[\![E]\!] \in Memory \to \mathbb{Z}_\perp$ 연속함수 CPO

while-문의 의미방정식은

$$\begin{aligned}&[\![\texttt{while}\ E\ C]\!]M\\&= \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M\end{aligned}$$

다시 쓰면,

$$\begin{aligned}&[\![\texttt{while}\ E\ C]\!] =\\&\lambda M.\mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M.\end{aligned}$$

즉, while-문의 의미 $[\![\texttt{while}\ E\ C]\!]$는 연속 함수

$$\begin{aligned}&\lambda X.(\lambda M.\mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ X([\![C]\!]M)\ \mathit{else}\ M)\\&\in (\mathit{Memory} \to \mathit{Memroy}) \to (\mathit{Memory} \to \mathit{Memroy})\end{aligned}$$

의 최소고정점

$$
\begin{aligned}
&[\![\texttt{while}\ E\ C]\!] \\
&= lfp\mathcal{F} \in Memory \to Memory \\
&= lfp(\lambda X.(\lambda M.if\ [\![E]\!]M \neq 0\ then\ X([\![C]\!]M)\ else\ M))
\end{aligned}
$$

정의되었는가? $[\![\texttt{while}\ E\ C]\!]$는 $[\![E]\!]$와 $[\![C]\!]$를 가지고 조립.

$$
\begin{aligned}
M \in Memory &= Var \to Value \\
z \in Value &= \mathbb{Z}_\perp \\
x \in Var &= ProgramVariable_\perp \\
[\![C]\!] &\in Memory \to Memory \\
[\![E]\!] &\in Memory \to Value
\end{aligned}
$$

$$
\begin{aligned}
[\![\texttt{skip}]\!] &= \lambda M.M \\
[\![x \texttt{ := } E]\!] &= \lambda M.M\{x \mapsto [\![E]\!]\, M\} \\
[\![\texttt{if } E\ C_1\ C_2]\!] &= \lambda M.\mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![C]\!]M\ \mathit{else}\ [\![C_2]\!]M \\
[\![C_1 \texttt{ ; } C_2]\!] &= \lambda M.[\![C_2]\!]\,([\![C_1]\!]\, M) \\
[\![\texttt{while } E\ C]\!] &= \mathit{lfp}(\lambda X.(\lambda M.\mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ X([\![C]\!]M)\ \mathit{else}\ M)) \\
[\![n]\!] &= \lambda M.n \\
[\![x]\!] &= \lambda M.M(x) \\
[\![E_1 \texttt{ + } E_2]\!] &= \lambda M.([\![E_1]\!]\, M) + ([\![E_2]\!]\, M) \\
[\![\texttt{- } E]\!]\, M &= \lambda M. -([\![E]\!]\, M)
\end{aligned}
$$

프로그램 C의 의미: 조립식

- C는 부품들로 조립됨

$$C = AB$$

- $[\![C]\!]$도 부품의미들로 조립됨

$$\begin{aligned} [\![AB]\!] &= \cdots [\![A]\!] \cdots [\![B]\!] \cdots \\ &= \cdots 1 + 2 \cdots \end{aligned}$$

프로그램 C의 의미: 고정점

예: $C = AB$

$$[\![AB]\!] = [\![A]\!] + [\![B]\!]$$

여기서

$$\begin{aligned} [\![A]\!] &= 1 \\ [\![B]\!] &= \cdots [\![B]\!] \cdots \end{aligned}$$

즉,

$$\begin{aligned} ([\![A]\!], [\![B]\!]) &= (1, \cdots [\![B]\!] \cdots) \\ &= \mathcal{F}([\![A]\!], [\![B]\!]) \end{aligned}$$

프로그램 C의 의미: 고정점

$[\![C]\!]$	$=$	$\mathcal{F}([\![C]\!])$	C의 의미 방정식
$[\![C]\!]$	$\in$	D	CPO
$\mathcal{F}$	$\in$	$D \to D$	연속함수
일 때,			
$[\![C]\!]$	$=$	$lfp\mathcal{F}$	C의 의미
	$=$	$\bigsqcup_{i \in \mathbb{N}}(\mathcal{F}^i \bot)$	
	$=$	$\bigsqcap\{x \in D \mid \mathcal{F}(x) \sqsubseteq x\}$	