Chapter 4. Denotational Semantics

Semantics With Applications

木谷有沙

お茶の水女子大学　人間文化創成科学研究科　理学専攻

November 27, 2008

はじめに

- 前章までで扱った operational semantics では、「プログラムがどのように実行されるか」に関心を置いていた。
- 本章で扱う denotational semantics ではそれよりも、「プログラムの実行によってもたらされる結果」、すなわち初期状態と最終状態の間にある関係に着目する。
- そのため、各 syntactic category （統語範疇？）に対し semantic function （意味関数？）を定義することを考える。
- semantic function とは、構文の各要素を、その実行結果を表現する数学的オブジェクト（たいていは関数）にマッピングするもの。

Denotational Semantics の特徴

semantics function は compositional に定義される。つまり、

- syntactic category の各基本要素 に対し semantic clause （意味項？）がある
- 複合要素に対しては、その構成要素に適用される semantic function によって、 semantic clause が定義される。

１章で定義した $\mathcal{A}, \mathcal{B}$ は denotational definition.
$\mathcal{S}_{ns}, \mathcal{S}_{sos}$ は compositional でないため、denotational definition ではない。

4.1 Direct style semantics : specification

定義

- S の実行によってもたらされる結果とは、state の変化。よって、S の意味を以下のような state 上の部分関数として定義する。

  $\mathcal{S}_{ds}$: Stm → (State ↪ State)

- 表 4.1 に $While$ の denotational semantics がまとめられている。
  - cond と FIX という補助関数を用いている。（詳細は後で説明）

表 4.1 の説明 1 (skip, sequencing)

- `skip` の項で使われている関数 id は State 上の恒等関数
  $\mathcal{S}_{ds}[\texttt{skip}]s = s$
- $S_1; S_2$ の実行結果は、S_1 の実行結果と S_2 の実行結果の合成。したがって関数合成の定義より、S_1 か S_2 のどちらかが undefined だと、$S_1; S_2$ の結果も undefined.(p.86 下部参照)

表 4.1 の説明 2 (conditional)

- **条件文では、以下のような補助関数** cond **が用いられている。**

  $$\text{cond} : (\text{State} \to \text{T}) \times (\text{State} \hookrightarrow \text{State}) \times (\text{State} \hookrightarrow \text{State}) \to (\text{State} \hookrightarrow \text{State})$$

  $$\text{cond}(p, g_1, g_2)s = \begin{cases} g_1\, s & \text{if } p\, s = \text{tt} \\ g_2\, s & \text{if } p\, s = \text{ff} \end{cases}$$

  選ばれたほうの分岐の結果が undefined **であれば、条件文全体も** undefined.
  $\mathcal{B}[b]$ **が全体関数なので、条件部分が** undef **になることはない。**

表 4.1 の説明 3 (while)

- while b do S の結果は次と等しくなるはず。

  if b then (S; while b do S) else skip

- これを $\mathcal{S}_{ds}$ の既に定義済みの部分を使って書くと

  $\mathcal{S}_{ds}$[while b do S] =

  cond($\mathcal{B}[b]$, $\mathcal{S}_{ds}$[while b do S] $\circ$ $\mathcal{S}_{ds}$, id) ...(*)

- (*) は $\mathcal{S}_{ds}$ が compositional でないためこのまま定義として使うことはできないが、$\mathcal{S}_{ds}$[while b do S] が次のような関数 F の不動点 (fixed point) であるということを表している。

  $F\ g$ = cond($\mathcal{B}[b]$, $g \circ \mathcal{S}_{ds}$, id)

  - つまり $\mathcal{S}_{ds}$[while b do S] = F ($\mathcal{S}_{ds}$[while b do S])

  これを使って表 4.1 の compositional な定義を得ている。

- 補助関数 FIX は不動点を返す

  FIX : ((State $\hookrightarrow$ State) $\rightarrow$ (State $\hookrightarrow$ State))

  $\rightarrow$ (State $\hookrightarrow$ State)

Example 4.1

`while` $\neg(\mathrm{x} = 0)$ `do skip`

これに対応する関数 F' **は次のように定義される。**

$$(F'\ g)\ s = \begin{cases} g\ s & \text{if } s\ \mathrm{x} \neq 0 \\ s & \text{if } s\ \mathrm{x} = 0 \end{cases}$$

このとき、関数 g_1 **を**

$$g_1\ s = \begin{cases} \underline{\text{undef}} & \text{if } s\ \mathrm{x} \neq 0 \\ s & \text{if } s\ \mathrm{x} = 0 \end{cases}$$

と定義すると、

$$(F'\ g_1)\ s = \begin{cases} g_1\ s & \text{if } s\ \mathrm{x} \neq 0 \\ s & \text{if } s\ \mathrm{x} = 0 \end{cases} = \begin{cases} \underline{\text{undef}} & \text{if } s\ \mathrm{x} \neq 0 \\ s & \text{if } s\ \mathrm{x} = 0 \end{cases} = g_1\ s$$

より、 g_1 **は** F' **の不動点である。**

Example 4.1

同様に、以下のような g_2 を考えてみる。
$g_2\ s$ = undef for all s

$s' = x$ となる s' において、
$(F'\ g_2)\ s' = s'$ となる一方で $g_2\ s'$ =undef となるため
g_2 は F' の不動点ではない。

この定義の問題点

じつはこれだけでは $S_{ds}[\texttt{while } b \texttt{ do } S]$ を満足に定義できない。
二つの問題点がある。

- 二つ以上不動点をもつ関数がある
- 不動点を一つも持たない関数がある

次節でこれらの問題の解決する。

Exercise 4.2, 4.3

- Exercise 4.2

  `while ¬(x = 0) do x := x-1`

  これに対応する関数 F を求めよ。また、 g_1 から g_5 のうち F の不動点であるものはどれか。

- Exercise 4.3

  `while ¬(x = 0) do (y := y*x; x:= x-1)`

  これに対応する関数 F と、その不動点を二つ以上求めよ。

Requirements on fixed theory

前述の二つの問題点を解決するために、次のようなことを行う。

- 不動点を選ぶ際に必要となる条件を課し、それらを満たす不動点は高々一つであることを示す。
- While のプログラムの関数 F は必ず上の条件を満たす不動点を持つことを示す。

３つのケース

条件を選びだすために、state s_0 からの `while` b do S の実行結果を考えてみる。すると次の三つが考えられる。

A 終了する。

B local にループ。

C global にループ。

各ケースにおいて、関数 F とその不動点について何が言えるかをを調べていく。

The case A

while b do S が終了する。つまり、次のような state $s_1, s_2, ...s_n$ がある。

$$\mathcal{B}[b]\, s_i = \begin{cases} \text{tt} & \text{if i} < \text{n} \\ \text{ff} & \text{if i} = \text{n} \end{cases}$$

かつ

$$\mathcal{S}_{ds}\, s_i = s_{i+1} \quad \text{for i} < \text{n}$$

これを満たす例 (x は正の値を持つ)

```
while 0 ≤ x do x := x - 1
```

The case A

g_o が F の不動点 ($F\ g_0 = g_0$ を満たす) だとすると i<n のとき

$$g_0\ s_i = (F\ g_0)\ s_i$$
$$= \text{cond}(\mathcal{B}[b],\ g_o \circ \mathcal{S}_{ds}[S],\ id)\ s_i$$
$$= g_o\ (\mathcal{S}_{ds}[S]\ s_i)$$
$$= g_o\ s_{i+1}$$

i=n のとき

$$g_0\ s_n = (F\ g_0)\ s_n$$
$$= \text{cond}(\mathcal{B}[b],\ g_o \circ \mathcal{S}_{ds}[S],\ id)\ s_n$$
$$= \text{id}\ s_n$$
$$= s_n$$

よって関数 F の全ての不動点 g_0 が以下の式を満たす。

$$g_0\ s_0 = s_n$$

ケース A では、不動点を選ぶ際に必要な条件は見つからなかった。

The case B

while b do S がローカルにループする。つまり、次のような state $s_1, s_2, ...s_n$ がある。

$\mathcal{B}[b]\, s_i$ = tt　　for i ≤ n

かつ

$$\mathcal{S}_{ds}[S]\, s_i = \begin{cases} s_{i+1} & \text{for i} < \text{n} \\ \underline{\text{undef}} & \text{for i} = \text{n} \end{cases}$$

これを満たす例 (x は正の値を持つ)

```
while 0 ≤ x do (if x=0 then (while true then do skip)
                else x := x - 1)
```

The case B

g_o が F の不動点 ($F\ g_0 = g_0$ を満たす) だとすると i<n のとき

$$g_0\ s_i = g_0\ s_{i+1}$$

i=n のとき

$$\begin{aligned} g_0\ s_n &= (F\ g_0)\ s_n \\ &= \mathrm{cond}(\mathcal{B}[b],\ g_o \circ \mathcal{S}_{ds}[S],\ id)\ s_n \\ &= (g_o \circ \mathcal{S}_{ds}[S])\ s_n \\ &= \underline{\mathrm{undef}} \end{aligned}$$

よって関数 F の全ての不動点 g_0 は以下の式を満たす。

$$g_0\ s_0 = \underline{\mathrm{undef}}$$

ケース B でも、不動点を選ぶ際に必要な条件は見つからなかった。

The case C

`while` b `do` S がグローバルにループする。つまり、次のような state $s_1, s_2, \ldots$ がある。

$\mathcal{B}[b]\, s_i = \mathrm{tt}$ for all i

かつ

$\mathcal{S}_{ds}[S]\, s_i = s_{i+1}$ for all i

これを満たす例 (x は 0 でない)

```
while ¬(x=0) do skip
```

The case C

g_o が F の不動点 ($F\ g_0 = g_0$ を満たす) だとすると、i≥0 をみたす全ての i に対し

$$g_0\ s_i = g_0\ s_{i+1}$$

よって

$$g_0\ s_0 = g_0\ s_i \text{ for all i}$$

となり、 $g_0\ s_o$ が定まらない。これは F が複数の異なる不動点を持つということ。
これは特に驚くべきことではなく、例えば Example 4.1 は

$$(F'\ g)\ s = \begin{cases} g\ s & \text{if s } x \neq 0 \\ s & \text{if s } x = 0 \end{cases}$$

という関数 F' を持っていたが、「$s\ x = 0$ のとき $g\ s = s$」さえ満たせば、どんな State ↪ State の部分関数 g も F' の不動点となれる。

The case C

とはいえ経験上、ループを区別するために、次のものが欲しくなる。

$$\mathcal{S}_{ds}[\texttt{while}\neg(\texttt{x=0})\ \texttt{do}\ \texttt{skip}]\, s_o = \begin{cases} \underline{\text{undef}} & \text{if } s_0\ \texttt{x} \neq 0 \\ s_0 & \text{if } s_0\ \texttt{x} = 0 \end{cases}$$

よって、より好ましい F' の不動点とは以下のように定義された関数 g_0 である。

$$g_0\ s = \begin{cases} \underline{\text{undef}} & \text{if } s\ \texttt{x} \neq 0 \\ s & \text{if } s\ \texttt{x} = 0 \end{cases}$$

この g_0 と F' の他の不動点 g' との違いは、$g_0\ s = s'$ であれば $g'\ s = s'$ が成り立つが、逆は成り立たないところである。

Generalizing

ここまでの話を一般化すると、欲しい不動点 FIX F は次の条件を満たす State $\hookrightarrow$ State の部分関数 g_0 ということになる。

- g_0 は F の不動点。つまり $F\ g_0 = g_0$
- g を F の他の不動点としたとき、全ての s, s' に対し、$g_0\ s = s'$ ならば $g\ s = s'$
  (注意：$g_0\ s =$ undef ならば $g\ s$ は何でも良い)

Exercise 4.4, 4.5

- Exercise 4.4
  Exercise 4.2 **で求めた不動点のうち、先の条件を満たす不動点はどれか。**
- Exercise 4.5
  Exercise 4.3 **で求めた関数に対し、先の条件を満たす不動点を求めよ。**

4.2 Fixed point theory

4.2 Fixed point theory

条件を満たす不動点が存在することを保証する前に、もう少し形式的に FIX F **の条件を定義し直そう。**
まず FIX F **が他の全ての不動点とその結果を共有しているという条件から定式化する。**
そのために、 State $\hookrightarrow$ State **の部分関数に順序** $\sqsubseteq$ **を定義する。**

$\sqsubseteq$ の定義

任意の s, s' に対して
$g_1\ s = s'$ ならば $g_2\ s = s'$
が成り立つとき、
$g_1 \sqsubseteq g_2$

Example 4.6, Exercise 4.7, 4.8

- Example 4.6
  $\sqsubseteq$ **の例。**
- Exercise 4.7
  $\sqsubseteq$ **を使って書かれた条件を満たす関数を定義する問題。**
- Exercise 4.8
  $\sqsubseteq$ **については次のようにも書けます。**
  $g_1 \sqsubseteq g_2$ if and only if graph$(g_1) \subseteq$ graph(g_2)
  これが本当に正しいことを示しなさい。
  (**関数** graph **については** p.214 **参照**)

partial order

集合 D と D 上の関係 $\sqsubseteq_D$ が以下を満たす時、組 $(D, \sqsubseteq_D)$ を半順序集合という。

- $d \sqsubseteq_D d$ (reflexivity)
- $d_1 \sqsubseteq_D d_2$ and $d_2 \sqsubseteq_D d_3$ **ならば** $d_1 \sqsubseteq_D d_3$ (transitivity)
- $d_1 \sqsubseteq_D d_2$ and $d_2 \sqsubseteq_D d_1$ **ならば** $d_1 = d_2$ (anti-symmetry)

関係 $\sqsubseteq_D$ を D 上の 半順序と言い、しばしば添字の D **を省いて** $\sqsubseteq$ **と書かれる。**
場合によっては $\sqsupseteq$ **も使う。**

least element

$d \sqsubseteq d'$ for all d' of D
を満たす D の要素 d を、D の least element **と呼び、「何の情報も含んでいない」と言う。**

- Fact 4.9
  半順序集合 $(D, \sqsubseteq)$ が least element d **を持っているとき、d は一意に定まる。**
  - **背理法で証明できる**

  D **の** least element **を $\bot_D$ もしくは $\bot$ と表記する。**

Example 4.10, Exercise 4.11, 4.12

- Example 4.10
  S **は、次を満たす空でない集合。**
  $\mathcal{P} = \{K | K \subseteq S\}$
  このとき $(\mathcal{P}, \subseteq)$ **が半順序集合であることの証明と、その** least element
- Exercise 4.11
  $(\mathcal{P}, \supseteq)$ **が半順序集合であることを証明し、その** least element **を求めよ。**
- Exercise 4.12
  S **は、次を満たす空でない集合。**
  $\mathcal{P}_{fin} = \{K | K$ is finite $andK \subseteq S\}$
  このとき、$(\mathcal{P}_{fin}, \subseteq)$, $(\mathcal{P}_{fin}, \supseteq)$ **が半順序集合であることを証明せよ。また、これらは** least element **をもつか答えよ。**

Lemma 4.13

(State $\hookrightarrow$ State, $\sqsubseteq$) **は半順序集合である。**

$\bot\ s = \underline{\text{undef}}$ for all s

を満たす部分関数 $\bot$: State $\hookrightarrow$ State **は、**
State $\hookrightarrow$ State **の** least element **である。**

- **証明**
  - $\sqsubseteq$ **が半順序の三つの条件を満たす事を示す**
  - $\bot$ **が** State $\hookrightarrow$ State **の** least element **であることを示す**

more precise statement of the requirements to FIX F

部分関数上の順序を導入して、FIX F **のより正確な定義を得られるようになった。**

- FIX F **は** F **の不動点。つまり** F (FIX F) = FIX F
- FIX F **は** F **の** least fixed point. **つまり** $F\ g = g$ **ならば** FIX $F \sqsubseteq g$

Exercise 4.14, 4.15

- Exercise 4.14
  Fact4.9 **と同様に、**F **が** least fixed point g_0 **を持つならば** g_0 **は一意であることを示せ。**
- Exercise 4.15
  Exercise 4.2, 4.3 **の関数の** least fixed point **を求めよ。また、**Exercise 4.4, 4.5 **と比較せよ。**

Least upper bound

- $(D, \sqsubseteq)$ を半順序集合、Y をその部分集合としたとき、次のような D の要素 d を Y の upper bound という。
  $\forall d' \in Y. \quad d' \sqsubseteq d$
- 以下を満たす Y の upper bound d を least upper bond という。
  d' が Y の upper bound $\Rightarrow d \sqsubseteq d'$
- Exercise 4.16
  Fact. 4.9 と同様に、Y が least upper bound d を持つならば d は一意であることを示せ。
- Y が least upper bound をもつならば、それを $\bigsqcup Y$ と表記する。

chain

- **次を満たす Y を** chain **と呼ぶ。**
  $\forall d_1, d_2 \in Y. \quad d_1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$
- Example 4.17
  Example 4.10 **の** $(\mathcal{P}(\{a, b, c\}), \subseteq)$ **の部分集合で** chain **になる例とならない例と、それぞれの** least upper bound**。**
- Exercise 4.18
  $(\mathcal{P}(S), \subseteq)$ **の空でない部分集合は必ず** least upper bound **を持つ事を証明せよ。**$(\mathcal{P}(S), \supseteq)$ **についても同様に証明せよ。**
- Exercise 4.19
  $(\mathcal{P}_{fin}(S), \subseteq)$ **が、**least upper bound **がない** chain **を持つような** S **の例を挙げよ。**

Example 4.20, Exercise 4.21, 4.22

- Example 4.20
- Exercise 4.21
- Exercise 4.22

chain complete

- **半順序集合 $(D, \sqsubseteq)$ の全ての** chain Y **に対し常に** $\bigsqcup Y$ **が存在するとき、半順序集合 $(D, \sqsubseteq)$ を** chain complete (ccpo) **であるという。**
  D の全ての部分集合 Y に $\bigsqcup Y$ **が存在するときは、**complete lattice(**完備束**) **であるという。**
- Example 4.23

Fact 4.24, Lemma 4.25

- Fact 4.24 $(D, \sqsubseteq)$ が ccpo であれば、$\bot = \bigsqcup \emptyset$ で与えられる least element $\bot$ を持つ
- Lemma 4.25
  (State $\hookrightarrow$ State, $\sqsubseteq$) は ccpo.
  Y の least upper bound $\bigsqcup Y$ は
  graph($\bigsqcup Y$) = $\bigcup$\{ graph(g) $|g \in Y$\}
  で与えられる。つまり、
  $(\bigsqcup Y)s = s' \Leftrightarrow g\,s = s'$ for some $g \in Y$

monotone

- $(D, \sqsubseteq)$, $(D', \sqsubseteq')$ は ccpo であるとし、関数 $f : D \to D'$ を考える。すべての d_1, d_2 に対し
  $$d_1 \sqsubseteq d_2 \Rightarrow f\ d_1 \sqsubseteq' f\ d_2$$
  となるとき、f は monotone（単調）であるという。
- Example 4.26
- Exercise 4.27
- Exercise 4.28

properties of monotone functions

- Fact 4.29
  $(D, \sqsubseteq)$, $(D', \sqsubseteq')$, $(D'', \sqsubseteq'')$ は ccpo であるとし、関数 $f : D \to D'$, $f' : D' \to D''$ は monotone であるとする。このとき、$f' \circ f : D \to D''$ も monotone.
- Lemma 4.30
  $(D, \sqsubseteq)$, $(D', \sqsubseteq')$ は ccpo であるとし、関数 $f : D \to D'$ は monotone であるとする。このとき、Y が D での chain であれば、$\{f\ d | d \in Y\}$ は D' での chain であり、さらに
  $$\bigsqcup' \{f\ d | d \in Y\} \sqsubseteq' f(\bigsqcup Y)$$
- 一般に、$\bigsqcup' \{f\ d | d \in Y\} = f(\bigsqcup Y)$ であるとは期待できない。
- Example 4.31
  $\bigsqcup' \{f\ d | d \in Y\} = f(\bigsqcup Y)$ でない例

continuous functions

- ccpo **である** $(D, \sqsubseteq)$ **と** $(D', \sqsubseteq')$ **上の関数** $f : D \to D'$ **が** monotone **で、かつ** Y **の全ての空でない** chain **について**
  $\bigsqcup'\{f\ d | d \in Y\} = f(\bigsqcup Y)$
  であるとき、この関数 f **を** continuous function **（連続関数）という。**
  空の chain **に対しても** $\bigsqcup'\{f\ d | d \in Y\} = f(\bigsqcup Y)$ **であれば、** f **は** strict **であるという。**
- Example 4.32
- Exercise 4.33
- Exercise 4.34

property of continuous functions

- Lemma 4.35
  $(D, \sqsubseteq)$, $(D', \sqsubseteq')$, $(D'', \sqsubseteq'')$ は ccpo であるとし、関数 $f : D \to D'$, $f' : D' \to D''$ は continuous であるとする。このとき、$f' \circ f : D \to D''$ も continuous.
- Exercise 4.36

define the required fixed point operator FIX !

- Theorem 4.37
  関数 $f : D \to D$ **は、**$(D, \sqsubseteq)$ **上で** least element $\perp$ **を持つ** continuous function **であるとする。このとき、**
  FIX $f = \bigsqcup\{f^n \perp | n \geq 0\}$
  は D **の要素であり、**f **の** least fixed point.
- Example 4.38
- Exercise 4.39
- Exercise 4.40

Fixed Point Theory

least fixed point **の存在を示すためにしたことのまとめ。**

1. chain complete partially ordered sets (ccpo) **に限定する。**
2. continuous function **に限定する。**
3. ccpo **上の** continuous function **は必ず** least fixed point **を持つことを示す。**

Exercise 4.41, 4.42

- Exercise 4.41
- Exercise 4.42

まとめ

- Denotational semantics で While を定義した。
- 不動点を求める関数 FIX を使う事で、compositionally に定義している。
- 関数 FIX で求める不動点の満たすべき条件について考え、FIX を定義した。

おしまい

おわりだよ！
来週につづく。