第1章　プログラミング言語の意味とは

1.1　はじめに

プログラミング言語の仕様を定める時には、その言語の “構文” — つまり、どのような記号の列が正当なプログラムとして受け入れられるのか — とともに、その言語の “意味” を定める必要がある。

プログラミング言語にはさまざまな構成要素がある。例えば、ほとんどのプログラミング言語には条件分岐・繰り返しなどの制御構造（C 言語では **`if`**〜**`else`** 文, **`while`** 文, **`for`** 文, **`do`**〜**`while`** 文など）がある。C++や Java には、例外処理のための **`try`**〜**`catch`** 文もあるし、Prolog にはユニフィケーション（単一化）・バックトラッキング（後戻り）などの仕組みがある。関数型言語 Haskell には遅延評価、オブジェクト指向言語にはクラス・インターフェースなど、やはり特有の仕組みが存在する。

これらの構成要素の “意味” については、現在のところ、「“条件式” が成り立つ間、“文” の実行を繰り返す」のように日本語・英語などの自然言語によって記述するのが普通である。しかし、自然言語による記述では、曖昧な点が多く、例えば次のような疑問が生じた時に厳密に議論することができない。

- コンパイラの正当性 — コンパイラは、本当に仕様書に定められた通りの動作をする目的プログラムを生成しているか？
- プログラムの同値性 — プログラムのある部分を、（おそらく効率の良い）別の形に書き直した時に、書き直しの前後でプログラムの意味は本当に同値か？

そのため、プログラムの “意味” を形式的に記述するために、さまざまな方法がこれまでに考えられて来た。

1.2　さまざまな意味論

プログラミング言語の意味論は大きく分けて、操作的意味論・公理的意味論・表示的意味論の 3 つに分類される。（ただし、この分類はある程度便宜的なもので、それぞれの境界がはっきりしているわけではない。）

操作的意味論は、仮想的な抽象機械を定義し、プログラムの意味をその抽象機械の内部状態の変化として記述しようという方法である。

公理的意味論は、プログラムの意味を公理と推論規則により与えようとする方法である。**ホーア論理**などがその代表的なものである。

表示的意味論は、集合や関数のような数学的概念を用いて、プログラムの意味を記述しようとする方法である。

この講義では、主に表示的意味論に関連する事柄を紹介する。

プログラミング言語の “構文” の記述については偉大な先人の努力により、BNF のような記法や yacc などの LALR パーサジェネレータのような道具が考え出され、仕様の記述や処理系の作成に欠かせないものとして、完全に定着している。

しかし、現在のところ、プログラミング言語の “意味” の記述については、現実のプログラミング言語の仕様書に標準的に使用されるような記法や道具が確立されているというところまでは到っていない。

それでも、プログラミング言語の意味論について、これまで蓄積された知見は、上に挙げたようなさまざまなプログラミング言語の構成要素についての理解を深めるために必要不可欠なものとなっている。本講義ではこのような “意味論” の簡単なところを “つまみ食い” して行く予定である。

1.3 この講義の概要

ラムダ計算（λ-calculus）は表示的意味論を展開するのに利用される計算体系である。 ラムダ計算は “関数” に関するもっとも単純な記法・体系であり、また、もっとも単純なプログラミング言語と考えることもできる。

本講義では、このラムダ計算を用いて、より複雑なプログラミング言語の構成要素の “意味” を記述していくことにする。

本来は表示的意味論では、ラムダ計算に加えて、D_∞ や $P\omega$ と呼ばれる領域（domain）に関する理論を展開する。これは、ラムダ計算では見かけ上異なる式が、プログラムとしては実は同じ意味を持つような場合があり、数学的に議論を行うためには充分でないからである。しかし、本講義では領域に関する理論については触れない。興味のある人は各自で参考文献を調べて欲しい。

ラムダ計算はシンプルでエレガントな体系ではあるが、これですべてを記述しようとすると、記述量がたいへん多くなり手に負えなくなる。そこで、ラムダ計算に基づくプログラミング言語である関数型言語 Haskell を紹介する。Haskell は、基本的にはラムダ計算にいくつかの構文上の糖衣（syntax sugar — 本質的ではないが便利な記法）を追加したものである。そのため、その気になれば、Haskell のプログラムは簡単にラムダ計算の式に書き直すことができる。

プログラミング言語のさまざまな構成要素の意味を記述する上でさまざまな概念を導入するが、なかでも**接続**（**継続**ともいう、continuation）の概念は制御構造・例外処理・後戻りなど、さまざまな事柄を説明するために、ひじょうに重要な役割を果たす。また、接続はコンパイラの設計の際にも用いられる。接続の概念を理解することは、プログラミング言語の意味論を理解する上でひじょうに重要である。

また、最後にオブジェクト指向の意味についても取り上げる予定である。

この章の参考文献

[1] 中島玲二「数理情報学入門 — スコット・プログラム理論」朝倉書店, 1982, ISBN4-254-11413-3

[2] 武市正人「プログラミング言語 — 岩波講座ソフトウェア科学 4」岩波書店, 1994, ISBN4-00-010344-X