プログラム言語論

亀山幸義

筑波大学コンピュータサイエンス専攻

筑波大学 情報科学類講義, 2009 年 4 月 13 日

Who am I ?

- 亀山幸義の基本データ
  - 筑波大学 大学院システム情報工学研究科 コンピュータサイエンス専攻
  - 総合研究棟 B-1008 (10 階)
  - 本講義 URL: `http://logic.cs.tsukuba.ac.jp/~kam/plm/`
  - E-mail: `kam` あっと `cs.tsukuba.ac.jp`
- 研究テーマ
  - プログラム理論・プログラム言語論
    - 関数型プログラム言語、型システム など。
  - ソフトウェア検証・システム検証
    - コンピュータによる定理証明、モデル検査による検証。

プログラム言語とは？

- プログラムを読み書きするため，人工的に作られた言語．
- なぜ必要か？
  - 現代コンピュータ == von Neumann Machine (Turing Machine)
  - プログラム内蔵方式
- 何のために？
  - 人間のため vs コンピュータ (ハードウェア) のため
  - 書きやすさ/理解しやすさ/保守のしやすさ vs 実行性能の高さ
- この授業は，主として，「人間のためのプログラム言語」(高級言語) を扱う．(cf. 低級 (low-level) 言語．)

プログラム言語論とは？

- プログラムについて，もっともらしく論じる学問．(?)
- 1 つ 1 つのプログラムについて語るのではなく，プログラム言語 (全体) について語る学問．(?)
- 1 つ 1 つのプログラム言語について語るのではなく，種々のプログラム言語に共通する (あるいは相違する) 概念について語る学問．
- そんなことは可能だろうか？そんなことに意味があるのだろうか？

この授業の目的は？

- プログラムを設計・実装するのが，容易になる，楽しくなる．
- 新しいプログラム言語を設計・実装するのが，容易になる，楽しくなる．

授業の目的については，シラバスも参考にしてください．

質問

「インタープリタ (I) とコンパイラ (C) はどう違うのですか？」

- I は実行が遅くて，C は速い．(?)
- I はプログラムを 1 行ずつ実行して，C は一気に全部実行する．(?)
- C 言語には C しかないし，shell には I しかない．(?)
- I は，プログラムを受け取って，それを実行して答えを返すプログラムであるが，C は，プログラムを受け取って，「それを実行して答えを返すプログラム」を返すプログラムである．
- 気分の違い (違いはない)．(?)

解釈系 (interpreter)

インタープリタは，言語 S で書かれたプログラムを解釈して実行するプログラム (それ自身は言語 L で書かれている) である．

- shell インタープリタ: S=shell, L=機械語
- perl インタープリタ: S=perl, L=機械語
- JVM 実行系: S=Java byte code, L=機械語
- あるインタープリタ: S=Scheme, L=Scheme
- もちろん，L は元々は，C なり Java なりで書かれていて，コンパイラを使って機械語に変換したものであろう．
- S=L のとき，meta-circular interpreter という．

$$[\![p]\!]_S(\vec{x}) = [\![\mathtt{int}]\!]_L(p, \vec{x})$$

翻訳系 (compiler) あるいは変換系 (translater)

コンパイラは，言語 S で書かれたプログラムを，言語 T で書かれたプログラムに翻訳 (変換) する (それ自身は言語 L で書かれている) プログラムである．

- C コンパイラ: S=C 言語, T=機械語, L=機械語
- Java コンパイラ: S=Java, T=byte code, L=機械語
- Java native コンパイラ: S=Java, T=機械語, L=機械語
- クロスコンパイラ: S=C 言語, T=PowerPC の機械語, L=Pentium の機械語
- ある変換器: S=Java, T=Scheme, L=Scheme

$$[\![p]\!]_S(\vec{x}) = [\![[\![\mathtt{comp}]\!]_L(p)]\!]_T(\vec{x})$$

部分評価 (プログラム特化) の話-1

プログラム mix:

$$[\![[\![mix]\!]_L(p,\vec{x})]\!]_L(\vec{y}) = [\![p]\!]_L(\vec{x},\vec{y})$$

プログラム p への入力 x, y のうち、x だけが、あらかじめわかっている時、「y をもらうと $[\![p]\!](x,y)$ を返す」プログラムを作成する。
部分評価 (partial evaluation) あるいは プログラム特化 (program specialization) という。
例. 「x の n 乗」を計算するプログラムがあるとき、$n = 3$ のときに (異なる x に対して) 頻繁に動かしたいなら、その n に特化した高速プログラムが欲しい。

部分評価 (プログラム特化) の話-2

二村射影 (Futamura Projections) [Futamura 1971]
mix の定義式で、$p = \mathtt{int}$, $\vec{x} = p$ とすると：

$$[\![[\![\mathtt{mix}]\!]_L(\mathtt{int},p)]\!]_L(\vec{y}) = [\![\mathtt{int}]\!]_L(p,\vec{y})$$

右辺は、インタープリタの定義式より、$[\![p]\!]_S(\vec{y})$ に等しい。
よって、

$$[\![[\![\mathtt{mix}]\!]_L(\mathtt{int},p)]\!]_L(\vec{y}) = [\![p]\!]_S(\vec{y})$$

結局、$[\![\mathtt{mix}]\!]_L(\mathtt{int},p)$ は、p と同じ動作をする (二村の第 1 射影)。

$$[\![\mathtt{mix}]\!]_L(\mathtt{int},p) = p$$

部分評価 (プログラム特化) の話-3

mix の定義式で、$p = \mathtt{mix}$, $\vec{x} = \mathtt{int}$, $\vec{y} = p$ とすると：

$$[\![[\![\mathtt{mix}]\!]_L(\mathtt{mix},\mathtt{int})]\!]_L(\vec{p}) = [\![\mathtt{mix}]\!]_L(\mathtt{int},\vec{p})$$

となり、右辺は第 1 射影なので、(言語 S で書かれている)p と、同じ動作をするプログラム (言語 L で書かれている) である。つまり、言語 S から言語 L へのコンパイラ (言語 L で書かれている) ができた。(二村の第 2 射影)

$$\mathtt{comp} = [\![\mathtt{mix}]\!]_L(\mathtt{mix},\mathtt{int})$$

部分評価 (プログラム特化) の話-4

同様に、mix の定義式で、$p = \mathtt{mix}$, $\vec{x} = \mathtt{mix}$ とすると以下が得られる。

$$[\![[\![\mathtt{mix}]\!]_L(\mathtt{mix},\mathtt{mix})]\!]_L(\vec{int}) = \mathtt{comp}$$

これより、$\mathtt{cogen} = [\![\mathtt{mix}]\!]_L(\mathtt{mix},\mathtt{mix})$ は「インタープリタが与えられると、コンパイラを生成するプログラム」(コンパイラジェネレータ) であるといえる。(二村の第 3 射影)

部分評価 (プログラム特化) の話-5

こんな机上の空論のような計算をして何が面白いか？

- 実際に、効率よい「コンパイラジェネレータ (cogen)」を生成する mix が、(Scheme 言語や C 言語に対して) 実装された!
- 人手で作成した cogen より効率が良い (ことがある)!
- 理論と実装の結びつきの好例。

ちなみに、、、現在は再び「手動で cogen を書く」ことへの関心が高まっている。
⇒マルチステージプログラミング言語の研究: たとえば、Walid Taha, "Gentle Introduction to Multi-Stage Programming" などを参照。

結局のところ，プログラム言語論とは？

- プログラム言語の中だけでなく，外からも眺めて，あーだ，こーだ，論じる学問.
- 単に，プログラム言語を比較するのではなく，新しいプログラム言語をどんどん作りだしてしまう考えてしまう学問.
  (cf. 「比較言語論」ではない.)
- とはいえ，1 つ 1 つのプログラム，1 つ 1 つのプログラム言語の設計や実装に役立つ貴重な情報を与えてくれる.
- Next Programming Language (Next Ruby) を考えたい人向け (!?)

この授業の参考書

John C. Mitchell, "Concepts in Programming Languages", Cambridge University Press, 2003.

amazon では 6,940 円! (2009/4/12 現在)

上記図書の誤植修正が、Mitchell 自身のホームページに置かれている。
`http://theory.stanford.edu/~jcm/books.html`

授業計画

週	内容
1	イントロダクション、インタープリタとコンパイラ、高級言語と低級言語 (第 1 章)、構文と意味、ラムダ計算、手続き型言語と宣言型言語 (第 4 章)
2　5	関数と手続き、ブロック構造と変数スコープ、評価順序、メモリ管理 (第 7 章)、データ構造と型システム (第 6 章)、制御構造 (第 8 章)
6　8	モジュラリティ、モジュール、抽象データ型 (第 9 章)、オブジェクト指向、クラスと継承 (第 10 章)
9　10	関数プログラミング (第 5 章)、論理プログラミング (第 15 章)、並列言語 (第 14 章)、授業のまとめ。

「第 n 章」は、Mitchell の参考書における章番号。ただし、授業内容と参考書は完全には対応していない。

手続き型言語 (命令型言語) vs 宣言型言語

- 命令型言語 imperative language: プログラムは、「命令の列 (手順、手続き)」である。順序付けられた一連の命令を順番に実行していくことにより、計算が行われる。
  - 例: C, Fortran, COBOL, ...
- 宣言型言語 declarative language: プログラムは、(関数や論理式などの) 宣言である。計算の手順は書かず、計算の意味を記述する。
  - 関数型言語の例: Lisp, Scheme, ML (Standard ML, OCaml), Haskell
  - 論理型言語の例: Prolog

命令型言語の例: miniC の構文 (更にやり直し)

BNF を多少崩した形がよく使われる。

$$
\begin{aligned}
p &::= s \mid p\ ;\ s \\
x &::= \ldots \\
c &::= \ldots \\
s, t &::= x := e \mid \mathit{while}\ e\ \mathit{do}\ s \mid \mathit{begin}\ p\ \mathit{end} \mid \mathit{if}\ e\ \mathit{then}\ s\ \mathit{else}\ t \\
e, f &::= x \mid c \mid e + f
\end{aligned}
$$

BNF は、文脈自由文法 (context-free grammar) を与える簡潔な記法。(⇒「オートマトンと形式言語」を参照)
しかし、実際のプログラム言語の構文は、文脈自由でないことが多い。
例えば、C 言語では、宣言されない変数を使ってはいけない。

宣言型言語の (うちの関数型言語の) 例: miniML の構文

$$
\begin{aligned}
x &::= \ldots \\
c &::= \ldots \\
e, f &::= x \mid c \mid e + f \mid \lambda x.e \mid e\ f \mid \texttt{let}\ x = e\ \texttt{in}\ f
\end{aligned}
$$

$\lambda x.e$ ラムダ式:

- $f(x) = e$ となる関数 f のことを $\lambda x.e$ と書く。
- $(\lambda x.e)(x) = e$ が成立する。
- $(\lambda x.e)(f)$ はどう定めたらよいか?

$\texttt{let}\ x = e\ \texttt{in}\ f$ let 式:

- $x = e$ として f を計算する。

構文 (syntax) と意味 (semantics)

- 言語の「構文」:
  - 日本語の場合、日本語の文字から構成される任意の文字列のうち、「日本語の文になっているもの」を定める規則が構文規則。
  - プログラム言語の場合も同様。
- 言語の「意味」:
  - 日本語の文の意味は、何だろう?
  - プログラムの意味は何か?
    - プログラムを計算すると、どういう結果が得られるか。
  - プログラムの意味をどうやって与えればよいか?
    - 次回のお楽しみ。

成績評価

- 期末試験をやります。(100 点満点で採点)
- 出席を取るかわりに short quiz を出すか、演習をやります。これが出席代わりになります。
- 2 回以下の欠席はカウントしませんが、それより多く (正当理由なく) 欠席すると、5 点ずつ減点します。
- short quiz の解答や演習については、「良いものを加点する」という精神で行ないます。
- 演習は、学類の計算機室で行ないます。

授業のホームページ

授業のスライドや配布資料などは、以下の場所に置きます.

`http://logic.cs.tsukuba.ac.jp/~kam/plm/`

質問等は下記へどうぞ.
電子メイル: `kam` あっと `cs.tsukuba.ac.jp`
直接訪問: 総合研究棟 B-1008