アルゴリズムとデータ構造入門

1.手続きによる抽象の構築

1.1 プログラムの構築

奥 乃 博
大学院情報学研究科知能情報学専攻
知能メディア講座 音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/05/IntroAlgDs/
okuno@i.kyoto-u.ac.jp
TA: 海尻 聡 (mod(学籍番号, 3) ≡0) 10号館4階奥乃1研
TA: 藤原 弘将(mod(学籍番号, 3) ≡1) 10号館4階奥乃1研
TA: 村瀬 昌満(mod(学籍番号, 3) ≡2) 10号館4階奥乃2研

教科書 Structure and Interpretation of Computer Programs (SICP)

1. 世界中のComputer Scienceのトップレベルの教科書
2. 1回生後期で前半を
3. 2回生前期で後半を(湯淺先生)
4. MIT Press 提供オンライン版(無料)
5. Emacs Texinfo 形式(無料)
6. 日本語訳(邦訳・訳あり)約4.5K円

教科書は持っているものとして進めます.

参考書・Scheme 処理系

1. ジョン・ベントリー(小林健一郎訳):『珠玉のプログラミング—本質を見抜いたアルゴリズムとデータ構造』(ピアソン)
2. 世界中にSICPのサイト・コースウェア等あり
3. 宿題は自分でやること(答えを見ない)
4. TUT Scheme(湯淺研開発・メディアセンタ)
5. 10月5日公開(WinO, Cygwin, Linux最新版×)
6. 他の処理系は自己責任で使用のこと
   1. MIT-Scheme-6001
   2. Chez Scheme, ・・・

成績評価

1. 試験　80%
2. 必修課題　20%
   - ① 計算機の歴史についてのレポート
   - ② 宿題で出した練習問題．提出はレポート箱（前日17時締切）
   - ③ 図形言語レポート（プログラムはメイルで提出）
3. 随意課題提出による“＋α”
   - ① 第2章までのすべての練習問題
   - ② Fixed-Point探索過程のSchemeによる可視化
   - ③ アルゴリズムのSchemeによる可視化
   - ④ これはすごいという抽象化を使ったSchemeプログラム（線形計画法，整数論，群論，組合せ論，古典力学，パズル解放，ゲーム）
   - ⑤ Lego Mindstorm用Lisp　XS　を使った自律ロボット
   - ⑥ 図形言語で circle-limit （難しい）
   - ⑦ 他の学生の支援

4

Computer Science を極めるには

1. 『計算機プログラムの構造と解釈』（SICP）を読む
2. 『コンピュータの構成と設計－ハードウェアとソフトウェアのインタフェース』（上下）を読む．
3. 『珠玉のプログラミング』を読む
4. 情報処理技術者試験を受ける
   - ① 2回生までに**基礎情報処理技術者試験**に合格
   - ② 3回生までに**ソフトウェア開発技術者試験**に合格
5. 自分の腕を磨く
   - ① ACM大学プログラミングコンテストに出場
   - ② Lego Mindstorm，・・・　で遊ぶ

5

10月4日・本日のメニュー

1. 京都大学での計算機の歴史
2. 課題1：計算機の歴史についてレポート（A4・5枚以上）
   - 期限は10月31日17時
   - 10号館1階レポート提出箱
3. SICP第1章　手続きによる抽象化(abstraction)
4. SICP 1.1　プログラムの要素
5. TUT Scheme の説明は10月18日　平石君（湯淺研D1)

6

計算機の歴史についての資料

1. 『誰がどうやってコンピュータを作ったのか？』星野力著、共立出版
2. 『A Computer Perspective、計算機創造の軌跡』アスキー出版局
3. 『コンピュータの構成と設計　ーハードウェアとソフトウェアのインタフェース』（第1章）David A. Patterson・John L. Hennesy, 日経BP社
4. Internetで Computer Museum が多数あり。
5. 出展は明記すること。
6. 自分なりの視点でレポートをまとめること。
7. 丸写しは、無効ないし減点。再提出もありうる。

7

京都大学での計算機の歴史

- **KDC-I ⇒ HITAC102**
- **KT-Pilot ⇒ TOSBAC3400**
- **三値論理計算機**
- **ADENA-1, ADENA-2**
- **QA-1, QA-2**
- **音声タイプライタ**

8

KDC-Iのお披露目（1960年10月21日）

湯川秀樹博士に説明するのは萩原宏現名誉教授

KDC-Iのコンソール：　右側にあるのは、紙テープ読取装置と紙テープ。当時、プログラムは機械語で開発され、紙テープに穿孔された。

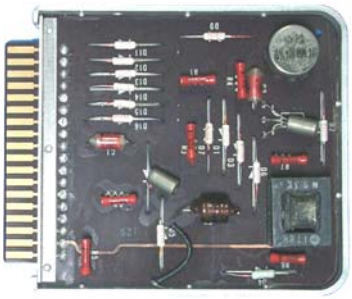
9

Kyoto Daigaku Digital Computer, KDC-1

形式	プログラム内蔵型10進電子計算機
回路方式	トランジスタおよびダイオードを使用した同期制御方式（クロック周波数約230 kC）内訳：ゲルマニュームトランジスタ約8500個、ダイオード約5万個）
記憶装置	中速磁気ドラム（4200語うち4000語平均待時間5ms、200語平均待時間1. 25ms）磁気コア（50語、磁気テープ記憶装置・本体間のバッファ，高速記憶装置（待時間0. 05ms）としても使用可）　磁気テープ（1台当たり約35万語、2台）
語の形式	数値語：固定小数点（符号1桁＋絶対値11桁），浮動小数点（符号1桁＋絶対値9桁＋指数部2桁） 命令語：操作部3桁、アドレス部4桁，インデックス部2桁，ブレークポイント部1桁　文字語：１文字2桁，数字，アルファベット，記号等63種
命令の数	基本命令95種（実質的には130以上）
アドレス方式	１・１/２アドレス方式（インデックスレジスタ：3個）
演算時間	固定小数点：　加減算0.5ms乗算5.8ms除算6.5ms, 浮動小数点：加減算1.3ms乗算5.2ms除算5.8ms
入出力装置	光電式テープリーダ200字／秒, 万能入出力装置480字／分
消費電力	本体約1.5 kW

10

KDC-1 の論理パッケージ



右端中央にあるのがゲルマニュームトランジスタ。1個の価格が当時の大卒初任給よりも高かったそうです。

KDC-Iの設計のために論理回路テスターも開発され、論理配線の自動化も1959年暮れに行われた。

11

KDC-1 のサブルーティン

Vol. 1 No. 4　　KDC-I のプログラミングについて　　187

P2-701	PRMXA	Print c(DM) (fx.pt., multi-length) *TM, TK (TK) TM	X3-023	RADEG	radian to degree *TM (←) TM
P2-702	PRMXB	Print c(DM) (fx.pt., multi-length) *TM, TK (TK) TM	RDIN	HH	initial input program *HH (←) HH
P3-010	PRREG	Print contents of registers *HH (HN) TM	TABL-001	TAB	table of functions (fl.pt.) *TK, TM (TT) TK, TM
P3-901	PRS	Print sentence *HH MN TM	TABL-002	TABX	table of functions (fx.pt.) *TK, TM MN) TK, TM
X1-201	RNDM	random numbers (fx.pt.) *HN, YD HH HN	TABL-003	TABD	table of functions (fl.pt., db.pr.) *TM, TK SK) TM, TK
X2-001	EVER	Everett's interpolation (fl.pt.) *TM MN) TM	TABL-004	TABDX	table of functions (fx.pt., db.pr.) *TM, TK (←) TM
X3-021	DERA 1	degree to radian *TM, SK JN) TM	DEBG-001	MDI	memory dump *SI, SY (←) SI
X3-022	DERA 2	degree to radian *TM (←) TM			

(1) fl.pt.: floating point, fx.pt.: fixed point, db.pr.: double precision, AC: accumulator, UA: upper accumulator, DM: drum, c( ): content of

(2) * に続く文字は作者（改訂者）検査者の氏名を示す：
YD：出口，HH：萬屋，SI：伊与田，YK：金沢，SK：加藤，TK：清野，TM：馬嶋，MN：長尾，JN：中西，HN：西堀，EN：西尾，TS：浜井，TT：津田，SY：久米．

ハードウェアの製造と同時に進められた機械語によるソフトウェアの開発。1960年10月10日現在の検査済みのライブラリリスト。紙テープに格納。

12

マイクロプログラミングのKT-Pilot

- 1961年，数理工学科萩原宏教授（現名誉教授）と東京芝浦電気が共同研究
- 我が国最初の非同期マイクロプログラム方式を採用したコンピュータ
- マイクロプログラムの書き換え方式や高速化の方式を確立
- 1963年末に，東芝が、これをベースにTOSBAC-3400を商用化．当時のベストセラー

13

三値論理計算機の開発

- 数理工学教室三根久教授、長谷川利治助教授(お二人は現名誉教授)、島田良作学振研究員（当時）が1970年に開発
- 9桁の3進四則演算装置
- 論理演算に +1, 0, -1 の三値
- 0, 1, 2 の3値より、演算方式が簡単。

14

データ並列計算機 ADENAの開発

- 数理工学教室野木達夫教授(現名誉教授）が開発
- 行列の演算は行に対する演算と列に対する演算から構成 ⇒ データ並列
- 転送能力の高いネットワーク
- ADENA1(1980)
- ADENA2(1989)
  松下電器
  64bit CPU×256
  1GFLOPS

15

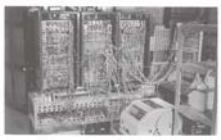
並列処理計算機 QA-1, 2 の開発

- 情報工学教室富田眞治助手（現教授）が1974年～1977年に開発
- VLIW (Very Long Instruction Word)
- 論理演算に +1, 0, -1 の三値
- 160 bit長、4個の演算器、4個のメモリアクセス、1個の分岐操作を指
- Real-Time Animation実現

16

音声タイプライタの開発

- 電気工学教室坂井利之現名誉教授が開発
- 『あいうえお』の認識、
- 零交差波分析
- 右はプロトタイプ
- 日本電気と共同試作(1960年）



17

聖徳太子ロボット

- 奥乃が科学技術振興事業団と共同開発
- ロボットが自分の耳で混合音を聞き分ける

SIG（姉）

Pino（弟）

SIG2（妹）

18

10月4日・本日のメニュー

1. 京都大学での計算機の歴史
2. 課題1：計算機の歴史についてレポート（A4・5枚以上）
   - 期限は10月31日17時
   - 10号館1階レポート提出箱
3. SICP第1章　手続きによる抽象化(abstraction)
4. SICP 1.1　プログラムの要素
5. TUT Scheme の説明は10月18日　平石君（湯淺研D1)

19

Lispによるプログラミング

1. 計算モデル：再帰方程式（recursion equation) という論理表現とその推論方式
2. Lispの方言ーscheme, CtCL, …
3. Lispの処理系(Implementation)
   - 解釈系（Interpreter): プログラムをそのまま解釈し、実行
   - コンパイラ（Compiler)：プログラムを機械語へ変換し、機械語をランタイムシステムの下で実行
4. 実装（implement)
5. 手続き（procedure)であるプログラムとデータが同じ形

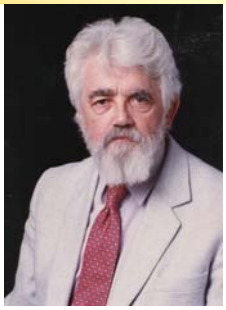
20

Lisp言語

1. John McCarthyが1959年に設計・開発
   http://www-formal.stanford.edu/jmc/recursive.html
2. Fortran 言語についで2番目に古い言語
3. 種々の方言・実装あり、Schemeもその一つ
   MacLisp, Interlisp, TAO, Kyoto Common Lisp, …
4. 今日のオブジェクト指向などさまざまなアイデアを創出してきた「原言語」
5. 統合的プログラミング環境が提供
6. TRON（Disney）最初のCGによる映画
7. Plutoの軌道がChaoticの計算による証明-Galileo 以来の open problemの解決

21

John McCarthy, Prof. Emeritus, SU

弟子さん：佐藤雅彦教授（情報），林晋教授（文学部），居候：奥乃

22

“TRON” Disney 映画 (1982)

A masterpiece of breakthrough CGI ingenuity, Disney celebrates the 20th anniversary of TRON, a dazzling film at the flashpoint of a continuing revolution in its genre. This special collector's edition showcases an epic adventure inside a brave new world where the action is measured in microseconds.

When Flynn (Jeff Bridges) hacks the mainframe of his ex-employer to prove his work was stolen by another executive, he finds himself on a much bigger adventure. Beamed inside by a power-hungry master control program, he joins computer gladiators on a deadly game grid, complete with high-velocity "light cycles" and Tron (Bruce Boxleitner), a specialized security program. Together, they fight the ultimate battle with the MCP to decide the fate of both the electronic world and the real world!

23

数学とコンピュータサイエンスの違い

1. 宣言的知識（Declarative Knowledge）

“What is true”という知識

$\sqrt{x}$ *is the* y *such that* $y^2 = x$ *and* $y \geq 0$

2. 規範的知識（命令的，Imperative Knowledge）

“How to”という知識

$x = 2$ *に対する* $\sqrt{x}$ *の値を求めるには*

予測 (guess)	相棒は商で求まる	予測とその相棒の平均値で改善
1	2/1 = 2	(1 + 2)/2 = 1.5
1.5	2/1.5 = 1.333	(1.5 + 1.333)/2 = 1.4167
1.4167	2/1.4167 = 1.4118	(1.4167 + 1.4118) = 1.4142
1.4142	2/1.4142 =	

25

“How to” 知識を概念化する

1. 手続き(procedure)
   所望の値を求めるステップ系列の概念 ー recipe のようなもの
2. 計算プロセス(computational process)
   具体的に計算機の中で実行されるステップの展開 ー 実際の調理
3. データ(data) ー 材料
4. プログラム(program)= 手続き ＋ データ
   計算プロセスはプログラム指示によりデータを操作
5. 指示誤り:バグ(虫, bug)、スリップ(glitches)
6. 間違い修正:虫とり(debug)
7. 言語(language) あるいはプログラミング言語
   計算プロセスを記述ために使用
   - Vocabulary (語彙)
   - Syntax (構文)ー 複合式を構築するためのルール
   - Semantics (意味)ー構成子に意味を付与するためのルール

26

“How to” 知識・概念化のポイント

複雑さとの戦いー単純なデータと手続き

- Vocabulary (語彙)
- Syntax (構文)
- Semantics (意味)

1. 手続き抽象化(procedure abstraction)
2. データ抽象化(data abstraction)

27

1.1 言語の要素

- **expressions (式)**
  primitives(基本式)とcombinations(合成式)で構成
- **means of abstraction(抽象化法)**
- **Creating procedure objects (手続きの作成法)**
- **Viewing the rules of evaluation from a computational perspective (計算という観点からの評価法)**

28

Scheme (Lisp) の基本

- 式(expression)は単純なものから構築.
- ほぼすべての式は，値(value)を持つ.
- 式は評価(evaluate)されて値を返す.
- すべての値には型(type)がある.

29

言語構文・構築子(language constructs)

- Primitives (基本式)
- Means of combination (合成法)
- Means of abstraction (抽象化法)

30

基本式(primitives)

- Self-evaluating primitives(評価すると自分自身の値)
  - Numbers (数): 38, 3.80, 1.4141, 2.3e-4, 3/5
  - Strings (文字列): “moji”, “a”,
  - Booleans (論理式): #t, #f
- Built-in procedure (組込み手続き)
  - 基本オブジェクト(primitive objects)の処理
  - Numbers(数): +, -, *, /, <, =, <=, ...
  - Strings(文字列): string-length, string=?
  - Boolenas(論理式): and, or, not, xor, nand,
- Names for built-in procedures
  - + ⇒ +という組込み手続き

31

合成法(combinations)

- Primitives を使って式を合成
- (+ 3 5)
  (演算子　引数 . . .)

- 評価法
  - 部分式(subexpressions)の評価し値を得る.
  - 演算子(operator)に引数(arguments)を適用.
  - procedure application （手続き適用)という

32

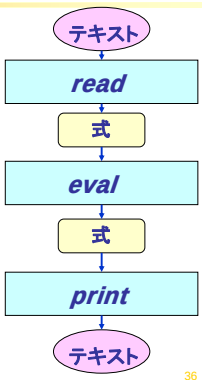
名前と値との連携

- define を使って式を合成
- (define foo (+ 3 5))
  foo の値は 8
- (define bar +)
  bar は + と同じ手続き
- (bar 3 5)

33

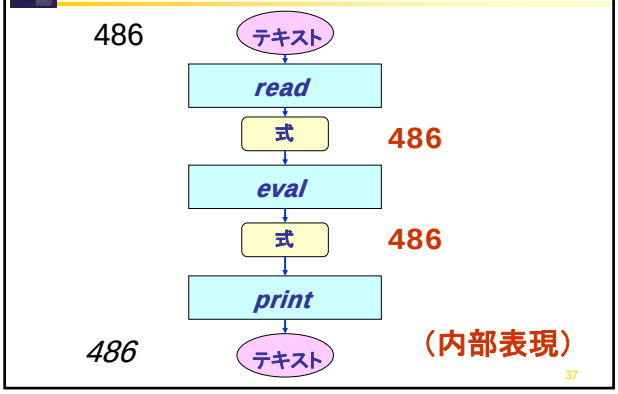
1.1.1 Expressions(式)

- An interpreter （解釈系):
  read-evaluate-print ループ (REPL) を繰り返す
- read: テキスト表現の式を内部表現に変換
- evaluate:式を評価
- print: 評価結果の式を内部表現からテキスト表現に変換, 出力



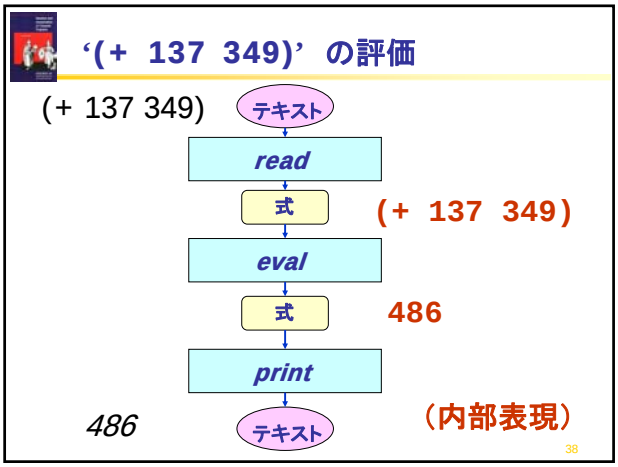
36

‘486’ の評価



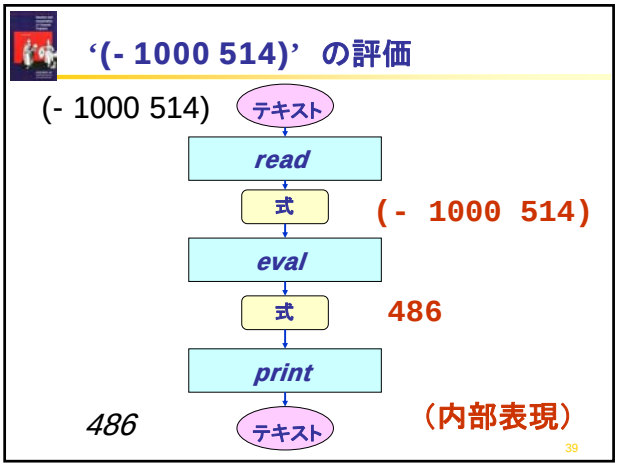
37

‘(+ 137 349)’ の評価



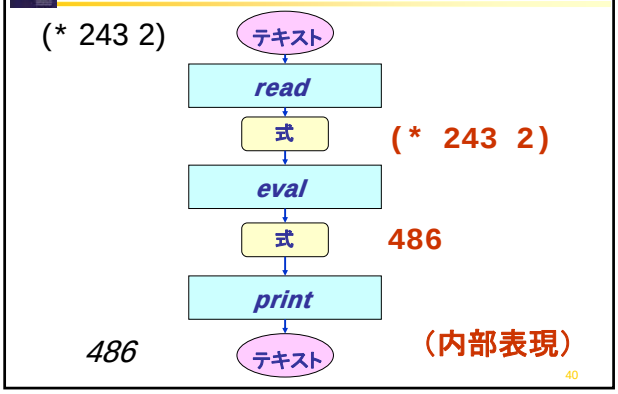
38

‘(- 1000 514)’ の評価



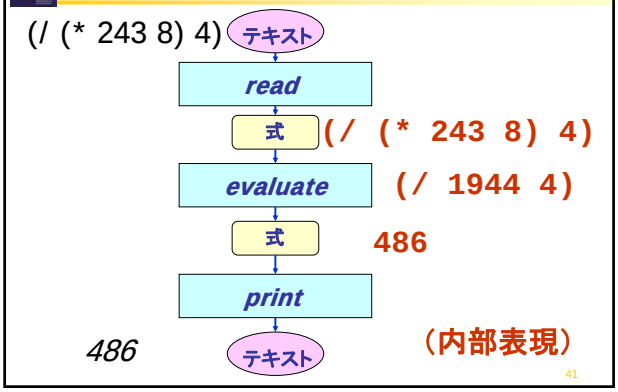
39

‘(* 243 2)’ の評価



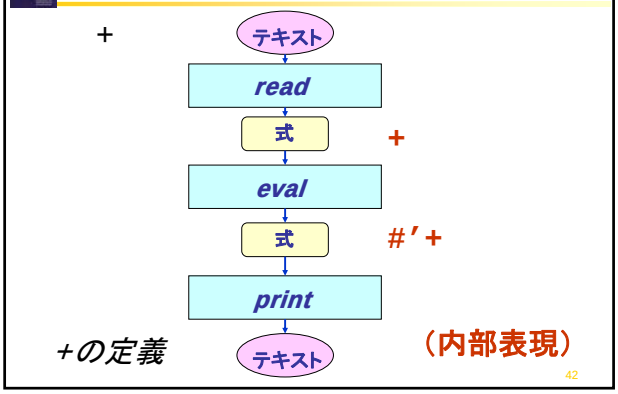
40

‘(/ (* 243 8) 4)’ の評価



41

‘+’ の評価



42

1.1.1 Expressions(式)

- 486
- (+ 137 349)
- (- 1000 334)
- (* 5 99)
- (/ 10 5)
- (+ 2.7 10)

43

1.1.1 Expressions(式)

- (+ 21 35 12 7)
- (* 25 4 12)
- (+ (* 3 5) (- 10 6))
- (+ (* 3 (+ (* 2 4) (+ 3 5)))
  (+ (- 10 7) 6))
- (+ (* 3
  (+ (* 2 4) (+ 3 5)))
  (+ (- 10 7) 6))

44

1.1.2 Naming (名前) Environment (環境)

- The critical aspect of a programming language is the means it provides for using a name to refer to computational object.
- The name identifies a **variable** whose **value** is the object.
- **What is a computational object?**
  - from simple data such as numbers
  - to Complex structures
- The **environment** provides some sort of memory that keeps track of the name-object pairs.

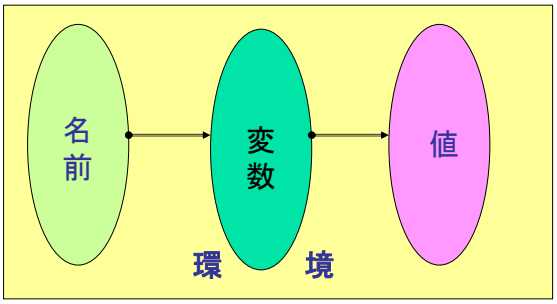
45

1.1.2 Naming and the Environment

- (define size 2)　処理系依存
- size
  2
- (* 5 size)
  10
- (define pi 3.14159)
- (define radius 10)
- (* pi (* radius radius))
  314.159
- (define cirumference (* 2 pi radius))
- circumference
  62.8318

46

1.1.2 Naming and the Environment

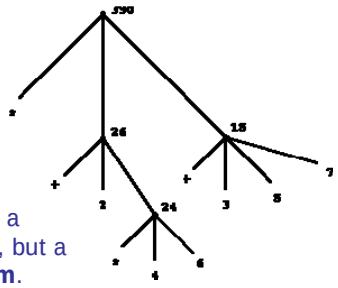


Global environment (大域環境)

47

1.1.3 Evaluating Combinations

- (* (+ 2 (* 4 6))
  (+ 3 5 7))



- define is not a combination, but a **special form**.

49

1.1.4 Compound Procedures(合成手続き)

- "To square something, multiply it by itself."
- (define (square x) (* x x))

To square something, multiply it by itself

- This is a compound procedure, of which name is "square".
- (define (<*name*> <*formal parameters*>)
  <*body*>)
  - <formal parameter> 仮パラメータ
  - <body> 本体

51

1.1.4 Compound Procedures(合成手続き)

- (square 21)
  441
- (square (+ 2 5))
  49
- (square (square 3))
  81
- (define (sum-of-square x y)
  (+ (square x) (square y)))
- (sum-of-square 3 4)
  25
- (define (foo a)
  (sum-of-squares
  (+ a 1)
  (* a 2)))
- (foo 5)
  136

Compound procedures are used in exactly the same way as primitive procedures. The latter is built into the interpreter.

52

1.1.5 The Substitution Model for Procedure Application(手続き適用の置換モデル)

- Assume that the mechanism for applying primitive procedures to arguments is built into the interpreter.
- For compound procedures, the application process is as follows:

To apply a compound procedure to arguments, evaluate the body of the procedure with each formal parameter replaced by the corresponding argument.

53

1.1.5 The Substitution Model for Procedure Application(手続き適用の置換モデル)

- (foo 5)

1. foo is the procedure defined before.
2. By retrieving the body of foo
   (sum-of-squares (+ a 1) (* a 2))
3. Replace the formal parameter a by the argument 5: (sum-of-squares (+ 5 1) (* 5 2))
4. Two arguments are evaluated and substituted for the formal parameters x and y:
   (+ (square 6) (square 10))
5. (+ 36 100)
6. 136

54

1.1.5 The Substitution Model for Procedure Application(手続き適用の置換モデル)

- A model that detemines the meaning of procedure application, insofar as the procedure in this chapter are concerned.
- The purpose of the substitution is just for explanation.
- The substitution model is the first step and we will present a sequence of increasingly elaborate models of how interpreters work.

55

1.1.5 The Substitution Model 注意

- **(|-∀) ∀右 と (∃|-) ∃左 は、eigenvariable condition が必要。要注意。**
- **a = b |- a = b**
  ---------- (|-∀)
  a = b ⊢ ∀x.(x = b)

  a = b |- a = b
  ----------(∃|-)
  ∃x.(x = b) ⊢ a = b

  上記は成立しない！

- **a = b |- a = b**
  ---------- (|-∃)
  a = b |- ∃x.(x = b)

  a = b |- a = b
  ----------(∀|-)
  ∀x.(x = b) |- a = b

  は成立！

56

Applicative order vs. normal order (適用順序と正規順序)

- 1.1.3: evaluates the operator and operands, and then applies resulting procedures to the resulting arguments.
- Alternative evaluation model: does not evaluate the operands until their values were needed.
- First substitute operand expressions for parameters until it obtained an expression involving only primitive operators, and would then perform the evaluation.

57

Alternative Model for Procedure Application

- (foo 5)

1. (sum-of-squares (+ 5 1) (* 5 2))
2. (+ (square (+ 5 1)) (square (* 5 2)))
3. (+ (* (+ 5 1) (+ 5 1)) (* (* 5 2) (* 5 2)))
4. (+ (* 6 6) (* 10 10))
5. (+ 36 100)
6. 136

The same answer, but different process.

The evaluations of (+ 5 1) and (* 5 2) are each performed twice.

58

Applicative order vs. normal order (作用的順序と正規順序)

- 1.1.3: "*Evaluate the arguments and then apply*" method called *applicative-order evaluation*
- Alternative evaluation model: "*fully expand and then reduce*" method called *normal-order evaluation*
- *Exercise 1.5 is an instance of an illegitimate value where these two evaluation methods do not give the same result.*

59

1.1.6 Condtional Expressions and Predicates (条件式と述語)

- 絶対値をcase analysis (場合分け)で定義

$$|x| = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -x & \text{if } x < 0 \end{cases}$$

- (define (abs x)
  (cond ((> x 0) x)
  ((= x 0) 0)
  ((< x 0) (- x))))
- General form of a conditional expression

60

1.1.6 Condtional Expressions and Predicates (条件式と述語)

- General form of a conditional expression
- (cond (<p1> <e1>)
  (<P2> <e2>)
  …
  (<pn> <en>))
- A pair of expressions (<p> <e>) called clauses.
- <p> predicate. Its value is interpreted as either true or false.
- <e> consequent expression
- Special <p>: else

61

1.1.6 Condtional Expressions (条件式)

- (define (abs x)
  (cond ((< x 0) (- x))
  (else x)))
- (define (abs x)
  (if (< x 0)
  (- x)
  x))
- If : special form
- (if <*predicate*> <*consequent*> <*alternative*>)

62

1.1.6 Predicates (述語)

- (and $<e_1>$ ... $<e_n>$) 論理積(左から評価)
- (or $<e_1>$ ... $<e_n>$) 論理和(左から評価)
- (not $<e>$) 論理否定

例:

- 5 < x < 10 ⇒
- (define (>= x y)
  (or (> x y) (= x y)))
- (define (>= x y)
  (not (< x y)))

63

1.1.7 Square Root by Newton's Method

$\sqrt{x}$ *is the* y *such that* $y^2 = x$ *and* $y \geq 0$

```
(define (sqrt-iter guess x)  Recursive definition
  (if (good-enough? guess x)
      guess
      (sqrt-iter (improve guess x) x) ))
(define (improve guess x)
  (average guess (/ x guess)) )
(define (average x y)
  (/ (+ x y) 2) )
(define (good-enough? guess x)
   (< (abs (- (square guess) x)) 0.001) )
```

64

宿題:10月11日正午締切

- 問題1．1〜1．5
- 問題1．5は難しい．
- レポート用紙に読める字で書くこと．
- ワープロ可．その場合名前等もワープロで．

66