1　復習

1.1　イントロダクション

process というものが event の連鎖で書ける.

1.2　絵図

process は event を矢印でつないだもので書ける.

たいていは上から下へ流れる木構造になるけど, recursive な定義があった場合は上に戻る.

1.3　規則

「process が等しい」という命題を厳密に定義するために規則を定めて, その規則「のみ」を使って等しいということを示す.

数学, 特に代数系で使われる定義の手法. この本の各章は「イメージの説明」→「規則で表現」→「Lisp もどきで実装」という流れになっている.

2　今回の話

2.1　Implementation of processes, プロセスの実装

P.18 の *interact* から.

最終行に括弧が足りない.

$$interact(A, P(car(k))), cdr(k)))$$

$interact(A, P, k) = cons(menu(A, P),$ if $car(k) =$ "END then

;; if the END inputed, terminate an interaction.

NIL

else if $P(car(k)) =$ "$BLEEP$ then

;; if the inputed command is not yet implemented,

;; just "beep" and do not change.

$cons($"$BLEEP,\ interact(A, P, cdr(k)))$

else

;; otherwise, accept the inputed command

;; and make the specified transition.

$interact(A, P(car(k)), cdr(k)))$

例 2.1. 「プログラムのメニューを選んで入力していく」というアナロジー

- P: 入力を受け付け, 次の入力待ちの状態に遷移するプロセス. ところどころ実装してない.
- A: P の入力一覧.
- k: Keyboard Input.

(ここから後は「LISP の実装系によってコードの書き方が様々になるよね」ということが書いてあるだけ.)

2.2 Traces, トレース

定義 2.1. トレースとはプロセスが動作した記録. 厳密には以下で定義される.

1. $\langle\rangle$ は trace. (空 trace)
2. x を event とするとき, $\langle x\rangle$ は trace.
3. $s = \langle s_0, s_1, s_2, \ldots, s_n\rangle, t = \langle t_0, t_1, t_2, \ldots, t_m\rangle$ を trace とするとき, $\langle s_0, s_1, s_2, \ldots, s_n, t_0, t_1, t_2, \ldots, t_m\rangle$ も trace. これを $s^\frown t$ と書く.

2.3 Operations on traces, トレース演算

2.3.1 Catenation, 結合

定義 2.1 で出てきた $\frown$ のこと.

規則 1 (identity element, 単位元)**.** $s\frown\langle\rangle = \langle\rangle\frown s = s$

規則 2 (associative law, 結合則)**.** $s\frown(t\frown u) = (s\frown t)\frown u$

ちなみに上の規則 1, 2 を満たすものをモノイド (monoid) と呼ぶ.

規則 3 (left cancellation, 左簡約)**.** $s\frown t = s\frown u \Leftrightarrow t = u$

規則 4 (right cancellation, 右簡約)**.** $s\frown t = u\frown t \Leftrightarrow s = u$

規則 5. $s\frown t = \langle\rangle \Leftrightarrow s = \langle\rangle \wedge t = \langle\rangle$

ちなみにここまでの 5 つの規則は公理である.

定義 2.2. trace → trace 型の関数 f が *strict* であるとは,

$$f(\langle\rangle) = \langle\rangle$$

を満たすということ.

定義 2.3. trace → trace 型の関数 f が *distributive* であるとは,

$$f(s\frown t) = f(s)\frown f(t)$$

を満たすということ.

規則 3, 4 から, distributive ならば strict であることが分かる. 数学では distributive な関数を (monoid 上の) homomorphism (準同型写像) と呼ぶ.

定義 2.4. 結合演算を乗算に見立ててべき乗を定義する.

規則 6. $t^0 = \langle\rangle$

規則 7. $t^{n+1} = t\frown t^n$

これから帰納法で以下が示せる.

規則 8. $t^{n+1} = t^n\frown t$

規則 9. $(s\frown t)^{n+1} = s\frown(t\frown s)^n\frown t$

2.3.2 Restriction, 絞り込み

定義 2.5. 定義 2.1 と同じ形式で記述する.

規則 1. $\langle\rangle \upharpoonright A = \langle\rangle$

規則 2. $\langle x\rangle \upharpoonright A = \begin{cases} \langle\rangle & \text{if } x \in A \\ \langle x\rangle & \text{otherwise} \end{cases}$

規則 3. $(s^\frown t) \upharpoonright A = (s \upharpoonright A)^\frown (t \upharpoonright A)$

この定義から以下が示せる.

規則 5. $s \upharpoonright \{\} = \langle\rangle$

規則 6. $(s \upharpoonright A) \upharpoonright B = s \upharpoonright (A \cap B)$

2.3.3 Head and tail, 頭と尾

要は Lisp の car と cdr. なので以下 (ry

2.3.4 Star, 星?

正規表現で出てくる asterisk と同じ意味.

定義 2.6.

規則 1. $\langle\rangle \in A^*$

規則 2. $\langle x\rangle \in A^* \iff x \in A$

規則 3. $(s^\frown t) \in A^* \iff s \in A^* \wedge\ t \in A^*$

2.3.5 Ordering, 順序付け

定義 2.7. trace には順序が入れられる. trace の集合は「順序付きモノイド」で右演算が順序を保つ.

$$s \leq t \equiv \exists u \text{ s.t. } s^\frown u = t$$

2.3.6 Length, 長さ

そのまんまの意味.