文法的機械（その2）

——計算機は計算するか？*——

原田　康也

1. アンドロイトは電気羊の夢を見るか？(1)

「計算機」という術語自体もそうだが,(2)一般に計算機関連の用語というのはわかりくにい印象がある。例えば,「ソフトウェア」対「ハードウェア」というわかったようなわからないような区別も,単に「形相」と「実質」,ソシュール言語学でいう form と substance の対立であるとわかれば話しは簡単なはずである。(3) VCR（ビデオ・カセット・レコーダ）のことをハードと呼び,映像を記録したカセット・テープのことをソフトと呼ぶような雑駁な用語法が「言葉の乱用にすぎない」ことは,ときおり強調することにも教育的な意義があるかもしれない。(4)

もっとも,言葉は乱用するところに命があるのであるから,(5)実際の用法を無視して「本来の使い方」を主張してみたところで意味のある議論はできない。例えば,RAM という術語は「読み出しはできるが書き込みはできない記憶媒体」を意味する術語 ROM に対立する用語として「読み出しも書き込みもできる半導体記憶素子」という意味で使われる用法が確立しているが,その本来の＜意味＞は Random Access Memory であるから「一次元的な順序に拘束されずに情報を読み出すことができる記憶媒体」ということであった。random access に対する術語はいまもって sequential access であるのだが,RAM という acronym にしたとたんに ROM が対立概念になってしまうのは,綴りの関

係もあるのだろう。

そもそもこうした2項対立そのものが概念上のものであり，比較的なものであることも忘れてはならない。「ハードウェア」対「ソフトウェア」の対立すら，どのレベルにおいて，という点を無視して論じることはできないし，いまではfirmwareというような中間的な概念・用語まで使われている。これはソフトウェアの中核的な部分をROM化することによって計算機のハードウェアの一部として組み込むと同時にreverse engineeringに対して半導体素子の設計・製作に関わる著作権によって対抗しようという手法である。firmには「堅い・しっかりした」という意味と「会社・工場」という意味があるが，firmwareとはまさにこの曖昧さ・多義性を逆手にとった巧みな用語であるといえよう。

ROMというのは「読み出しはできるが書き込みはできない記憶媒体」を意味する術語であると述べたが，「初期状態では書き込みができる読み出し専用の記憶素子」を意味するPROM (Programmable ROM) という術語もある。「初期状態では書き込みができ，何らかの強制的な手段で書き込んだ記憶を消去し初期状態に復元できる読み出し専用の記憶素子」を意味するEPROM (Erasable Programmable ROM) というものもある。「初期状態では書き込みができ，電子的な手段で書き込んだ記憶を消去し初期状態に復元できる読み出し専用の記憶素子」を意味するEEPROM (Electronically Erasable Programmable ROM) というものもある。ここまで来るとRAMとの違いがどこにあるのかは，機能上からはほとんどわからない。商品系列，コスト，内部構造，利用法などを見ないことには，いわゆるRAMとEEPROMの違いはわからないことになる。

というように用語の逐語的な解釈にこだわって話しを進めてみると，果たして「計算機」というものが，適切な用語なのかどうかを議論の対象にしなければならないであろう。とはいえ，日本語の場合には，「計算」という表現と「機」という表現が連続した場合，その両者の間に成立することが想定される関係として＜主語－動詞＞あるいは＜動作主－述語＞といった関係に限られるわけで

はない，というなやましさがある。「計算機」といっても，「計算する機械」のつもりではなく「計算に使う機械」なのかもしれない。あるいは，「計算する機械」といっても，「それが自ら計算する処の機械」ではなく，「それによって人が計算する処の機械」のつもりかもしれない。いずれにせよ，「計算機は計算するか？」という設問こそ，我々が「計算機」を前にしたとき第一に発するべき質問であることは間違いない。しかしながら，「計算機は計算するか？」という設問よりは「計算機は思考するか？」という設問の方が一般の興味をひきつけ易いようである。

「計算機は思考するか？」というのは，人工知能研究，特にそれに関する哲学的なメタ議論の中心的課題のようである。[6] たとえば，*Mind's I* というタイトルの Douglous Hoffstadter の＜作ならびに編＞の＜妄想と反趣の集成＞[4] は全巻この課題に関連する議論で終始しているといっても過言ではない。もちろん，「計算機とは何であるか」という点と「思考する」とは何を指すかを規定することなくこのような疑問を提示したところで，噛み合った議論が成立するはずもないのだが，こうした議論を通じてそれぞれの論者が「計算機」として何を想定しているか，「思考する」ということに対してどのような思い入れを込めているかが明かとなってくる。従って，「計算機は思考するか？」という設問をめぐる議論は，「計算機」の特徴を明らかにするというよりも「思考する」とは何を意味するのかという人間に関する哲学的議論の色彩を帯びることになるのもまた自然の成行きである。

「計算機は思考するか？」という設問に関連してしばしば言及されるのが Turing test である。これは極端に単純化して言い替えると「計算機の振舞いがとても巧妙で，あたかも人間が思考しているのと見分けがつけられないほどであるなら，その計算機は思考している，と言ってしまうことにしよう」という提案である。[7] あるいは「相手が愚かしいことを言って何も考えていないことが分かるまでは，相手が何事か考えていることにしておこう」という optimism と言ってもいいかも知れない。Hoffstadter の[3]はこの Turing test

が何を意味するものであるかを，物理学，生物学，哲学を専攻する学生たちの会話を通して説きほぐすという＜作品＞である。次に紹介するのは[11]から[6]に引用・紹介されている Alan Turing 自身の作による＜チューリング・テスト＞の一例である。

Interrogator: In the first line of the sonnet which reads "Shall I compare thee to a summer's day," would not "a spring day" do as well or better ?

Witness: It wouldn't scan.

Interrogator: How about "winter's day" ? That would scan all right.

Witness: Yes, but, nobody wants to be compared to a winter's day.

Interrogator: Would you say Mr. Pickwick reminded you of Christmas ?

Witness: In a way.

Interrogator: Yet, Chrismas is a winter's day, and I do not think Mr. Pickwick would mind the comparison.

Witness: I don't think you are serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas.

一方，[2]の中で重要な働きをする Voigt-Kampff test は見かけ上人間と区別のつかないアンドロイド（これを Dick は sim (=simulacrum) と呼ぶのであるが）を見つけ出すための教理問答のようなものである。[2]の中から典型的なシーンを台詞だけ抜粋して紹介しよう。ここで，先に紹介した＜チューリング・テスト＞との関連性を考えるのも面白いであろうし，また映画 *Blade Runner* での対応するシーンと比較するのも面白いかもしれない。

"In a magazine you come across a full-page color picture of a nude girl."

"Is this testing whether I'm an android or whether I'm homosexual ?"

"Your husband likes the picture. The girl is lying face down on a large and

beautiful bearskin rug. Your husband hangs the picture on the wall of his study."

"I certainly wouldn't let him."

"Okey. Now consider this. You're reading a novel written in the old days before the war. The characters are visiting Fisherman's Wharf in San Francisco. They become hungry and enter a seafood restaurant. One of them orders lobster, and the chef drops the lobster into the tub of boiling water while the characters watch."

"Oh god. That's awful! Did they really do that? It's depraved! You mean a *live* lobster?"

「計算機は思考するか？」という設問を放棄する手だてとしての Turing test も面白い思考実験であるが，「本物の人間とはなんであるか？」を明らかにする小説的機構としての Voigt-Kampff test も興味深い。だが，より現実的な設問として，「計算機は計算するか？」という問題点を避けることはできない。この点に関しては John Searle が[8]の中で次のような指摘をしている。機械には，人間が理解するという時と同じ意味で「理解」という言葉を使えない，というのが[8]と[9]における Searle の一貫した主張である。

My car and my adding machine, on the other hand, understands nothing: they are not in that line of business. We often attribute "understanding" and other cognitive predicates by metaphor and analogy to cars, adding machines, and other artifacts, but nothing is proved by such attributions.

人は「サーモスタットが温度を感じる」と比喩的に言うとしてもそれは人間が温度を感じる感じ方とは全く異なる，そして「計算機が計算する」と言うとしても，それは「人間が計算する」というときとは全く異なることをさしてい

る，というのがSearleの論点である。

Searleのあげている例は，日本語と英語で比喩的な表現の可能性が微妙に異なりむしろわかりにくいので，次のような例を考えて見よう。次のような言い方は日本語として可能であろうか。意味があるであろうか。[8]

(1)

a．テレビが夢を見る。

b．VCRが映画を見る。

c．カメラが人物や景色を見る。

おそらく，多くの人にとって，比喩的な意味かＳＦ的な状況以外で(1)のような表現は許されないであろう。つまり，日本語を母国語とする多くの人にとって，次のような暗黙の了解がある。

(2)

a．テレビは夢を見ない。

b．VCRは映画を見ない。

c．カメラは人物も景色も見ない。

従って，日本語では通常の状況において，(1)のようなものの言い方をしない。それではなぜ人は「計算機が計算する」といった間違ったものの言い方をするのであろうか。あるいは，なぜそのような間違った表現に対して寛容なのであろうか。

「計算機は計算のための道具である」という通念からすれば，「計算機が計算するかどうか？」というのは愚かしい設問に思えるかも知れない。では設問の目先を代えて「算盤は計算するか？」「電卓は計算するか？」としてみよう。次のような表現は，日常的な文脈において，日本語として自然な表現であろう

か。

(3)

a．算盤は計算する。

b．電卓は計算する。

c．計算機は計算する。

(4)

a．算盤が計算した結果

b．電卓が計算した結果

c．計算機が計算した結果

(5)

a．算盤で計算した結果

b．電卓で計算した結果

c．計算機で計算した結果

算盤というものについては小学校で教えられた知識しか持っていないが，人間が計算していく上での鍵のような部分の覚え書を残す道具と理解しても間違いではないであろう。算盤に関しては暗算というものがある。算盤の操作を内化することによって，熟達した算盤使いは算盤無しで計算ができるようになるのであろう。計算を行っているのは人間であり，算盤はその計算過程の途中経過と結果を記録するための表象の道具なのであろう。従って，「算盤が計算を行う」という表現に対しては，比喩的にせよ正当性を認めることは難しい。

一方「計算機が計算する」という表現は，哲学的な立場から異議が唱えられることがあっても，日常生活ではよく耳にし，また使いがちな表現である。「電卓が計算する」と人は言うのであろうか。それとも「電卓で計算する」と

いう表現に留まるのであろうか。実は前者を認めないのであれば，「計算機が計算する」という表現もおかしいことに気がつくべきである。なぜなら，計算機の行っていることも，物理的なレベルにおいてはある種の電気信号の系列を入力としてある種の電気信号を出力するだけであり，それが何事かの「計算」であるのは，そうした電気信号の系列を「数」の表象として人間が用いていることに依存している点で算盤の場合となんら変わることがないからである。これに関して Searle も似たような点を[8]の中で次のように述べている。

[T]he programmed computer does not do "information processing." Rather, what it does is manupulate formal symbols... Thus, if you type into the computer "2 plus 2 equals ?" it will type out "4." But it has no idea that "4" means 4 or that it means anything at all.

ではなぜ，「電卓が計算する」と考えるより「計算機が計算する」と考えることに抵抗が少ないのであろうか。我々は一般常識として「計算機」が「計算機言語」によって書かれたプログラムという指令の集まりに基づいて動作することを知っているが，この常識が事態の正しい認識をより困難にしている面があるのかも知れない。上に紹介した Searle の議論も，計算機が行っていることを物理的なレベルで促えて「計算」をしているかどうかを検討することよりは，「記号列としてのプログラム」が人間の心的過程を理解する手だてとなるかどうかという点に注目している。「情報処理」ではないと主張しつつ，「形式的な記号操作である」と述べている点では，むしろ一般常識的な「誤解」に固まっていると見なすべきであろう。こうした「誤解」はなぜ生じるのであろうか。

話を具体的にするために，非常に単純な例として，ふたつの正の整数2と3の和を出すような指令を「計算機」に対して与えることを考えてみよう。これまた単純な例として BASIC というプログラミング言語を使うことを考えてみ

よう。これはおよそ(6)のようなプログラムとなろう。

(6)

```
10  PRINT 2+3
```

これではいくらなんでもプログラムという感じがしないかも知れないので，もう少し一般化してふたつの正の整数に対応する変数 A と B を考え，それぞれに2と3を代入し，その和を出力するような指令を計算機に対して与えることを考えてみよう。このためのプログラムはおよそ(7)のようになる。

(7)

```
10  A＝2
20  B＝3
30  PRINT  A＋B
```

これをもう少し一般化してふたつの正の整数に対応する変数 A と B を考え，それぞれに数値を代入し，その和を出力するような指令を計算機に対して与えることを考えてみよう。これはおよそ(8)のようになろう。

(8)

```
10  INPUT A
20  INPUT B
30  PRINT A+B
```

ここまでくれば，いかに単純とはいえ，たとえば前期試験の成績と後期試験の成績を加算して成績を求めるプログラムの一部などとして十分実用性が生じてくる。

常識的な理解としてはこうしたプログラムを「計算機」が「理解」して「計算を実行する」結果としてさまざまな数値や出力が得られる，ということになろう。しかし，そうした理解はあくまでも＜比喩＞としてのみ正しい。

ハードウェアとしての計算機の中核的な部分はデジタル信号を処理する機能を中心に設計されている。ＣＰＵ（中央演算操置）と呼ばれる仮想的な装置に対してデジタル信号の列としての入力があると，設計が仕様通りに行われていれば，その仕様に基づいた出力をデジタル信号の列として返してくる。こうした bit 列として構成された計算機に対する指令を BASIC などのプログラミング言語と対比して機械語と呼ぶが，このレベルにおいては，指令もデータも出力も bit 列であり，従って，「計算機」が行っていることの物理的な実態はデジタル信号処理であって，それを「計算」と呼ぶかどうかは語用論の問題となる。「計算機」の入力と出力を人間が見たときそこに計算が成立している，ということと，「計算機が計算している」ということを等価と見なせるかどうかは，「計算する」という表現にどれだけ「意識」ないし「(メタ) 認識」を仮定するかに依存するであろう。

プログラミング言語はしばしば「計算機言語」と呼ばれるが，これを「計算機のための言語」と考えるのはまったくの誤解である。計算機の中核的な演算機構がハードウェアとして「理解」するのは「機械語」だけであり，それ以外の「高級言語」は人間が機械に対する指令を書き易くするために設計されたものであり，「計算機を使うための言語」ではあっても「計算機のための言語」ではない。「高級言語」は「機械語」に翻訳されてはじめて機械による実行が可能になるのであり，人間と計算機のインタフェースを定めるものであるということもできよう。人間が「他者」と「対話」をするために用いるものであるという点で「言語」と呼ぶのがふさわしい。

翻訳と実行との関連で計算機言語にもおおまかに分けて二つのタイプを考えることが出来る。一つはインタプリンタと呼ばれる＜解釈系＞に対する入力となる計算機言語である。これは計算機システム（＝ハードウェア＋解釈系）が

対話的に実時間でコマンドを受け付け，それを機械語に翻訳して実行するものである。例えばBASICであるとか，LISPであるとかprologであるとかがこうしたインタプリタ用言語としては一般に有名である。もう一つは，コンパイラに入力される言語である。コンパイラというのは，もととなるテキストとしてのプログラムを機械語のプログラムに翻訳するシステム（ソフトウェア）のことである。こうした翻訳の入力となるテキストとしてのプログラムをソース・プログラム（ソース・コード），翻訳の結果出来上がる機械語の集まりをオブジェクト・プログラム（オブジェクト・コード）と呼ぶ。こうしたコンパイラ系の言語としてはFortran, C, Pascal, Adaなどが有名である。

もちろん，インタプリタ系対コンパイラ系という二項対立も相対的なものであり，BASIC, LISP, prologといったインタプリンタ系言語に関してもコンパイラが作られることが普通である。しかし，インタプリタ系言語の場合，コンパイラによって作られたオブジェクト・コードをインタプリタから呼び出して使うことが多いといった特徴もある。（つまり，一般にはその場で翻訳＋解釈＋実行を行うインタプリタ形式よりも，あらかじめコンパイルを済ませておいたオブジェクト・コードを実行する方が動作が速いので，インタプリタを利用する場合でも，しばしば使われるルーチンに関してプリコンパイルしたオブジェクト・コードを利用する，といった複合的な利用法があるわけである）また，コンパイラ系の言語で書かれたソース・コードはオブジェクト・コードに「翻訳」されて始めて機械による「実行」が可能となるが，そのコードを「起動」するにはオペレーティング・システムの介入が必要であり，このオペレーティング・システムは全体としては一種のインタプリタ系をなしてもいる。この様に，計算機の利用に際しては，両者を適宜組み合わせて利用することが一般的である。

インタプリタの場合には投入されたコマンドをその場で解釈系が機械語に翻訳してそれがまたCPUへの入力となり実行されるのであるし，コンパイラの場合にはソース・コードをオブジェクト・コードに翻訳するのがその最大の仕

事となる。いずれも，入力としてのプログラムはテキスト・ファイルであり，その意味で計算機はこうした作業の中で「計算」をしているというよりは「テキスト処理」をしているという方がふさわしい。従って，大方の計算機が行っている処理の多くが「高級言語」によって書かれたプログラムの「翻訳」と，より機械語に近い「低級言語」によって書かれたプログラムの「解釈」であることは忘れてはならない。[9] 実際，計算機の「基本プログラム」とされる OS (Operating System) の中核とはファイル・システムの設計とコマンド・インタプリタの仕様決定にあるといっても過言ではない。[10]

プログラム言語が計算機とユーザのインタフェースを規定しているということは，ユーザから見た場合，計算機が内部的にどのようなデジタル信号処理を行っているかを考えるよりは，＜ハードウェア＋言語処理系＞を一つの仮想的な機械と考えて，それに対して必要に応じてプログラムを与えて使用していると見なした方が簡単である，ということを意味する。このような考え方に慣れ親しむと，「プログラムを計算機が理解して計算を実行する結果としてさまざまな数値や出力がえられる」という見方に拘束されることにもなりかねないが，これまた「言葉の乱用」であることをときおり思い起すのも教育的な意義はあろう。

2. こことあそこ[11]

計算機に対する指示が計算機言語という言語システムに基づいて行なわれる以上，そこには人間同士が言語を用いて対話を行う際と同じ様な曖昧性，多義性，状況依存性が生じ，その解決に文脈的情報が用いられることは当然である。計算機科学において自然言語処理がマン・マシン・インタフェースなどとの関連で話題になるとき，この点はしばしば誤解されているようである。多くの場合，こうした曖昧性，多義性，状況依存性は，自然言語に固有の特徴として論じられ，その点を解決・処理することが自然言語処理の中核であるかのような記述が与えられることが多い。例えば[11]では自然言語を計算機で処理する場

合の問題点として，次のような特徴を挙げている。(後の説明との関連で若干表現を改めたり順序を変更した部分もある)

＜１＞ ambiguity

＜1.1＞ lexical ambiguity

eg. The pitcher is angry.

The pitcher is empty.

＜1.2＞ syntactic ambiguity

eg. I hit the man with the hammer.

＜1.3＞ anaphora

eg. John hit Bill because he sympathized with Mary.

＜２＞ incompleteness

eg. John went out to a restaurant last night. He ordered steak. When he paid for it, he noticed that he was running out of money.

＜３＞imprecision

eg. I've been in the doctor's office for a long time.

The crops died because it hadn't rained for a long time.

The dinosaurs ruled the earth a long time ago.

＜４＞ inaccuracy

＜4.1＞ spelling errors

＜4.2＞ transposed words

＜4.3＞ ungrammatical constructions

＜4.4＞ incorrect syntax

＜4.5＞ incomplete sentence

＜4.6＞ improper punctuation

しかしながら，こうした点はむしろ自然言語・計算機言語に関わらず，コミュニケーションのモードである「言語」ならびに「言語の使用」に特徴的な現象であると考える方が正しい。例えば＜4＞にあげた inaccuracy に関わるものは，人間であれ計算機であれ，対話の相手が充分に知的であって対話の内容の把握が正確であればそれに基づいて補うことが可能であり，少くとも不正確であることを検出して相手に問い返すことが出来ることがらである。計算機言語で人間がプログラムを組む場合にも，こうした不正確さは避けられるはずもなく，多くの場合プログラムにはこうした＜構文的な誤り＞が付きものである。しかし，エディタが優れていれば，＜機械的な誤り＞は最小限に抑えられるし，多くの場合，コンパイラがこうした＜構文上の誤り＞を指摘してくれる。高度に intelligent なコンパイラないしデバッガであれば，＜誤りの指摘＞だけでなく＜訂正の示唆＞までしてくれることが期待される。

しかし，こうした＜機械的な誤り＞を見つけそれに対する＜訂正の示唆＞をしてくれることを計算機に期待するのは何も＜計算機言語＞に限定する必要なない。最近の欧文ワープロは（ソフトウェアであれハードウェアであれ）綴りをチェックする機能を備えている例が多い。構文や文体をチェックするソフトウェアもある。こうした，文章作成支援の機能は実は，文字列としてのテキストを処理するという意味で，プログラム作成支援の機能と本質的な所で類似している。そして，プログラム作成ならびに開発支援環境の作成が計算機の利用の重要な一部である以上，文章作成支援に計算機を利用するのは全く自然なことなのである。

＜4＞の inaccuracy というものが人間のあり方に関わっている以上，「自然言語」であろうと「計算機言語」であろうと人間が言語を使う際には避けることが出来ないわけであるが，これは＜言語＞の内在的な特徴というよりは人間の＜言語運用＞の特徴と促えることもできよう。同様に＜3＞の imprecision も人間の＜言語運用＞の特徴と促えることができる。言語の内在的ないし形式的な特徴としては，従って，＜1＞の ambiguity や＜2＞の incompleteness を

考える方が面白い。そしてこれらは言語が状況に依存して使われ，常識に従って理解されるという点を反映している。[12]

日本語の「曖昧」という単語は「曖昧な」単語である。「多義的（ambiguous）」という意味で用いられる場合もあるし，「あやふや（fuzzy）」という意味で用いられる場合もある。前者が＜１＞にあたり，後者が＜２＞に相当し，日本語では両者をあわせて「曖昧」と呼んでいるのであろう。自然言語の特徴として「曖昧さ」があげられることは多いのだが，その「曖昧な表現」の解釈には，状況による解決・処理と常識による解決・処理が見られる。例えば上記の＜1.１＞の二つの文で pitcher が何を意味するかは文の内容から常識で理解されることになろう。それに対して＜1.2＞が「ハンマーで人を殴った」か「ハンマーの男を殴った」かはその場の状況がわからなければ解決できない。

とはいえ，「計算機言語」が「自然言語」と違って「曖昧でない」と考えるのは誤解でしかない。言語の本質がその乱用にあるように，多義性をもたない記号体系は窮屈で使いづらい。例えば，前節にあげた二つの数の和を求めて表示する「プログラム」にしても，変数に代入される数値が整数か，実数か，またその有効桁数はいくつか，という指定によって，計算機の内部的な動作は変わってくる。プログラムとして安全に動作することを求めるのであれば，使われている変数がどのような数値に対する表現であるかを「宣言」することが必要である。つまり，“＋”という加法を表すような基本的な記号ですら，計算機の扱い得る数値の種類の数だけ「多義的」なのである。

人間が会話をする上で，物を（具体的ないし話し手と聞き手の了解において）指し示しながら「あれ」とか「これ」と指示する表現がなければ対話の進行が著しく阻害されることは想像に難くない。同様に，計算機に対する指令においても，「さっきのあれ」といった状況に依存した指示は有効である。例えば MS-DOS のような素朴なシステムでさえ，function-3 を押すことによって直前のコマンドが再現されるような機能は備えている。UNIX の csh といったコマンド・インタプリタでは，history 機能がある。!! によって直前のコマンド

が再現されるだけでなく，^string 1^ string 2 による string 1 から string 2 への置換などの機能も実現されている。また，例えばアプリケーション・ソフトウェアのレベルでも WordStar などでは cntl-R によって，直前に作業をしていたファイル名を呼び出すことができるような設計となっている。

「あれ」「これ」といった話し手と聞き手の了解に関わる指示的指標と同様に重要なものとして，「いま」「ここ」という指標がある。これは計算機の場合，ファイル指示に関して current directory/current disk といった概念として頻繁に利用されることになる。例えば MS-DOS の場合を例に取って考えて見よう。[13]

MS-DOS 上のファイルを指定するにはドライブ名とファイル名の指定が必要である。例えば Aというドライブに EXAMPLE というテキスト・ファイルがあったとして，これを画面に出力することを考えてみよう。指示的指標を利用しないでこれを実行しようという場合には例えば

1＞ TYPE A:EXAMPLE

といったコマンドの投入が必要である。（コロン（“：”）は＜ドライブ指定＞と＜ファイル名＞の間を区切る記号である）ところが，A が current drive である場合には，＜ファイル指定＞の中で＜ドライブ指定＞の省略が可能となる。

2＞ TYPE EXAMPLE

というコマンド投入がコマンド・インタプリタによって解釈されると，先の1＞の場合と同じ動作が実行される。B が current drive である状況では

3＞ TYPE B:EXAMPLE

として解釈される。従って，Bというドライブに EXAMPLE というファイルがあればそれが画面に出力されるし，そうでなければ

「ファイルが見つかりません」

といった類のエラー・メッセージが画面に表示される。

MS-DOS のファイルは階層ディレクトリとなっている。つまり，コマンドにおける＜ファイル指定＞は＜ドライブ指定＞と＜パス指定＞の連結となる。ただし，＜パス指定＞は＜ディレクトリ指定＞と＜ファイル名＞を連結したものとする。＜ドライブ指定＞と＜パス指定＞の間の＜句切り記号＞としてはコロン（":"）が使用される。また，＜ディレクトリ指定＞と＜ファイル名＞の間の＜句切り記号＞としては＜円記号＞("¥") が使用される。(これは本来の PC-DOS で＜バックスラッシュ＞（"\"）を使っていたものが JIS 記号の都合で奇妙なものになってしまったのである)

例えば先ほどのAドライブにあった EXAMPLE というファイルが実は¥（ルート）というディレクトリにあったとすると，指示的指標を全く利用せずに目的の動作を実行させるには

4＞ TYPE A:¥EXAMPLE

というコマンド投入が必要であった。また，このファイルが，¥EXADIR というディレクトリにあったとすれば，

5＞ TYPE A:¥EXADIR¥EXAMPLE

というコマンド投入が必要であった訳である。(ここでディレクトリ指定とファイル名の間に¥という区切り記号が挿入されている。この記号が root を示す

記号と同じと言うのも計算機言語特有の ambiguity である）

さて，current drive と同様の概念として current directory というものがある。例えば current drive が A であって，current directory が ¥EXADIR という状況では

6＞ TYPE EXAMPLE

というコマンド投入によって A: ¥EXADIR¥EXAMPLE が画面出力される。ところが，current drive が A であって current directory が ¥EXBDIR という状況では同じコマンド投入で A: ¥EXBDIR¥EXAMPLE が画面出力される。（もしくは「ファイルが見つかりません」のエラーメッセージが出力される）この状況下で A: ¥EXADIR¥EXAMPLE を画面出力するには，例えば，

7＞ TYPE ¥EXADIR¥EXAMPLE

といったコマンド投入が必要となる。もちろん，

8＞ TYPE A: ¥EXADIR¥EXAMPLE

のようにドライブ名もパス名も指定しておけば，状況によらず（すなわち current drive と current directory に関わらず）同一のファイルが指定されるので安全ではあるが，対話的環境で作業する場合には，安全性と簡便性のトレードオフは簡便性に片寄るのは当然である。

"current" と似ていながらまったく異なるものとして，"default" という概念もある。これは明に指示を与えなかった場合にシステム側が「適当な解釈」を＜常識によって＞補うというものである。＜常識的な解釈＞はシステムの設計段階でさまざまな場合に分けて詳細に規定されるわけであるが，ことが常識に

関わるだけに，システムの使い心地は default の設計に依存するところが大きい。またしても MS-DOS で頻繁に使う COPY というコマンドを例に取って説明してみよう。

本来的には COPY というコマンドは引き数を二つ取る。第一引き数が複写元のファイル指定，第二引き数が複写先のファイル指定である。例えば次のようなコマンド投入によって A というドライブの ¥EXADIR というディレクトリの EXAMPLE 1 というファイルが B というドライブの ¥EXBDIR というディレクトリの EXAMPLE 2 というファイルに複写される。

```
9＞　COPY　A: ¥EXADIR¥EXAMPLE 1　B: ¥EXBDIR¥EXAMPLE 2
```

いずれのファイル指定も先ほどの場合と同様に current drive と current directory が何であるかという状況に応じていくぶんかの省略した指定が可能となる。しかし，この current という概念に依存した省略に加えて第二引き数に関しては更に default という概念に依存した省略の可能性もある。例えば第二引き数を全部省略することも可能である。

```
10＞　COPY　A: ¥EXADIR¥EXAMPLE 1
```

この時何が起こるかは二つの場合が考えられる。current drive が A で current directory が ¥EXADIR の時には，

```
「コピーすることはできません
        0個のファイルをコピーしました」
```

というエラーメッセージが返ってくる。それ以外の場合には current drive の current directory に新たに EXAMPLE 1 というファイルを作り，そこに

A: EXADIR¥EXAMPLE 1 の内容を複写する。(既に EXAMPLE 1 というファイルが存在する場合，その内容は失われる)

この場合，実際にコマンド投入の結果何が＜実行＞されるかは＜状況＞すなわち current drive と current directory の値に依存する。しかし，その＜解釈＞が何であるかは一意に定まっている。*drv* を current drive を現す meta-variable, *dir* を current directoy を現す meta-variable として使うと，11＞ のコマンド投入は常に 12＞ のコマンド投入と等価である。

11＞ COPY A: ¥EXADIR¥EXAMPLE 1

12＞ COPY A: ¥EXADIR¥EXAMPLE 1 *drv*: *dir* ¥EXAMPLE 1

current drive が A で current directory が ¥EXADIR の時には，11＞ は 13＞ と等価であるために上で述べたようなエラーメッセージが返ってくるわけである。

13＞ COPY A: ¥EXADIR¥EXAMPLE 1 A: ¥EXADIR¥EXAMPLE 1

つまり，current drive/current directory は実行に関する環境を指定するものであるのに対して，default とは解釈に関わるものであるということが出来る。

第二引き数を全く省略する場合のほかに，ドライブ名ないしドライブ名とディレクトリの指定だけしてファイル名を省略する場合も有り得る。例えば，

14＞ COPY A: ¥EXADIR¥EXAMPLE 1 B:

とした場合には，ドライブB上の current directory に新たに EXAMPLE 1 というファイルを作り，そこに A: ¥EXADIR¥EXAMPLE 1 の内容を複写する。

（既に EXAMPLE 1 というファイルが存在する場合，その内容は失われる）
また，

15＞ COPY A: ¥EXADIR¥EXAMPLE 1 B: ¥EXBDIR

とした場合には，ドライブBの上の ¥EXBDIR というディレクトリに新たに EXAMPLE 1 というファイル作り，そこに A: ¥EXADIR¥EXAMPLE 1 の内容を複写する。(既に EXAMPLE 1 というファイルが存在する場合，その内容は失われる）ただし，このときドライブBに ¥EXBDIR というディレクトリがない場合，MS-DOS のコマンド・インタプリタは第二引き数をファイル名の指定として扱い，ドライブB上に新たに ¥EXBDIR というファイルを作り，そこに A: ¥EXADIR¥EXAMPLE 1 の内容を複写する。(既に ¥EXBDIR というファイルが存在する場合，その内容は失われる）

ここで見られるのは COPY というコマンドには "COPY *arg*" という書式と "COPY *arg 1 arg 2*" という書式があり，この二つの書式の間で ambiguous であること，二つ目の書式のコマンド投入に対する動作が *arg 2* がファイル指定であるかディレクトリ指定であるかによって異なること，そして *arg 2* がドライブ指定かディレクトリ指定である場合には複写先のファイル名が *arg 1* に従って決まることが重要である。つまり COPY という MS-DOS のコマンドは多重に ambiguous であり COPY を使ったコマンド投入の解釈は状況に依存して決まり，その実行には default が関与することがある，ということである。こうしたことが示しているのは，MS-DOS のコマンド実行といった場合にも，ユーザとシステム・インタプリタの「対話」があり，そこに「言語の使用」が見られ，言語の使用は「文法的原理」に従って行われている，ということである。

MS-DOS のようなパーソナル・コンピュータのオペレーティング・システムのコマンド・インタプリタであれば，対話的な環境で使用されることが当然

であり，そのためその解釈と実行に関してある種の状況依存が生じるのは当然のことである。しかしながら，こうした状況依存性はむしろ言語一般の特徴であり，自然言語だけでなく，計算機言語にもあまねく見られる。プログラムは[9]のタイトルが示唆する通り，文字列ないし記号列としてのテキストであり，プログラムをコンパイルしたり解釈・実行することは「言語処理」にほかならない。(実際，BASIC というのは Beginner's All Purpose Symbolic Instruction Codeの acronym であったりする) 計算機言語と自然言語との違いがあるとすれば，計算機言語の方がより厳密に形式的であり，より厳格に文法的であるということであろう。

3. What you see and what you get.

電子機器をコミュニケーションに利用するようになると，複製というものの存在が我々の実在に対する認識を混乱させて，言語表現がますます実体との遊離を深めて行く。メディアの融合と変容に関して *The Whole Earth Catalog* の制作者 Stewart Brand は[1]で次のように言う。

With digitalization all of the media become translatable into each other – computer bits migrate merrily – and they escape from their traditional means of transmission. A movie, phone call, letter, or magazine article may be sent digitally via pahone line, coaxial cable, fiberoptic cable, microwave, satellite, the broadcast air, or a physical storage medium such as tape or disk. If that's not revolution enough, with digitalization the content becomes totally plastic – any message, sound or image may be edited from anything into anything else.

例えば，「原稿」という日本語を考えてみよう。この単語の「意味」は何であろうか。「意味」というより「指示対象」という方がより正確かも知れない。「指示作用」を支持する「実体」と言うべきかも知れない。黒い文字や赤い文

字が乱雑に書き散らかされた「紙」が「原稿」なのだろうか。それとも「書かれた文字」が原稿なのだろうか。「印刷物を原稿にする」というのはどのような意味なのだろうか。ワープロ・ファクスの時代以前にもこのような問題はあったはずである。テキスト・クリティークなり，文献学なりというのは，作者の「意図」とこのような「実在」の相克の問題をはらんでいたはずである。

電話だとかファクスなどというものの出現によって，「原稿」が紙を指すものでないことは明らかになってしまった。あるいは「紙に定着したなにものか」以外のものを指し示す用法が規約的習慣として定着した，というべきかもしれない。「電話送稿」だとか，ファクスで「原稿を送る」だとかいう表現をとってみれば，書かれた紙としての「原稿用紙」が問題なのではなく，そこに書かれた「内容」が原稿の「実体」だとわかる。

これが電子化されることによって，ますますわけがわからなくなる。ぼくのように，未発表の原稿をフロッピーやハードディスク上のファイルとして電子的な手段で蓄積している場合，それは最終的には磁気媒体の上の bit 列に帰着する。これが原稿なのだろうか。とはいえ，文章を書く上で bit 列を想定して書いた覚えはない。

「計算機がプログラムに従って計算する」というのが「誤解」であるのと同じように，「人がワープロによって文書を作る」というのも「誤解」であるかも知れない。最終的にプリンタなり写植機なりから印字を得る段階に達することをもって文書を作ると言うのであれば，「人がワープロによって文書を作る」と考えるのが正しいのであろうが，文書をファイルとして電子的に記録している状態にとどめているとき，これは「文書を作成した」ことになるのであろうか。これまた，「文書」が何を指すか，意味論と語用論を再検討する必要があろう。そのために，いわゆる日本語ワープロなるものがどのような機能を持つものであり，人がそれをどのように使っているのかを考え直してみよう。[14]

いわゆる日本語ワープロと呼ばれるものが持っている機能はおよそ次の3つ

の下位の機能に分解して考えることが出来る。

＜＃1＞　かな漢字変換機能
＜＃2＞　テキスト編集機能
＜＃1＞　出力整形機能

＜＃1＞のかな漢字変換機能はかなやローマ字による入力を変換キーなどを押すことにより漢字かな混じりの文字列に変換する機能である。日本語入力方式としてはかな漢字変換方式のほかにも，多段式キーボードを利用するハードウェアによる方式からTコードや touch 16 などの無想式漢字入力方式まで多様なものが実用化されているが，親指シフトによるかな入力やローマ字入力を初期入力とする漢字変換が広く普及している。

専用ワープロを使っている限りは特定の入力方式しか利用できず，これが独立した機能であると実感できないかもしれないが，パーソナル・コンピュータを文書処理に利用する場合には，MS-DOS などのフロント・エンド・プロセッサとして組み込まれるかな漢字変換機能を利用することになる。ソフトウェアのコンフィギュレーションを変更することによって，同じパーソナル・コンピュータを同じオペレーティング・システムで使用する場合でも，異なるかな漢字変換機能を利用することが可能となる。また逆に，異なるワープロ・ソフトで共通のかな漢字変換の機能を利用できるだけでなく，ワープロ以外の表計算，データベース，通信端末用エミュレータなどといったアプリケーション・ソフトの上でも同じかな漢字変換が利用できる。[15]

ワープロの特徴はむしろ＜＃2＞のテキスト編集機能にある。ここでいうテキスト編集機能としては，例えば特定の部分の（文字単位また行単位の）削除，復活，複写，移動などが中心となる。[16] また，編集対象のテキストの特定の位置まで編集の焦点を移動する機能，特定の文字列を検索して編集の焦点をその位置まで移動する機能，特定の文字列を検索して別の文字列に置き換える機能

などは最低限必要である。複数のテキストを編集対象として扱い，その間で複写や移動ができる，というのも便利な機能である。

＜＃3＞の出力整形機能はワープロのもっとも word processing 的なところであろう。これは，行の始めに句読点が来ては行けないといった禁則処理から始めて，[17] テキストを印刷物としてふさわしい形式に整える機能である。パーソナル・コンピュータ用のワープロ・ソフトであれば，出力に使う用紙の大きさから始めて，上下左右のマージン，字詰め行詰め，字間行間，段組などの指定が可能となっている。また，斜体字や太字といった字種の指定や，拡大文字，網掛けといった飾りの指定が可能な場合も多い。[18]

ここで強調しておきたいのは，専用ワープロではテキスト編集機能とかな漢字変換機能と出力整形機能を別々に意識することは少ないであろうが，これら3者は機能としては独立である，という点である。特に，かな漢字変換機能は入力に関わるので，これなしでテキスト編集機能があっても実際に日本語の文章を入力することはできないが，出力整形機能は編集機能とはまったく独立に考えることができる。実際，こうした機能が比較的独立してさまざまなソフトウェアとして流通しているパーソナル・コンピュータの場合，ワープロとは若干性質の異なるエディタというものが広く使われている。これはテキスト編集機能のみをそなえたソフトウェアなのだが，パーソナル・コンピュータの場合にはかな漢字変換機能はオペレーティング・システムのフロント・エンド・プロセッサとして組み込まれるので，これだけで文章を綴るには十分なのである。

エディタというのは元々はプログラム開発者がプログラムを書く為に利用されるものであったのだが，テキスト編集の機能は十分以上に備えているので，かな漢字変換機能さえあれば，文章を綴る上に不便はない。パーソナル・コンピュータとエディタがあれば，ワープロ・ソフトやプリンタがなくても原稿を書く上で困らない。原稿用紙とペンさえあれば，編集者や印刷所がなくても原稿が書けるようなものである。もっとも，出力して人に見せる必要が生じると，とたんにプリンタが必要になるし，また，行詰め字詰めの指定をしなければい

けないので，ワープロ・ソフトの類を使わなければならなくなることもある。

しかし，最終的にワープロ・ソフトで若干の修正を加えて打ち出すにしても，テキストの編集と出力整形が別物であるという認識は必要である。というのは，LaTeX などの高度な出力整形機能を利用しようとすると，編集しているテキストと最終的な出力が似ても似つかなくなるという状況で文書の編集をすることになるからである。

word processing のシステムには二通りの考え方がある。まず，高品位の出力機の機能を解像度の低い端末装置での編集でも十分に利用できるように，出力機に対する指令をある種の拡張コードとして定義して，入力データを用意するという方針が考えられる。一方解像度の高いディスプレーを備えたワークステーションなどが普及すると，画面で見たままの状態が出力として得られる方が編集作業がわかりやすい，という立場がある。後者のことを What you see is what you get. の頭文字をとって WYSIWIG ということがある。

前者の場合，エディタなどで編集する文書ファイルというのは写植機やプリンタやフォーマッタなどのインタープリタやコンパイラに対するデータの様なものであるが，そこにインタプリタやコンパイラに対するコマンドが含まれている。文書そのものが，実はインタプリタやコンパイラに対するコマンドの集まり，すなわちプログラムであると見ることもできる。LaTeX の場合など，テキスト・ファイルをいったんコンパイラに通し device independent file というオブジェクト・コードを得るという所などますますプログラミングに近い。パーソナル・コンピュータやワークステーションで文書を書くというとなにか高級な機械を本来の用途以外に流用しているような後ろめたい気持ちになることがあるかもしれないが，ある種のプログラミングをしているのであるから，まさに本来の使い方なのである。校正と言うのはデバック作業に他ならない。

計算機というのは＜ハードウェア＋言語処理系＞を一つの「仮想的な機械」として考えるのが分かりやすいと既に述べた。専用ワープロというのは実は，ドット数の比較的多いプリンタが付属している点と，キーボードがそれなりに

使いやすく特殊化してある点を徐けば，内部的にはほとんどパーソナル・コンピュータと変わるところがない。ただ，ワープロという作業に特化した＜コマンド・インタプリタ＞がハードウェアと一体化しているところが特徴である。汎用のパーソナル・コンピュータも，ワープロ・ソフトを起動してしまえば＜ハードウェア＋ワープロ・ソフト＞で全体として仮想的な＜ワープロ＞をなしていると考えてよい。そして，いずれの場合も文書ファイルというのはこの＜仮想的な機械＞に対するプログラムとしてあり，この＜仮想的ワープロ＞が＜解釈と実行＞を行うと，その結果求める文書が出力されるというわけである。

LaTeX などのように，インタプリタに対するコマンドをあらわにユーザ自身が編集できる場合にはこの点はもっと明確に意識できる。しかし，普通のいわゆる＜ワープロ＞の類であっても WYSIWYG の状況であれこれと編集作業を行って望む出力が得られるようなデータの集まりを文書ファイルとして保存するわけであるから，本質的な点で変わりがあるわけではない。

ここで，「文書」という表現が，「文書ファイル」という意味と「印刷出力」という意味とで多義化している点が重要である。ワープロは，「文書ファイル」の指令に従って，紙などの上に白黒の密度分布としての文字を表現する機械であり，インタプリタである。従って，ワープロで書いた「文書」というのは，そのワープロがなければ，あるいは，そのワープロが「文書」を記録する仕様を知らなければ，ただのノイズである。言い方を変えれば，ワープロで書いた「文書」というのは，そのワープロに意図した「文書」を出力させるためのプログラムであるといえる。

もちろん，ここで言う「文書」というのは bit 列としてのそれであり，日常的な意味での「文書」とは大きく異なっている。人間が言語を用いて表現するテキストのある側面は bit 列として表現可能であり，その限りにおいて，計算機にとっても処理が可能となる。しかしながら，我々が普通の日常語で「文書」というときにさす，活字や手書きの文字は計算機にとってはテキスト以前の画像認識の対象としてある。また，抽象的にテキストという時，われわれは単な

る一次元的な流れだけでなく，暗黙の相互参照なども含めたより複雑な構造を想定して理解しているのだが，bit 列としてのテキストは通常の意味でのテキストの極めて部分的な表現にしかなっていないことをわれわれは強く意識する必要がある。計算機の世界でも，ハイパー・テキストと称して，単なる文字列としてのそれを越えたテキスト表現を追求する動きがある。bit 列を越えるテキストというのが，計算機にとってのこれからの課題となろう。

4. まとめ

以上見てきたことをまとめると，次のようになろう。

- 計算機が行っていることを物理的なレベルで記述すればデジタル信号処理であり，それを「計算」と呼ぶか「情報処理」と呼ぶか「テキスト処理」と呼ぶかは語用論の問題である。
- 並列ベクトル計算機やスーパー・コンピュータなどの例外を徐いて，計算機の多くは広義のテキスト処理に使われていると考えてよい。特に，ワークステーション，パーソナル・コンピュータなどは，プログラム開発かアプリケーション・ソフトの利用に使用されるわけだが，前者ではまさにソース・プログラムを対象としたテキスト編集と「言語処理」が中心となるし，後者も表計算，データベースも含め，広義のテキスト処理と考えることができる。
- 計算機を使うためのインタフェースとなる「計算機言語」はオペレーティング・システムのコマンド系も含め，厳密な統語論と意味論を備えた「文法的体系」をなしている。
- 計算機言語（の使用）には ambiguity や状況依存性といった，通例自然言語に関する特徴と考えられている特性が見られる。計算機言語の使用に際して，こうした ambiguity は状況や常識によって文法的に解決される。
- このような「文法的機械」としての特性を備えた計算機をワード・プロセシング，テキスト編集，デスクトップ・パブリッシングなどの自然言語のより高度な伝達に利用するのは極めて自然なことである。

• ワープロなどで作られる文書ファイルは一種のプログラムであり，計算機環境を利用して文章を綴るということはプログラミング作業そのものである。
• しかしながら，現在多くの計算機で扱うことのできるのは，bit 列としてのテキストであり，人間の理解するテキストの持つ多様性，構造性，歴史性などには十分対応できていない。

「計算機」,「電子計算機」,「コンピュータ」というと「計算のための機械」とか「計算にしか使えない機械」という印象を与えることが多いであろう。しかも，ここでいう「計算」というのが,「数字の操作」とか「数値の処理」という連想を呼び起こすため,「コンピュータ」というのは「電卓が自己増殖して複雑化したもの」というイメージで促えられることが多い。確かに「計算機」の機能の本質は「計算」にあるが，それは物理的にはデジタル信号処理であり，非常に一般的な意味での「記号処理」としての計算であって,「数値計算」は計算機によって処理できることがら，あるいは利用法，の内で極めて特殊な一つであるに過ぎない。例えば，銀行のオンライン・システムを利用して現金を口座から引き出す場合でも，伝票を打ち出す時点では，数字はその計量的な意味よりも文字としての意味が重要になっている。その意味で計算機は「計算」ではなく,「文字列処理」をしていることになる。

数値計算の場合も含めて「計算機」を使う上での特質は，それが「文法的機械」である点にある。「計算機」に対する指令は，なんらかの信号系の上の bit 列として「統語的規則」に従って構成されたものでなければならない。故に「計算機」はそれが存在した時から既に，自らに対する指令をある文法規則に従って「解釈」し「実行」する「機械」として存在した。多くの場合，与えられた「計算機」が直接に解釈し得る文法規則によって構成された指令は，人間にとって「意味論的」に把握し難いものであり，そのため，より「高次の言語」を設定し，これを機械が解釈し得る言語に中間的に翻訳する，という処理をすることが多い。こうした意味において，計算機にとって,「言語」を「解釈」し，

あるいは「翻訳」する，というのは，それなしには計算機の存在が不可能となるような，基本的な動作なのである。

そして，人間もまた「文法的機械」である限りにおいて「計算機」であるのかもしれない。

（続く）

注

＊「するか？」と言いきってしまうのは関東方言ないし「標準語」の疑問文としては舌足らずの印象を免れない。関西方言の一種ならびに学生言葉の一部で非難をこめた反語的な表現として使う方が自然であろう。しかしながら，関連する論考や小説などの（翻訳の）タイトルがしばしばこの表現で終っているので，ここでは敢えてこの舌足らずの表現を流用することにする。

なお，本稿は1989年夏に Stanford 大学 CSLI を訪問中に書いた草稿を基に再構成したものである。

(1) これは Philip K. Dick の小説 *Do Androids Dream of Electric Sheep?* [2]の日本語訳の題名である。なお，これを原作とした Ridley Scott 監督の映画 *Blade Runner* はいわゆるサイバーパンク的な雰囲気を先取りした作品としてしばしば言及されるが，あくまでもハリウッド的なＳＦ映画であって，そこには P. K. D. の小説に見られる reality と authenticity に対する不安も William Gibson の小説に見られるような計算機ネットワークと人間の精神的な相互依存に対する関心も見い出すことはできない。

(2) ふつうの日本語では「電子計算機システム」と呼ばれるであろう対象を本稿では仮に「計算機」と呼んでる。これをどのように定義するかはいわゆる「人工知能」の研究方法論ならびに実在論との関係で微妙な問題をはらんでいる。

(3) フランス語では「計算機」のことを ordinateur という見慣れぬ術語で表現するが，この区別は physiciel と logiciel という明快な表現で表している。「デジタル・レコーディング」というとわかったようなわからないような感じになるが，enregistrement numerique といえば，誤解のしようがない。と書いて，綴りが不安になって，[13]を調べてみたところ，ordinateur は出てくるものの，physiciel と logiciel が出てこない。[14]を見ても同様である。[10]をひっくり返すと logiciel は出てくるが physiciel は出てこない。どこで覚えたのか思い出せない。

(4) 言うまでもないことを改めて説明しておくと，カセット（ケーシング）だのテープだのは物理的実在物であって，本来の意味でのソフトウェアはテープに記録された情報内容そのものを指すわけであるから，カセットテープを「ソフトウェア」と呼ぶことは本来の用法から言えば間違いである。しかし，逆に「カセットテープ」という表現で，物理的実在物ではなくそこに記録された内容を指す用法も日本語として不自然ではない。従って，こうした「間違い」を指摘することに，「教育的配慮」以上の意味はない。

(5) 「言葉の乱用によって」というのはフランスの数学者集団ブルバキが，その教科書でよく使用する表現である。比喩ということが数学の理解において本質的であることが，この一事からもよくわかる。

(6) この「特に」という表現は一見すると異様で，「ならびに」でなければ日本語として不自然であるような印象を与えるかも知れないが，メタ議論が人工知能研究の（重要な）一部である以上，適切な用法である。なお，人工知能研究に関する通俗的な入門書としては[6]が読み易い。

(7) もちろん，Turing test がこのように粗っぽい規定をしているわけではない。

(8) 最近の言語学の著作の例にならって，容認可能性については明示しないことにする。

(9) 「解釈」の後に行うのが「鑑賞」ではなく「実行」であるところが「国文学者」と「計算機」の違いである。

(10) エンド・ユーザから見た場合には，OSについては自分が利用したいアプリケーション・プログラムの起動方法とファイル・アクセスの手だてを知ることだけで，基本的には十分である。

(11) この項目で以下に記す事柄は，ここ数年計算機に触れながら考えてきたことに基づいているが，それをまとめながら，Stanford 大学 CSLI の Ivan Sag 教授が resolution problem と称して考察していることがらに酷似していることに気が付いてきた。同じような言語学的バックグラウンドの人間が同じような時期に同じような計算機科学との接触を持ったわけであるから，同じ様な問題意識を持つことも当然であろう。なお，Ivan Sag 教授の resolution problem に関する考察については，project proposal や project report などのほか，[7]の草稿がある。

(12) 機械翻訳に関連した自然言語処理研究の最大の功績は，テキストの理解にいかに「常識」が関与しているかを明らかにしたことであると言ってよいだろう。

(13) MS-DOS に関しては例えば [6] や [7] を参照されたい。ここでの筆者の議論が計算機言語における指示的指標の利用についての議論にかこつけて MS-DOS のファイル・システムの紹介と MS-DOS のコマンド入力についての常識の紹介を試みているのではないかと賢明な読者が考えるとしたら，それは正しい推測であるかも知れない。なお，MS-DOS の操作そのものについては，例えば [12] などを参照されたい。

(14) ここでいうワープロとは専用のハードウェアでもパソコン上のワープロでもワークステーション上のデスクトップ・パブリッシングの類でもかまわないとしておこう。

(15) VAX や SUN など UNIX をオペレーティング・システムとして利用する環境では Wnn の jserver をかな漢字変換に利用することが一般的である。

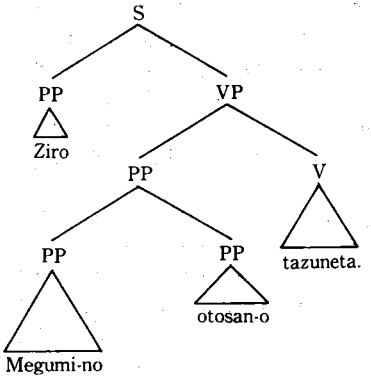
(16) 削除と復活の機能があれば，複写や移動はその組合せとして表現できる。実際 UNIX 上でいちばん広く使われている Emacs というエディタでは複写や移動はおよそその様に利用する。

(17) 英文用なら justification や hyphenation まで含めてより高度な処理が必要となろう。

(18) UNIX などで広く普及している LaTeX というファーマッタでは，斜体，太字，スモールキャピタルなどの字種の指定，文字毎のポイントの指定，段組の指定，数学記号やギリシャ文字などの特殊記号の使用が簡単にできるだけでなく，脚注を組むことができ，しかも注，文献，例題などの番号を自動的に振り，相互参照できるようになっている。詳しくは[5]などを参照して頂きたい。LaTeX はレーザビームプリンタがあれば，PC-9801 などのパーソナル・コンピュータ・システムでも利用できる模様である。

例えば次頁の(i)のような図版を LaTeX で組みたければ(i')のようにテキスト・ファイルを用意して「計算機」上で若干の処理を行ったのち，その出力データをレーザ・ビーム・プリンタなどに送ればよい。ちなみに，これは本文（に若干のコマンドを含めたもの）の中に含まれるのだが，出力のための処理を行う段階で，この図が複数のページや段などにわたることのないように，本文中の位置の若干の調節と節の切れ目のスペースの調節などが自動的に行われる。

(i)



(i')

```
\enumsentence {
\unitlength=.1 ex
\tree {
\node {S}
 {\larc {13}   \node {PP}   \Triangle 7 {Ziro}}
\rarc {13}
        \node {VP}
         {\larc {13}
                \node {PP}
                 {\larc {13}   \node {PP}   \Triangle 7 {Megumi-no}}
                 \rarc {13}   \node {PP}   \Triangle {10} {otosan-o}
        }
        \rarc {13}
                \node {V}   \Triangle 7 {tazuneta\rlap {.}}}}
}}
```

また，次頁の(ii)の20に見られるような式を含む引用は(ii')のような指示で得られる。これは本文に対する脚注として付けたものであるが，本文の中の脚注を指示する番号が置かれるべき箇所の直後にこのままで用意すると，脚注の番号を自動的に振った上でこの脚注相当部分が全体として紙面上の然るべき所に印字されるように処理が行われる。(なお，文献からの引用の部分では当然のことながら規則の番号をテキストとして指定している)

(ii)

[19]See the discussion in [2] for restrictions on right node raising.

[20]This is essentially a reformulation of (116) in [13] (pp. 161).

(116) $V^2[\text{CONJ } \alpha] \rightarrow \alpha, X^{2+}$ where $\alpha \in$ {*and, but, nor, or*}.

With respect to semantic interpretation of this construction, they state as follows;

> The semantic interpretation for structures admitted by (116) may be given by the rule informally stated in (119):
>
> (119) The interpretation of an elliptical construction is obtained by uniformly substituting its immediate constituents into some immediately preceding structure and computing the interpretation of the results.

(ii')

```
\footnote {\leftmarginii=4em
This is essentially a reformulation of (116) in\cite {coordination} (p. 161)
\begin {quote} \begin {list} {} {}
\item [(116)] V$^2$[CONJ $\alpha$] \ra\ $\alpha$, X$^{2+}$ \hsp where
\hsp $\alpha \in$ \ {\em and, but, nor, or}\ . \end {list} \end {quote}
With respect to semantic interpretation of this construction, they state as follows;
\begin {quote} The semantic interpretation for structures admitted by (116) may be given by the rule informally stated in (119):
\begin {list} {} {} \item [(119)] The interpretation of an elliptical construction is obtained by uniformly substituting its immediate constituents into some immediately preceding structure, and computing the interpretation of the results. \end {list} \end {quote} }
```

参考文献

[1] Steward Brand. *The Media Lab – inventing the future at M.I.T.*, Penguin Books, 1988.

[2] Philip K. Dick, *Do Androids Dream of Electric Sheep?*, 1968.

[3] Douglas R. Hofstadter, "The Turing Test: A Coffeehouse Conversation", in [4].

[4] *The Minds's I – Fantasies and reflection on self and soul*, composed and arranged by Douglas R. Hofstadter and Daniel C. Dennett, Basic Books, Inc. 1981.

[5] Leslie Lamport, *LaTeX: A Document Preparation System*, Addeison-Wesley Publishing Company, 1986.

[6] Henry C. Mishkoff, *Understanding Artificial Intelligence*, Howard W. Sams & Co., Indianapolis, 1985.

[7] Ivan A. Sag, "The Resolution Problem for Natural Language Processing", MS, 1988.

[8] John R. Searle, "Minds, Brains, and Programs", in [4].

[9] J. R. サール,「論争――機械はものを考えるか No: プログラムは記号でしかない」, サイエンス Scientific American (日本版), 株式会社 日経サイエンス, 1990.

[10] Judy Tatchell & Bill Bennett, *Introduction à la micro informatique*, Hachette, 1983.
[11] Alan M. Turing, "Computing Machinery and Intelligence," Mind, LIX.
[12] 標準 MS-DOS ハンドブック，アスキー出版局編著，アスキー出版局，1984.
[13] クラウン仏和辞典，三省堂，1978.
[14] 仏和理工辞典 三訂正増補版，白水社，1985.