SKY 1996 Yearbook of the Linguistic Association of Finland. Pp. 53-64.

Esa Itkonen

Is There a 'Computational Paradigm' within Linguistics?

The advent of computers has considerably changed the landscape of linguistics, as shown by the increasing number of publications and conferences devoted solely to computational linguistics. The emergence of this subdiscipline is not due to the fact that a heretofore neglected realm of linguistic phenomena would finally have attained the recognition that it deserves (as has happened e.g. with neurolinguistics or pidgin-and-creole studies). If this were the case, then computational linguistics would simply occupy one more or less well-defined subdomain within the overall field of linguistics. However, computational linguistics purports to investigate those *same* phenomena (roughly: sentences and texts) that have been investigated since the inception of linguistics. The question now has to be asked whether this new way of seeing differs from the old one sufficiently much to justify speaking of a *paradigmatic shift* (cf. Winograd 1983: 13–22).

Before proceeding any farther, however, I have to answer a possible objection. It is sometimes claimed that a given theoretical framework, whether or not we choose to call it a 'paradigm', literally *constitutes* its own data. As can be seen from the preceding paragraph, I do not share this view. (This entails that in my opinion Saussure asserted only a half-truth in claiming that "c'est le point de vue qui crée l'objet".) There is always an *atheoretical* (or pretheoretical) level at which it makes perfect sense to speak of *different* theories (or 'paradigms') dealing with the *same* data. In practice no one has ever doubted this, even if in one's philosophical moments one might feel tempted to endorse the existence of

of the different frameworks (cf. Itkonen 1991: 325–328). For instance, it would be perverse to deny that one and the same set of sentences may be analyzed in dissimilar ways by representatives of different schools.

Now let us see whether there is any justification for speaking of a computational paradigm. In presenting my argument, I shall make use of a very simple artificial language L, namely a language whose sentences are of the form $(ab)^n$. (This is, in the present context, the 'same data' which the different types of description have to come to grips with.) I do not think, however, that the simplicity of my example undermines my argument. That is, I do not think that making the example increasingly more complex would at any point bring about a *qualitative* change in the mutual relations between the types of descriptions that I shall discuss.

Already in the 'pre-computational' days linguists made use of rewriting rules. Thus, when we have to present a grammar for L within the framework of mainstream, non-computational linguistics, it looks like this:

I) r_1: $S \rightarrow abS$
r_2: $S \rightarrow ab$

A sentence like *ababab* is generated in three steps, namely by applying the rule r_1 twice, and then by applying the rule r_2 (cf. figure 1).

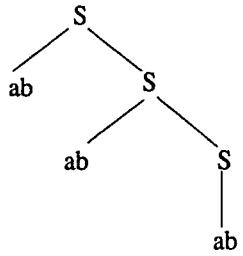

Figure 1.

An even older formalism for describing L is provided by predicate logic (plus the notion of set-membership). In this case the grammar contains one universally quantified implication and one singular statement:

II) $\forall x\,(x \in S \supset abx \in S)$
$ab \in S$

The sentence *ababab* (or, more precisely, the truth that *ababab* is a sentence) is generated by two joint applications of Universal Instantiation and Modus Ponens:

$\forall x\,(x \in S \supset abx \in S)$
$ab \in S$

$abab \in S$

$\forall x\,(x \in S \supset abx \in S)$
$abab \in S$

$ababab \in S$

The grammars I and II are tested in the same way, namely by finding out whether they generate *all* and *only* sentences of L. These two aspects correspond to the notions of *completeness* and *soundness*, as they are used in the theory of logic. On the one hand, we have a sentence of L, and we ask whether it is generated by our grammar. If the answer is 'yes' every time we ask this question, the grammar generates *all* sentences of L, and is therefore complete. On the other hand, we have our grammar, and we ask whether what it generates is a sentence of L. If the answer is 'yes' every time we ask this question, our grammar generates *only* sentences of L, and is therefore sound. (It is of no consequence that L happens to be so simple as to

make, in reality, any questioning superfluous.) The notions (or viewpoints) of soundness and completeness correspond to the notions of 'prediction' and 'explanation', as they are used in the philosophy of the natural sciences (cf. Itkonen 1978: 4–9, 254–253).

All natural-language grammars are tested as to their soundness and completeness, with the qualification that an additional criterion is constituted by the 'adequacy' of the structural descriptions (whatever concrete interpretation is given to this difficult notion). A natural-language grammar is concerned with the (intuitive) correctness of sentences while an axiomatization of formal logic is concerned with the (intuitive) validity of formulae. But apart from this difference, they are tested exactly in the same way. It is crucially important to understand that, apart from such simple cases as propositional or predicate logic, a logical axiomatization is falsified if it, although *provably* sound and complete, fails to be *intuitively* sound and/or complete, i.e. if it generates intuitively non-valid formulae and/or fails to generate intuitively valid formulae. In this respect, axiomatizations of natural sciences differ from either grammatical or logical axiomatizations. They purport to generate all and only *empirically true* sentences (of the relevant domain), but empirical truth, unlike intuitive correctness or intuitive validity, is a property which cannot be assigned to a sentence just by inspecting it (cf. Itkonen 1978: 276–287).

Next, we shall consider a computational grammar of *L*. I submit that computers are taken to have inaugurated a new era in linguistics, mainly because they seem to enable us to have *dynamic* descriptions, where we previously had only *static* descriptions. Previously we investigated linguistic *structures*; but now we investigate linguistic *processes*, more precisely processes of speaking and understanding. (Notice that since cognitive-computational processes apply to structures—and are indeed represented as just successions of structures—, it would be more accurate to speak of 'structures-and-processes', rather than of just 'processes'.) As a consequence, it is particularly important to determine the extent to which this idea of a

'dynamic turn' is justified.

I shall choose PROLOG as the language in which the computational grammar of *L* is couched. The grammar contains one 'rule' and one 'fact':

III) s(ab(X)):–s(X).
s(ab).

As inputs, the grammar may be given two types of 'questions', exemplified here by *A* and *B*:[1]

A) ? s(ab(ab(ab))).
B) ? s(X).

In response to *A*, the grammar produces the output 'yes.' (which means that *ababab* has the property of being a sentence). In response to *B*, it produces the following output:

X = ab;
X = ab(ab);
X = ab(ab(ab));
....

It is clear that *A* and *B* exemplify sentence-recognition and sentence-generation, respectively. They may be considered as loose analogues of the corresponding psychological processes.

A comparison of the grammars I, II, and III reveals both similarities and dissimilarities. On the one hand, the structure of the grammar III is exactly the same as that of the grammar II (and, by implication, that of the grammar I). On the other, in the case of the grammars I and II, it is the grammarian (or the logician) *himself* who has to perform the tasks of sentence-generation and sentence-recognition, whereas in the case of the grammar III, the grammarian needs only to give the input in

[1] I disregard the questions involving the 'anonymous variable' (_).

form of the relevant questions; after this, it is the grammar which performs the tasks of sentence-generation and sentence-recognition.

It seems clear to me that, from the viewpoint of actual descriptive practice, the dissimilarities are outweighed by the similarities. Writing the grammar III is an undertaking qualitatively similar to writing the grammars I and II. Of course, the grammar III contains the idea of a *machine* which performs the tasks assigned to it, but from the viewpoint of the grammarian this idea remains *hidden*. The affinity with mainstream linguistics is enhanced by the fact that the PROLOG notation may be replaced by the rewriting notation, resulting in the 'definite clause grammar' (cf. Pereira & Shieber 1987: 70–79).

Finally, we shall consider a grammar which is literally a *machine*. I have chosen the Turing machine for this purpose. This choice may need some justification.

First, Turing machines are normally regarded as *abstract* machines defining mathematical functions; but it is also possible to regard them as machines in the literal sense of the word. In fact, it is quite convenient to illustrate the notion of *causality* with the aid of a (concrete) Turing machine. On this interpretation, the symbol which the machine reads at t is an *external cause*, while the state in which the machine is at t is an *internal cause*. The combination of these two causes brings about one *internal effect*, namely the state in which the machine is at $t+1$, and two *external effects*, namely the symbol which, having been printed in lieu of the earlier symbol, will be on the tape at $t+1$, and the movement either to the left or to the right (or, perhaps, the halting) which the machine will have performed at $t+1$ (cf. Itkonen 1983: 22, 287–288).

Second, the language L is so simple that it could also be described by a finite-state machine. At present, however, I am not interested in conceptual parsimony, but rather in conceptual clarity; and, as I just noted, the Turing machine is well suited to illustrate the functioning of a causal process.

Third, the grammar which I am about to present handles

only the aspect of sentence-recognition (cf. table 1).

IV)

	q_0	q_1	q_2
A	R		
a	q_1, R	q_2, R	R
b	q_2, R	q_0, R	R
B	1, stop	0, stop	0, stop

Table 1.

In the 'machine-table' (see table 1) the column outside the rectangle contains the 'external causes', i.e. the symbols on the tape, whereas the row outside the rectangle contains the 'internal causes', i.e. the machine-states. The effects of these pairs of causes are located inside the rectangle, in such a way that non-changes are not explicitly mentioned. For instance, the machine starts in q_0, reading A. As a result, it replaces A by A (i.e. leaves it unchanged), enters q_0 (i.e. remains in it), and moves one step to the right.— The only new symbols that are printed (in lieu of B) are 1 and 0, which mean 'yes' and 'no', respectively.

Let us see how the grammar IV recognizes that *ababab* is a sentence, whereas *abbaba*, *aababa*, and *a* are not sentences (cf. table 2).

When there is a correct sentence on the tape, the machine moves from A to B, while alternating between q_0 and q_1, replaces B by 1, and stops. When the machine encounters an error, i.e. when it reads either b in q_0 or a in q_1, it enters q_2, remains in it until it reaches B, replaces B by 0, and stops. The sentence a, which entails reading B in q_1, constitutes an error type of its own.

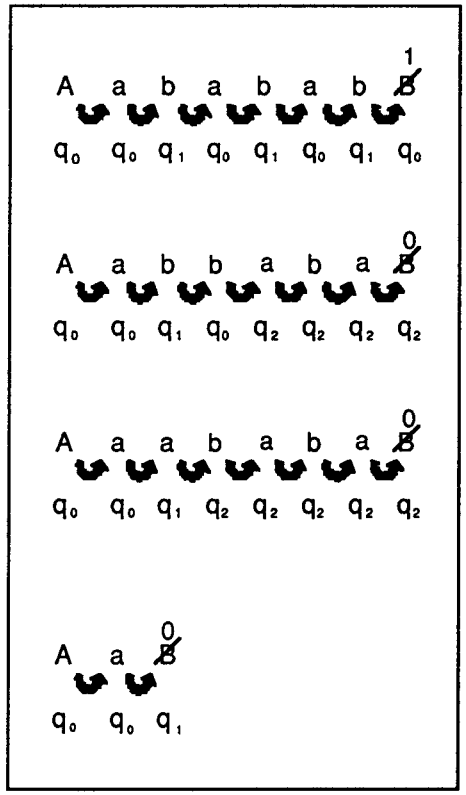

Table 2.

When we compare the grammar IV to the other three grammars, we finally discover a genuine dissimilarity. The grammar IV is dynamic, in the sense of describing a temporal process; it incorporates the notion of *change of state*. By contrast, the grammars I, II, and III describe static, atemporal structures; any processes, whether they are performed by the grammarian or by the computer, remain hidden. To put it in a nutshell, I claim that the notion of a computational *paradigm* in

linguistics is justified only to the extent that computer-based descriptions resemble the grammar IV.

Once I have stated my basic claim, I must immediately qualify it somewhat. It is quite clear that PROLOG programming *implicitly* contains the idea of a process. Answering a question means either *proving* the sentence asked (as in 'understanding') or *proving* the sentences resulting from replacing the variable(s) in the sentence asked by constants (as in 'production'); and this happens by *searching* and *finding* facts that qualify as instantiations of the bodies of rules (i.e. of the antecedent clauses of universal implications). The programmer knows that rules are read from top to bottom and that conjunctions in the bodies of rules are read from left to right; and observing the correct order may make the difference between a program that works and one that does not. This knowledge may, however, be incorporated into the grammatical conventions. Therefore it still remains the case that it is possible to practice PROLOG-based grammatical description without having any very clear notion of the computational processes involved.

The point that I have made here rather informally has been made quite explicitly, and at a more general level, by Petre & Winder (1990). They discuss the difference between *declarative* and *imperative* computer languages, which roughly corresponds to the difference between our grammars III and IV. A declarative language specifies *what* is the problem to be solved, without indicating *how* it is solved. Because "declaration, by nature, excludes algorithm" (p. 176), the solution of the *how*-question is deferred to the language implementation. Imperative languages, by contrast, still reflect the basic machine operations: as a set of instructions, they *show* the process of computation. Between pure types of imperative and declarative languages there are intermediate types, i.e. declarative languages with an "expression of algorithmic intent". Now it is precisely Petre & Winder's (1990) claim that the 'imperative vs. declarative' distinction should be thought of as a *continuum*; and they proceed to place various programming languages on it. For instance, FORTRAN is a typical imperative language; LISP is

situated exactly at the 'imperative vs. declarative' divide; PROLOG lies somewhere between LISP and typical declarative languages. Peter & Winder reach this conclusion: "The basic difference between programming styles lies in the hiding of the computational model" (p. 180). This entails, interestingly enough, that it is questionable whether PROLOG-based descriptions should be considered as part of *computational* linguistics at all.

Thus, whether or not computers have brought about a change of perspective within linguistics, is a *matter of degree*, and this in a *twofold* sense. First, one language (e.g. PROLOG) may be more 'conservative' than another (e.g. LISP). Second, even within one and the same language the grammarian may be more or less aware of the (more or less) hidden computational process. I submit that the *gradual* nature of this change of perspective does not agree too well with the way that paradigmatic shifts are generally conceived of. More precisely, some computer linguists (i.e. those operating with languages close to the imperative end of the continuum) may have experienced a genuine paradigmatic shift in their way of thinking, but others may not.

I shall conclude this paper with a few remarks of a more general nature, relating the preceding results to questions of language use and pragmatics. First, the distinction between declarative and imperative languages is analogous to that between logical, set-theoretic semantics and algorithmic or procedural semantics. In Itkonen (1983: 149–151, 311–313) I argued that, in terms of psychological import, algorithmic models are more informative than logical ones; and this conclusion may now be generalized so as to apply to the distinction between imperative and declarative languages.

Second, algorithmic *semantics* is a misnomer to the extent that the computational processes are meant to be even rough analogues of *cognitive* processes, simply because this amounts to committing the psychologistic fallacy (*op. cit.* p. 313). That this fallacy is nearly ubiquitous in today's cognitive science, does nothing to lessen its objectionable character. This also means

that any comparison between Turing and Wittgenstein (see e.g. Leiber 1991: 81–88) is misconceived, unless the processes that the machine performs are thought of as (embedded in) *public* actions governed by *socially* valid norms.

Third, Leiber's (1991: Ch.10) attempt to align Chomsky with Turing and Wittgenstein is misconceived *tout court*. In addition to the fact that he is interested in mental structure, not in mental process (cf. Jackendoff 1987: 38–39), Chomsky has consistently denied the relevance of behavioral, public criteria, thus explicitly opposing the Wittgensteinian position (cf. Itkonen 1983: 227–233, 243–248).

It is often said that computers may simulate processes of any kind, whether physical, psychological, or social. It should be clearly understood, however, that computers may be, and typically are, quite unable to simulate those surroundings and/or accompaniments which, to a large extent, constitute a process as what it is. Therefore a program alone is seldom enough.

References

Itkonen, Esa (1978) *Grammatical Theory and Metascience*. Amsterdam: Benjamins.

Itkonen, Esa (1983) *Causality in Linguistic Theory*. London: Croom Helm.

Itkonen, Esa (1991) *Universal History of Linguistics: India, China, Arabia, Europe*. Amsterdam: Benjamins.

Jackendoff, Ray (1987) *Consciousness and the Computational Mind*. Cambridge, MA: MIT Press.

Leiber, Justin (1991) *An Invitation to Cognitive Science*. Oxford: Blackwell.

Pereira, Fernando & Shieber, Stuart (1987) *Prolog and Natural-Language Analysis*. Stanford: CSLI.

Petre, Marian & Winder, R. (1990) On Languages, Models, and Programming Styles. *The Computer Journal* 33: 173–180.

Winograd, Terry (1983) *Language as a Cognitive Process*. Reading, MA: Addison-Wesley.