Ariel Transmission Coversheet

www.infotrieve.com/ariel

NATIONAL LIBRARY OF NEW ZEALAND

Te Puna Interloan Batch Report

University of Waikato Central Library (HU)

Request date: 13-Apr-2008 11:12

Interloan no:	3216747	TGQ:	185491
Service type:	Copy non returnable	Service level:	Normal
Media type:	Photocopy	Supply before:	20-Apr-2008
Call number:	P299.Q3G465 1987 OC storage.		

Author:	Gardenfors, Peter.
Title:	Generalized quantifiers : linguistic and logical approaches
Publisher:	D. Reidel Pub. Co.;Sold and distributed in the U.S.A. and Canada by Kluwer Academic Publishers
Pub. place & date:	Dordrecht;Boston;Norwell, MA, U.S.A., c1987.
ISBN / ISSN:	1556080174 : $74.00 (U.S.)
Series:	Studies in linguistics and philosophy ; v. 31
Details of article:	Author: Johan van Benthem Title: Toward a Computational Semantics
Issue:	Pages: 31-71
Verify source:	National Library of New Zealand/VOYAGER
Copyright decl:	This copy request is for the private study or research of the above user, or complies with s54 or s55 of the Copyright Act 1994.

Requester	University of Auckland General Library (UA)		
Delivery address:	University of Auckland General Library 5 Alfred Street Private Bag 92019 Auckland	Hold:	No
		Maximum cost:	NZ$20.00
		Delivery method:	
		Recip. agreement	2
		Payment method:	IBS billing
Fax:	09 373 7092		
Phone:	09 373 7599		
Email:	au.interloans@auckland.ac.nz		

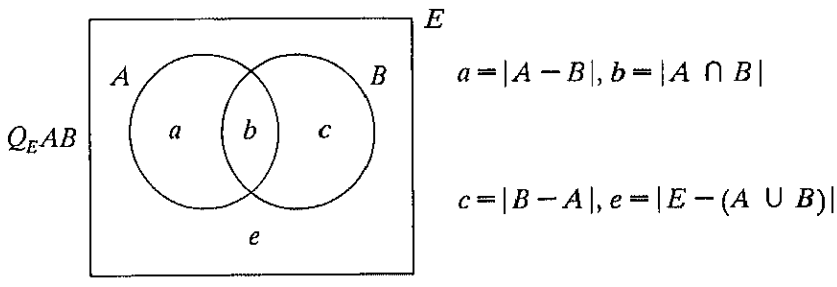
JOHAN VAN BENTHEM

TOWARDS A COMPUTATIONAL SEMANTICS

1. INTRODUCTION

In ordinary model-theoretic semantics, set-theoretic denotations are assigned to linguistic expressions without regard to computational complexity. Yet, there is a reasonable *prima facie* case for the assumption that at least basic items in natural language correspond to simple procedures, that are easy to learn. What is needed to investigate such ideas is a way of thinking 'procedurally' about the usual semantic denotations. In line with our earlier paper van Benthem 1984b, the basic notion here is that of an *automaton*, as developed extensively in mathematical linguistics (see Hopcroft & Ullman 1979). Thus, one of the main supports of formal syntax is enlisted in the service of semantics.

The prime example in the above-mentioned paper was the computation of *generalized quantifiers*, viewed as relations Q between subsets A, B of universes of individuals E:



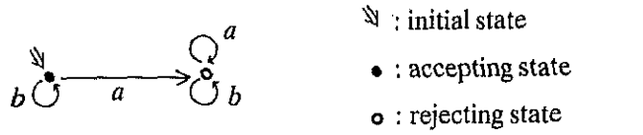
An automaton for Q searches through all individuals in E, marked for their membership of the four relevant 'slots', say, using symbols a, b, c, e [alternatively, the automaton might perform A, B-tests on actual individuals], and accepts or rejects when E has been enumerated completely.

Indeed, under the usual assumptions on quantifiers Q, being

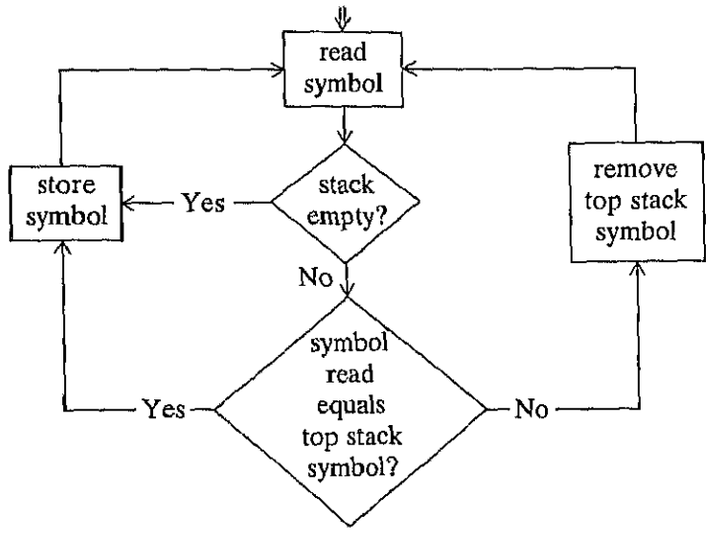
Peter Gärdenfors (ed.), Generalized Quantifiers, 31–71.

Quantity (Q_EAB depends only on the cardinalities a, b, c, e), *Extension* (Q_EAB depends only on $A \cup B$) and *Conservativity* (Q_EAB holds if and only if $Q_EA(B \cap A)$), only the labels 'a' and 'b' will matter. Of course, many generalizations are possible subsequently.

EXAMPLE. A finite state machine computing *all A* are *B*:



EXAMPLE. A push-down store automaton computing *most A* are *B*:



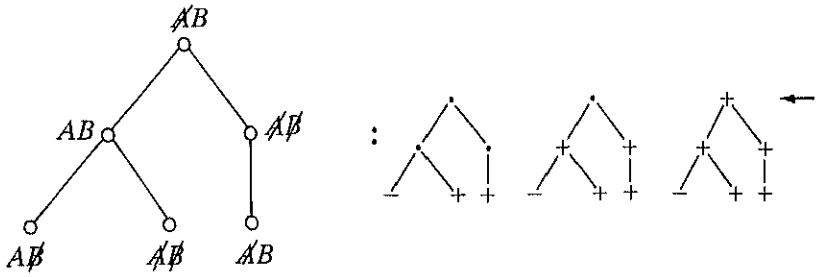
Recognition: at the end of a sequence read, if the stack contains only symbols *b*.

EXAMPLE. A tree automaton computing *if A, then B*: (in the sense of 'all *A*-worlds in the possible worlds tree which lie closest to the root [modelling the 'vantage point'] are *B*-worlds')

> move upward from the leaves towards the root,
> leaving markers +/− ('accept', 'reject') on nodes,
> according to the instruction:

check features of the current node:
— if A, then, if B: write +
else : write −
— if not A, then, if all daughters have +
already: write +
else : write −

For instance:



In this third case, the relation computed is not 'quantitative', as the pattern of the individuals in the universe surveyed is crucial to the evaluation: conditionals are analogous to, but not quite identical with quantifiers.

In this paper, both quantitative and nonquantitative automata will be studied, starting with the former.

2. FINITE STATE MACHINES

2.1. *Permutation Invariance*

Many quantifiers can already be computed at the lowest level of the automata hierarchy, by means of finite state machines. The above example of *all* can be taken as a paradigm for computing, e.g., *some, no, not all,* but also, allowing more than two states, *one, two, three, . . . , all but one, ... , between three and nine, . . .* , etcetera.

One immediate effect of Quantity is *permutation-closure* of the 'languages' (E-descriptions) accepted by such automata: if, e.g., *abbaaba* is accepted, then so are all strings with 4 *a*'s and 3 *b*'s. In the given transition graph for *all,* this property follows from *permutation-invariance* of the machine itself: if reading some string of *a, b* takes the

machine from a state q_1 to a state q_2, then reading any permutation of that string will result in the same state transition.

THEOREM. *The permutation-closed regular languages are precisely those recognizable by some permutation-invariant finite state machine.*

Proof. Permutation-invariant machines obviously recognize permutation-closed languages. (Look at trajectories from the initial state to accepting states.)

Conversely, let L be a permutation-closed regular language. Define an equivalence relation $\sim_L$ among strings as follows:

$$s \sim_L t \text{ if, for all strings } u,\ s^\frown u \in L \Leftrightarrow t^\frown u \in L.$$

There will be only finitely many equivalence classes $\ulcorner s \urcorner$ (for a regular language, that is), which can be taken as 'states', with a transition convention

$$\ulcorner s \urcorner \xrightarrow{a} \ulcorner s^\frown \langle a \rangle \urcorner.$$

Accepting states $\ulcorner s \urcorner$ are those having $s \in L$; the initial state is $\ulcorner \langle\ \rangle \urcorner$. By the Nerode Theorem, this automaton recognizes precisely the strings in L. Moreover, if L is permutation-closed, then it will even be permutation-invariant:

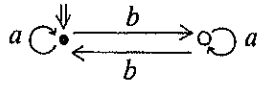
Let α be a sequence $\langle a_1, \ldots, a_k \rangle$, sending $\ulcorner s \urcorner$ to $\ulcorner s^\frown \alpha \urcorner$. Now, let α' be any permutation of α. It will send $\ulcorner s \urcorner$ to $\ulcorner s^\frown \alpha' \urcorner$, which equals $\ulcorner s^\frown \alpha \urcorner$ (and we are done). The latter equality follows from $s^\frown \alpha \sim_L s^\frown \alpha'$: if u is any string, $s^\frown \alpha'^\frown u$ is a permutation of $s^\frown \alpha^\frown u$, and hence $s^\frown \alpha'^\frown u \in L \Leftrightarrow s^\frown \alpha^\frown u \in L$. ∎

Natural language also has examples of 'ordinal' uses of quantifiers, where the order of inspection is important, such as "Every third prisoner was beheaded", "You will make a dollar for every hundred words in the manuscript". Moreover, there are so-called 'branching' uses, where parallel enumeration seems to occur: "Most girls in this corner and most boys in that corner hate each other". The automata perspective certainly seems capable of handling such cases; but, they will not be pursued here. (Another example with such a more 'dynamic' flavour: the so-called 'cumulative' reading of "Five authors produed one hundred poems".)

2.2. *First-Order Quantifiers*

The earlier examples were all *first-order*, differing only in the number of states for their computation. (E.g., *one* requires at least three states.) But, there are other finite state quantifiers too.

EXAMPLE. A finite state machine computing *an even number of A* are *B*:



A conspicuous difference with the earlier diagram is the occurrence of a '2-cycle' here, needed for keeping track of the required periodicity. First-order quantifiers lack such devices (except for 1-cycles): their associated machines can be taken to be *acyclic*.

THEOREM. *The first-order quantifiers are precisely those which can be computed by permutation-invariant acyclic finite state machines.*

The proof exploits the special geometric form of first-order quantifiers, when represented in the Tree of Numbers (van Benthem 1984b). This is only one instance of a parallel between truth-value patterns for quantifiers (as displayed in that Tree) and an automaton representation. Another example are the various notions of 'homogeneity' for such patterns (cf. van Benthem 1985a), which can be matched with finite state computability using varying numbers of states.

2.3. *Testable Languages*

The above characteristic of first-orderness has an independent motivation, as will be illustrated by connecting it up with the main notion of McNaughton & Papert (1971), a monograph devoted to so-called *testable* languages — for which the authors claim special psychological relevance.

First, some definitions are needed. A language L is 'k-testable' ($k \geqslant 1$) if it is closed under the following equivalence relation:

$s \sim_k t$ if (i) s, t have length $\leqslant 2k$ and are identical, or else
(ii) s, t have length $> 2k$ and they have
(a) identical initial segments of length k,

(b) identical final segments of length k,
(c) identical 'occurrence sets' of sequences of length k in between their first and last elements.

For instance, $111000 \sim_2 11110000$, $1100100 \sim_2 1100100100$. The intuitive idea is that recognition of such languages only requires 'local tests' on subsequences of fixed length. Next, a language is 'locally testable' if it is k-testable for some k. Finally, a 'testable' language is any construct out of locally testable ones by repeated *Boolean Operations* and *Sequencing* (these being obvious effective operations).

For the special case of quantifiers, e.g., the four members of the Square of Opposition are locally testable (*all, some, no, not all*), in fact, 1-testable. On the other hand, *at least two* already becomes non-locally testable:

the sequence $a^k aba^k$ is not in its language,
whereas its $\sim_k$-equivalent $a^k aba^k baa^k$ is.

The latter quantifier is testable, however, being expressible as the sequence "at least one; at least one". The general situation is as follows.

THEOREM. *The testable languages are precisely those having an acyclic finite state recognizer.*

Proof. First, here is the inclusion $\supseteq$. The set of accepted strings for an acyclic automaton may be described as a finite disjunction (a Boolean Operation) of accepted 'trajectories', one for each accepting state. Because there are no non-unit cycles, each such state accepts only a finite set of trajectories (another disjunction), of a form exemplified in the regular notation

$$a.b.a^*.b.c^*$$

The latter are sequences (another admissible operation) of two types of basic case:

— a single symbol a (and singleton languages are 1-testable)
— 'homogeneous' languages a^* (again, a 1-testable case).

For the reverse inclusion $\subseteq$, some closure properties of 'acyclic languages' are useful. The latter are closed under

(1) *complements*:
reversing accepting and rejecting states introduces no loops;

(2) *intersections*:
the usual construction of a 'product-automaton' out of two separate (acyclic) ones creates no loops.

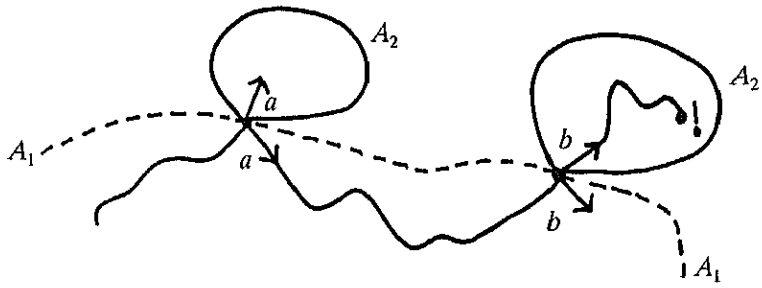
Thus, we have Boolean closure. Moreover, we have closure under

(3) *sequencing*:

Proof. Let A_1 be acyclic, recognizing L_1, and A_2, likewise, L_2. Attach disjoint copies of A_2 to each recognizing state in A_1, fusing the latter with the starting state in the copy of A_2. The new recognizing states will be only those in the A_2-copies. The new automaton has no loops. Also, it recognizes exactly $L_1; L_2$ — be it only *non-deterministically*.

For, everything recognized is evidently in $L_1; L_2$.

Conversely, every sequence in $L_1; L_2$ can be recognized by a judicious sequence of moves, choosing the right moment to enter a copy of A_2:



To return to deterministic recognition, a final lemma is needed. ■

(4) languages with a non-deterministic recognizing acyclic automaton also have a deterministic acyclic recognizer.

Proof. The usual construction of a deterministic 'power-set automaton' for the same language introduces no new loops. (Recall that $X_1 \xrightarrow{a} X_2$ if X_2 is the set of all states reachable from some state in X_1 by reading a, in the old automaton. Then, a loop $X_1 \to X_2 \to \ldots \to X_1$ would always presuppose the existence of a non-trivial loop between single states as well.) ■

Already, these four observations take care of the recursion steps in the definition of testable languages. There remains the basic step:

LEMMA. Every k-testable language is acyclically recognizable ($k \geqslant 1$).

Proof. Every such language can be described as a finite union (Boolean, and hence admissible) of accepted elementary cases, given by a triple:

- fixed initial sequence of length k,
- fixed final sequence of length k,
- some 'interior set' of k-sequences.

(In addition, there may be some isolated single sequences of length $\leqslant 2k$: which each have an obvious acyclic recognizer.) Now, every sequence satisfying the above triple description is in the intersection (again, Boolean) of three languages which can be acyclically recognized,

- fixed expression $\in$ followed by an arbitrary sequence: this is a sequencing of '$\in$ only' and 'all sequences', both of which are acyclic.
- arbitrary sequence followed by fixed expression $\in$: likewise.
- a finite intersection of sequencings of 'all non-empty sequences' (acyclic), 'one single expression $\in$' (acyclic), 'all non-empty sequences' [this enforces the occurrence of all k-sequences in the 'interior set'] and complements of similar expressions for k-sequences outside of the interior set. ■

This completes the proof of the theorem. ■

In particular, the promised characterization follows:

COROLLARY. *The first order quantifiers correspond exactly to the permutation-closed testable languages.*

Finally, here is a result inspired by a question of Frans Zwarts, who observed that the basic logical quantifiers are locally testable, but hardly any others. (It is Sequencing and Boolean Operations which bring in the other first-order cases.) Using some combinatorial reasoning in the Tree of Numbers, we have the following

THEOREM. *The permutation-closed* k*-testable languages* L *are pre-*

cisely those which can be represented in the Tree of Numbers by an arbitrary top triangle followed by a 'flattened cone' definable by some disjunction of quantifiers in the Square of Opposition.

Proof. Here are two illustrations, for a two-symbol alphabet. First, consider the tree level $2k + 2$, with entries $+/-$ at the positions

$$a^{2k+2}, - \ldots a^{k+1}, b^{k+1} \ldots -, b^{2k+2}$$

The middle entry determines (non-)membership of the language for all sequences with at least $k + 1$ occurrences of a and also for b. To see this, write $a^{k+1}\, b^{k+1}$ as follows: $a^k abb^k$, and observe how arbitrary a, b can be inserted to obtain $\sim_k$-equivalent sequences: $a^k aa^* b^* bb^k$.

Next, at this same level, consider any entry at a^i, b^i, with $i \leqslant k$. This determines the language behaviour of all sequences with additional symbols a, since $a^{k+1}a^*\, a^{k-i+1}b^i \sim_k a^{k+1}a^*a^*\, a^{k-i+1}b^i$. But also, as long as $k \geqslant 1 > 0$, adding symbols b makes no change, by the equivalence $ba^k(ba^k)^*a^m \sim_k ba^k(ba^k)^*(ba^k)^*a^m$ (using the previous observation to disregard additional symbols a). ■

2.4. *Further Topics*

There are various further topics in this area. For instance, the above results can also be obtained for arbitrary *finite alphabets*. E.g., acyclic automata will then recognize finite unions of languages having the following type of description, 'the number of occurrences of a_i equals n_i/ is at least n_i $(1 \leqslant i \leqslant k)$'.

Another interesting case is rather that of *one-symbol* alphabets: the preceding cases seem to reduce to 'compositions' of the latter. All one-symbol languages are permutation-closed, and hence are precisely characterized by their *Parikh-tuples* (cf. van Benthem 1984b): which gives them a canonical representation in terms of 'semi-linear' sets of natural numbers. (This amounts to a reduction in complexity of the usual regular set notation for this case — as one can reduce all nested occurrences of Kleene stars to one single layer.)

Another general theme of interest is the comparison of independently motivated *semantic constraints* on quantifiers with special properties of the machines computing them. For instance, *monotonicity* (i.e., QAB, $B \subseteq B'$ implies QAB') will show up as follows:

whenever a string α makes the machine go from state s_1 to

s_2, and s_2 is accepting, then any string α' obtained from α by replacing symbols a by b will drive the machine from s_1 to an accepting state too.

The proof uses the Nerode representation (cf. Section 2.1). In this way, the question may be shown to be *decidable* if a given finite-state computable quantifier is monotone.

3. PUSH-DOWN STORE AUTOMATA

Computing higher-order quantifiers will in general require the use of machines with memory, in the simplest case: push-down store automata. One first question then becomes how much new power of expressing quantifiers has been added in this way — say, viewed as numerical relations on the occurrence numbers for the symbols *a, b*. The answer is in van Benthem 1984b:

THEOREM. *The binary quantifiers computable by push-down store automata correspond precisely to those relations on* a, b *which are definable in purely additive first-order Pressburger Arithmetic.*

This description still leaves a multitude of theoretical possibilities, only a few of which are realized in natural language. Here, the earlier example *most* exemplifies a general tendency toward 'proportionality', a notion which also seems to lie behind the intuitively most plausible readings for, e.g., *many, few*. Therefore, additional constraints were formulated in the above-mentioned paper, leading to a typical result such as the following.

THEOREM. *The bilinear continuous quantifiers (among the above) are precisely those in the following 'Squares of Opposition'* ($n \geqslant 0$):

at least $\frac{1}{n+1}$, *at most* $\frac{n}{n+1}$,

more than $\frac{n}{n+1}$, *less than* $\frac{1}{n+1}$.

Actually, it would perhaps be preferable to have these additional constraints in the form of restrictions on the automata themselves. (See Section 8 for some reasonable restrictions on the action of push-down

store automata.) For instance, returning to more general semantic constraints,

> when does a given push-down store automaton recognize a *permutation-closed* language? a *monotone* language? And, are these properties *decidable*?

One striking feature of this area of complexity is that many such questions will indeed be decidable, because Pressburger Arithmetic is a decidable subtheory of full arithmetic (the latter being undecidable, and indeed highly complex, by Gödel's Incompleteness Theorem). For instance, with a quantifier presented as a push-down store automaton M_Q, we can effectively determine its additive equivalent μ_Q, and then check if, say,

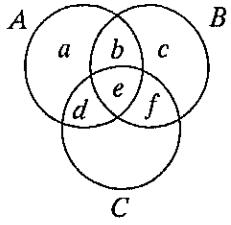
$\forall aba'b'((a + b = a' + b' \wedge a' \leq a \wedge \mu_Q(a, b)) \rightarrow \mu_Q(a', b'))$ (Monotonicity)
$\forall ab(\mu_{Q_1}(a, b) \leftrightarrow \mu_{Q_2}(a, b))$ (Equivalence).

The full power of arithmetic will only be involved with quantifiers (or other linguistic expressions) requiring *multiplication* in their meaning. This has indeed been claimed for certain readings of e.g., *many*, or the modifier *very*. But, the case is far from being conclusive.

Another possible extension beyond the present area is more plausible. The above characterization of push-down computability works only for *two-symbol* alphabets. Higher-up, non-context-free cases arise, even in additive arithmetic, such as '$a = b = c$': representing the (context-sensitive) language of all sequences with equal numbers of occurrences of the symbols *a, b, c*. Do such cases occur naturally in ordinary language?

EXAMPLE. Polyadic determiners (see Keenan & Moss 1985). Constructions such as *more A than B* are *C* require a six-element alphabet, at least, assuming the appropriate form of Conservativity here:

$QAB, C \Leftrightarrow QAB, C \cap (A \cup B)$.



The numerical condition here remains push-down computable, however: $d > f$. Similarly, *as many A as B* are *C* becomes $d = f$. Only the iterated *as many A as B as C* are *D* leads to a more complex type: and this is certainly of doubtful grammaticality.

Still, it should be clear that nothing hinges on a general restriction to push-down computability: the purpose of this paper is rather to point out where 'jumps' in complexity arise in semantics.

In any case, the main thrust from now on is not toward the study of more complex machine action, but toward the topic of simple machines operating on more complex data than the present linear sequences of symbols.

4. TREE AUTOMATA

4.1. *Setting up the Format*

Many expressions operate on 'structured' domains. For instance, intensional operators were computed on possible worlds graphs in Section 1. This is a frequent phenomenon in semantics: one has a semantic domain in the form of a relational graph, which the automaton is to survey in some order. (For a non-intensional example, think of a comparative order when computing measure adjectives, or some 'possession order' when computing possessive expressions (*my*, *'s*).) One important case which admits of a rather straightforward kind of survey is that of (finite) *acyclic graphs*, or even just *finite trees*. Here, an automaton can start at the bottom leaves, working its way up to the top node in an obvious inductive manner. Trees are widespread, also in semantic modelling — and so we shall study this special case here, as a pilot example for the feasibility of our enterprise. (But of course, in the end, one should be able to operate on arbitrary graphs).

A similar move from linear sequences to trees has been made already in mathematical linguistics (cf. Perrault 1984). The two cases are not completely analogous, however, in that syntactic structure trees usually have a fixed set of branching patterns (unary, binary, perhaps ternary) — whereas no such constraints need be expected for semantic structures. Moreover, left-right order of descendants carries no semantic information, whereas it may do so in syntax.

The simplest kind of automaton to be of semantic use on trees

operates as follows. There is a facility for testing if nodes carry certain features, as well as a finite state machine inspecting final states already reached on immediate descendants, and finally, a device for printing state markers on nodes surveyed. The total procedure is as follows:

- the tree is surveyed, level by level, starting at the bottom leaves,
- when a node is inspected, its features are checked, to determine in which state to start the finite state automaton surveying its immediate descendants (or rather, the string of final state markers left there after the preceding round),
- the final state reached by the automaton is printed on the current node.

Thus, there is a 'conditional program' here: "if features$_1$, then do M_1; if features$_2$, then do M_2; etcetera". Then, the automaton M may be just a disconnected union of components $M_1, M_2, \ldots$.

EXAMPLE. Computing the property of 'alternation':

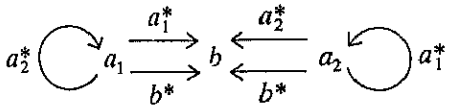
$$\forall x(Ax \to \forall y(R^+xy \to \neg Ay)) \wedge \forall x(\neg Ax \to \forall y(R^+xy \to Ay)).$$

Here, quantification is over nodes in the tree, which can carry one feature (A). 'R' expresses dominance of nodes, 'R^+' immediate dominance.

Our machine has three states:

a_1 ('accept, with top node A')
a_2 ('accept, with top node $\neg A$')
b (reject').

Its diagram is this (with markers q^* for state q):



Here, b is an absorbing state. The starting convention is this: 'with feature A, start at a_1; without feature A, start at a_2'.

4.2. *An Alternative*

As is usual with automata, there are attractive variants of the above set-up. For instance, at each new level, one could let the machine search

first through all final state markers on immediate predecessors (starting from some fixed initial state), looking at the features of the current node only afterwards, to decide the 'exit state'.

Every property of trees recognized in the original format can be recognized with the new one. For, let a 'feature to entry' convention be given for a machine M (old-style). In the new format, we cannot influence where to start our machine: it always departs from the same initial state — but, we have the power to 'interpret' its final states. So, take a new automaton whose states are n-tuples of former states in M (with n the total number of old states). Transitions are the obvious ones: copy the old transitions for the marker read, coordinate-wise. Then, final states encode all outcomes for the original M, from all its possible starting states, with respect to the current input. So, the final convention only has to let the features of the current node pick out the final state at the coordinate given by its former entry convention. Actually, this does require a liberalization of the earlier scheme, in that the markers printed do not correspond one-to-one with the new states. But in any case, separating 'states' from 'auxiliary output symbols' seems a reasonable policy — to be followed henceforth.

With this more liberal perspective, a converse simulation is possible too. States now become couples of (old state, initial feature), with transitions computed as before, on the first coordinate. We enter the machine in (initial state, feature read), and then use the former 'exit convention' to decide the eventual (state)marker to be printed.

5. RECOGNITION AND RECURSION

5.1. *Second-Order Definitions*

Any given tree automaton M obviously recognizes a definite property π_M of trees, whose extension is the class of trees accepted by M. (Here, we assume the original format of Section 4.1.) Can this property also be described in a more explicit manner? At least, there is a straightforward way of describing the machine action in a fragment of higher-order logic, using only sentences of the form

$$\exists Q_1 \ldots \exists Q_k \phi(R, S, A_1, \ldots, A_n, Q_1, \ldots, Q_k),$$

where 'Q_1', ..., 'Q_k' are unary set variables, and ϕ is a first-order

sentence involving the dominance order R, some linear ordering S, $A_1, \ldots, A_n$ (viewed as 'feature properties'), and $Q_1, \ldots, Q_k$ (now viewed as 'auxiliary properties'). Such sentences are called *monadic* Σ^1_1-*sentences.*

THEOREM. *For every finite state tree automaton* M, *its property computed* (π_M) *is monadic* Σ^1_1.

Proof. Our task is just to check that everything explained in Section 4 can be expressed in this formalism. Here are the main steps.

(i) The tree has really just the structure type (T, R, $A_1, \ldots, A_n$), where the latter induce 2^n exhaustive 'features' (or 'node descriptions'). The latter each have an associated state in M, via the 'entry convention'.

Notation: feature set F, state set Q,
for each $f \in F$, there is an associated $q(f) \in Q$.

(ii) The linear order S represents one particular way of enumerating the tree, which induces an order on the immediate predecessors of any given node x.

Define: $F_x(y)$: = 'y is S-first among x's immediate predecessors',
$L_x(y)$: = 'y is S-last in that set'
$N_x(y_1, y_2)$: = 'y_1 S-precedes y_2 immediately in that set'.

(iii) Now, for each state $q \in Q$, take two unary predicate letters,

q^*x (intuitively, 'x finally gets Q-marker printed')
qx (intuitively, 'M has state Q immediately after inspecting x').

State that: both the q^*'s and q's partition T exhaustively.

(iv) 'Bottom leaves':

state that: $\forall x(\neg\exists y Rxy \rightarrow \bigvee_{f \in F} (f(x) \wedge q(f)^*(x)))$.

(The initial state here is also the final state; as there are no predecessors.)

(v) 'Climbing the tree':

state that: $\forall x(\exists y Rxy \rightarrow \bigvee_{f \in F} (f(x) \wedge \bigvee_{q \in Q} (\mu(q(f), x, q) \wedge q^*x)))$;

where $\mu(q_1, x, q_2)$ says that, starting in state q_1, M will end in state q_2 after having surveyed all of x's immediate predecessors.

To define this, let $M[q_i, q_j^*]$ be the state assumed by M when reading marker q_j^* in state q_i. Then, we merely write out the transition diagram for M:

$$\exists y(F_x(y) \wedge \bigvee_{q \in Q} (q^*(y) \wedge M[q_1, q^*](y))) \wedge$$

$$\forall y_1 y_2(N_x(y_1, y_2) \rightarrow \bigvee_{q, q' \in Q} (q(y_1) \wedge q'^*(y_2) \wedge$$

$$M[q, q'^*](y_2))) \wedge \exists y(L_x(y) \wedge q_2(y)).$$

(vi) 'Success on top':

state that: in the top node, q^* holds for some accepting state $q \in Q$.

π_M can now be presented as the conjunction of all these (first-order) statements, prefixed by existential quantifiers over the unary predicates $q, q^*(q \in Q)$. ■

If one wishes, the linear order S can also be quantified away existentially in the above formula π_M.

A somewhat greater change would be necessary if the machine is allowed to inspect, not just the immediate predecessors, but all predecessors of a given node. In that case, many 'visits' will occur, and the global q/q^*-trick does not work. One solution is then to postulate the existence of suitable predicates q (representing M's inspection) 'locally' at each node. Again, second-order logic allows transformation into a Σ_1^1-sentence here, through the equivalence

$$\forall x \exists Q^{(1)} \phi(x, Q) \leftrightarrow \exists R^{(2)} \forall x \phi(x, `\lambda y.Rxy').$$

Note, however, that this will introduce second-order quantification over predicates of higher arities, not just sets.

5.2. *Unwinding Recursion*

Still, the above description does not count as a genuine 'explanation' of what a machine M recognizes: it merely restates the recognition procedure. What would be a more satisfactory solution? For instance, if

M is *acyclic*, checking some first-order condition on predecessor states, we would like to obtain some explicit *first-order* definition of π_M also, in terms of the tree order and the feature predicates (such as those occurring in earlier example). This can be quite difficult already.

EXAMPLE. (1) Consider a machine M with the following action:

- if a node has feature A, then it ends up accepting it if and only if no predecessor was accepted,
- if the node lacks feature A, then M accepts it if and only if some predecessor was accepted.

It is not easy to describe π_M (although one can form an impression, by looking at small trees first).

(2) And difficulties increase with more complex (though still first-order) recursions such as

- if feature A, then accept iff exactly one predecessor was accepted,
- otherwise, accept iff no predecessor was accepted.

The general problem is this: we want to turn a *recursive implicit* definition of π_M into an *explicit* one, in the same language used in stating the recursive clauses. The general scheme here is this:

$$\begin{array}{ll} q_1(x) \leftrightarrow & \bigvee_{f \in F} (f(x) \wedge \text{'first-order condition on occurrences} \\ \vdots & \quad \text{of states } q_1, \ldots, q_n \text{ among } x\text{'s predecessors'}) \\ & \vdots \\ q_n(x) \leftrightarrow & \text{[likewise]}. \end{array}$$

This is a simultaneous recursive definition for final state predicates on nodes, by recursion on the (well-founded) relation of precedence in trees. (And, of course, π_M will be the statement that the top node has some accepting state predicate.)

5.3. *A Modal Logic Perspective*

Now, consider the simplest possible special case of this, being a one-line recursion with just basic quantifiers ($\forall$, $\exists$) over predecessors. The latter can be viewed as *modal operators* ($\Box$, $\Diamond$): an analogy that will

turn out useful. For instance, (1) in the preceding example has the modal form (with 'Acc' for *accept*):

$$Acc \leftrightarrow (A \wedge \Box\neg Acc) \vee (\neg A \wedge \Diamond Acc)$$

This perspective points at a relevant result in modal logic, *viz.* the *De Jongh-Sambin Fixed-Point Theorem* (cf. Smoryński 1984). This theorem says that there is an effective procedure π for obtaining fixed-points for modal schemata

$$p \leftrightarrow \phi(p, \vec{q}\,)$$

(with p only occurring in ϕ in the scope of at least one modal operator: which enforces the required recursion). I.e., $\pi(\phi)$ is a modal formula in the parameters $\vec{q}$ only such that the following equivalence is valid on all finite trees:

$$\pi(\phi) \leftrightarrow \phi(\pi(\phi), \vec{q}\,).$$

REMARK. There is one important proviso here: the modal operators are to range over *all* predecessors of a node, not just *immediate* predecessors (cf. Section 5.1). In the remainder of this section, we shall make this assumption about our machines too (returning to this issue in Section 6).

The exact method for calculating $\pi(\phi)$ is as follows:
(I) for all modal formulas $\Box C(p)$, one has (with 'T' for *true*):

$$\Box C(T) \leftrightarrow \Box C(\Box C(T))$$

(II) for recursions $p \leftrightarrow \phi(p) = B[\Box C(p)]$, where $B = B[r, \vec{q}\,]$ is a p-free modal formula with one formula $\Box C(p)$ substituted for r, the solution is:

$$\phi(B[T]).$$

(III) for recursions $p \leftrightarrow \phi(p) = B[\Box C_1(p), \ldots, \Box C_k(p)]$, solutions are obtained by iterated application of II, using auxiliary proposition letters.

EXAMPLE. For the above example (1) (section 5.2), one gets:

$$Acc \leftrightarrow (A \wedge \Box\neg Acc) \vee (\neg A \wedge \neg\Box\neg Acc);$$

with $B = (A \wedge r) \vee (\neg A \wedge \neg r)$
$C = \neg Acc.$

Then, $B[T] = (A \wedge T) \vee (\neg A \wedge \neg T) \leftrightarrow A$,
$\phi(B[T]) = (A \wedge \Box\neg A) \vee (\neg A \wedge \neg\Box\neg A)$

is the eventual solution.

It is enlightening to check this solution by direct semantic reasoning: π_M is the class of all trees whose top node satisfies this modal disjunction.

5.4. *Extending the Modal Framework*

But, our questions go beyond the original De Jongh-Sambin theorem. For, we want to admit arbitrary first-order quantifiers over predecessors, not just the standard ones — witness (2) in the earlier example. And indeed, De Jongh has shown that such an extension is possible. (See also Smoryński 1985.) The crucial observation here is that step (I) in the above procedure works for all quantifiers Q over predecessors of nodes x which are *hereditary*, in the sense that

if $Q_x A$ and Rxy, then $Q_y A$ ('$Q \rightarrow \Box Q$').

Examples of such quantifiers are: "in all predecessors" ($\Box$), "in at most k predecessors". Non-examples are: "in at least k predecessors", "in most predecessors". (But note that, for *any* quantifier Q, the combination $\Box Q$ is hereditary.)

CLAIM. On well-founded orders (including all finite trees), the equivalence $QT \leftrightarrow QQT$ is valid at every node.

Proof. Suppose that QT holds at x. Then QT holds at all predecessors y of x (Heredity). So, on predecessors of x, QT and just T define the same set. Therefore, substitution yields QQT at x (Q being a quantifier over predecessors).

Next, suppose that $\neg QT$ at x. Let y be R-minimal in the non-empty set consisting of x and all its predecessors where QT fails. Thus, $\neg QT$ and $\Box QT$ hold at y. Again by substitution, then, $\neg QQT$ holds at y. By Heredity, it follows that $\neg QQT$ holds at x too. ■

Now, arbitrary first-order quantifiers have Boolean definitions in terms of *at most k* (*not*); and these will be covered by step (II) of the fixed-point algorithm, just as before.

EXAMPLE. By this method, a solution may be obtained for the earlier (2)-recursion:

$$Acc \leftrightarrow (A \wedge \textcircled{1}Acc) \vee (\neg A \wedge \Box\neg Acc)$$

(with '①' expressing *in exactly one predecessor*).

The corresponding explicit condition is this:

$$(\neg A \wedge \Box A) \vee (A \wedge \textcircled{1}(\neg A \wedge \Box A) \wedge \neg\Diamond(\neg A \wedge \Box A)).$$

Again, it is useful to check this solution independently.

Finally, to solve our original problem in its full generality, a multiple fixed-point solution is required to the set of equations. Again, a relatively simple iteration of the one-line case turns out to work here (an observation due to Boolos), and so we have our desired result:

THEOREM. *Acyclic finite state tree automata recognize first-order properties of tree order and node-features.*

5.5. *Further Extensions, and Limitations*

The preceding result by no means exhausts all questions in this area. For instance, one would like to determine the exact range of the Fixed-Point Theorem. It breaks down, for instance, for non-hereditary quantifiers, such as *in most predecessors* (μ).

EXAMPLE (De Jongh): There is no solution to the equation

$$p \leftrightarrow \neg\mu p$$

in its own language. For, on the frame $(\mathbb{N}, >)$, this defines the subset p = all even numbers (letting μ be false at 0). But, the latter set is not definable using only μ, T, $\bot$, $\neg$, $\wedge$. ■

The same example shows that allowing first-order quantification in a language allowing (unlike the modal one) explicit reference to R, can also block the fixed-point result. To see this, consider the equation (with R^+ as in Section 4.1)

$$Px \leftrightarrow \neg\exists y(R^+xy \wedge Py);$$

which has the same effect on $(\mathbb{N}, >)$.

But, the even numbers are not purely first-order $R, =$ –definable in this structure.

From the present point of view, a more urgent question is the behaviour of other finite state computable quantifiers in the Fixed-Point Theorem. For instance, consider *an even number of predecessors* ($\Subset$).

EXAMPLE. The equation $p \leftrightarrow \Subset p$ describes the set p of all finite trees whose top node has an *even* number of *immediate predecessors*.

First, consider any tree whose top node has an even number of immediate predecessors, say $2n$; with k nodes validating p, and $2n - k$ not-p.

Case i: k is even.

Then, the total number of 'p-nodes' under the top must be even, being a sum of: k p-nodes (even) + a k-sum of their underlying p-nodes (in each case, an even number) + a $(2n - k)$-sum (again, an even number) of underlying p-nodes (in each case, an odd number). All three summands are even.

Case ii: k is odd.

This time, the summands become: odd + even (odd sum of even contributions) + odd (odd sum of odd contributions), again an even total.

On the other hand, if the top node has an odd number of immediate predecessors, a similar count will always find an odd numbers of p-nodes underneath. So, by the above equation, p will always be true in the first type of tree, and never in the second. ■

Note that this argument (unlike the earlier ones) depends on our using finite trees, rather than mere finite acyclic graphs. Moreover, it does not produce a fixed-point solution in the obvious language.

EXAMPLE. On the structure $(\mathbb{N}, >)$, the equation $p \leftrightarrow \Subset p$ defines the subset $p = \{0\}$ (by the preceding characterization). But, no formula in the language $\Subset$, T, $\bot$, $\neg$, $\wedge$ defines this singleton set. (Every non-contradictory formula in this formalism defines a subset of $\mathbb{N}$ closed under at least one 'period' 2^N.)

Still, the above fixed-points are definable in languages relatively close to the sparsest $\in$-formalism. For instance, adding simple modalities gives $\{0\}$ as the extension of the modal formula $\Box\bot$. And the above general fixed-point is definable as '$\in^+T$'; where '$\in^+$' is the variant of $\in$ with respect to immediate successors only.

So, there remains a general question. Can all finite tree recursions employing arbitrary finite state quantifiers over predecessors always be solved explicitly in this same formalism?

5.6. *A General Fixed-Point*

To conclude, here is another perspective on the definability of fixed-points (again due to De Jongh).

Let Q be an arbitrary quantifier over predecessors. Let Q^{end} be the (fixed) truth value for statements $Q\phi$ in end-points. In terms of the latter, there are at least 'local' solutions to our leading question:

THEOREM: *On trees of depth* $\leqslant$ n, *the equation* p $\leftrightarrow$ Qp *has the solution* $p = Q^nQ^{end}$.

Proof. By induction on n. In end-points, $Q\phi \leftrightarrow Q\psi$ holds for all ϕ, ψ, and hence in particular $Q^{end} \leftrightarrow QQ^{end}$. This takes care of the case $n = 0$. Next, for depth $n + 1$, the crucial step is this. $Q^{n+1}Q^{end} \leftrightarrow QQ^nQ^{end} \leftrightarrow QQQ^nQ^{end}$ (by the inductive hypothesis, and the fact that all predecessors lie in trees of depth $\leqslant n$) $\leftrightarrow QQ^{n+1}Q^{end}$. ■

Such local solutions can be wrong in general.

E.g., in the earlier example of *not most* on the natural numbers (Section 5.5), $Q^{end} = T$, and the iteration $Q^1Q^{end} = \neg\mu T$ indeed defines p at level 1 (i.e., on $\{0, 1\}$). It fails already at level 2, however, where the point 2 verifies p as well as μT. (But, at this level, $\neg\mu\neg\mu T$ still works; etcetera.)

Now, a general solution to the equation $p \leftrightarrow Qp$ may be defined as follows: $p = Q^*$, where Q^* is true at a node x iff $Q^{d(x)}Q^{end}$ is true at x. (Here, $d(x)$ is the *depth* of x, being the maximum length of a branch leading from x to some end-point.)

Thus, the question in earlier sections may be rephrased as follows. In which languages for defining quantifiers Q, does Q^* have an explicit definition?

6. COMPUTABLE PROPERTIES OF TREES

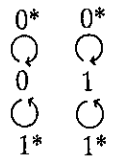
6.1. *Recognizing Modal First-Order Properties*

The general extent of finite state computability on finite trees was found early in Section 5.1: all computable properties must be (monadic) Σ^1_1.

The obvious converse question then becomes if *all* properties definable in the latter language are computable. (This is a tree-analogue of a question concerning finite state automata operating on linear sequences of symbols found already in Büchi 1960, a pioneering paper in the area of automata and logic.) No solution to this problem will be found here. We shall only consider a special case, *viz.* that of *first-order* sentences in the tree order R and the feature predicates A.

The obvious approach here is an inductive one, looking at simple computable cases $\pi(R, A, x)$ (where 'x' denotes the top node of the tree), and then generating more complex cases. What can be recognized is first: (I) the *atomic* case $\pi = Ax$.

The automaton is this:



with an entry convention "$A \rightarrow 1$, $\neg A \rightarrow 0$". (1 is an accepting state.) Then, there is closure under

(II) *negations* $\neg\pi(x)$
(by interchanging accepting and rejecting states), as well as

(III) *conjunctions* $\pi_1(x) \wedge \pi_2(x)$
(computed by a suitable product automaton).

(IV) Finally, there is a computable form of *restricted quantification* over predecessors of x (where Q is any finite state computable quantifier in the sense of Section 2).

Proof. Let M compute $\pi(x)$, and M_Q Q. The following machine will compute the condition

$$Q\{y \mid Rxy\}\{y \mid Rxy \wedge \pi(y)\}.$$

Form new states (s, i) of states s for M and i for M_Q. The new transitions will be of the following form:

state:	symbol read:	new state:
s, i	t^*, j	u, k

Here, the new symbols consist of state markers for states of M, plus *truth values* $j = 0$ or 1 — encoding rejection or acceptance by M_Q so far. (This is one instance of the greater liberality as regards output allowed in Section 4.2.) Then, the transition convention is this:

- u is the state that M gets into from s by reading t^*,
- k is the state that M_Q gets into from i by reading a symbol b (if t was accepting in M), or a (if t was rejecting in M).

Thus, M, M_Q work together — M_Q processing the outcomes of M. Accepting states will be those having an accepting M_Q-component. ■

The resulting restricted formalism is reminiscent (again) of that of *modal logic* (when transcribed into first-order predicate logic; cf. van Benthem 1984a). We have proved, amongst others:

THEOREM. *All modal first-order properties of trees are finite-state computable.*

Still, this result is by no means the best possible. For instance, 'upward-directed' properties can be computed too, such as 'every A-node is dominated by at least one B-node' (see Section 6.2). Another case in point are the *single trees*: their categorical descriptions in the matching first-order language are all computable (using different state labels for distinct subtrees). In fact, here is a

Conjecture: All first-order properties of trees are finite state computable.

6.2. *Second Thoughts*

The preceding analysis needs one important qualification. The proof of the main theorem was neutral between the two earlier formats: 'inspecting *all* predecessors' versus 'inspecting all *immediate* predecessors'. But, definability in general is affected by this, witness the following

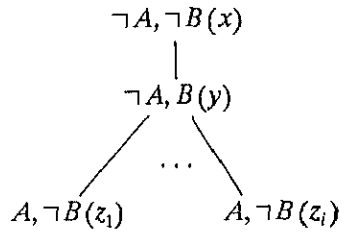
EXAMPLE. To compute $\forall x(Ax \rightarrow \exists y(Ryx \wedge By))$.

In the immediate predecessor format, this upward-looking property can be recognized in an obvious way, using three states:

- accept_1 (in nodes $\neg A, B$, regardless of what lies underneath)
- accept_2 (in nodes $\neg A, \neg B$, with all immediate predecessors having either accept_1 or accept_2)
- reject (in all other nodes, including all those carrying feature A).

But, this procedure does not work when surveying *all* predecessors (compare 'accept_2'). In fact, then, the above property is not computable at all!

Proof. Suppose that finite-state machine M computed it. Consider the family of trees T_i:



Each of these has the above property, and will therefore be accepted by M. Now, M's computation produces identical states on all bottom nodes z_j ($1 \leqslant j \leqslant i$), but perhaps varying ones for y, x (depending on i). In any case, only finitely many will be available for y — so, for some i_1, i_2 with $i_1 < i_2$, M will reach the same state on y, when processing T_{i_1}, T_{i_2}. But then, compare T_{i_2} with the tree T' obtained from it by severing y's links with $z_{i_1+1}, \ldots, z_{i_2}$. (The latter are still connected with the top node x.) When processing T', M will assign identical states below x as in T_{i_2} — and hence, it must also treat x identically in both cases. I.e., T' will be recognized: even though it lacks the above first-order property. ■

In fact, similar problems arise with 'downward-directed' modal properties, such as

$$\forall x(Ax \rightarrow \exists y(R^+xy \wedge By));$$

which cannot be computed on the latter option either. Our *conjecture* is that this is an asymmetric affair: everything computable on the

'arbitrary predecessor' option is also computable on the 'immediate predecessor' option; but not *vice versa*.

This observation emphasizes the independent interest of our original machines. Therefore, it is worth-while exploring the earlier fixed-point theory (see Sections 5.3—5.6) for this case too, in some appropriate formalism.

For instance, the general fixed-point solution of Section 5.6 works here as well (as its proof was neutral between the two options).

Again, the local solutions obtained there can be incorrect in general.

EXAMPLE. Consider Q = '$\exists\neg$' on $(\mathbb{N}, >)$, with the equation $p \leftrightarrow Qp$:

$$\underset{+}{3} \rightarrow \underset{-}{2} \rightarrow \underset{+}{1} \rightarrow \underset{-:p}{0}$$

E.g., until depth 2, $Q^2Q^{end} = \exists\neg\exists\neg\bot$ $(= \exists\forall\bot)$ is a correct definition of p: but at level 3, it fails. (What works there, however, is indeed $Q^3Q^{end} = \exists\forall\exists T$.)

Of course, with the other option, p would have been globally definable, *viz.* by $p \leftrightarrow \neg\Box\bot$. Such uniform solutions are very scarce on the present option — arising only in special cases (such as $Q = \exists$ or $\forall$). Therefore, it becomes of interest to also have more local versions of the De Jongh-Sambin Theorem — and these do exist:

PROPOSITION. *Let* A(p) = B(Qp), *with the only occurrences of* p *as indicated.*

Define C = (BQ)^{end}. *On trees of depth* $\leqslant$ n, *one has the equivalence*

$$A^nC \leftrightarrow AA^nC.$$

The proof is similar to that of the Theorem in Section 5.6. One can check this outcome with the above cases.

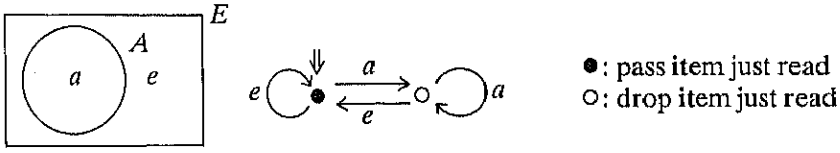
Finally, these results can also be studied on *non-well-founded* structures. The difficulty, there, is that the above equations do not have unique solutions, e.g., once loops are admitted. But, options can still be explored systematically. (Compare the unraveling of 'guarded' systems of recursion equations in Process Algebra; cf. Milner 1980, Bergstra & Klop 1984.) This topic will not be pursued here.

7. AUTOMATA IN OTHER CATEGORIES

As all compound types in a categorial language may be regarded as denoting semantic functions, it might be possible to develop one general perspective on automata computing these. But, if such an enterprise is to be worthwhile, there should be some additional examples of *attractive* machine models for specific types of expression. We discuss a few cases.

First, in addition to 'relational' types, such as with the earlier quantifiers, also more 'operational' types from natural language should be considered, such as with connectives or adjectives. Here, automata will act most naturally as *transducers* rather than mere *recognizers*. In the simplest case, such a transducer will not change its input, but merely select items from it.

EXAMPLE. An automaton for computing *not*:



In fact, all earlier finite state automata can also be re-interpreted as devices for computing operations.

EXAMPLE. The automaton for *all* (Section 1) will pass precisely all items *b*, up till the first *a* encountered. That for *an even number* (Section 2) will pass *every second b*, together with alternate intervals of *a*'s.

So, permutation-invariant recognizers need not turn into *order-indifferent* transducers, with a yield independent of the order of presentation of individuals. The latter will require the following, for every symbol *a*:

> either, all *a*-arrows end in a passing state ●
> or, all *a*-arrows end in a blocking state ○.

It follows that order-indifferent operations are few, as only two states will be involved. Among the connectives, this leaves just the *Boolean*

polynomials. (This was to be expected, as order indifference is an analogon of the earlier Condition of *Quantity* for operations: cf. van Benthem 1985a.)

This restriction to what might be called *logical* items in various categories is not unreasonable. These are the more 'theoretical' expressions, whose degrees of complexity are worth studying by formal means. (Other examples would be 'formal' operations modifying numbers of arguments in predicates, etcetera.) But obviously, it would not be reasonable to expect an automata hierarchy for, say, adjectives or adverbs, whose meaning is dependent on lexical content.

In line with the earlier treatment of quantifiers, two directions of extension are open now. One is to consider *more complex automata*. For instance, a transducer with a push-down store could pick up items and replace them somewhere else, thus rearranging input patterns. It is not quite clear if such facilities are needed for natural language meanings. Another direction leads to *more complex data*. As in Section 4, operations will have to work, not just on flat sequences of objects presented, but also (at least) on tree — or graph-like patterns.

One example would be that of computing R-maximal items in a certain set A, say, when determining the extension of the superlative *A-est*, given some underlying comparative order *A-er*. The tree automaton of Section 1 would now have to drop nodes lacking the feature A, while, upon arrival at an A-node, dropping its predecessors altogether. Again, all earlier tree-recognizers can be used as tree-transducers too.

A more complex case of a tree operation is provided (curiously) by the positive adjectives underlying the above superlatives. For instance, there is a recurrent proposal to the effect that a measure adjective like "tall" applies to precisely those items in the "taller"-tree which have more predecessors than successors (in the upward order). (Thus, the reading proposed is "taller than most".) Computing this property goes beyond the earlier resources; one must not only 'look back', when climbing the tree, but also 'look ahead'. In fact, this problem would already arise with linear sequences. In the simplest case, how is a machine to operate on a sequence, leaving only those items beyond its mid-point? No finite state machine can do this, and even a push-down store automaton seems inadequate: as back-and-forth movement seems required along the sequence surveyed. So, are we forced to cross the boundary to full *Turing machine* action after all?

The preceding conclusion may be somewhat premature, as one

important issue has been left implicit, *viz.* the *representation* of the data fed to the procedure associated with a linguistic expression. There is a matter of 'division of labour' here, which can affect judgements of complexity. For instance, another way of computing the above sense of "tall" would be this. Given any individual x and a "taller"-order, we can form the two sets $\{y|x \text{ taller than } y\}$ and $\{y|y \text{ taller than } x\}$, and then use an ordinary *most*-automaton (as in Section 3) to compare their cardinalities.

The point of the present approach is not so much to support one particular analysis here, as to make such issues of representation and complexity amenable to systematic analysis.

8. REALISTIC AUTOMATA

The introduction of automata into semantics carries the promise of making semantic sense of questions of actual *mental processing* and *learnability* of natural language. Of course, it is not clear a priori that the present approach will be less controversial here than it has been in mathematical linguistics in general. But, it is worth speculating about some more realistic interpretations, or modifications of our framework.

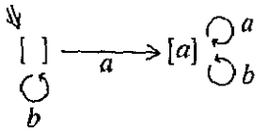
8.1. *Computability*

For instance, how plausible are the earlier machines as models of processing expressions? Already with finite state machines, there is the problem to find some significance for the *states*. In computational practice, these might encode *instruction* labels of some program being executed. But, this does not seem quite plausible in our case. Let us try something simple and direct.

In line with common psychological assumptions, we postulate a fixed *finite working memory*, together with a facility for reading new symbols. Now, there may be various actions allowed here. A totally 'passive' reader would only store new symbols (losing the bottom-most one stored), a somewhat more active one could decide whether or not to store (depending on the symbol and current memory-contents), and eventually, more drastic rewriting of memory-contents is possible too. Finally, in all cases, acceptance/rejection conventions could be based on current memory-contents as well. This perspective turns out to

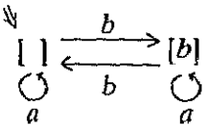
throw some additional light on the earlier finite state automata (Section 2).

EXAMPLE. Computing with a one-place memory; on an alphabet $\{a, b\}$. In this case, there are three possible memory-contents ('states'), *viz.* [ ], $[a]$, $[b]$. Of these, [ ] is the obvious initial state; where any of the three can be accepting/rejecting. To compute, e.g., *all*, one makes the stipulation of the following diagram



[ ]: accept
$[a]$: reject

To compute, e.g., *an even number*, one needs,



[ ]: accept
$[b]$: reject

Note that the latter machine rewrites memory-contents.

Some further analysis of these examples shows that

(i) the one-place memory case yields only two-state computable quantifiers,

(ii) all first-order quantifiers are computable without rewriting of memory,

(iii) with the latter facility, recognizing power is exactly that of finite state automata (encoding states as memory-contents, and *vice versa*).

The next reasonable enrichment of this scheme is to add a *long-term memory*, potentially unbounded, to the short-term one, with (restricted) facilities for transfer between the two. Then, e.g., the *proportions* of Section 3 become computable, within tight constraints.

EXAMPLE. Computing *at least two-thirds*.

A two-symbol short-term memory is required, with the obvious transfer to the long-term memory (push-down store). Rewriting is only allowed in the short-term memory as follows:

symbol read:	short-term memory:	action:
a	*aa*, $-a$, $-$	store *a* in short-term memory
	bb	empty short-term memory
	ab	store *a* in short-term memory in left-most position
	$-b$	rewrite to *ab*
b	*bb*, $-b$, $-$	store in short-term memory
	aa, $-a$	store *b* in short-term memory in right-most position
	ab	empty short-term memory

What will happen here is this. At each stage, there will be a homogeneous stack of either all *a* or all *b*, with a working memory containing (perhaps) occurrences of the other symbol as well. The crucial transition is that where one symbol *a* 'neutralizes' two symbols *b*. By a simple argument, it follows that there are at least two-thirds occurrences of *b* in the sequence read if and only if the short-term memory registers one of the following:

bb, $-b$, $-$

Therefore, these may be chosen as the accepting 'states' for the procedure.

All procedures of this kind can be simulated by push-down store automata. The converse is probably false, in view of our lack of states, and the restrictions on rewriting.

Perhaps the most important 'realistic' difference with the usual push-down store automata has to do with a feature not discussed up till now. The above automata are *deterministic*; whereas non-determinism is crucial to push-down store automata as usually employed (e.g. in getting all context-free languages recognized). From a realistic point of view, there are good reasons for studying deterministic subcases — a restriction, moreover, which seems in line with our ideas about actual quantifier expressions in this range.

E.g., all the continuous bi-linear quantifiers, isolated in Section 3, are deterministically computable in the above sense.

Still, there may be psychological arguments for allowing a certain amount of non-determinism after all in our account of semantic com-

petence. For instance, categorial semanticists are fond of the metaphor of understanding a sentence as fitting together a *jigsaw-puzzle*. As those who have experience with such puzzles can testify, competence here often consists in applying a judicious blend of deterministic fitting, of obviously matching pieces, and random fitting, in cases with little contrast. The latter 'stupid' procedure can actually be a lot faster than an over-all deterministic solution!

8.2. *Learnability*

The preceding discussion revolved around the issue of *complexity*: what is 'easy to compute'. What about the companion topic of *learnability*: what is 'easy to learn'? Perhaps, an answer to the first implies an answer to the second: 'less complex procedures are easier to learn'. (But, what about a simple Turing machine instruction versus, say, one hundred pages of regular rewrite rules?) At the present stage, alternative approaches may be just as plausible, stressing various aspects of the learning process itself. (See Section 9 for some further speculation.)

Moreover, an issue which would have to return in this setting is the choice of *representations*, to do our computations on. Obviously, much learning consists in finding the most informative representations of knowledge to cope with the world. (Perhaps, these are even chosen so as to *minimize* computation.) In connection with this issue, the approach taken in this paper could also be developed, not on extra-linguistic models, but, say, on discourse representation structures.

Finally, successful learning implies the ability both to *recognize* and to *produce* situations of the kind which was studied. In mathematical linguistics, this duality between *recognition and generation* has been studied extensively. It also makes sense in semantics, however, which knows various systematic methods of model construction (such as 'semantic tableaus' and the like).

So, computational semantics has a full agenda for research.

9. APPENDIX ON LEARNING THEORY

This section is a discussion as to how learnability of semantic meanings can be treated mathematically, in the spirit of this paper. The guiding idea here comes from Osherson and Weinstein 1986 (even though the direction taken is different eventually).

As a learning model, suppose that one learns the meaning of some linguistic expression by being presented with situations where it holds, and situations where it fails. On the basis of these examples, perhaps acquired in childhood, one hypothesizes some uniform meaning, which is then used subsequently (subject to revision by further examples). If this process is regular, it comes in the form of some *learning function*, producing a hypothesis, for every finite sequence of data, about the general meaning behind these. If all works well, this function should, for each of the types of expression that we are interested in, produce the correct hypothesis after having seen some finite segment of its semantic behaviour.

There are still many mathematical options in this outline, which are explored in detail in Osherson, Stob and Weinstein 1986. (The model itself dates back to work by Gold in the sixties, however.) Their main concern is with language learning in a more syntactic sense (with grammars being hypothesized) — but many of their points are of such a general recursion-theoretic nature that they fit a wide range of other cases (including the present). For instance, what can be 'learnt' successfully depends on such assumptions on the data as the following:

— does one see only positive instances of the notion to be learnt or both positive and negative ones (as suggested above)?
— does the order in which data are received matter/and likewise, repetitions of data?

Moreover, answers will depend on assumptions about the (complexity of the) learning function: which could be computed by a Turing machine, but also by simpler devices. And finally, there is also a variety of outcomes associated with different requirements on the quality of recognition. E.g., should the learning function also reject, or at least, fail to identify correctly, every sequence of data for meanings not in our intended class?

Evidently, there is a wide variety of questions here — going from proposed learning functions to classes of languages/meanings recognized, as well as *vice versa*. One interesting line is to start from natural assumptions about human learning functions, and then locate the induced constraints on 'natural languages/meanings' that can be humanly learnt.

In this appendix, we look only at a very simple special case — so special, that some of the central tools in the general analysis (such as

the 'Blum & Blum Lemma') do not apply. Even so, some interesting connections emerge with our previous topics.

Let us restrict attention to learning *quantifiers* — the data being progressive information about their true/false patterns in the Tree of Numbers (cf. van Benthem 1984b). Thus, both positive and negative information is provided. Moreover, let us assume that information about smaller situations will, on the whole, precede information about larges ones — resulting in the idealization that we are being fed the tree pattern line by line, starting from the top. In fact, knowing this fixed enumeration, having only the positive cases for a quantifier Q automatically supplies the negative ones.

EXAMPLE. (Tree pattern for *all*): The Tree of Numbers represented all possible configurations a, b with $a = |A - B|$, $b = |A \cap B|$, for a quantifier QAB:

$\|A\| = 0$			0,0			
1		1,0		0,0		
2	2,0		1,1		0,2	
3	3,0	2,1		1,2		0,3

etcetera.

E.g., the pattern for *all* is this (with + for YES, − for NO).

			+			
		−		+		
	−		−		+	
−		−		−		+
			⋮			

This will come out as the following 'learning sequence':

$$+, -, +, -, -, +, -, -, -, +, \ldots$$

In the most general case, a learning function will now be any map f assigning quantifiers (or, suitable *names* for these) to finite sequences of YES/NO answers to successive nodes in the Tree.

As there are only countably many such finite sequences, f will

identify at most countably many quantifiers. But also conversely, for any countable class of quantifiers $X = \{Q_1, Q_2, \ldots\}$, there is some learning function f_X identifying it, via the following rule:

> 'for any sequence $\in$, $f_X(\in)$ is the first Q_i in the enumeration whose Tree pattern is consistent with $\in$; f_X is undefined if no such Q_i exists'.

Thus, exactly the *countable* classes of quantifiers are learnable by unrestricted (possibly partial) learning functions. Note that the above learning function is *prudent*: every temporary hypothesis is consistent with the data so far. This feature will be assumed henceforth.

To create more structure, one may plausibly require that learnable families be *recursively enumerable* sets of quantifiers which are *decidable* on finite models. In that case, the above learning function becomes (general) *recursive*. Actually, for this conclusion, it suffices to assume that all quantifiers to be learnt have *RE* patterns of accepted nodes. (To compute $f_X(\in)$, one then starts enumerating all Q_i-ranges in the usual diagonal way: picking the first to embrace $\in$.) A converse holds too. If f is recursive, then its range, on some effective enumeration of the finite sequences, will be *RE*. Moreover, for each of the quantifiers occurring here (in encoded form), its class of accepted nodes may also be enumerated effectively. (Compute f in ever increasing depth, over ever increasing finite sets of sequence arguments — using the fact that node x belongs to Q_i iff f assigns Q_i to some sequence $\in$ having YES at its x-position. (Prudence is needed here.)) There are further possible refinements — but, the above will suffice for a general impression.

Now, let us consider some *fine-structure* of learning functions, by bringing in the earlier hierarchy of automata. For instance, the simplest kind of learning device would be a finite state machine, whose states represent some finite number of possible conjectures.

EXAMPLE 1. The *all* machine of Section 1. Let the accepting state represent the *universal* quantifier (true everywhere), and the rejecting state its negation. Then, the machine correctly identifies this family $\{Q_1, Q_2\}$. But, it will also produce incorrect identifications outside of this family (for, the pattern for any quantifier other than Q_1 will receive Q_2 eventually). An alternative would be to let the rejecting state represent "No Hypothesis". Then, $\{Q_1\}$ would be uniquely identified.

EXAMPLE 2. The *even number* machine of Section 2.2. Here, the same multiplicity arises. E.g., with the accepting state representing some hypothesis Q, the latter will be assigned to *any* quantifier pattern consisting of some finite even number of YES-markers, distributed arbitrarily in the Tree of Numbers.

What these examples suggest is that permutation-invariance is not desirable now (unlike in Section 2.1): the *order* of data is important. Moreover, we have changed to recognizing *infinite* sequences: an area with its own peculiarities (cf. van Benthem 1984b). And most importantly, automata now operate *one level higher* up, so to speak: which should make one beware of apparent analogies.

This shows even more when we reverse the question. Consider the Tree pattern for the quantifier *all*, whose learning sequence was already given in an earlier example.

What kind of automaton will recognize exactly this sequence? It turns out that a *push-down automaton* is needed, which alternatively stores and erases the sequences of $-$ (NO) markers, making sure that the next one is exactly one longer than the preceding one. In its most obvious form, this requires *two stacks* (one for comparing, one for tallying). But, push-down store automata with two stacks have *Turing Machine* power: and we have arrived at the most complex case after all.

So, what would be the proper notion of computational fine structure here? For instance, one would still like to say that all *first-order* quantifiers are of roughly the same complexity as the above example. (Consider their Fraïssé thresholds (van Benthem 1984b): in the corresponding learning sequence, the crux will always be to recognize one-step growth of some interval of $-$ or $+$ in a fixed environment.) On the other hand, e.g., the *higher-order* case of *most* requires the use of more than two stacks (intuitively speaking), when checking its learning sequence

$$(-)-+--+---++---++---+++\ldots$$

Finally, the above automata can also be used to identify *classes* of quantifiers. For instance, when automata are given identifying the single quantifiers $Q_1, \ldots, Q_n$ uniquely, then $\{Q_1, \ldots, Q_n\}$ can also be recognized as follows:

First, follow sequences until the first depth where all differences between $Q_1, \ldots, Q_n$ have shown: this can be registered in some initial finite state part. Then continue the final states reached in this way with the appropriate automata $M(Q_1), \ldots, M(Q_n)$.

The preceding discussion underscores the difference between our two levels of using automata: as learning machines *or* as semantic procedures hypothesized by the former. For instance, computing a first-order quantifier may be of finite state complexity — while learning that one is dealing with this particular quantifier may be an essentially more complex problem of *pattern recognition*. Even so, the question remains which connections exist between the internal mechanism of a learning machine and the structure of its conjectures.

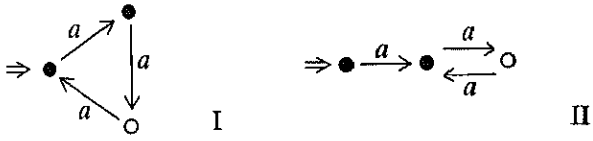
For instance, can a learning machine recognize if a given learning sequence corresponds to a finite state computable quantifier — and if so, identify which one?

By an earlier general result, the answer is positive — as this is a recursively enumerable class of decidable sets. But, the learning function given in the proof was highly abstract. On the other hand, scientists engaged in pattern recognition seem to predict behaviour with great certainty, after having seen only a few evolutions in, say, the computer simulation of a cellular automaton (cf. Wolfram 1984). But then, in such a case, the learning task is rather this: *assuming* that some simple system lies behind the observed evolution, identify *which one*.

The latter perspective raises questions of *speed of recognition*. E.g., if we know that a given set of sequences is produced by some N-state finite state machine, then there exists some finite length before which all differences between the finitely many possibilities must have shown already. Can this length be estimated?

In general, the obvious conjecture (being N itself) does not work:

EXAMPLE (1 symbol, 3 states):



I II

Machine I recognizes: $\langle\ \rangle$, a, a^3, a^4, . . . , while machine II recognizes $\langle\ \rangle, a, a^3, a^5, \ldots$

PROPOSITION. *If two finite automata with at most* N *states each recognize different languages; then they differ already on some string of length at most* 2 N.

Proof. In Arbib 1969, Chapter 3, item 32, it is shown that any two non-equivalent states in an N-state finite automaton can be told apart by feeding in some string of length at most N. Now, consider our two finite automata as one ($2N$-state) joint automaton, and the assertion follows. ■

We can specialize this result to the earlier case of binary quantifiers. Recall how machine states show in the Tree representation of a quantifier (cf. van Benthem 1986): as nodes generating identical downward subtrees. If only finitely many types of subtree occur, then Q is computable by some finite state machine.

PROPOSITION. *If* Q *is computable by a finite* N*-state machine, then all its states will have occurred already in the upper triangle of depth* N.

Proof. This follows from the following claim: if only m types occur ($m \leqslant N$) in the top triangle of depth $N+1$, then only m types occur in the whole Tree. The argument is by induction on N.

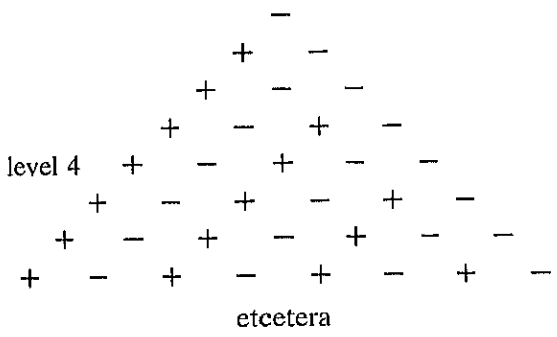
— $N = 1$. If the top pattern is like this: $t_1^{\,t_1}$, then the whole Tree has pattern t_1 (as t_1 'propagates').

— $N+1 \rightarrow N+2$. Case i: The top triangle with depth $N+1$ contains occurrences of only m types, with $m \leqslant n$. By the inductive hypothesis, only m types occur altogether. Case ii: That triangle contains occurrences of $N+1$ types. But then, in the $N+2$-triangle, every occurring type has two immediate successor nodes with types from among these $N+1$ possibilities. So, again by propagation, the whole Tree pattern is fixed in the downward direction, displaying only these $N+1$ types. ■

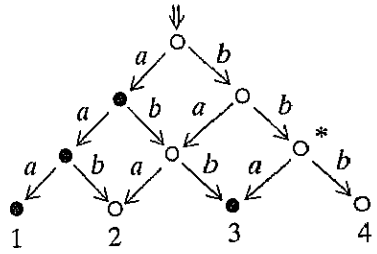
REMARK. The same result could be proved less pictorially (but faster) by means of the Pumping Lemma for regular languages.

Now by developing the Tree with depth $2N$, the earlier identification method can be applied.

EXAMPLE. Consider the following Tree-pattern for a 4-state finite state quantifier

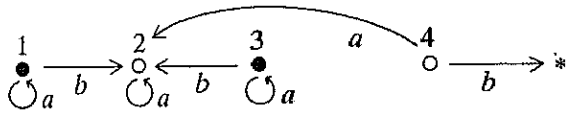


For the initial part of the diagram, take the upper triangle *itself*:

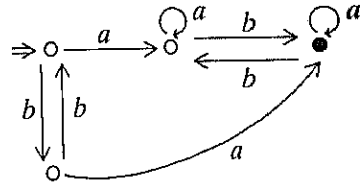


It remains to find transition arrows on the final row.

By comparing generated downward subtrees of depth 4, one can complete this now, e.g., as follows:



This may be simplified to the equivalent 4-state machine:



The corresponding quantifier is ('at least one a, and an even number of b'):

$$QAB \text{ iff } A - B \neq \emptyset \text{ and } A \cap B \text{ has even cardinality.}$$

Another way of thinking about these temporary conjectures is via the Nerode Representation of regular languages (see Section 2.1). A *Nerode-Learner* could act as follows, in the above setting. At each stage, it knows some finite set A of outcomes for Q — and it can make guesses as to its automaton by identifying tree nodes if these are not Q-distinguishable by continuations *within* A. Moreover, there are obvious a, b transition functions. (Actually, one must be somewhat more careful, creating different approximations for fixed 'comparison depths'.) Now, provided that some finite state automaton lies behind the observed Q-pattern, the Nerode learner will arrive at a stable conjecture sooner or later. By way of contrast, compare its continuing oscillations on the non-finite state quantifier "exactly half", which accepts only Tree nodes x, x. It would be of interest to have similar generators of conjectures for push-down computable quantifiers too. In fact, a very rapid change in finite state estimates would probably force us to postulate some higher automaton at work. What kind of decision mechanism would describe this?

Of course, this model of learning and hypothesizing is still rather crude. See e.g. Winston 1984, Charniak and McDermott 1985 for more sophisticated accounts of learning grammars — which might be adapted to the present semantic concerns.

ACKNOWLEDGEMENTS

I would like to thank Dick de Jongh for a stimulating correspondence on the topics of Section 5, resulting in his many contributions mentioned here.

Various participants at the Lund workshop on Generalized Quantifiers (May 9—11, 1985) offered interesting suggestions and criticisms, especially Manfred Bierwisch and Lars Johnsen.

Another useful round of comments was received at a presentation of this work at Tübingen (July, 1986), from Hans Kamp, Donald Nute, Stan Peters and Heinz Volger.

Finally, as always, Frans Zwarts has been a source of questions and inspiration.

REFERENCES

Arbib, M.: 1969, *Theories of Abstract Automata*, Prentice-Hall, Englewood Cliffs, New Jersey.

van Benthem, J.: 1984a, 'Modal Correspondence Theory', in D. Gabbay and F. Guenthner (eds.), *Handbook of Philosophical Logic*, Vol. II, Reidel, Dordrecht, pp. 167—247.

van Benthem, J.: 1984b, *Semantic Automata*, report, Center for the Study of Language and Information, Stanford. (To appear in: D. de Jongh *et al.* (eds.), 1987, *Studies in the Theory of Generalized Quantifiers and Discourse Representation*, Foris, Dordrecht, (GRASS series, vol. 8), pp. 157—181.)

van Benthem, J.: 1986, *Essays in Logical Semantics*, Reidel, Dordrecht (Studies in Linguistics and Philosophy).

Bergstra, J. and J. Klop: 1984, 'The Algebra of Recursively Defined Processes and the Algebra of Regular Processes', in: J. Paredaens (ed.), *Proceedings 11th ICALP, Antwerpen 1989*, Springer Lecture Notes in Computer Science **172**, 82—95.

Büchi, J.: 1960, 'Weak Second-Order Arithmetic and Finite Automata', *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* **6**, pp. 66—92.

Charniak, E. and D. McDermott: 1985, *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts.

Hopcroft, J. and J. Ullman: 1979, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading (Mass.).

Keenan, E. and L. Moss: 1985, 'Generalized Quantifiers and the Expressive Power of Natural Language', in: J. van Benthem and A. ter Meulen (eds.), *Generalized Quantifiers in Natural Language*, Foris, Dordrecht (GRASS-series, Vol. 4), pp. 73—124.

McNaughton, W. and S. Papert: 1971, *Counter-Free Automata*, MIT Press, Cambridge (Mass.).

Milner, R.: 1980, *A Calculus of Communicating Systems*: Springer Lecture Notes in Computer Science 92.

Osherson, D. and S. Weinstein: 1985, 'Identification in the Limit of First-Order Structures', *Journal of Philosophical Logic* **15**, 55—81.

Osherson, D., M. Stob and S. Weinstein: 1986, *Systems that Learn*, MIT Press, Boston.

Perrault, R.: 1984, *On the Mathematical Properties of Linguistic Theories*, Report 84-18, Center for the Study of Language and Information, Stanford.

Smoryński, C.: 1984, 'Modal Logic and Self-Reference', in: Gabbay and Guenthner, o.c., pp. 441—495.

Smoryński, C.: 1985, *Self-Reference and Modal Logic*, Springer, Berlin.

Winston, P.: 1984, *Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts.

Wolfram, S.: 1984, 'Cellular Automata as Models of Complexity', *Nature* 311.