Recognizing tokens in a finitary algebra

Basil A. Karádais

(a note — draft of 30 Nov 2012, 2:11 p.m.)

Abstract

I describe here a technique of recognizing in linear time trees over a free algebra given by constructors. The technique is arithmetical rather than automata theoretic, in that it depends on the arities of the constructors rather than on their names.

1 Algebras, terms, and the recognition problem

A *finitary (free) algebra* is given by a *ranked alphabet* or a *signature*, that is, a finite collection of symbols C^i, $i = 1, \dots, k$, each of them paired with a non-negative integer r_i, their *arity* or *rank*. With the prospect of denotational semantics as in [6], one thinks of the ranked symbols as *constructors* and requires that one of these is the special *partiality constructor* $*$ of arity 0, and that at least one of the rest is also nullary; the first requirement is not essential in general, while the second one is crucial, since otherwise empty algebras sneak in the picture.

Let K denote a fixed signature; write $\mathrm{ar}(C)$ to denote the arity of its constructor C. A *string* or *word* over K is just a string over the (unranked) alphabet that underlies K, that is, a string that may neglect arities. Strings may be defined inductively in the following manner: ε is a string, the so-called *empty string*; if a is already a string, then Ca is a string, where C is a given symbol in K.

For example, consider the signature K given by the pairs $(*, 0)$, $(0, 0)$, $(1, 0)$, $(S, 1)$, and $(B, 2)$. The following are strings: ε, 1, $0S$, $S10$, $SB{*}1$—by convention we write a instead of $a\varepsilon$ when a is nonempty.

A *term* or *token* over K is a string of the form $Ca_1 \cdots a_r$, where C is in K with arity r, and $a_1, \dots, a_r$ are already terms, if any are needed—note that the definition is again inductive. The second and the last of the previous strings are also terms, the others not.

Write $S(K)$ for the collection of strings over K and $T(K)$ for the collection of terms over K. The *recognition problem* of K is this: given an arbitrary $a \in S(K)$, decide if $a \in T(K)$ or not. It is clear that the recognition problem in this setting is decidable, but the question is how to solve it in an efficient way.

To illustrate the issue, consider the following string over our toy signature:

$$a = BBBBBBBBBB * * * * * * * * * S0SB1* \; .$$

We want to recognize a. Based on a direct interpretation of the definition of terms, we would reason more or less as follows: The string a starts with a B, whose arity is 2; so it will be a term if we make sure that its tail (the substring left if we drop the head) is the concatenation of two strings that are themselves terms. So start recognizing the tail; if you get a term before it exhausts, recognize the rest of the string; if it turns out to be a term, then you're done and $a \in T(K)$. In all other cases it is $a \notin T(K)$.

The above shows two things clearly enough: (a) due to branching constructors, that is, constructors of superunary arity, there are potentially many simultaneous recursive calls of "recognize" that raise the complexity of the procedure exponentially; (b) the names of the symbols do not play a role so much as their arity. I present a way to recognize a term that circumvents (a) by exploiting (b).

2 A linear solution through the tree rank of a string

There are some very well-known mappings that send strings or terms to non-negative integers, which serve as measures, like the *length* or *size* of a string, that is, the number of involved symbols, or the *height* or *depth* of a term, that is, the number of nodes in a branch of maximal length in its treeform. Here I make use of a similar mapping that intends to capture some of the "treeness" of a given string.

Let a be a string over K. Define its *tree rank*, and write $\mathrm{tr}\,(a)$, by

$$\begin{aligned} \mathrm{tr}\,(\varepsilon) &:= 1\ , \\ \mathrm{tr}\,(Ca) &:= \mathrm{ar}\,(C) + \mathrm{tr}\,(a) - 1\ . \end{aligned}$$

For example, the tree ranks of the terms ε, 1, $0S$, $S10$, and $SB{*}1$ are 1, 0, 0, -1, and 0 respectively. Note that the tree rank can take negative values.

A possibly suggestive intuition for this choice is the following: emptiness needs to be appended 1 well-formed term to trivially yield a well-formed term; if we have a string a that already needs r well-formed terms to "complete" to a well-formed term, and we further extend it by a constructor C of arity r', in order to complete the concatenated string Ca we will need r well-formed terms for a, minus 1 because of C, plus r' for what C needs in turn to be completed.

Suggestive or not, the intuition goes only half way: *zero* tree rank does not necessarily mean that we have a proper term in our hands, as we just saw in the examples above. To wholly capture "treeness" we need to do some work.

Let $a = C_1 \cdots C_m$, for C_i constructors in K of arities r_i, $i = 1, \ldots, m$; the length of a, which we write $|a|$, is here m. It is direct to see that the tree rank mapping admits the explicit definition

$$\mathrm{tr}\,(a) = 1 + \sum_{i=1}^{m} (r_i - 1) = 1 - m + r_1 + \cdots + r_m\ ,$$

and the following formula for any $a, a' \in S(K)$ is also direct to calculate:

$$\mathrm{tr}\,(aa') = \mathrm{tr}\,(a) + \mathrm{tr}\,(a') - 1\ . \qquad (\star)$$

We say that a string a' is a *proper prefix* of a, when it has length $m < |a|$, and consists of a's first m constructors, that is, $a'(i) = a(i)$, for all $i = 1, \ldots, m$. The main observation of the note is the following.

Proposition. *Let $a \in S(K)$. It is $a \in T(K)$ if and only if* $\operatorname{tr}(a) = 0$*, and* $\operatorname{tr}(a_0) > 0$*, for all proper prefixes a_0 of a.*

Proof. Let a be a string over K. Assuming that a is a term, we proceed by induction on a as a term. It must have the form $Ca_1 \cdots a_r$, for some constructor C of arity r in K. It is

$$\operatorname{tr}(Ca_1 \cdots a_r) \overset{(\star)}{=} \operatorname{ar}(C) + \operatorname{tr}(a_1) + \cdots + \operatorname{tr}(a_r) - (r-1) - 1 \overset{\text{(IH)}}{=} 0\,,$$

so $\operatorname{tr}(a) = 0$. Now let a_0 be a proper prefix of a. If a consists of one *nullary* constructor C, that is, if $r = 0$, there is nothing to show. If not, and $r > 0$, suppose that $a_0 = Ca_1 \cdots a_m a_{m+1,0}$, where $a_{m+1,0}$ is a prefix of a_{m+1}, for some $m < r$. It is

$$\begin{aligned}\operatorname{tr}(Ca_1 \cdots a_m a_{m+1,0}) &= \operatorname{ar}(C) + \operatorname{tr}(a_1) + \cdots + \operatorname{tr}(a_m) + \operatorname{tr}(a_{m+1,0}) - (m+1)\\ &= \underbrace{r - m - 1}_{\geq 0} + \underbrace{\operatorname{tr}(a_{m+1,0})}_{>0 \text{ by (IH)}}\,,\end{aligned}$$

so $\operatorname{tr}(a_0) > 0$.

Conversely, assume that (i) $\operatorname{tr}(a) = 0$ and (ii) $\operatorname{tr}(a_0) > 0$, for all of its proper prefixes a_0. We will perform induction on a as a string. We first observe that assumption (i) forces us to disregard the case $a = \varepsilon$, so we only consider strings of the form $a = C_0 a'$, with $|a'| \geq 0$, in other words, such that $a = C_0 C_1 \cdots C_m$, where C_i's are constructors of arities r_i, for $i = 0, \ldots, m$, and $m \geq 0$. The induction takes the form of an induction over m, that is, the length of the tail a':

- For $m = 0$, it is $a = C_0$ with $\operatorname{tr}(C_0) = \operatorname{ar}(C_0) = 0$; a nullary constructor does indeed form a term.

- For $m \geq 1$ and $n \leq m$, the induction hypothesis is: if the tail a' of a is of length n, and assumptions (i) and (ii) hold, then a is a term. Since with respect to arities assumption (i) yields

$$r_0 + r_1 + \cdots + r_n = n\,, \tag{1}$$

  and assumption (ii) yields

$$\mathop{\forall}_{i=0}^{n-1} r_0 + r_1 + \cdots + r_i > i\,, \tag{2}$$

  we may reread the induction hypothesis as: *if the sequence $r_0, \ldots, r_n$ is a solution to the diophantine equation* (1) *under the conditions* (2)*, then it corresponds to (the constructor sequence of) a term.* Let us denote this associated diophantine system of (1) and (2) by $\mathcal{S}(n, r_0)$.

- For $m + 1$, the associated diophantine system is $\mathcal{S}(m+1, r_0)$:

$$\begin{gathered} r_0 + r_1 + \cdots + r_m + r_{m+1} = m + 1\,,\\ \mathop{\forall}_{i=0}^{m} r_0 + r_1 + \cdots + r_i > i\,. \end{gathered}$$

  By induction hypothesis, it suffices to show that for any fixed r_0, the *sequence* $r_1, \ldots, r_{m+1}$ is uniquely partitioned into r_0 subsequences $s_{j,0}, \ldots, s_{j,n_j}$, the j-th of which solves $\mathcal{S}(n_j, s_{j,0})$, for $j = 1, \ldots, r_0$. We do this by inner induction on r_0, which by (1) and (2) may take the values $1, \ldots, m$:

- For $r_0 = 1$, the partition that we seek is trivial, since it should correspond to an argument for the unary constructor C_0. The associated system $\mathcal{S}(m+1, 1)$ is

$$1 + r_1 + \cdots + r_m + r_{m+1} = m + 1 \; ,$$
$$\bigvee_{i=0}^{m} 1 + r_1 + \cdots + r_i > i \; .$$

  Set $s_i := r_{i+1}$, for $i = 0, \ldots, m-1$. Then $\mathcal{S}(m+1, 1)$ becomes

$$s_0 + \cdots + s_m = m \; ,$$
$$\bigvee_{i=0}^{m-1} s_0 + \cdots + s_i > i \; ,$$

  which is $\mathcal{S}(m, s_0)$, which is granted by the (outer) induction hypothesis.
- For $r_0 \geq 1$ and $r \leq r_0$, the inner induction hypothesis is: if the sequence $r, r_1, \ldots, r_{m+1}$ solves $\mathcal{S}(m+1, r)$, then its subsequence $r_1, \ldots, r_{m+1}$ breaks in turn uniquely into r subsequences $s_{j,0}, \ldots, s_{j,n_j}$, the j-th of which solves $\mathcal{S}(n_j, s_{j,0})$, for $n_j \leq m$, $s_{j,0} \leq r$ and $j = 1, \ldots, r$.
- For $r_0 + 1$ the system $\mathcal{S}(m+1, r_0+1)$ reads

$$(r_0 + 1) + r_1 + \cdots + r_{m+1} = m + 1 \; ,$$
$$\bigvee_{i=0}^{m} (r_0 + 1) + r_1 + \cdots + r_i > i \; .$$

  Since all initial partial sums add up to something less or equal to $m + 1$ (all arities are nonnegative), the inequality for $i := m$ gives

$$(r_0 + 1) + r_1 + \cdots + r_m = m + 1 \; ;$$

  by the equality, it follows that $r_{m+1} = 0$. But then $\mathcal{S}(m+1, r_0+1)$ reduces to $\mathcal{S}(m, r_0)$:

$$r_0 + r_1 + \cdots + r_m = m \; ,$$
$$\bigvee_{i=0}^{m-1} r_0 + r_1 + \cdots + r_i > i \; .$$

  The two induction hypotheses yield that the sequence $r_1, \ldots, r_m$ breaks up into r_0 terms, and since $r_{m+1} = 0$ also corresponds to a term, we have $r_0 + 1$ terms in total, and we are done.

□

So the solution to the recognition problem of K for an arbitrary $a \in S(K)$ is the following: for $i = 1, \ldots, |a| - 1$, check if $\mathrm{tr}\,(a_i) > 0$, where a_i is the initial segment of a with $|a_i| = i$; then check if $\mathrm{tr}\,(a) = 0$; the string is a well-formed term exactly when all checks turn positive. Observe that the complexity of the procedure is linear: for an n-letter string a, in the worst case (which is when we indeed have a valid term), we have to calculate n tree ranks, namely, the ranks of a's prefixes.

Example. Consider again the strings of page 1.The tree rank of ε is 1 by definition, and it is of course no term. The rank of 1 is 0 (we silently omit checking ε although it is strictly speaking a proper prefix), so it is a term. For $0S$ we have $\operatorname{tr}(0) = 0$, so we stop already and announce that the string is no term. Similarly, for $S10$, the algorithm halts at the prefix $S1$ and announces "no". For $SB{\ast}1$, we successively have:

$$\operatorname{tr}(S) = 1 > 0\,,\ \operatorname{tr}(SB) = 2 > 0\,,\ \operatorname{tr}(SB{\ast}) = 1 > 0\,,\ \operatorname{tr}(SB \ast 1) = 0\,,$$

so we have a valid term. For $BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast S0SB1{\ast}$ we have:

$$\begin{aligned}
&\operatorname{tr}(B) = 2 > 0\,,\\
&\operatorname{tr}(BB) = 3 > 0\,,\\
&\operatorname{tr}(BBB) = 4 > 0\,,\\
&\operatorname{tr}(BBBB) = 5 > 0\,,\\
&\operatorname{tr}(BBBBB) = 6 > 0\,,\\
&\operatorname{tr}(BBBBBB) = 7 > 0\,,\\
&\operatorname{tr}(BBBBBBB) = 8 > 0\,,\\
&\operatorname{tr}(BBBBBBBB) = 9 > 0\,,\\
&\operatorname{tr}(BBBBBBBBB) = 10 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB) = 11 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB{\ast}) = 10 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast) = 9 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast) = 8 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast) = 7 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast) = 6 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast) = 5 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast) = 4 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast) = 3 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast) = 2 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast S) = 2 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast S0) = 1 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast S0S) = 1 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast S0SB) = 2 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast S0SB1) = 1 > 0\,,\\
&\operatorname{tr}(BBBBBBBBBB \ast\ast\ast\ast\ast\ast\ast\ast\ast S0SB1{\ast}) = 0\,,
\end{aligned}$$

so we again have a valid term. □

3 Language-theoretic considerations

The notion of a free algebra, as given in the first section, is found in the literature of algebraic semantics as the *initial* algebra over $\{C^i \mid i = 1, \ldots, k\}$ (see [9], [10]). Indeed, the recognition problem—in this context also known as *word problem*—rightfully belongs to the realm of algebraic semantics; having a free algebra given by constructors, the first thing to do towards implementation is to build a parser, and this in turn needs a recognizer—in fact, recognizers and parsers are often identified and sometimes confused in nonexpert everyday parlance[1].

A standard way to go about this—described in classics like [3] or lecture notes like [4] or even relevant Wikipedia articles—is as follows. Let K be a signature, for example the one that we had in the first section. Firstly, determine the grammar $G(K)$ that generates the *language* of K, that is, that generates $T(K)$. Secondly, build the corresponding automaton $M(K)$ that recognizes the elements of $T(K)$.

Concerning the grammar, in our case we would necessarily have one *non-terminal* (also called "axiom") A, and all constructors would be *terminals*: $*$, 0, 1, S, B; as for the *production rules* of the grammar, we would have one per constructor:

$$A \to * \, , \; A \to 0 \, , \; A \to 1 \, , \; A \to SA \, , \; A \to BAA \; ;$$

Following the standard nomenclature of formal language theory, we first notice that the grammar is *context-free*. It is also *unambiguous* (every string has a unique leftmost derivation)—the term $BB01S*$ for example is produced in the following left-first steps:

$$A \Rightarrow BAA \Rightarrow BBAAA \Rightarrow BB0AA \Rightarrow BB01A \Rightarrow BB01SA \Rightarrow BB01S* \; .$$

Then, the grammar is not *linear* (because of the rule for B), so it cannot be *regular*.

Furthermore, concerning determinism, one has to look at pushdown automata. The default *non-deterministic* automaton $M(K)$ that corresponds to the grammar has one *state* q, its *inputs* are the constructors (terminals) $*$, 0, 1, S, B, together with the empty input ε^i, and its stacks are the "axiom" A together with the empty stack ε^s. Finally, its *behavior* δ consists of two kinds of quintuples, one *push* for each production rule:

$$(q, \varepsilon^i, A, q, *), \; (q, \varepsilon^i, A, q, 0), \; (q, \varepsilon^i, A, q, 1), \; (q, \varepsilon^i, A, q, SA), \; (q, \varepsilon^i, A, q, BAA),$$

and one *pop* for each constructor input:

$$(q, *, *, q, \varepsilon^s), \; (q, 0, 0, q, \varepsilon^s), \; (q, 1, 1, q, \varepsilon^s), \; (q, S, S, q, \varepsilon^s), \; (q, B, B, q, \varepsilon^s).$$

The automaton is non-deterministic, since the initial triple (q, ε^i, A) is present in more than one quintuples.

To sum it up, the grammar that we get following the standard textbook procedure up to this point is an unambiguous non-deterministic context-free grammar, and for such grammars, by Earley's algorithm [2], we know that the recognition problem is of quadratic complexity. But by the algorithm that we gave in section 2, we know we can achieve linear time; indeed, one can devise a *deterministic* pushdown automaton $M^d(K)$ that recognizes our language, hence certifying the grammar to be deterministic as well.

Here is an appropriate definition of $M^d(K)$ due to Fredrik Nordvall Forsberg. In addition to the state q, we employ two more states, a *starting state* q_0 and an *error state* q_∞. We allow here only nonempty inputs that is, the constructors $*$,

[1]The following post on the Theoretical Computer Science forum of StackExchange is relevant, though not nonexpert: `http://cstheory.stackexchange.com/questions/6411/formal-definition-of-parser?`.

input	action	stack status
—	—	ε^s
$\underline{B}B01S\ast$	push BAA	BAA
	pop B	AA
$\underline{B}01S\ast$	push BAA	$BAAA$
	pop B	AAA
$\underline{0}1S\ast$	push 0	$0AA$
	pop 0	AA
$\underline{1}S\ast$	push 1	$1A$
	pop 1	A
$\underline{S}\ast$	push SA	SA
	pop S	A
$\underline{\ast}$	push $\ast$	$\ast$
	pop $\ast$	ε^s
ε^i	halt	ε^s

input	action	stack status
—	—	ε^s
$\underline{\ast}S$	push $\ast$	$\ast$
	pop $\ast$	ε^s
$\underline{S}$	push SA	SA
	pop S	A
ε^i	halt	A

Figure 1: The automaton $M(K)$ processing the inputs $BB01S\ast$ (left) and $\ast S$ (right). In the first case it halts with an empty stack while in the second one not.

0, 1, S, B, while we preserve the stacks A and ε^s. The behavior will consist of different kinds of quintuples for each constructor input: one *start*,

$$(q_0, \ast, \varepsilon^s, q, \varepsilon^s),\ (q_0, 0, \varepsilon^s, q, \varepsilon^s),\ (q_0, 1, \varepsilon^s, q, \varepsilon^s),\ (q_0, S, \varepsilon^s, q, A),\ (q_0, B, \varepsilon^s, q, AA),$$

and one *process*,

$$(q, \ast, A, q, \varepsilon^s),\ (q, 0, A, q, \varepsilon^s),\ (q, 1, A, q, \varepsilon^s),\ (q, S, A, q, A),\ (q, B, A, q, AA),$$

as well as *abort* quintuples

$$(q, C, \varepsilon^s, q_\infty, \varepsilon^s) \text{ and } (q_\infty, C, \varepsilon^s, q_\infty, \varepsilon^s)\,,$$

for each constructor C; here, the process rules combine the corresponding push and pop rules of $M(K)$, by simultaneously refering to both constructor inputs and their corresponding production rules in the grammar.

It is interesting to notice that the automaton $M^d(K)$, by the definition above, operates in close correspondence to the tree rank criterion of page 3: the tree rank of the accessed input essentially corresponds to the stack memory of the machine at each step. In other words, the stack memory measures (in a unary notation) the tree rank of the accessed input up to that point—or prompts to the abort state when the tree rank drops too soon below one (see Figure 2).

Endnote

The arithmetical parsing technique described in this note stems from an (unfinished) attempt to code the structure of *coherent Scott information systems induced by finitary algebras*, as in [6], into Wolfram's *Mathematica* system[2]—which has yet to notice the interactive theorem proving boom of the recent decades, and

[2]See `http://www.wolfram.com/mathematica/`.

input	action	state	stack status
—	—	q_0	ε^s
$\underline{B}B01S*$	start with B	q	$\underline{AA}$
$\underline{B}01S*$	process B	q	$\underline{AA}A$
$\underline{0}1S*$	process 0	q	AA
$\underline{1}S*$	process 1	q	A
$\underline{S}*$	process S	q	$\underline{A}$
$\underline{*}$	process $*$	q	ε^s
ε^i	halt	q	ε^s

input	action	state	stack status
—	—	q_0	ε^s
$\underline{*}S$	start with $*$	q	ε^s
$\underline{S}$	abort on S	q_∞	ε^s
ε^i	halt	q_∞	A

Figure 2: The automaton $M^d(K)$ processing the inputs $BB01S*$ (up) and $*S$ (down). In the first case the word is accepted, since we halt with an empty stack while in the second one not, since we end up in the abort state. One can see how at every step no interaction is involved, that is, no non-deterministic choices.

consequently lacks a lot of basic relevant tools, the least of which is the matter of recognition in user-defined algebras.

Regarding parsing itself there is a vast literature out there. One could begin scratching the surface with the early account of [1] or the later [5] and [8, 7].

PS – Thanks to Rhea, Brent and Fredrik for the fun I had discussing the issue with them.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling. Vol. I: Parsing*. Prentice-Hall Inc., Englewood Cliffs, N. J., 1972. Prentice-Hall Series in Automatic Computation.

[2] Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, 1970.

[3] John E. Hopcroft and Jeffrey D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1969.

[4] Juhani Karhumäki. Automata and formal languages. Retrieved in September 2012 from `http://www.math.utu.fi/en/home/karhumak/`, 2005.

[5] Anton Nijholt. *Context-free grammars: covers, normal forms, and parsing*, volume 93 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[6] Helmut Schwichtenberg and Stanley S. Wainer. *Proofs and computations*. Perspectives in Logic. Cambridge University Press, Cambridge, 2012.

[7] Seppo Sippu and Eljas Soisalon-Soininen. *Parsing theory. Vol. I*, volume 15 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988. Languages and parsing.

[8] Seppo Sippu and Eljas Soisalon-Soininen. *Parsing theory. Vol. II*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1990. L$R(k)$ and L$L(k)$ parsing.

[9] TeReSe, editor. *Term rewriting systems*. Cambridge University Press, Cambridge, 2003. Cambridge Tracts in Theoretical Computer Science, Vol. 55.

[10] Wolfgang Wechler. *Universal algebra for computer scientists*, volume 25 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1992.