Harvard CS 121 and CSCI E-207

Lecture 8: Minimizing DFAs
Context-Free Grammars

Harry Lewis

September 26, 2013

Reading: Sipser, §2.1 (except Chomsky Normal Form).

A Final Note on Regular Languages and Finite Automata

For any regular language L, there are infinitely many different finite automata accepting L, and infinitely many different regular expressions representing L. Why?

A Final Note on Regular Languages and Finite Automata

For any regular language L, there are infinitely many different finite automata accepting L, and infinitely many different regular expressions representing L. Why?

For any regular language L, there is a **unique** *minimal* finite automaton M such that $L(M) = L$.

That is, $L(M) = L$, and for any other finite automaton M' such that $L(M') = L$, either M' has more states than M or its state diagram is isomorphic to that of M.

The minimal equivalent finite automaton can be found constructively.

Minimizing DFAs

Finding the minimal equivalent DFA:

- Let M be a DFA
- Without loss of generality assume all states are reachable
- Say that states p, q of M are *distinguishable* if there is a string w such that exactly one of $\delta^*(p, w)$ and $\delta^*(q, w)$ is final.
- Plan: Merge indistinguishable states
- Start by dividing the states of M into two equivalence classes: the final and non-final states

Minimizing DFAs, continued

- Break up the equivalence classes according to this rule: If p, q are in the same equivalence class but $\delta(p, \sigma)$ and $\delta(q, \sigma)$ are not equivalent for some $\sigma \in \Sigma$, then p and q must be separated into different equivalence classes
- When all the states that must be separated have been found, form a new and finer equivalence relation
- Repeat
- How do we know that this process stops?

Generalizations of FA

Can add:

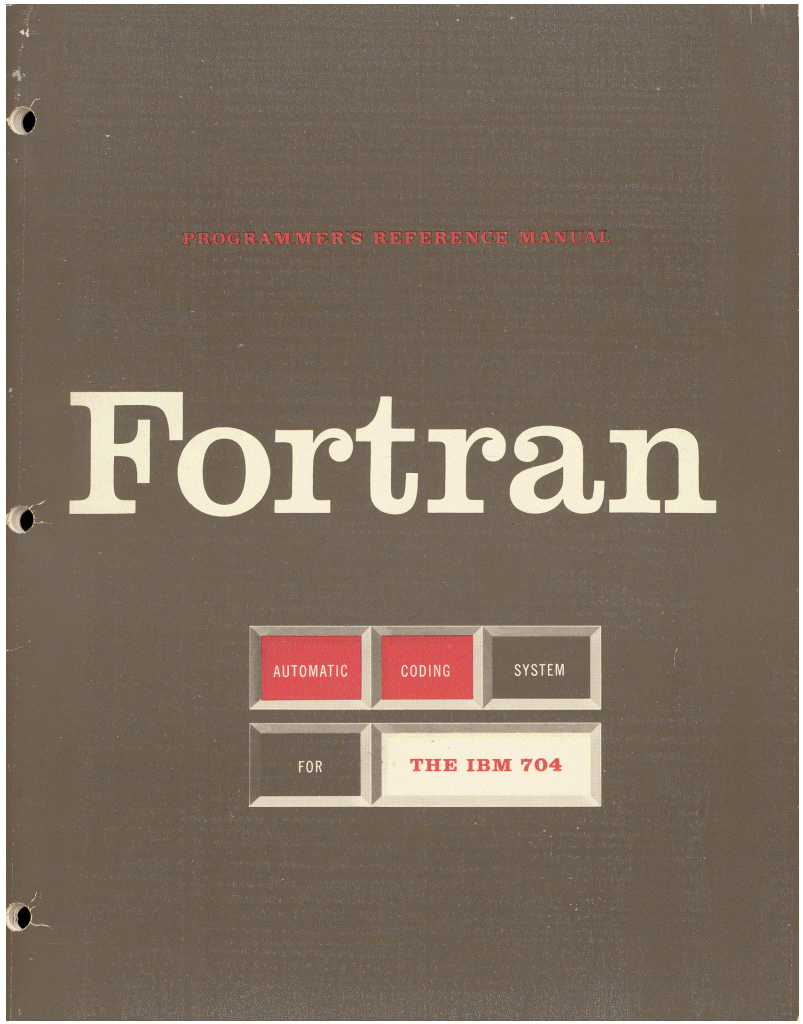
- probabilistic transitions (like Markov chains)
- outputs at each state
- rewards at each state
- infinite state spaces

Often referred to as “state machines”.

FORTRAN

John Backus

The Fortran Automatic Coding System for the IBM 704 EDPM (October 15, 1956)



A Fortran Lexical Definition

Functions

As in the above example, a FORTRAN expression may include the name of a *function* (e.g. the sine function SINF), provided that the routine for evaluating the function is either built into FORTRAN or is accessible to it as a pre-written subroutine in 704 language on the master FORTRAN tape.

GENERAL FORM	EXAMPLES
The name of the function is 4 to 7 alphabetic or numeric characters (not special characters), of which the last must be F and the first must be alphabetic. Also, the first must be X if and only if the value of the function is to be fixed point. The name of the function is followed by parentheses enclosing the arguments (which may be expressions), separated by commas.	SINF(A+B) SOMEF(X,Y) SQRTF(SINF(A)) XTANF(3.*X)

A Fortran Syntactic Definition

Formal Rules for Forming Expressions. By repeated use of the following rules, all permissible expressions may be derived.

1. Any fixed point (floating point) constant, variable, or subscripted variable is an expression of the same mode. Thus 3 and I are fixed point expressions, and ALPHA and A(I,J,K) are floating point expressions.
2. If SOMEF is some function of n variables, and if E, F, ... , H are a set of n expressions of the correct modes for SOMEF, then SOMEF (E, F, ... , H) is an expression of the same mode as SOMEF.
3. If E is an expression, and if its first character is not + or −, then +E and −E are expressions of the same mode as E. Thus −A is an expression, but +−A is not.
4. If E is an expression, then (E) is an expression of the same mode as E. Thus (A), ((A)), (((A))), etc. are expressions.
5. If E and F are expressions of the same mode, and if the first character of F is not + or −, then

```
E + F
E - F
E * F
E / F
```

are expressions of the same mode. Thus A−+B and A/+B are not expressions. The characters +, −, *, and / denote addition, subtraction, multiplication, and division.

Peter Naur

Revised Report on the Algorithmic Language Algol 60 (1962)

1.1 Formalism for syntactic description.

The syntax will be described with the aid of metalinguistic formulae (1).

(1) Cf. J. W. Backus, The syntax and semantics of the proposed international algebraic language of the Zuerich ACM-GRAMM conference. ICIP Paris, June 1959.

Their interpretation is best explained by an example:

```
<ab> ::= ( | [ | <ab> ( | <ab> <d>
```

Sequences of characters enclosed in the bracket <> represent metalinguistic variables whose values are sequences of symbols. The marks ::= and | (the latter with the meaning of **or**) are metalinguistic connectives. Any mark in a formula, which is not a variable or a connective, denotes itself (or the class of marks which are similar to it). Juxta position of marks and/or variables in a formula signifies juxtaposition of the sequences denoted. Thus the formula above gives a recursive rule for the formation of values of the variable <ab>. It indicates that <ab> may have the value (or [or that given some legitimate value of <ab>, another may be formed by following it with the character (or by following it with some value of the variable <d>. If the values of <d> are the decimal digits, some values of <ab> are:

```
[(((1(37(
(12345(
(((
[86
```

Noam Chomsky

1956

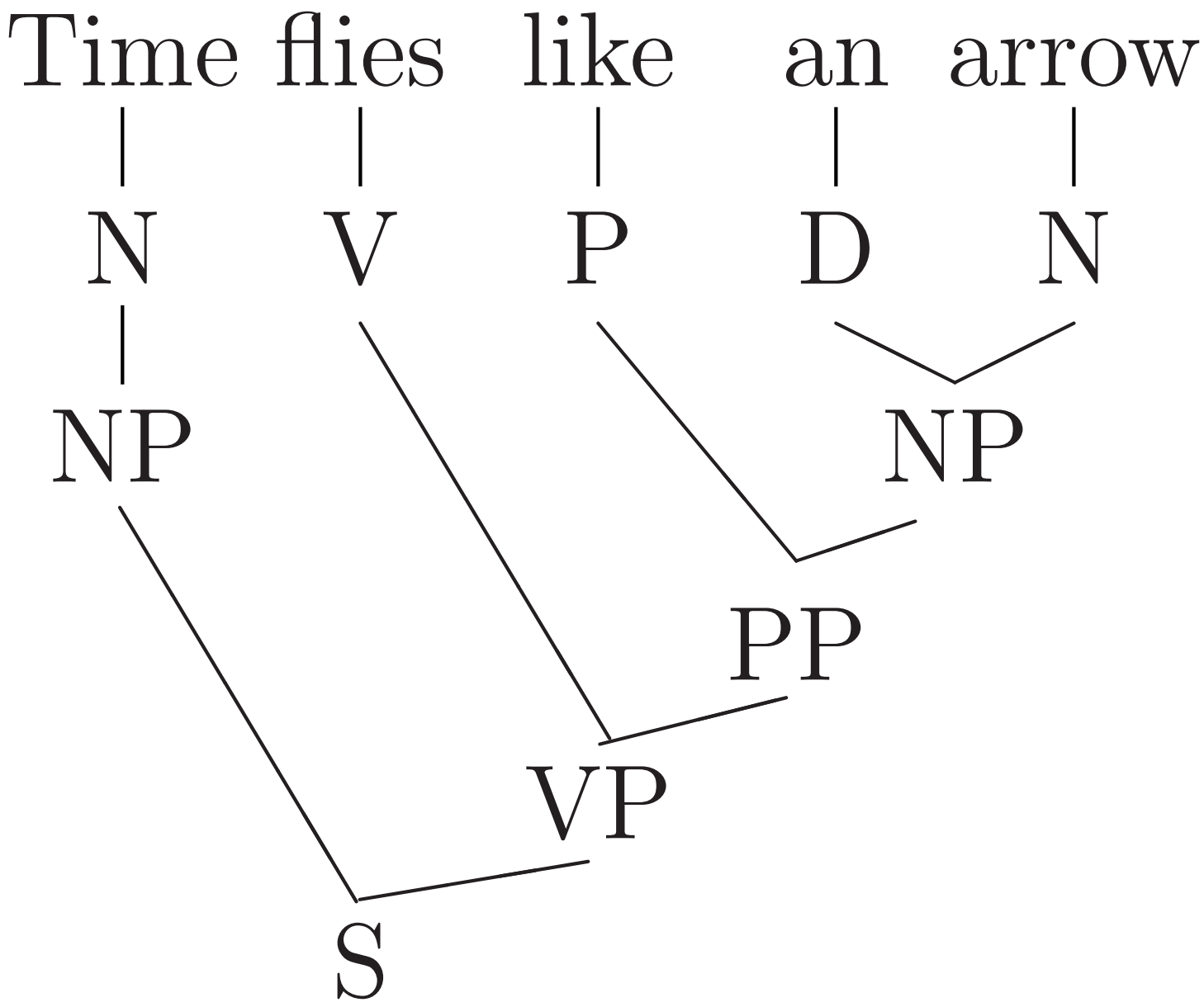
THREE MODELS FOR THE DESCRIPTION OF LANGUAGE*

Noam Chomsky
Department of Modern Languages and Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts

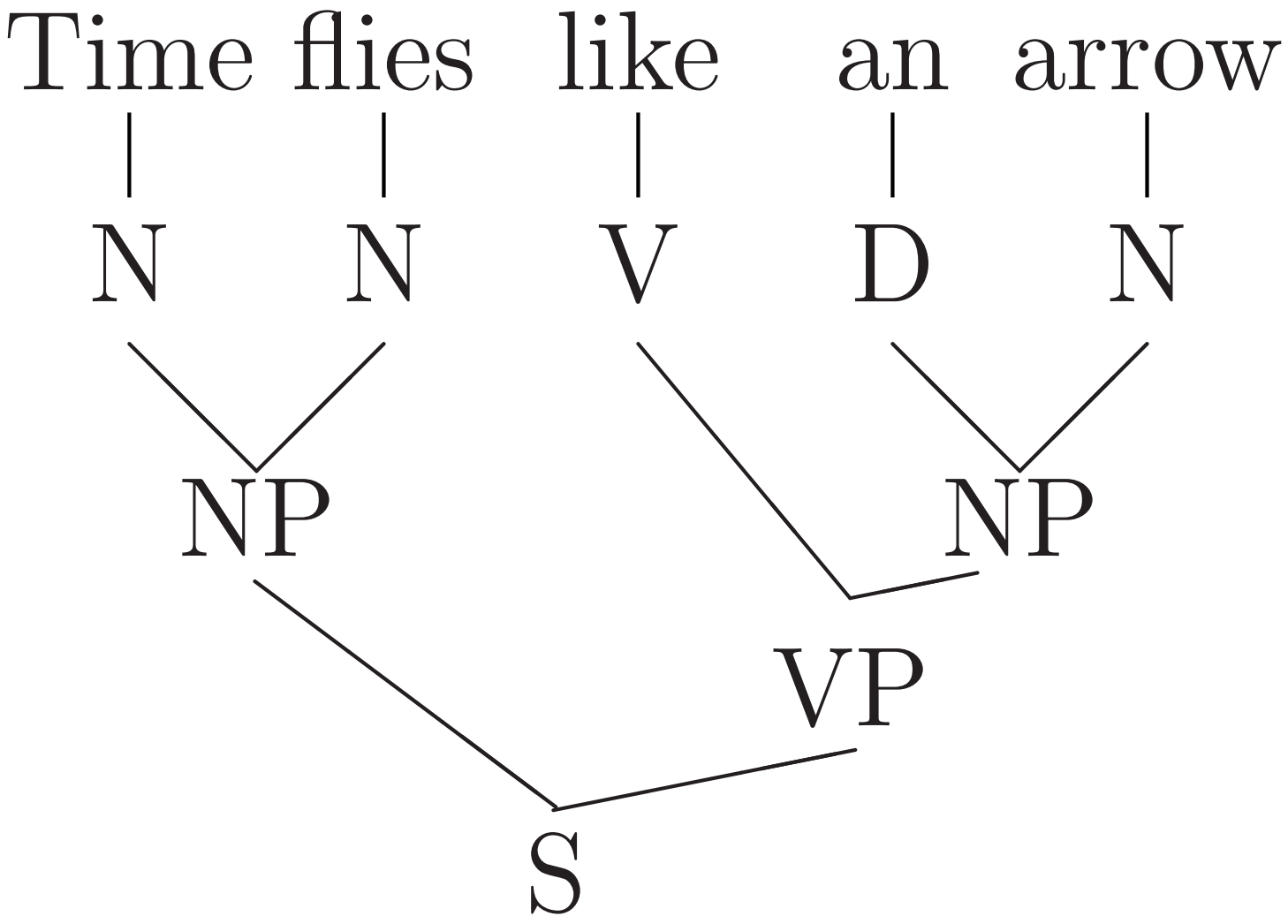
Abstract

We investigate several conceptions of linguistic structure to determine whether or not they can provide simple and "revealing" grammars that generate all of the sentences of English and only these. We find that no finite-state Markov process that produces symbols with transition from state to state can serve as an English grammar. Furthermore, the particular subclass of such processes that produce n-order statistical approximations to English do not come closer, with increasing n, to matching the output of an English grammar. We formalize the notions of "phrase structure" and show that this gives us a method for describing language which is essentially more observations, to show how they are interrelated, and to predict an indefinite number of new phenomena. A mathematical theory has the additional property that predictions follow rigorously from the body of theory. Similarly, a grammar is based on a finite number of observed sentences (the linguist's corpus) and it "projects" this set to an infinite set of grammatical sentences by establishing general "laws" (grammatical rules) framed in terms of such hypothetical constructs as the particular phonemes, words, phrases, and so on, of the language under analysis. A properly formulated grammar should determine unambiguously the set of grammatical sentences.

Parse Trees



Parse Trees



Context-Free Grammars

- Originated as abstract model for:
  - Structure of natural languages (Chomsky)
  - Syntactic specification of programming languages (Backus-Naur Form)
- A context-free grammar is a set of generative rules for strings

  e.g.

$$G = \begin{array}{l} S \to aSb \\ S \to \varepsilon \end{array}$$

- A derivation looks like:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

$$L(G) = \{\varepsilon, ab, aabb, \ldots\} = \{a^n b^n : n \geq 0\}$$

Equivalent Formalisms

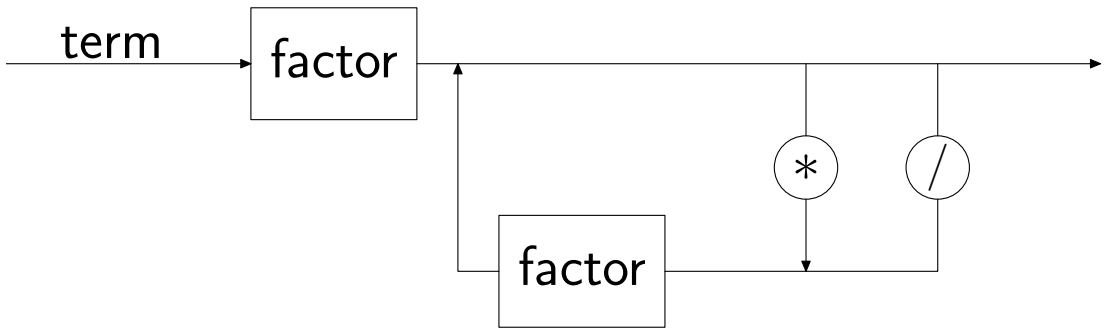
1. Backus-Naur Form (aka BNF, Backus Normal Form)

   due to John Backus and Peter Naur

   $$< \text{term} > ::= < \text{factor} > \mid < \text{factor} > * < \text{term} > \mid < \text{factor} > / < \text{term} >$$

   "$\mid$" means "or" in the metalanguage $=$ same left-hand side

2. "Railroad Diagrams"



Formal Definitions of CFG $G = (V, \Sigma, R, S)$

$V =$ Finite set of variables (or nonterminals)

$\Sigma =$ The alphabet, a finite set of terminals ($V \cap \Sigma = \emptyset$).

$R =$ A finite set of rules, each of the form $A \to w$ for $A \in V$ and $w \in (V \cup \Sigma)^*$.

$S =$ The start variable, a member of V

e.g. $(\{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S)$

Formal Definitions of CFG $G = (V, \Sigma, R, S)$

$V =$ Finite set of variables (or nonterminals)

$\Sigma =$ The alphabet, a finite set of terminals ($V \cap \Sigma = \emptyset$).

$R =$ A finite set of rules, each of the form $A \Rightarrow w$ for $A \in V$ and $w \in (V \cup \Sigma)^*$.

$S =$ The start variable, a member of V

e.g. $(\{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S)$

- **Derivations**: For $\alpha, \beta \in (V \cup \Sigma)^*$ (strings of terminals and nonterminals),

  $\alpha \Rightarrow_G \beta$ if $\alpha = uAv, \beta = uwv$ for some $u, v \in (V \cup \Sigma)^*$ and rule $A \to w$.

  $\alpha \Rightarrow_G^* \beta$ ("α yields β") if there is a sequence $\alpha_0, \ldots, \alpha_k$ for $k \geq 0$ such that $\alpha_0 = \alpha$, $\alpha_k = \beta$, and $\alpha_{i-1} \Rightarrow_G \alpha_i$ for each $i = 1, \ldots, k$.

- $L(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$ (strings of terminals only!)

Another example of a CFG

$L = \{x \in \{a, b\}^* : x$ has the same # of a's and b's$\}$.

Same Number of a's and bs

$G = (\{S, \}, \{a, b\}, R, S)$ where R has rules:

$S \to \varepsilon$

$S \to SS$

$S \to aSb$

$S \to bSa$

Claim: $L(G) = \{x : x$ has the same # of a's and b's$\}$

$x \in L(G) \Rightarrow x$ has the same # of a's and b's

Pf: Easy, every RHS has the same number of a's and b's. Formal proof by induction on length k of the derivation.

(a) $k = 1$: The derivation is $S \Rightarrow e = x$ and e has 0 a's, 0 b's. ✔

(b) $k > 1$: Then either:

1) $S \Rightarrow SS \Rightarrow^* xy$

2) $S \Rightarrow aSb \Rightarrow^* axb$

3) $S \Rightarrow bSa \Rightarrow^* bxa$

Since $S \Rightarrow^* x$, $S \Rightarrow^* y$ by derivs. of length $< k$, x, y have equal #'s of a's and b's (IH)

But then so do xy, axb, and bxa. ✔

x has the same # of a's and b's $\Rightarrow x \in L(G)$

Proof: by induction on $|x|$ (omit base case, let $|x| = k + 2$)

4 subcases depending on first and last symbols of x

(i) $x = ayb$ for some $y \in \Sigma^*$:

So $|y| = k$, and y has the same # of a's & b's.

By induction hypothesis, $S \Rightarrow^* y$

Hence $S \Rightarrow aSb \Rightarrow^* ayb = x$

(ii) $x = aya$ for some $y \in \Sigma^*$.

$|y| = k$ and y has 2 more b's than a's.

$y = uv$ for some u and v such that:

- u, v each has one more b than a.

Hence $S \Rightarrow^* au$ and $S \Rightarrow^* va$

So $S \Rightarrow^* SS \Rightarrow^* auva = x$, etc.