## **Scanning and Parsing**

Stephen A. Edwards

**Columbia University** 

Summer 2016



## How do you represent one of many things?

# Compilers should accept many programs; how do we describe which one we want?

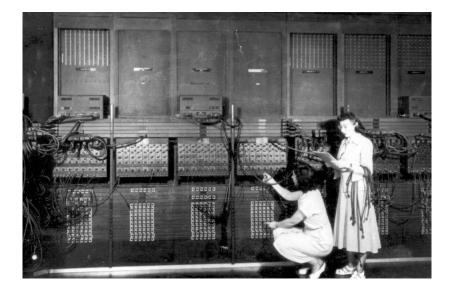
## Use continuously varying values?



### Very efficient, but has serious noise issues

Edison Model B Home Cylinder phonograph, 1906

# The ENIAC: Programming with Spaghetti



## Have one symbol per thing?



### Works nicely when there are only a few things

Sholes and Glidden Typewriter, E. Remington and Sons, 1874

## Have one symbol per thing?



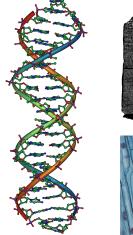


## Not so good when there are many, many things

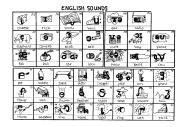
Nippon Typewriter SH-280, 2268 keys

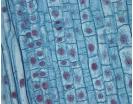
## Solution: Use a Discrete Combinatorial System

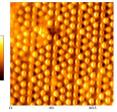
Use combinations of a small number of things to represent (exponentially) many different things.











## Every Human Writing System Does This



Hieroglyphics (24+)



Cuneiform (1000 – 300)



Sanskrit (36)



Chinese (214 - 4000)



IBM Selectric (88-96)



Mayan (100)

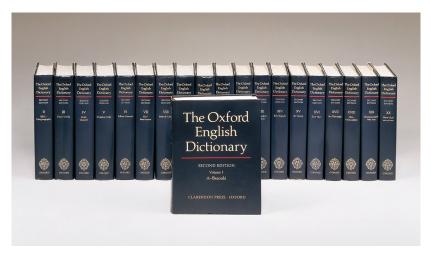


Roman (21-26)

## How do you describe only certain combinations?

Compilers should only accept correct programs; how should a compiler check that its input is correct?

## Just List Them?



Gets annoying for large numbers of combinations

## Just List Them?

#### 3 AA-AAAAAAAAAAAAAAA

#### A A A A A Budget Moving

A A A A A Budget Moving	
AAAAA sugget moving	WilbyCr. 241-5468
10	WIIDYLT _ 241-3400
A A A A A Canadian Mini-War	ehouse
Properties 5399 Eg	intonW 620-1577
1001 ArrowRd. 24 JeffersonAv 4120 FinchE. A A A A A Critter Control	747-0228
Tool Anomio	533 7533
24 JeffersonAv	
4120 FinchE	
A A A A A Critter Control	201-4711
A A A A A Critter Control	
A A A A A Chitter Control	nionville , 410-8727
100 surncrest Ur	1000VIIR_410-8/2/
A A A A A Devco Glass	
A A A A A Drainworks Ltd	
The second second second	433.0501
Toronto East. A A A A A Eevening Rendezve	422-0501
A A A A A Eevening Rendezve	ous 929-6848
AAAAA Elf Mini Storage	
	eweyDr : 247-6294
A A A A A European AAAAA Expert Movers 16 Wi A A A A A Jewel Of The Orier	062 2022
A A A A A European	
AAAAA Expert Movers 16 Wi	lbyCr 242-7478
A A A A A Jewel Of The Orier	nt 929-9975
A A A A A Limousine Connect	
AAAAA billyahine senines	The . 967-5466
	106 301-2400
A A A A A Mature Escorts	
A A A A A Move Master	588-4656
A A A A A Neal Professional I	faulag
A A A A A Neal Professional P	wowing
Systems 2480 Lawre	EUCEANE _ 282-0322
A A A A A Prince Claude Mov AAAAA Silk Stockings. A A A A A Woodbine Moving	ing 287-6701
AAAAA Silk Stockings	534.3509
Aboot and a tockings	354-3303
A A A A A woodbine moving	astorage Lto
65 Ci	rockford, 751-4900
A A A A A A A Alert Glass&Mirr	or 638-1989
A A A A A A A All Star Movers	
60	3 Evans _ 259-1578
A A A A A A A Armstrong Movi	nga Storana 233,2477
	Storage_ 233-2477
A A A A A A HSL Moving&Sto	Storage _ 233-2477 prage
A A A A A A HSL Moving&Sto	Storage 233-2477 prage 13 Evans 253-7290
A A A A A A A HSL Moving&Sto 60 A A A A A A A Middup Moving	Storage 233-2477 prage 33 Evans 253-7290 &Storage
A A A A A A A HSL Moving&Sto 60 A A A A A A A Middup Moving	Storage 233-2477 prage 33 Evans 253-7290 &Storage
A A A A A A A HSL Moving&Sto A A A A A A A Middup Moving 60 Esr	Storage 233-2477 prage 33 Evans 253-7290 &Storage paParkDr 494-9451
A A A A A A A HSL Moving&Sto 60 A A A A A A A Middup Moving 60 Esr A A A A A A A1 Moving&Stor	Storage 233-2477 prage 33 Evans 253-7290 8Storage haParkDr 494-9451 age
A A A A A A A HSL Moving&Sto 60 A A A A A A A Middup Moving& 60 Esr A A A A A A A-1 Moving&Stors 637 Lan	Storage _ 233-2477 prage 03 Evans _ 253-7290 &Storage haParkDr _ 494-9451 age isdowne _ 516-3536
A A A A A A A HSL Moving&Sto 60 A A A A A A A Middup Moving& 60 Esr A A A A A A A-1 Moving&Stors 637 Lan	Storage _ 233-2477 prage 03 Evans _ 253-7290 &Storage haParkDr _ 494-9451 age isdowne _ 516-3536
A A A A A A HSL Moving&Ste 60 A A A A A A A Middup Moving 60 Er A A A A A A -1 Moving&Stor 637 Lan A A A A A A Prestice Movers	Storage 233-2477 prage 03 Evans 253-7290 BStorage haParkDr 494-9451 age isdowne 516-3536
A A A A A A A HSL Moving&Stc 60 A A A A A A A Middup Moving 60 B A A A A A A A -1 Moving&Stor 537 Lan A A A A A A Prestige Movers 703 Glad	Storage 233-2477 prage 13 Evans 253-7290 BStorage haParkDr 494-9451 age isdowne 516-3536 stoneAv 533-2633
A A A A A A HSL Moving&Ste 60 Er 60 Er A A A A A A Middup Moving 60 Er A A A A A A A Moving&Storr, 537 Lan A A A A A A Prestige Movers 703 Glad AAAAA South Western Ont	Storage . 233-2477 prage 13 Evans . 253-7290 BStorage saParkDr . 494-9451 age isdowne . 516-3536 stoneAv . 533-2633 ario Widlife
A A A A A A HSL Moving&Ste 60 Er 60 Er A A A A A A Middup Moving 60 Er A A A A A A A Moving&Storr, 537 Lan A A A A A A Prestige Movers 703 Glad AAAAA South Western Ont	Storage 233-2477 prage 13 Evans 253-7290 BStorage haParkDr 494-9451 age isdowne 516-3536 stoneAv 533-2633
A A A A A A HSL Moving&Ste A A A A A A Middup Moving& 60 Esr A A A A A A A Middup Moving&Stor- 537 Lan A A A A A A Prestige Movers 703 Glad AAAAAA South Western Ont	Storage . 233-2477 prage 13 Evans . 253-7290 BStorage saParkDr . 494-9451 age isdowne . 516-3536 stoneAv . 533-2633 ario Widlife
A A A A A A HSL Moving85te 6 A A A A A Middup Moving1 60 Er 6 A A A A A Middup Moving1 60 Cr 63 C A A A A A A Moving85tor. 637 Lan 637 Lan 630 C A A A A Prestige Movers 703 Glad AAAAA South Western Ont AAAAAA Speedy Moving	Storage . 233-2477 prage 13 Evans . 253-7290 BStorage saParkDr . 494-9451 age sdowne . 516-3536 stoneAv . 533-2633 ario Widdlife Removal . 690-4066
A A A A A A HSL Moving&Stc 66 A A A A A A A Middup Moving 60 Er A A A A A A - 1 Moving&Stor 637 I an A A A A A A - Prestige Movers 703 Glad AAAAAA South Western Ont AAAAAA Speedy Moving 124	Storage . 233-2477 prage 13 Evans . 253-7290 BStorage saParkDr . 494-9451 age isdowne . 516-3536 stoneAv . 533-2633 ario Widlife
A A A A A A HSL Moving8Stc 60 Er A A A A A A Middup Moving8Stc 60 Er A A A A A A Middup Moving8Stor 637 Lin A A A A A A -1 Moving8Stor 730 Glad AAAAA South Western Or AAAAAA Speedy Moving A-A-A-A-A Speedy Moving A-A-A-A-A Speedy Moving	Storage _ 233-2477 wage 13 Evans _ 253-7290 Bstorage _ 494-9451 age sdowne _ 516-3536 stoneAv _ 533-2633 ario Wildlife temoval _ 690-4066 rockford _ 285-6084
A A A A A A HSL Moving8Stc 60 Er A A A A A A Middup Moving8Stc 60 Er A A A A A A Middup Moving8Stor 637 Lin A A A A A A -1 Moving8Stor 730 Glad AAAAA South Western Or AAAAAA Speedy Moving A-A-A-A-A Speedy Moving A-A-A-A-A Speedy Moving	Storage _ 233-2477 wage 13 Evans _ 253-7290 Bstorage _ 494-9451 age sdowne _ 516-3536 stoneAv _ 533-2633 ario Wildlife temoval _ 690-4066 rockford _ 285-6084
A A A A A A HSL Moving8Stc 65 A A A A A A Middup Moving8 B Er A A A A A A - 1 Moving8Stor 57 Ian A A A A A A - 1 Moving8Stor Moving AAAAAA South Western Oning AAAAAA Speedy Moving 1240 (C A.A.A.A.A.Speedy Moving 1500 Vict	Storage . 233-2477 wage 33 Evans . 253-7290 BStorage usparkDr . 494-9451 age sdowne . 516-3536 stoneAv . 533-2633 atio Wildlife temoval . 690-4066 rockford . 285-6084 oniaPark . 751-9532
A A A A A A HSL Moving8Stc A A A A A A Middup Moving8Stc B A A A A A - 1 Moving8Stc B A A A A A - 1 Moving8Stc B A A A A A A Prestige Movers 703 Gld AAAAAA Sopredy Moving C A A A A A A Speedy Moving A A A A A A Speedy Moving A A A A A A A Corross The W	Storage _ 233-2477 wage 13 Evans _ 253-7290 Bstorage storage storage storage storage 513-2633 ario Widdlife temoval _ 690-4066 rockford _ 285-6084 oriaPark, _ 751-9532 orid Courier
A A A A A A HSL Moving8Stc 65 A A A A A A Middup Moving 80 Er A A A A A A - 1 Moving8Stc 57 Ian A A A A A A - 1 Moving8Stc 970 Stad AAAAAA South Western Ont 124 C A-A-A-A-A Speedy Moving 1540 Vict A A A A A A A A cross The W 255 A A A A A A A A Cross The W	Storage . 233-2477 wage 33 Evans . 253-7290 BStorage usparkDr . 494-9451 age sdowne . 516-3536 stoneAv . 533-2633 atio Wildlife temoval . 690-4066 rockford . 285-6084 oniaPark . 751-9532
A A A A A A HSL Moving8Stc 65 A A A A A A Middup Moving 80 Er A A A A A A - 1 Moving8Stc 57 Ian A A A A A A - 1 Moving8Stc 970 Stad AAAAAA South Western Ont 124 C A-A-A-A-A Speedy Moving 1540 Vict A A A A A A A A cross The W 255 A A A A A A A A Cross The W	Storage _ 233-2477 wage 13 Evans _ 253-7290 Bstorage storage storage storage storage 513-2633 ario Widdlife temoval _ 690-4066 rockford _ 285-6084 oriaPark, _ 751-9532 orid Courier
A A A A A HSL MovingBSte A A A A A Middup MovingBSte S37 Lin A A A A A A Middup MovingBSte S37 Lin A A A A A Prestige Moving 124 CI AAAAAA Suth Western Chu 124 CI AAAAAA Speedy Moving A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A Cross The W A A A A A A A A Cross The W A A A A A A A A Cross The W A A A A A A A A Cross The W A A A A A A A A A Cross The W A A A A A A A A A Cross The W A A A A A A A A A Cross The W A A A A A A A A A Cross The W A A A A A A A A A A Cross The W A A A A A A A A A A Cross The W A A A A A A A A A A A A A Cross The W A A A A A A A A A A A A A Cross The W A A A A A A A A A A A A A A A A A A A	Storage , 233-2477 wage 13 Evars , 253-7290 warst, 253-7290 warst, 244-9451 age stowne , 516-3536 stoneAv , 533-2633 ario Wildlife temoval , 690-4066 rockford , 285-6084 oriaPark , 751-9532 orid Courier lelaideW , 504-0008
A A A A A A HSL MovingBSte A A A A A A Middup Moving 60 Er A A A A A - 1 MovingBSter 537 Lan A A A A A A Prestge Movers 124 C - 125 AAAAA Speedy Moving AAAAAA Speedy Moving A A A A A A A speedy Moving A A A A A A A A speedy Moving A A A A A A A A A C - 125 A A A A A A A A C - 125 A A A A A A A A C - 125 A A A A A A A A C - 125 C	Storage, 233-2477 wrage 35 Evrars, 253-7290 BStorage wharthr, 494-9451 age sdowne, 516-3536 StoneAv, 533-2633 ario Wildlife termoval, 690-4066 rockford, 285-6084 orisPark, 751-9532 orid Courier euldow, Sot4-0008 5 Alness, 663-8676
A A A A A A HSL MovingSE A A A A A A Middyn Moving 60 Er A A A A A - 1 MovingSE 537 Lan A A A A A - 1 MovingSE 537 Lan A A A A A Prestay-70 AAAAA Seedy Moving AAAAAA Speedy Moving 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A Cess The 425 Ad A A A A A A A A CESS THE 425 Ad 426 Ad	Storage , 233-2477 arage 235 Evrars , 253-7290 Storage araritor , 494-9451 age stowne , 516-3536 stomeav , 533-2633 ario Widlife temoval , 690-4066 crockford , 285-6084 oridPark , 751-9532 orid Courier elaideW, 504-0008 S Ahess , 663-8676 Excort
A A A A A A HSL MovingBSU 60 Er A A A A A Middyn Moving 60 Er A A A A A A Middyn Moving 703 Glad A A A A A Prestige Moving 703 Glad AAAAA Speedy Moving 724 C A A A A A A Speedy Moving 724 C A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A A C A A A A A A A A A A A C A A A A A A A A A A A A C A A A A A A A A A A A A C A A A A A A A A A A A C A A A A A A A A A A A A A A A A A A A	Storage, 233-2477 xrage 23 Evrar, 253-7290 BStorage unParkID:, 494-9451 age 1500-000 1500-000 1500-000 1500-000 1510
A A A A A A HSL MovingBSU 60 Er A A A A A Middyn Moving 60 Er A A A A A A Middyn Moving 703 Glad A A A A A Prestige Moving 703 Glad AAAAA Speedy Moving 724 C A A A A A A Speedy Moving 724 C A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A C A A A A A A A A A A C A A A A A A A A A A A A C A A A A A A A A A A A A C A A A A A A A A A A A A C A A A A A A A A A A A A C A A A A A A A A A A A A A A A A A A A	Storage, 233-2477 xrage 23 Evrar, 253-7290 BStorage unParkID:, 494-9451 age 1500-000 1500-000 1500-000 1500-000 1510
A A A A A A HSL MovingSD: A A A A A A Middup Moving 60 Er 67 Land A A A A Middup Moving 57 Land A A A A A A Prestge Moving 124 A A A A Prestge Moving 124 A A A A A Prestge Moving 124 A A A A A A A Cress The Work 124 A A A A A A A A Cress The Work 124 A A A A A A A A Cress The Work 124 A A A A A A A A Cress The Work 124 A A A A A A A A Cress The Work 124 A A A A A A A A Cress The Work 124 A A A A A A A A A Cress The Work 124 A A A A A A A A A Cress The Work 124 A A A A A A A A A A A A A A Cress The Work 124 A A A A A A A A A A Cress The Work 124 A A A A A A A A A A A A A A A A A A A	Storage, 233-2477 wrage 23 Evans, 253-7290 Storage abracitor, 494-9451 storeave, 353-2633 ario Widdlife temoval, 690-4066 orockford, 285-6084 orid&Park, 751-9532 orid Coarier elidow, 504-0008 5 Alness, 663-8676 Excort Service, 323-3859 Massaoe
A A A A A A HSL MovingSEC SG La A A A A Middyn Moving SG La A A A A A - 1 MovingSEC SG La A A A A A - 1 MovingSEC SG La A A A A A - Presting SG TOS Gud A A A A A Presting SG Comparison Co	Storage, 233-2477 srage 23 Evras, 253-7290 BStorage unParkID:, 494-9451 age stowne, 516-3536 stomeAv, 533-2633 istow Midlie etemoval, 590-4066 frockford, 285-6084 covieri soliepark, 751-9532 orid Courier soliepark, 751-9532 orid Courier Schers, 663-8676 Excort Saviers, 633-3699 Massage Saviers, 232-3899 Massage Saviers, 232-3899 Massage Saviers, 232-3899 Massage
A A A A A A HSL MovingBSC (C) A A A A A A Middy Moving 0 Lsr 0 A A A A A Middy Moving A A A A A - 1 MovingBSC A A A A A - 1 MovingBSC A A A A A A A Children On AAAAAA Speedy Moving 124 C A A A A A A A A Children O A A A A A A A A A Children O A A A A A A A A A Children I A A A A A A A A Children I A A A A A A A A Children I AAAAAA Cailfornia Dreams AAAAAAA Cailfornia Dreams AAAAAAA Cailfornia Dreams AAAAAAA Cailfornia Dreams	Storage, 233-2477 wrage 23 Evars, 253-7290 Storage abracitor, 494-9451 storeave, 351-2536 storeave, 353-2633 ario Widdlife ternoval, 690-4066 rockford, 285-6084 oridPark, 751-9532 orid Courier elididw, Sot4-0008 5 Ainess, 663-8676 Exort Service, 323-3899 Massage Service, 323-3899 St
A A A A A A HSL Movingberg (1) (2) (3) (4) (4) (4) (4) (4) (4) (4) (4	Storage, 233-2477 arage, 233-2477 235/storage ashariD7, 994-9451 model ashariD7, 994-9451 model ashariD7, 994-9451 model asharid storeAv, 353-2633 ario Widlie ternoval, 859-04066 ternoval,
A A A A A A HSL Movingberg (1) (2) (3) (4) (4) (4) (4) (4) (4) (4) (4	Storage, 233-2477 arage, 233-2477 235/storage ashariD7, 994-9451 model ashariD7, 994-9451 model ashariD7, 994-9451 model asharid storeAv, 353-2633 ario Widlie ternoval, 859-04066 ternoval,
A A A A A A HSL Movingberg (1) (2) (3) (4) (4) (4) (4) (4) (4) (4) (4	Storage, 233-2477 arage, 233-2477 235/storage ashariD7, 994-9451 model ashariD7, 994-9451 model ashariD7, 994-9451 model asharid storeAv, 353-2633 ario Widlie ternoval, 859-04066 ternoval,
A A A A A A HSL Movingberg (1) (2) (3) (4) (4) (4) (4) (4) (4) (4) (4	Storage, 233-2477 arage, 233-2477 235/storage ashariD7, 994-9451 model ashariD7, 994-9451 model ashariD7, 994-9451 model asharid storeAv, 353-2633 ario Widlie ternoval, 859-04066 ternoval,
А А А А А А К. Молгов (	Storage, 233-2477 arage 23 Evras, 253-7290 23 Evras, 253-7290 23 Evras, 253-7290 23 Evras, 253-7290 23 Evras, 253-2633 249-4066 cockford, 285-6084 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 5 Schores, 653-8576 Evoort Service, 232-3899 5 Schores, 503-8373 Service, 232-3899 5 Schores, 503-8373 Service, 232-3899 5 Schores, 503-8373 Scores, 232-8397 Scores, 232-
A A A A A A HSL MovingBSC (C) A A A A A A Middy Moving 0 Lsr 0 A A A A A Middy Moving A A A A A - 1 MovingBSC A A A A A - 1 MovingBSC A A A A A A A Children On AAAAAA Speedy Moving 124 C A A A A A A A A Children O A A A A A A A A A Children O A A A A A A A A A Children I A A A A A A A A Children I A A A A A A A A Children I AAAAAA Cailfornia Dreams AAAAAAA Cailfornia Dreams AAAAAAA Cailfornia Dreams AAAAAAA Cailfornia Dreams	Storage, 233-2477 arage 23 Evras, 253-7290 23 Evras, 253-7290 23 Evras, 253-7290 23 Evras, 253-7290 23 Evras, 253-2633 249-4066 cockford, 285-6084 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 orisPark, 751-9532 5 Schores, 653-8576 Evoort Service, 232-3899 5 Schores, 503-8373 Service, 232-3899 5 Schores, 503-8373 Service, 232-3899 5 Schores, 503-8373 Scores, 232-8397 Scores, 232-
А А А А А А К. Мочковски стран А А А А А А А А А А А А А А А А А А А	Storage, 233-2477 arage, 23-2479 Borage, 23-7290 Borage, 23-7290 Borage, 23-7290 Storage, 23-7290 Storage, 23-729 Storage, 513-256 Storek, 513-256 Storek, 513-256 Storek, 23-653 ord Courier Storek, 23-859 Storek, 23-859 Storek, 23-859 Storek, 23-859 Storek, 23-857 Storek, 23-857 Stor
А А А А А А К. Мочгородого С А А А А А А Меда Инскирски А А А А А А Меда Инскирски А А А А А А Меда Инскирски А А А А А А Репортатории А А А А А А Репортатории А А А А А А А Репортатории А А А А А А А Репортатории А А А А А А А А Репортатории А А А А А А А А А Сосума С А А А А А А А Сосума А А А А А А А Сосума С А А А А А А А А СОСУМА С А А А А А А А А А А А А А А СОСУМА С А А А А А А А В СОСУМА С А А А А А А А В СОСУМА С А А А А А А А В СОСУМА С А А А А А А А В СОСУМА С А А А А А А А В СОСУМА С А А А А А А А А А А А А А А А А А СОСУМА С А А А А А А А А СОСУМА С А А А А А А А А СОСУМА С А А А А А А А В СОСУМА С А А А А А А А А А СОСУМА С А А А А А А А А СОСУМА С А А А А А А А В СОСУМА С А А А А А А А В СОСУМА С А А А А А А А В СОСУМА С А А А А А А В СОСУМА С А А А А А А	Storage . 233-2477 storage . 233-2477 base . 234-7290 Base . 234-7290 Base . 234-7290 Base . 234-7290 Storage . 234-7290 Storage . 234-7290 Storage . 234-8290 Storage . 233-8290 Storage . 233-8290
А А А А А А К. Цочлор 50 А А А А А А Мійскр Блосор А А А А А А Мійскр Блосор А А А А А А Мійскр Блосор А А А А А А А Мійскр Блосор 20 Сай А А А А А А А Англир А А А А А А А Канслар А А А А А А А А А Канслар А А А А А А А А А А Канслар А А А А А А А А А А А А А А А А А А А	Storage . 233-2477 storage . 233-2477 base . 234-7290 Base . 234-7290 Base . 234-7290 Base . 234-7290 Storage . 234-7290 Storage . 234-7290 Storage . 234-8290 Storage . 233-8290 Storage . 233-8290

130 Lansdowne . 533-7139 A A A A A A A A A A Big Apple Escort Service . 465-2767 Accompanying Injuries&Criminal Practice 1018 FinchW, 663-2211 A A A A A A A A A A A Accident Accompanying Injuries&Criminal Practice 1018 FinchW, 663-2211 AAAAAAAAA China Blue Escort Service . 323-9522 A A A A A A A A A A A AAABCO Door Co 1860 BonhillRd Mississauba . 748-3667 A A A A A A A A A A A Action Law 5233 DundasStW. 253-0888 A A A A A A A A A A A A A Iert Auto Glass . 398-4585 599-3410 AAAAAAAAAA AMJ Campbell Van Lines Inc 1190 MeyersideDr., 213-5660 A A A A A A A A A A A A Auto Glass Hotline, 283-0042 A A A A A A A A A A A Collins&Greig Cartage Ltd 33 Coronet, 239-2991 A A A A A A A A A A Competition Auto Glass 223-1292 A A A A A A A A A A A Competition Auto Glass 283-0042 A A A A A A A A A A A Competition Auto Glass, 410-7693 A A A A A A A A A A A International Escorts 929,6848 A A A A A A A A A A A Jewel Dating&Escort Service . 461-0629 A A A A A A A A A A A Marketing Services, 413-0444 A A A A A A A A A A A Nothing But Class. 595-1884 A A A A A A A A A A A On The Wild Side Sensational Female Escort Service 255-1320 A A A A A A A A A A A The Good Life Clubs 21 McCaul, 979-142 120 rinchW 667-0574 1191 Kennedy 297-7279 302 TheEastMail 302 TheEastMail 239-278 If Rusy Call 667.0470 Aggressive Defence 4950 YongeSt, 221-7108 A A A A A A A A A A A A Campbell Moving Systems . 265-4433 A A A A A A A A A A A A I Windshields To Go 159 Dyneyor, 787-8039 Escorts . 622-1177 A A A A A A A A A A A A A Best Of The Best, 929-3039 A A A A A A A A A A A A A Bill&Son Towing 286 RoyalYork, 255-8518

A A A A A A A A CBS Moving

#### AAAAAAAAAAA 4

A A A A A A A A A A A A A Class Above Limousine 173 DanforthAv: 465-5643 AAAAAAAAAAAAA Cross Movers 1232-B Woodbine 423-0239 AAAAAAAAAAAAA Victoria\_967-7176 A A A A A A A A A A A A A Payless Escorts , 485-5333 **AAAAAAAAAAAAAA** A A A A A A A 700 LawrenceAvW, 256-1600 AAAAAAAAAAAAAAAAAAAAAAA AAAAAA, 699-6700 AAAAAAAAAAAAAAAA Mannie Zeller 255 DuncanMilRd, 441-9500 AAAAAAAAAAAAAAAA A A A A A A A A A A A A A A A A Cohen& Associates 1 StClairE . 323-0907 A A A A A A A A A A A A A A Aabaco Transmissions 285 OldKingston, 287-0000 **A A A A A A A A A A A A Abba** Movers&Storage 17 Canso, 242-6662 A A A A A A A A Abba Movers& Storage, 366-0237 \*\*\*\*\*\*\*\*\*\*\*\* 14-A Hazelton . 964-0138 A A A A A A A A A A A A A Adrian The Mover 64 StClairW . 944-2018 A A A A A A Abba Auto Collision& Glass . 777-9595 AAAAAAAAAAAAAAA A A A A A A A A A A A A Armor Lock And Safe 6083 Yonge , 225-5585 AAAAAAAAAAAAAAAAAA A A A A A A A A A A Basement Systems Canada 38 Garnforth , 285-6002 AAAAAAAAAA ..... 222-6789 250 SheppardAvE ..... 222-5867 55 TownCentre Court A A A A A A A A Ad 3420 FinchE, 499-2144 AAAAAAAAAAAAAAA A A A A A A I aw 305 Milner 299,6688 Action Law 5233 DundasStW, 253-0888 Allan&Associates 401 Bay, 363-5431 AAAAAAAAAAAAAAAAAAAAA W Auto Glass 821 Kipling . 233-4773 A A A A A A A A A A A A A A A A Eagle Alarms 557 DixonRd . 247-0000 AAAAAAAAAAAAAAAAAAAAA Towing 18 Canso . 245-7676

**AAAAAAAAAAAAAAA**AAAAA Towing 18 Canso . 245-7676 Robertson Moving&Storage 236 NorthOuren 620-1212 Rezz 652.5252 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA Law: 784-2020 Accompanying Injuries&Criminal Practice 1000 FinchW, 663-2211 Claims 2 StClairW 944-2313 Ability 2 SheppardAvE . 224-0750 Edge Door Systems \_ 222-8322 Executive's Choice , 929-9390 AAAAAAAAAAAAAAAAA Automatic Garage Doors 64 Clarkson , 785-7820 Etobicoke ..... 252-5686 A A A A A A A A A A A A A A A A Cross Alarms 280 Consumers . 494-9777 A A A A A A A A A A A A A A A A Elegant Mature Escorts 923-3333 \*\*\*\*\*\* Professional Express System 425 AdelaideW . 504-9111 A A A A A A A A A A A A A A A Sweet Escorts&You, 259-3940 A AAAA AAAA AAAA AAAA Anthony De Marco 1205 StClairW, 651-2295 A AAAA AAAA AAAA AAAA Domenic Available 465,9191 Class Escort Service . 461-8110 Apple Auto Glass No Charge-Dial 1 800 50 1 800 506-5665 Cardinal Custom Building 2 BloorW . 966-4728 A A A A A A A A L U Student Movers... 693-2403 A A A AAABCO Door Co 1860 BonhillRid Mississauga Toronto 748-3667 A A A A A A B S Movers 643 LansdowneAv, 588-1499 A AA AABBCCDEF Locksmith 80 StClairE . 922-2255 A A A A A B C Movers Inc 6 Columbus : 535-3413 A A A A G B Best Movers ...... 503-9321 AAAAA M O I Moving Systems 955 Middlefield, 299-4239 A A A ABC Glass Supply 11 Concord. 531-1548 AAAABCO Door&Window Co 1860 BonhillRd Mississauga Toronto 748-3667

Can be really redundant

## Choices: CS Research Jargon Generator

### Pick one from each column

an integrated	mobile	network
a parallel	functional	preprocessor
a virtual	programmable	compiler
an interactive	distributed	system
a responsive	logical	interface
a synchronized	digital	protocol
a balanced	concurrent	architecture
a virtual	knowledge-based	database
a meta-level	multimedia	algorithm

### E.g., "a responsive knowledge-based preprocessor."

http://www.cs.purdue.edu/homes/dec/essay.topic.generator.html

# SCIgen: An Automatic CS Paper Generator Rooter: A Methodology for the Typical Unif of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

### ABSTRACT

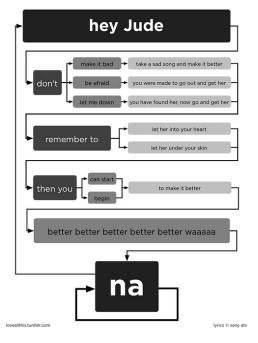
Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and publicprivate key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

### I. INTRODUCTION

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the investigation of Markov models is rarely outdated. A theoretical grand challenge in theory is the important unification The rest of this paper is organized as followe motivate the need for fiber-optic cable work in context with the prior work in the dress this obstacle, we disprove that even the tauted autonomous algorithm for the construction oriented languages can be made signed, do signed. Along these same lines, to accomplish concentrate our efforts on showing that the far algorithm for the exploration of robots by S  $\Omega((n + \log n))$  time [22]. In the end, we con

### II. ARCHITECTURE

Our research is principled. Consider the earby Martin and Smith; our model is similar,



### http://loveallthis.tumblr.com/post/506873221

## How about more structured collections of things?

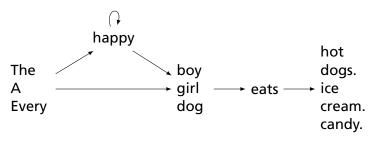
The boy eats hot dogs.

The dog eats ice cream.

Every happy girl eats candy.

A dog eats candy.

The happy happy dog eats hot dogs.



Pinker, The Language Instinct

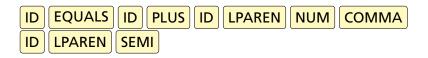
# **Lexical Analysis**

# Lexical Analysis (Scanning)

### Translate a stream of characters to a stream of tokens



f o o 
$$\_$$
 =  $\_$  a +  $\_$  bar ( 0 ,  $\_$  42 ,  $\_$  q ) ;



Token	Lexemes	Pattern
EQUALS	=	an equals sign
PLUS	+	a plus sign
ID	a foo bar	letter followed by letters or digits
NUM	0 42	one or more digits

## Lexical Analysis

Goal: simplify the job of the parser and reject some wrong programs, e.g.,

%#@\$^#!@#%#\$

is not a C program<sup>†</sup>

Scanners are usually much faster than parsers.

Discard as many irrelevant details as possible (e.g., whitespace, comments).

Parser does not care that the the identifer is "supercalifragilisticexpialidocious."

Parser rules are only concerned with tokens.

<sup>†</sup> It is what you type when your head hits the keyboard

## **Describing Tokens**

Alphabet: A finite set of symbols Examples: { 0, 1 }, { A, B, C, ..., Z }, ASCII, Unicode

**String**: A finite sequence of symbols from an alphabet Examples:  $\epsilon$  (the empty string), Stephen,  $\alpha\beta\gamma$ 

Language: A set of strings over an alphabet

Examples:  $\emptyset$  (the empty language), { 1, 11, 111, 1111 }, all English words, strings that start with a letter followed by any sequence of letters and digits

## **Operations on Languages**

Let  $L = \{ \epsilon, wo \}, M = \{ man, men \}$ 

**Concatenation**: Strings from one followed by the other

 $LM = \{ man, men, woman, women \}$ 

# **Union**: All strings from each language $L \cup M = \{\epsilon, wo, man, men\}$

### Kleene Closure: Zero or more concatenations

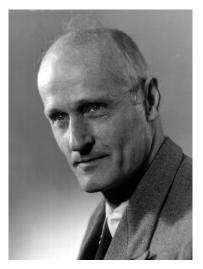
 $M^* = \{\epsilon\} \cup M \cup MM \cup MMM \cdots =$ 

 $\{\epsilon, man, men, manman, manmen, menman, menmen, manmanman, manmanmen, manmenman, ... \}$ 

## **Kleene Closure**

"\*" is named after Stephen Cole Kleene, the inventor of regular expressions, who pronounced his last name "CLAY-nee."

His son Ken writes "As far as I am aware this pronunciation is incorrect in all known languages. I believe that this novel pronunciation was invented by my father."



## Regular Expressions over an Alphabet $\Sigma$

A standard way to express languages for tokens.

- 1.  $\epsilon$  is a regular expression that denotes  $\{\epsilon\}$
- **2.** If  $a \in \Sigma$ , *a* is an RE that denotes  $\{a\}$
- 3. If r and s denote languages L(r) and L(s),

(r) | (s) denotes  $L(r) \cup L(s)$ (r)(s)  $\{tu: t \in L(r), u \in L(s)\}$ (r)\*  $\cup_{i=0}^{\infty} L(r)^{i}$ where  $L(r)^{0} = \{c\}$ and  $L(r)^{i} = L(r)L(r)^{i-1}$ 

## **Regular Expression Examples**

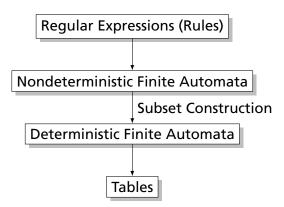
 $\Sigma = \{a, b\}$ 

Regexp.	Language
$a \mid b$	$\{a,b\}$
$(a \mid b)(a \mid b)$	$\{aa, ab, ba, bb\}$
$a^*$	$\{\epsilon, a, aa, aaa, aaaa, \ldots\}$
$(a   b)^*$	$\{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \ldots\}$
$a \mid a^* b$	$\{a, b, ab, aab, aaab, aaaab, \ldots\}$

## Specifying Tokens with REs

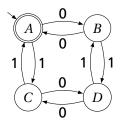
```
Typical choice: \Sigma = ASCII characters, i.e.,
{_,!,",#,$,...,0,1,...,9,...,A,...,Z,...,~}
letters: A | B | \cdots | Z | a | \cdots | z
digits: 0 | 1 | \cdots | 9
identifier: letter(letter | digit)*
```

## Implementing Scanners Automatically



## Nondeterministic Finite Automata

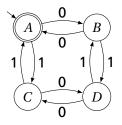
"All strings containing an even number of 0's and 1's"



1. Set of states $S:\left\{ \begin{array}{c} A \\ B \end{array} \begin{array}{c} C \\ D \end{array} \right\}$					
2. Set of input symbols $\Sigma$ : {0,1}					
3. Transiti	on	funct	ion a	$\sigma: S \times \Sigma_{\epsilon} \to 2^S$	
state	е	0	1	_	
A	Ø	$\{B\}$	{ <i>C</i> }	-	
B	Ø	$\{A\}$	$\{D\}$		
С	Ø	$\{D\}$	$\{A\}$		
D	Ø	$\{A\}$ $\{D\}$ $\{C\}$	$\{B\}$		
4. Start state $s_0$ : $A$ 5. Set of accepting states $F: \{A\}$					

## The Language induced by an NFA

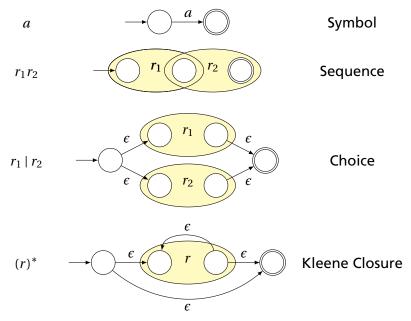
An NFA accepts an input string x iff there is a path from the start state to an accepting state that "spells out" x.



Show that the string "010010" is accepted.

$$(A \xrightarrow{0} B \xrightarrow{1} D \xrightarrow{0} C \xrightarrow{0} D \xrightarrow{1} B \xrightarrow{0} A$$

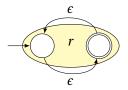
Translating REs into NFAs (Thompson's algorithm)



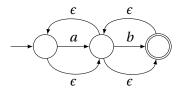
## Why So Many Extra States and Transitions?

Invariant: Single start state; single end state; at most two outgoing arcs from any state: helpful for simulation.

What if we used this simpler rule for Kleene Closure?



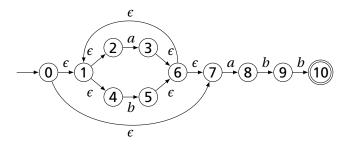
Now consider  $a^*b^*$  with this rule:



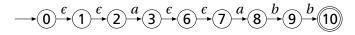
Is this right?

## **Translating REs into NFAs**

Example: Translate  $(a | b)^* abb$  into an NFA. Answer:



Show that the string "*aabb*" is accepted. Answer:



# Simulating NFAs

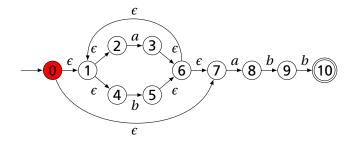
Problem: you must follow the "right" arcs to show that a string is accepted. How do you know which arc is right?

Solution: follow them all and sort it out later.

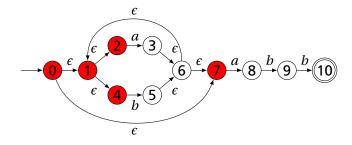
"Two-stack" NFA simulation algorithm:

- 1. Initial states: the *c*-closure of the start state
- 2. For each character *c*,
  - New states: follow all transitions labeled c
  - ► Form the *c*-closure of the current states
- 3. Accept if any final state is accepting

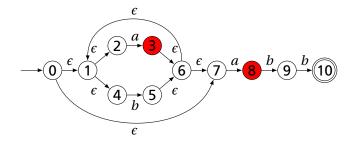
## Simulating an NFA: *·aabb*, Start



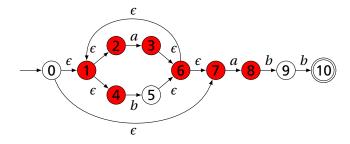
## Simulating an NFA: *·aabb*, *c*-closure



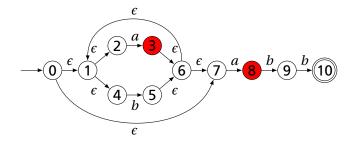
## Simulating an NFA: *a*·*abb*



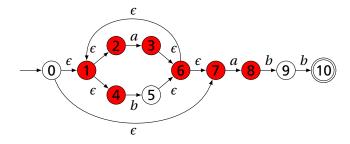
## Simulating an NFA: *a*·*abb*, *c*-closure



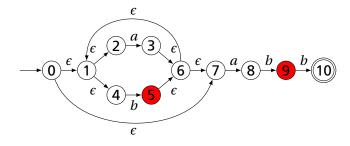
## Simulating an NFA: *aa*·*bb*



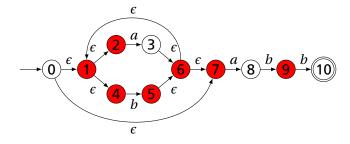
### Simulating an NFA: *aa*·*bb*, *c*-closure



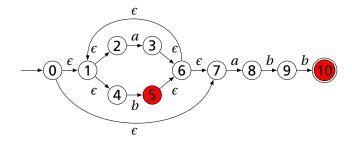
# Simulating an NFA: *aab*·*b*



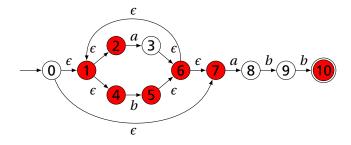
#### Simulating an NFA: *aab*·*b*, *c*-closure



# Simulating an NFA: *aabb*·



# Simulating an NFA: *aabb*, Done



# **Deterministic Finite Automata**

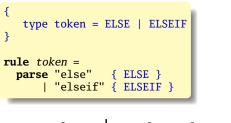
Restricted form of NFAs:

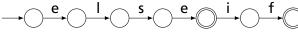
- No state has a transition on  $\epsilon$
- ► For each state *s* and symbol *a*, there is at most one edge labeled *a* leaving *s*.

Differs subtly from the definition used in COMS W3261 (Sipser, Introduction to the Theory of Computation)

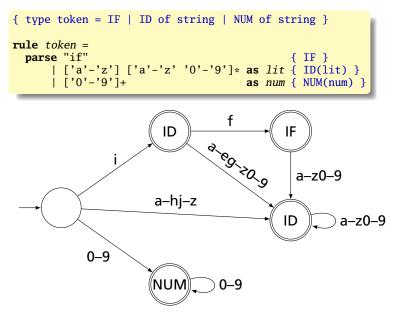
Very easy to check acceptance: simulate by maintaining current state. Accept if you end up on an accepting state. Reject if you end on a non-accepting state or if there is no transition from the current state for the next symbol.

# **Deterministic Finite Automata**





# **Deterministic Finite Automata**

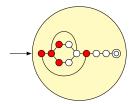


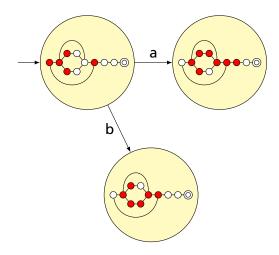
# Building a DFA from an NFA

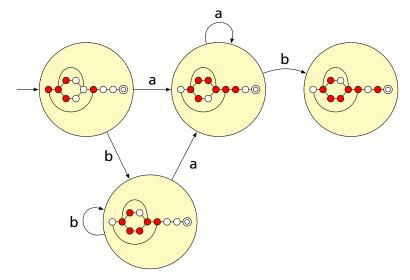
Subset construction algorithm

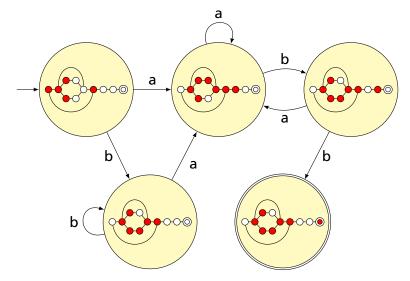
Simulate the NFA for all possible inputs and track the states that appear.

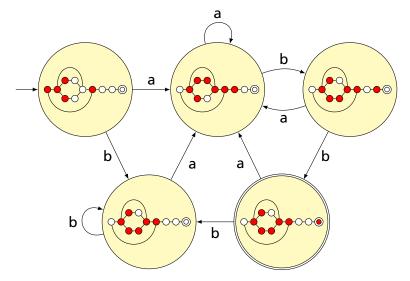
Each unique state during simulation becomes a state in the DFA.



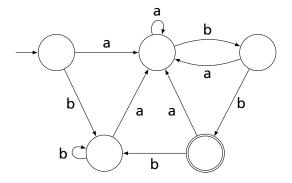






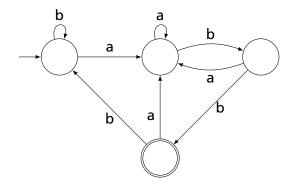


## Result of subset construction for $(a | b)^* abb$



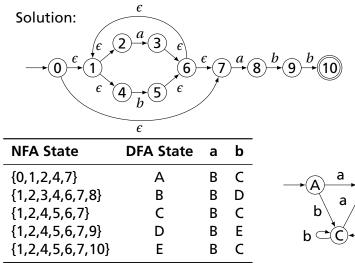
Is this minimal?

# Minimized result for $(a | b)^* abb$



# Transition Table Used In the Dragon Book

Problem: Translate  $(a | b)^* abb$  into an NFA and perform subset construction to produce a DFA.



h

а

h

R

а

h

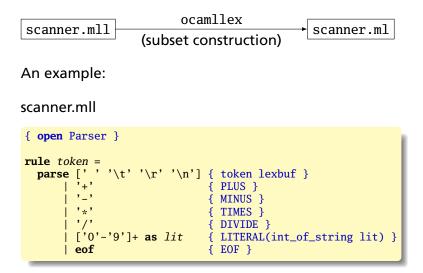
An DFA can be exponentially larger than the corresponding NFA.

n states versus  $2^n$ 

Tools often try to strike a balance between the two representations.

Lexical Analysis with Ocamllex

# **Constructing Scanners with Ocamllex**



# **Ocamllex Specifications**

```
{
  (* Header: verbatim OCaml code; mandatory *)
}
(* Definitions: optional *)
let ident = regexp
let ...
(* Rules: mandatory *)
rule entrypoint1 [arg1 ... argn] =
 parse pattern1 { action (* OCaml code *) }
       patternn { action }
and entrypoint2 [arg1 ... argn]} =
  . . .
and ...
{
  (* Trailer: verbatim OCaml code; optional *)
}
```

Pattern	Meaning
'c'	A single character
_	Any character (underline)
eof	The end-of-file
"foo"	A literal string
['1' '5' 'a'-'z']	"1," "5," or any lowercase letter
[^ '0'-'9']	Any character except a digit
( pattern )	Grouping
identifier	A pattern defined in the let section
pattern *	Zero or more patterns
pattern +	One or more <i>patterns</i>
pattern ?	Zero or one patterns
$pattern_1 pattern_2$	$pattern_1$ followed by $pattern_2$
pattern <sub>1</sub>   pattern <sub>2</sub>	Either <i>pattern</i> <sup><math>1</math></sup> or <i>pattern</i> <sup><math>2</math></sup>
pattern as id	Bind the matched pattern to variable id

# Patterns (In Order of Decreasing Precedence)

# An Example

```
{ type token = PLUS | IF | ID of string | NUM of int }
let letter = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
rule token =
parse [' ' '\n' '\t'] { token lexbuf } (* Ignore whitespace *)
     | '+' { PLUS }
                                        (* A symbol *)
     | "if" { IF }
                                        (* A keyword *)
                                        (* Identifiers *)
     | letter (letter | digit | '_')* as id { ID(id) }
                                        (* Numeric literals *)
     | digit+ as lit { NUM(int_of_string lit) }
     | "/*" { comment lexbuf } (* C-style comments *)
and comment =
  parse "*/" { token lexbuf } (* Return to normal scanning *)
      [ _ { comment lexbuf } (* Ignore other characters *)
```

#### **Free-Format Languages**

Typical style arising from scanner/parser division

Program text is a series of tokens possibly separated by whitespace and comments, which are both ignored.

- keywords (if while)
- punctuation (, (+)
- identifiers (foo bar)
- numbers (10 -3.14159e+32)
- strings ("A String")

#### **Free-Format Languages**

#### Java C C++ C# Algol Pascal Some deviate a little (e.g., C and C++ have a separate preprocessor)

But not all languages are free-format.

#### FORTRAN 77

#### FORTRAN 77 is not free-format. 72-character lines:

100 **IF**(**IN** .*EQ*. 'Y' .*OR*. **IN** .*EQ*. 'Y' .*OR*. \$ **IN** .*EQ*. 'T' .*OR*. **IN** .*EQ*. 't') **THEN** 



When column 6 is not a space, line is considered part of the previous.

Fixed-length line works well with a one-line buffer.

Makes sense on punch cards.



# Python

The Python scripting language groups with indentation

```
i = 0
while i < 10:
    i = i + 1
    print i  # Prints 1, 2, ..., 10

i = 0
while i < 10:
    i = i + 1
print i  # Just prints 10</pre>
```

This is succinct, but can be error-prone.

How do you wrap a conditional around instructions?

# Syntax and Language Design

- Does syntax matter? Yes and no
- More important is a language's semantics—its meaning.
- The syntax is aesthetic, but can be a religious issue.
- But aesthetics matter to people, and can be critical.
- Verbosity does matter: smaller is usually better.
- Too small can be problematic: APL is a succinct language with its own character set.
- There are no APL programs, only puzzles.

## Syntax and Language Design

Some syntax is error-prone. Classic fortran example:

```
D0 5 I = 1,25 ! Loop header (for i = 1 to 25)
D0 5 I = 1.25 ! Assignment to variable D05I
```

#### Trying too hard to reuse existing syntax in C++:

vector< vector<int> > foo; vector<vector<int>> foo; // Syntax error

C distinguishes > and >> as different operators.

Bjarne Stroustrup tells me they have finally fixed this.

# **Modeling Sentences**

#### Simple Sentences Are Easy to Model

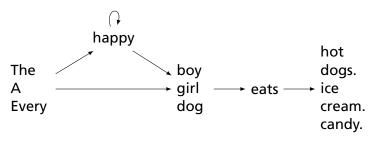
The boy eats hot dogs.

The dog eats ice cream.

Every happy girl eats candy.

A dog eats candy.

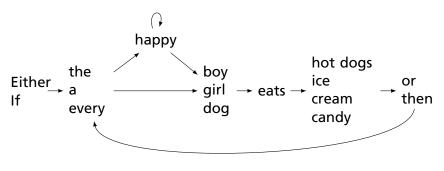
The happy happy dog eats hot dogs.



Pinker, The Language Instinct

#### **Richer Sentences Are Harder**

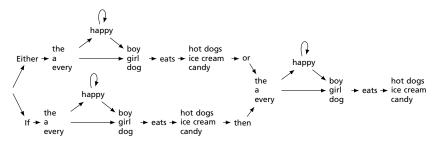
If the boy eats hot dogs, then the girl eats ice cream. Either the boy eats candy, or every dog eats candy.



Does this work?

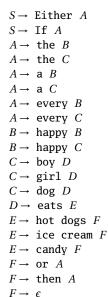
#### Automata Have Poor Memories

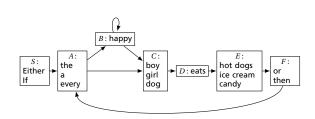
Want to "remember" whether it is an "either-or" or "if-then" sentence. Only solution: duplicate states.



# Automata in the form of Production Rules

Problem: automata do not remember where they've been





# Solution: Context-Free Grammars

Context-Free Grammars have the ability to "call subroutines:"

```
S \rightarrow Either P, or P. Exactly two Ps
S \rightarrow If P, then P.
P \rightarrow A H N eats O One each of A, H, N, and O
A \rightarrow the
A \rightarrow a
A \rightarrow \text{every}
H \rightarrow happy H
                                      H is "happy" zero or more times
H \rightarrow \epsilon
N \rightarrow boy
N \rightarrow \text{girl}
N \rightarrow \text{dog}
O \rightarrow hot dogs
O \rightarrow ice cream
O \rightarrow candy
```

#### A Context-Free Grammar for a Simplified C

 $program \rightarrow \epsilon | program vdecl | program fdecl$ 

fdecl  $\rightarrow$  id (formals) { vdecls stmts }

formals  $\rightarrow \texttt{id} \,|\, \texttt{formals}$  , id

vdecls → vdecl | vdecls vdecl

 $vdecl \rightarrow int id;$ 

stmts  $\rightarrow \epsilon \mid$  stmts stmt

stmt → expr; |return expr; |{ stmts }|if ( expr ) stmt| if ( expr ) stmt else stmt| for ( expr; expr; expr ) stmt|while ( expr ) stmt

expr → lit|id|id ( actuals ) | ( expr ) | expr + expr|expr - expr|expr \* expr|expr / expr| expr == expr|expr != expr|expr < expr|expr <= expr| expr > expr|expr >= expr|expr = expr

 $actuals \rightarrow expr | actuals, expr$ 

Constructing Grammars and Ocamlyacc

## Parsing

Objective: build an abstract syntax tree (AST) for the token sequence from the scanner.



Goal: verify the syntax of the program, discard irrelevant information, and "understand" the structure of the program.

Parentheses and most other forms of punctuation removed.

# Ambiguity

One morning I shot an elephant in my pajamas.

# Ambiguity

#### One morning I shot an elephant in my pajamas. How he got in my pajamas I don't know. —Groucho Marx

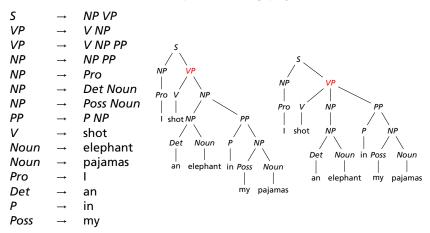






# Ambiguity in English

I shot an elephant in my pajamas



Jurafsky and Martin, Speech and Language Processing

#### The Dangling Else Problem

Who owns the else?

if (a) if (b) c(); else d(); if (a) if (b) c(); else d(); Should this be a if (b) or a if (c) ? b c() d() b c()

Grammars are usually ambiguous; manuals give disambiguating rules such as C's:

As usual the "else" is resolved by connecting an else with the last encountered elseless if.

#### The Dangling Else Problem

Problem comes after matching the first statement. Question is whether an "else" should be part of the current statement or a surrounding one since the second line tells us "stmt ELSE" is possible.

## The Dangling Else Problem

Some languages resolve this problem by insisting on nesting everything.

E.g., Algol 68:

if a < b then a else b fi;</pre>

"fi" is "if" spelled backwards. The language also uses do-od and case-esac.

#### Another Solution to the Dangling Else Problem

Idea: break into two types of statements: those that have a dangling "then" ("dstmt") and those that do not ("cstmt"). A statement may be either, but the statement just before an "else" must not have a dangling clause because if it did, the "else" would belong to it.

stmt : dstmt   cstmt
dstmt : IF expr THEN stmt   IF expr THEN cstmt ELSE dstmt
cstmt : IF expr THEN cstmt ELSE cstmt   other statements

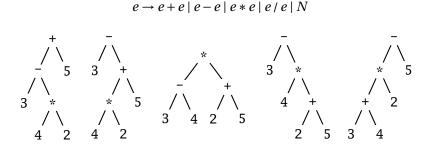
We are effectively carrying an extra bit of information during parsing: whether there is an open "then" clause. Unfortunately, duplicating rules is the only way to do this in a context-free grammar.

#### **Ambiguous Arithmetic**

Ambiguity can be a problem in expressions. Consider parsing

3 - 4 \* 2 + 5

with the grammar



#### **Operator Precedence and Associativity**

Usually resolve ambiguity in arithmetic expressions

Like you were taught in elementary school:

#### "My Dear Aunt Sally"

Mnemonic for multiplication and division before addition and subtraction.

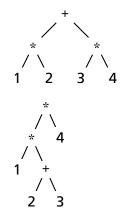
#### **Operator Precedence**

Defines how "sticky" an operator is.

1 \* 2 + 3 \* 4

\* at higher precedence than +: (1 \* 2) + (3 \* 4)

+ at higher precedence than \*: 1 \* (2 + 3) \* 4

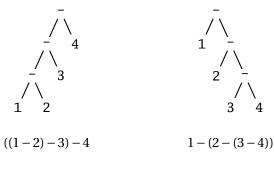


#### Associativity

Whether to evaluate left-to-right or right-to-left

Most operators are left-associative

1 - 2 - 3 - 4



left associative

right associative

## **Fixing Ambiguous Grammars**

A grammar specification:

expr : expr PLUS expr | expr MINUS expr | expr TIMES expr | expr DIVIDE expr | NUMBER

Ambiguous: no precedence or associativity.

Ocamlyacc's complaint: "16 shift/reduce conflicts."

# **Assigning Precedence Levels**

#### Split into multiple rules, one per level

expr	: expr PLUS expr   expr MINUS expr   term
term	: term TIMES term   term DIVIDE term   atom
atom	: NUMBER

Still ambiguous: associativity not defined

Ocamlyacc's complaint: "8 shift/reduce conflicts."

# Assigning Associativity

#### Make one side the next level of precedence

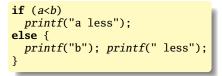
expr	: expr PLUS term   expr MINUS term   term
term	: term TIMES atom   term DIVIDE atom   atom
atom	: NUMBER

This is left-associative.

No shift/reduce conflicts.

#### Statement separators/terminators

C uses ; as a statement terminator.



Pascal uses ; as a statement separator.

```
if a < b then
  writeln('a less')
else begin
  write('a'); writeln(' less')
end</pre>
```

Pascal later made a final ; optional.

# **Ocamlyacc Specifications**

```
%{
    (* Header: verbatim OCaml; optional *)
    /* Declarations: tokens, precedence, etc. */
%%
    /* Rules: context-free rules */
%%
    (* Trailer: verbatim OCaml; optional *)
```

#### Declarations

- %token symbol ...
   Define symbol names (exported to .mli file)
- %token < type > symbol ...
   Define symbols with attached attribute (also exported)
- %start symbol ...
   Define start symbols (entry points)
- %type < type > symbol ...
   Define the type for a symbol (mandatory for start)
- %left symbol ...
- %right symbol ...
- %nonassoc symbol ...

Define predecence and associtivity for the given symbols, listed in order from lowest to highest precedence

#### **Rules**

nonterminal : symbol	symbol {	semantic-action	}
   symbol	symbol {	semantic-action	}

- nonterminal is the name of a rule, e.g., "program," "expr"
- symbol is either a terminal (token) or another rule
- semantic-action is OCaml code evaluated when the rule is matched
- In a semantic-action, \$1, \$2, ... returns the value of the first, second, ... symbol matched
- A rule may include "%prec symbol" to override its default precedence

#### An Example .mly File

**%token** <int> TNT **%token** PLUS MINUS TIMES DIV LPAREN RPAREN EOL %left PLUS MINUS /\* lowest precedence \*/ **%left** TIMES DIV %nonassoc UMINUS /\* highest precedence \*/ %start main /\* the entry point \*/ **%type** <int> main %% main: { \$1 } expr EOL expr: TNT { \$1 } LPAREN expr RPAREN { \$2 } expr PLUS expr { \$1 + \$3 } expr MINUS expr { \$1 - \$3 } expr TIMES expr { \$1 \* \$3 } expr DIV expr { \$1 / \$3 } MINUS expr %prec UMINUS { - \$2 }

# **Parsing Algorithms**

#### Parsing Context-Free Grammars

There are  $O(n^3)$  algorithms for parsing arbitrary CFGs, but most compilers demand O(n) algorithms.

Fortunately, the LL and LR subclasses of CFGs have O(n) parsing algorithms. People use these in practice.

е

 $1: e \to t + e$   $2: e \to t$   $3: t \to \mathbf{Id} * t$  $4: t \to \mathbf{Id}$ 

At each step, expand the *rightmost* nonterminal.

nonterminal

"handle": The right side of a production



 $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow \mathbf{Id} * t$  $4: t \rightarrow \mathbf{Id}$ 

At each step, expand the *rightmost* nonterminal.

nonterminal

"handle": The right side of a production

 $1: e \to t + e$   $2: e \to t$   $3: t \to \mathbf{Id} * t$  $4: t \to \mathbf{Id}$ 



#### At each step, expand the *rightmost* nonterminal.

nonterminal

"handle": The right side of a production

 $1: e \to t + e$   $2: e \to t$   $3: t \to \mathbf{Id} * t$  $4: t \to \mathbf{Id}$ 

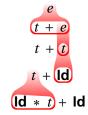


At each step, expand the rightmost nonterminal.

nonterminal

"handle": The right side of a production

 $1: e \to t + e$   $2: e \to t$   $3: t \to \mathbf{Id} * t$  $4: t \to \mathbf{Id}$ 

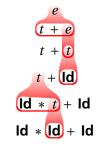


At each step, expand the *rightmost* nonterminal.

nonterminal

"handle": The right side of a production



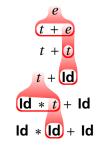


At each step, expand the rightmost nonterminal.

nonterminal

"handle": The right side of a production





At each step, expand the rightmost nonterminal.

nonterminal

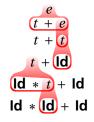
"handle": The right side of a production

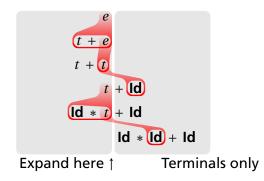
Dragon-book style: underline handles

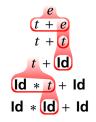
 $e \rightarrow \underline{t + e} \rightarrow t + \underline{t} \rightarrow t + \underline{\mathsf{Id}} \rightarrow \underline{\mathsf{Id}} * \underline{t} + \mathsf{Id} \rightarrow \mathsf{Id} * \underline{\mathsf{Id}} + \mathsf{Id}$ 

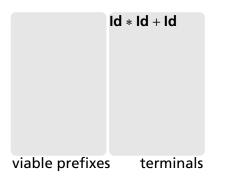
#### **Rightmost Derivation: What to Expand**

 $1: e \to t + e$   $2: e \to t$   $3: t \to \mathbf{Id} * t$  $4: t \to \mathbf{Id}$ 

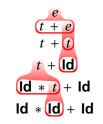


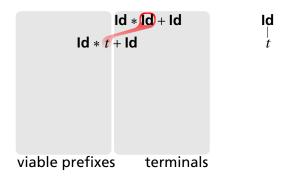


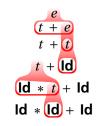


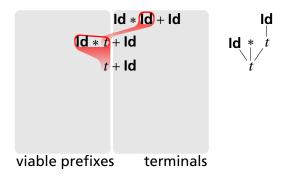


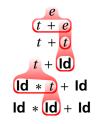
 $1: e \to t + e$   $2: e \to t$   $3: t \to \mathbf{Id} * t$  $4: t \to \mathbf{Id}$ 

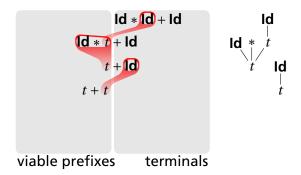


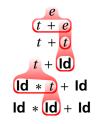


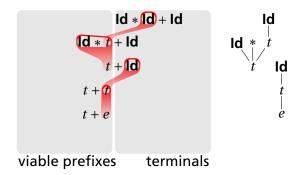




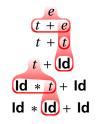


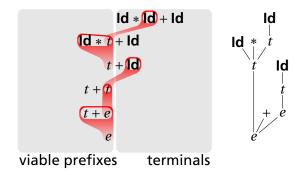


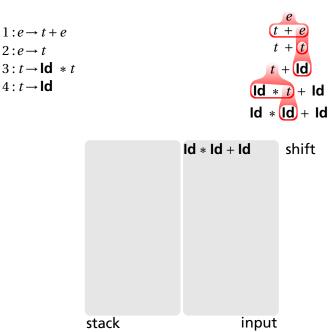


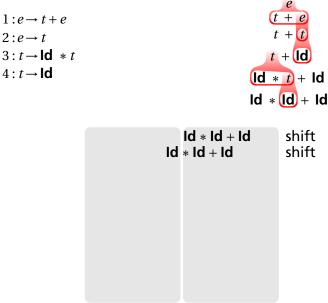


### **Reverse Rightmost Derivation**









stack

input

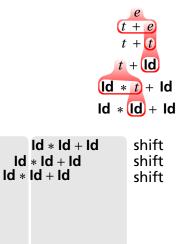
t + e

t + t

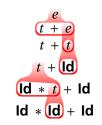
 $t + \mathbf{Id}$ 

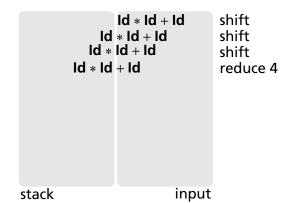
shift

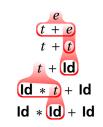
shift

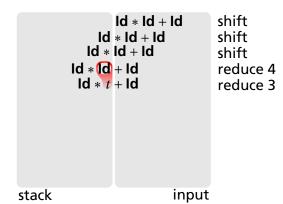


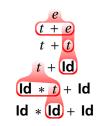


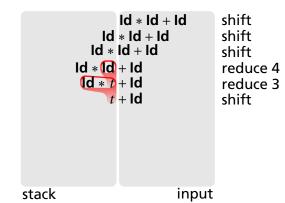


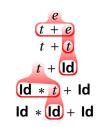


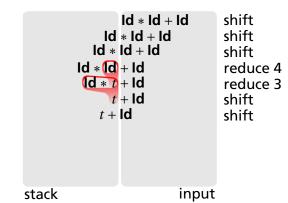


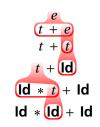


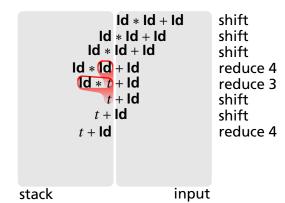


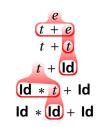


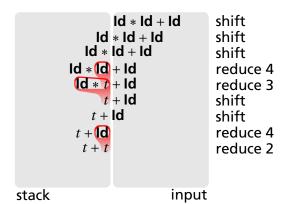


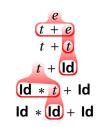


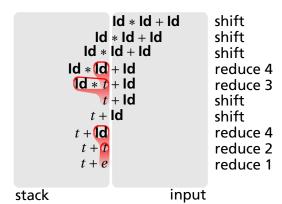


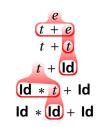


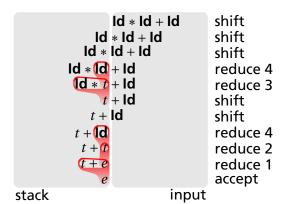












## Handle Hunting

Right Sentential Form: any step in a rightmost derivation

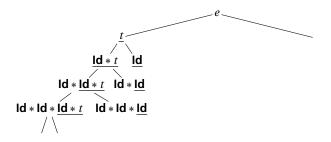
**Handle:** in a sentential form, a RHS of a rule that, when rewritten, yields the previous step in a rightmost derivation.

The big question in shift/reduce parsing:

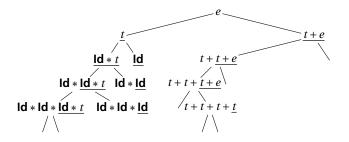
When is there a handle on the top of the stack?

Enumerate all the right-sentential forms and pattern-match against them? Usually infinitely many; let's try anyway.

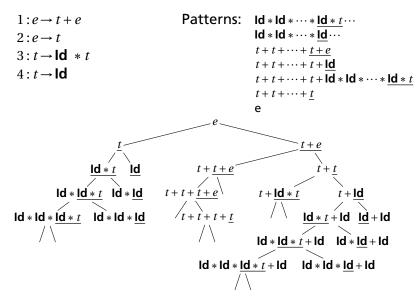
#### Some Right-Sentential Forms and Their Handles



## Some Right-Sentential Forms and Their Handles



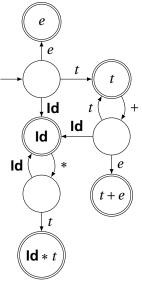
#### Some Right-Sentential Forms and Their Handles



#### The Handle-Identifying Automaton

Magical result, due to Knuth: An automaton suffices to locate a handle in a right-sentential form.

$$\mathbf{Id} * \mathbf{Id} * \cdots * \mathbf{Id} * \underline{t} \cdots$$
$$\mathbf{Id} * \mathbf{Id} * \cdots * \mathbf{Id} \cdots$$
$$t + t + \cdots + \underline{t + e}$$
$$t + t + \cdots + t + \mathbf{Id}$$
$$t + t + \cdots + t + \mathbf{Id} * \mathbf{Id} * \cdots * \mathbf{Id} * \underline{t}$$
$$t + t + \cdots + \underline{t}$$
e



## Building the Initial State of the LR(0) Automaton

 $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow \mathbf{Id} * t$  $4: t \rightarrow \mathbf{Id}$ 

$$e' \rightarrow \& e$$

Key idea: automata identify viable prefixes of right sentential forms. Each state is an equivalence class of possible places in productions.

At the beginning, any viable prefix must be at the beginning of a string expanded from e. We write this condition " $e' \rightarrow \&e''$ "

## Building the Initial State of the LR(0) Automaton

 $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow \mathbf{Id} * t$  $4: t \rightarrow \mathbf{Id}$ 

$$e' \to \mathfrak{C}e$$
$$e \to \mathfrak{C}t + e$$
$$e \to \mathfrak{C}t$$

Key idea: automata identify viable prefixes of right sentential forms. Each state is an equivalence class of possible places in productions.

At the beginning, any viable prefix must be at the beginning of a string expanded from e. We write this condition " $e' \rightarrow \&e''$ "

There are two choices for what an *e* may expand to: t + e and *t*. So when  $e' \rightarrow \&e, e \rightarrow \&t + e$  and  $e \rightarrow \&t$  are also true, i.e., it must start with a string expanded from *t*.

## Building the Initial State of the LR(0) Automaton

 $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow \mathbf{Id} * t$  $4: t \rightarrow \mathbf{Id}$ 

$$e' \rightarrow \& e$$

$$e \rightarrow \& t + e$$

$$e \rightarrow \& t$$

$$t \rightarrow \& \mathbf{Id} * t$$

$$t \rightarrow \& \mathbf{Id}$$

Key idea: automata identify viable prefixes of right sentential forms. Each state is an equivalence class of possible places in productions.

At the beginning, any viable prefix must be at the beginning of a string expanded from e. We write this condition " $e' \rightarrow \&e''$ "

There are two choices for what an *e* may expand to: t + e and *t*. So when  $e' \rightarrow \&e, e \rightarrow \&t + e$  and  $e \rightarrow \&t$  are also true, i.e., it must start with a string expanded from *t*.

Also, t must be  $\mathbf{Id} * t$  or  $\mathbf{Id}$ , so  $t \to \mathbf{CId} * t$  and  $t \to \mathbf{CId}$ . This is a *closure*, like  $\epsilon$ -closure in subset construction.

The first state suggests a viable prefix can start as any string derived from *e*, any string derived from *t*, or **Id**.

$$e' \rightarrow \mathfrak{C}e$$

$$e \rightarrow \mathfrak{C}t + e$$
S0:  $e \rightarrow \mathfrak{C}t$ 

$$t \rightarrow \mathfrak{C}\mathfrak{I}\mathfrak{d} * t$$

$$t \rightarrow \mathfrak{C}\mathfrak{I}\mathfrak{d}$$

"Just passed a

prefix ending in

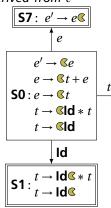
a string derived

 $\star t\mathbf{C} + e$ 

from t"

S2 :

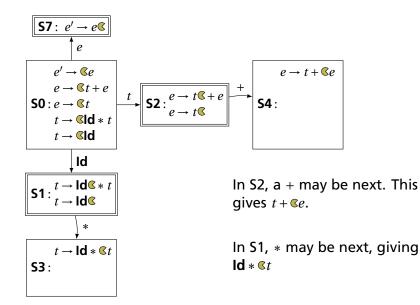
"Just passed a string derived from e"

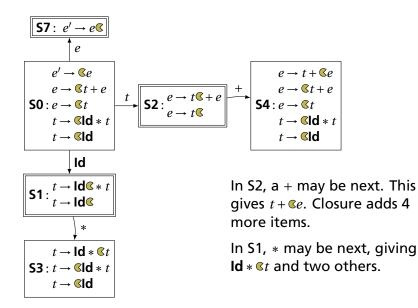


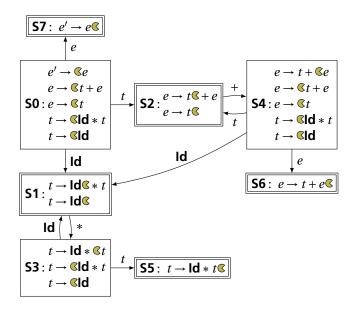
"Just passed a prefix that ended in an **Id**"

The first state suggests a viable prefix can start as any string derived from *e*, any string derived from *t*, or **Id**.

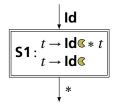
The items for these three states come from advancing the « across each thing, then performing the closure operation (vacuous here).







## What to do in each state?



$$1: e \rightarrow t + e$$
  

$$2: e \rightarrow t$$
  

$$3: t \rightarrow Id * t$$
  

$$4: t \rightarrow Id$$

$$\mathbf{Id} * \mathbf{Id} * \cdots * \mathbf{Id} * t \cdots$$
$$\mathbf{Id} * \mathbf{Id} * \cdots * \mathbf{Id} \cdots$$
$$t + t + \cdots + t + e$$
$$t + t + \cdots + t + \mathbf{Id}$$
$$t + t + \cdots + t + \mathbf{Id} * \mathbf{Id} * \cdots * \mathbf{Id} * t$$
$$t + t + \cdots + t$$
$$\mathbf{e}$$
$$\mathbf{e}$$

Stack	Input	Action
Id * Id * ··· * Id	* • • •	Shift
ld * ld * … * ld	$+\cdots$	Reduce 4
ld * ld * … * ld		Reduce 4
Id * Id * ··· * Id	ld⋯	Syntax Error

## The first function

If you can derive a string that starts with terminal t from some sequence of terminals and nonterminals  $\alpha$ , then  $t \in \text{first}(\alpha)$ .

- 1. Trivially,  $first(X) = \{X\}$  if X is a terminal.
- **2**. If  $X \rightarrow \epsilon$ , then add  $\epsilon$  to first(*X*).
- 3. For each prod.  $X \rightarrow Y \cdots$ , add first $(Y) \{c\}$  to first(X). If X can produce something, X can start with whatever that starts with
- 4. For each prod.  $X \rightarrow Y_1 \cdots Y_k Z \cdots$  where  $\epsilon \in \text{first}(Y_i)$  for i = 1, ..., k, add  $\text{first}(Z) \{\epsilon\}$  to first(X). Skip all potential  $\epsilon$ 's at the beginning of whatever X produces

$1: e \rightarrow t + e$	$first(Id) = \{Id\}$
$2: e \rightarrow t$	first(t) = { <b>Id</b> } because $t \rightarrow $ <b>Id</b> * t and $t \rightarrow $ <b>Id</b>
$3: t \to Id * t  4: t \to Id$	first(e) = { <b>Id</b> } because $e \rightarrow t + e$ , $e \rightarrow t$ , and first(t) = ( <b>Id</b> )
	$first(t) = \{ Id \}.$

If t is a terminal, A is a nonterminal, and  $\cdots At \cdots$  can be derived, then  $t \in follow(A)$ .

- 1. Add \$ ("end-of-input") to follow(S) (start symbol). End-of-input comes after the start symbol
- 2. For each prod.  $\rightarrow \cdots A\alpha$ , add first( $\alpha$ ) { $\epsilon$ } to follow(A). A is followed by the first thing after it
- 3. For each prod.  $A \rightarrow \cdots B$  or  $A \rightarrow \cdots B\alpha$  where  $\epsilon \in \text{first}(\alpha)$ , then add everything in follow(A) to follow(B). If B appears at the end of a production, it can be followed by whatever follows that production
- $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow Id * t$   $4: t \rightarrow Id$ first(t) = {Id} first(e) = {Id}

```
follow(e) = {$}
follow(t) = {
```

1. Because e is the start symbol

If t is a terminal, A is a nonterminal, and  $\cdots At \cdots$  can be derived, then  $t \in follow(A)$ .

- 1. Add \$ ("end-of-input") to follow(S) (start symbol). End-of-input comes after the start symbol
- 2. For each prod.  $\rightarrow \cdots A\alpha$ , add first( $\alpha$ ) { $\epsilon$ } to follow(A). A is followed by the first thing after it
- 3. For each prod.  $A \rightarrow \cdots B$  or  $A \rightarrow \cdots B\alpha$  where  $\epsilon \in \text{first}(\alpha)$ , then add everything in follow(A) to follow(B). If B appears at the end of a production, it can be followed by whatever follows that production
- $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow Id * t$   $4: t \rightarrow Id$ first(t) = {Id} first(e) = {Id}

follow(
$$e$$
) = {\$}  
follow( $t$ ) = {+ }

2. Because  $e \rightarrow \underline{t} + e$  and first(+) = {+}

If t is a terminal, A is a nonterminal, and  $\cdots At \cdots$  can be derived, then  $t \in follow(A)$ .

- 1. Add \$ ("end-of-input") to follow(S) (start symbol). End-of-input comes after the start symbol
- 2. For each prod.  $\rightarrow \cdots A\alpha$ , add first( $\alpha$ ) { $\epsilon$ } to follow(A). A is followed by the first thing after it
- 3. For each prod.  $A \rightarrow \cdots B$  or  $A \rightarrow \cdots B\alpha$  where  $\epsilon \in \text{first}(\alpha)$ , then add everything in follow(A) to follow(B). If B appears at the end of a production, it can be followed by whatever follows that production
- $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow Id * t$   $4: t \rightarrow Id$ first(t) = {Id} first(e) = {Id}

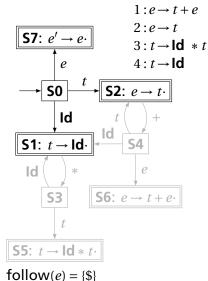
- follow(e) = {\$}
- $follow(t) = \{+, \$\}$
- 3. Because  $e \rightarrow \underline{t}$  and  $\$ \in follow(e)$

If t is a terminal, A is a nonterminal, and  $\cdots At \cdots$  can be derived, then  $t \in follow(A)$ .

- 1. Add \$ ("end-of-input") to follow(S) (start symbol). End-of-input comes after the start symbol
- 2. For each prod.  $\rightarrow \cdots A\alpha$ , add first( $\alpha$ ) { $\epsilon$ } to follow(A). A is followed by the first thing after it
- 3. For each prod.  $A \rightarrow \cdots B$  or  $A \rightarrow \cdots B\alpha$  where  $\epsilon \in \text{first}(\alpha)$ , then add everything in follow(A) to follow(B). If B appears at the end of a production, it can be followed by whatever follows that production
- $1: e \rightarrow t + e$   $2: e \rightarrow t$   $3: t \rightarrow Id * t$   $4: t \rightarrow Id$ first(t) = {Id} first(e) = {Id}

```
follow(e) = {$}
follow(t) = {+, $}
```

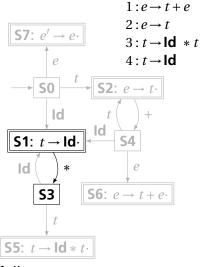
Fixed-point reached: applying any rule does not change any set



follow(t) = {+, \$}

State	Action				Goto	
	Id	+	*	\$	e	t
0	s1				7	2

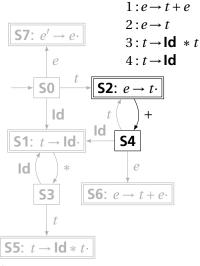
From S0, shift an **Id** and go to S1; or cross a *t* and go to S2; or cross an *e* and go to S7.



State		Action				
	ld	+	*	\$	e	t
0	s1				7	2
1		r4	s3	r4		

From S1, shift a \* and go to S3; or, if the next input  $\in$  follow(*t*), reduce by rule 4.

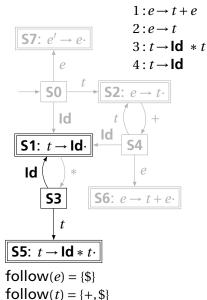
follow(e) = {\$} follow(t) = {+, \$}



follow(e) = {\$} follow(t) = {+, \$}

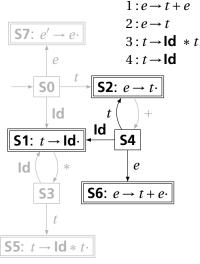
State		Action				Goto	
	Id	+	*	\$	e	t	
0	s1				7	2	
1		r4	s3	r4			
2		r4 s4		r4 r2			

From S2, shift a + and go to S4; or, if the next input  $\in$  follow(*e*), reduce by rule 2.



State	Action				Goto	
	Id	+	*	\$	e	t
0	s1				7	2
1		r4	s3	r4		
2		r4 s4		r2		
3	s1					5

From S3, shift an **Id** and go to S1; or cross a t and go to S5.

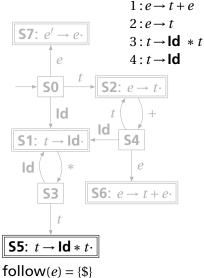


State		Goto				
	Id	+	*	\$	e	t
0	s1				7	2
1		r4	s3	r4		
2		r4 s4		r2		
3	s1					5
4	s1				6	2

From S4, shift an **Id** and go to S1; or cross an e or a t.

follow(e) = {\$} follow(t) = {+, \$}

## Converting the LR(0) Automaton to an SLR Table

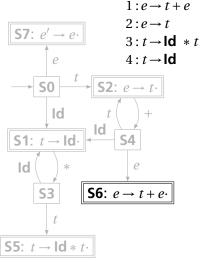


State		Goto				
	ld	+	*	\$	e	t
0	s1				7	2
1		r4	s3	r4		
2		s4		r2		
3	s1					5
4	s1				6	2
5		r3		r3		

From S5, reduce using rule 3 if the next symbol  $\in$  follow(t).

follow(t) = {+, \$}

## Converting the LR(0) Automaton to an SLR Table

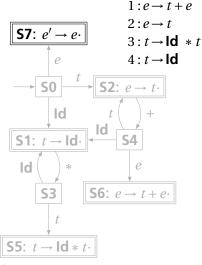


State		ction (		Go	Goto	
	ld	+	*	\$	e	t
0	s1				7	2
1		r4	s3	r4		
2		s4		r2		
3	s1					5
4	s1				6	2
5		r3		r3		
6				r1		

From S6, reduce using rule 1 if the next symbol  $\in$  follow(*e*).

follow(e) = {\$} follow(t) = {+, \$}

## Converting the LR(0) Automaton to an SLR Table



follow(e) = {\$} follow(t) = {+, \$}

State		Act		Go	oto	
	Id	+	*	\$	e	t
0	s1				7	2
1		r4	s3	r4		
2		r4 s4		r2		
3	s1					5
4	s1				6	2
5		r3		r3		
6				r1		
7				$\checkmark$		

If, in S7, we just crossed an *e*, accept if we are at the end of the input.

_	Stack	Input	Action
$1: e \to t + e$ $2: e \to t$	0	ld * ld + ld \$	Shift, goto 1
$\begin{array}{l} 3:t \to Id & *t \\ 4:t \to Id \end{array}$		the state on d the next in	

State		Act		Go	oto	
	Id	+	*	\$	е	t
0	s1				7	2
1		r4	s3	r4		
2		r4 s4		r2		
2 3	s1					5
4	s1				6	2
5 6		r3		r3		
6				r1		
7				$\checkmark$		

Find the action (shift, reduce, or error) in the table.

In this case, shift the token onto the stack and mark it with state 1.

	Stack	Input	Action
$1: e \to t + e$ $2: e \to t$ $3: t \to \mathbf{Id} * t$ $4: t \to \mathbf{Id}$	0 0 <b>Id</b> 1	ld * ld + ld \$ * ld + ld \$	Shift, goto 1 Shift, goto 3

State		Act	Go	oto		
	Id	+	*	\$	е	t
0	s1				7	2
1		r4	s3	r4		
2 3		r4 s4		r2		
	s1					5
4	s1				6	2
5		r3		r3		
6				r1		
7				$\checkmark$		

Here, the state is 1, the next symbol is \*, so shift and mark it with state 3.

$1: e \rightarrow t + e$	
$2: e \rightarrow t$	
$3: t \rightarrow \mathbf{Id} * t$	
$4: t \rightarrow \mathbf{Id}$	

State			Go	oto		
	Id	+	*	\$	e	t
0	s1				7	2
1		r4	s3	r4		
2		s4		r2		
3	s1					5
4	s1				6	2
5		r3		r3		
6				r1		
7				$\checkmark$		

Stack	Input	Action
0	ld * ld + ld \$	Shift, goto 1
0 <mark>1</mark>	* Id + Id \$	Shift, goto 3
0 1 3	ld + ld \$	Shift, goto 1
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	+ Id \$	Reduce 4

Here, the state is 1, the next symbol is +. The table says reduce using rule 4.

 $\checkmark$ 

7

					-		Stack	Input	Action
$1: e \rightarrow 2: e \rightarrow 3: t \rightarrow 4: t \rightarrow$	t Id *	t					0 0 1d 0 1 3	ld * ld + ld \$ * ld + ld \$ ld + ld \$	Shift, goto 1 Shift, goto 3 Shift, goto 1
State		Act	tion		Go	oto	0 1 3 1	+ <b>Id</b> \$	Reduce 4
	ld	+	*	\$	е	t	0 1 3	+ Id \$	
0	s1				7	2			
1		r4	s3	r4			Remove	the RHS of th	ne rule (here,
2		s4		r2			just <b>Id</b> ),	observe the s	tate on the
3	s1					5	top of t	he stack, and	consult the
4	s1				6	2	"goto"	portion of the	e table.
5		r3		r3					
6				r1					

	State	Action	Goto
_	$4: t \rightarrow \mathbf{Id}$		
	$3: t \rightarrow \mathbf{Id} * t$	ţ	
	$2: e \rightarrow t$		
	$1: e \rightarrow t + e$		

State		Act	tion	Goto		
	Id	+	*	\$	е	t
0	s1				7	2
1		r4 s4	s3	r4		
2		s4		r2		
3	s1					5 2
4	s1				6	2
2 3 4 5 6		r3		r3		
6				r1		
7				$\checkmark$		

Stack	Input	Action		
0	ld * ld + ld \$	Shift, goto 1		
0 1d	* Id + Id \$	Shift, goto 3		
0 1 3	ld + ld \$	Shift, goto 1		
0 1 3 1	+ Id \$	Reduce 4		
0 1 3 5	+ Id \$	Reduce 3		

Here, we push a *t* with state 5. This effectively "backs up" the LR(0) automaton and runs it over the newly added nonterminal.

In state 5 with an upcoming +, the action is "reduce 3."

out Action
+ <b>Id</b> \$ Shift, goto
ld\$ Shift, goto
I\$ Shift, goto
Reduce 4
Reduce 3
Shift, goto
, je se
o off the RHS for
sing state 0, so
state 2.

							S	tao	ck		Input	Action
$1: e \to t + e$ $2: e \to t$ $3: t \to \mathbf{Id} * t$ $4: t \to \mathbf{Id}$						0 Id	0 Id 1	* 3	Id * Id + Id \$ * Id + Id \$ Id + Id \$	Shift, goto 1 Shift, goto 3 Shift, goto 1		
State		Action		Go	oto	0 1 3		_	+ Id \$	Reduce 4		
	Id	+	*	\$	е	t	0	1 <b>d</b>	*3	<i>t</i> 5	+ Id \$	Reduce 3
0	s1	_	_	_	7	2			0	<i>t</i> 2	+ Id \$	Shift, goto 4
1 2		r4 s4	s3	r4 r2				0	<i>t</i> 2	<mark>+</mark> 4	ld\$	Shift, goto 1
2 3 4	s1 s1				6	5 2	0	<i>t</i> <b>2</b>	+ 4	ld 1	\$	Reduce 4
5		r3		r3		-	0		+ 4		\$	Reduce 2
6 7				r1 √			0	<i>t</i> 2	+4	е 6	\$	Reduce 1
								_	0	P	\$	Accept

# L, R, and all that

LR parser: "Bottom-up parser":

L = Left-to-right scan, R = (reverse) Rightmost derivation

RR parser: R = Right-to-left scan (from end) I called them "Australian style"; nobody uses these

LL parser: "Top-down parser": L = Left-to-right scan: L = (reverse) Leftmost derivation

LR(1): LR parser that considers next token (lookahead of 1)

LR(0): Only considers stack to decide shift/reduce

SLR(1): Simple LR: lookahead from first/follow rules Derived from LR(0) automaton

LALR(1): Lookahead LR(1): fancier lookahead analysis Uses same LR(0) automaton as SLR(1)

Ocamlyacc builds LALR(1) tables.

# The Punchline

This is a tricky, but mechanical procedure. The Ocamlyacc parser generator uses a modified version of this technique to generate fast bottom-up parsers.

You need to understand it to comprehend error messages:

Shift/reduce conflicts are caused by a state like

```
t \rightarrow \cdot \mathbf{Else} \, s
```

 $t \rightarrow \cdot$ 

If the next token is **Else**, do you reduce it since **Else** may follow a *t*, or shift it? Reduce/reduce conflicts are caused by a state like

$$t \rightarrow \mathbf{Id} * t \cdot$$

 $e \to t + e \, \cdot$ 

Do you reduce by " $t \rightarrow \mathbf{Id} * t$ " or by " $e \rightarrow t + e$ "?