

Earley's Algorithm (1970)

Nice combo of our parsing ideas so far:

- no restrictions on the form of the grammar:
 - $A \rightarrow B C \text{ spoon } D x$
- incremental parsing (left to right, like humans)
- left context constrains parsing of subsequent words
 - so waste less time building impossible things
 - makes it faster than $O(n^3)$ for many grammars

About Jay



Jay Earley, Ph.D., is a transformational psychologist, group leader, psychotherapist, coach, author, teacher, and theorist.

Jay is trained in Internal Family Systems Therapy and assists with professional trainings in IFS. He leads IFS Classes for the general public which teach IFS as a practice for self-help and peer counseling. He is active in the IFS community and has presented a number of workshops at IFS annual conferences. He also teaches classes on Communication from the Heart, based on IFS, interactive groups, and the Pattern System.

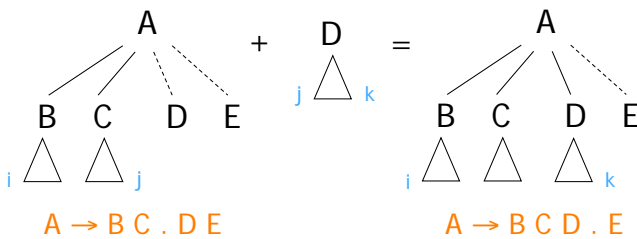
He is nationally known for his innovation in the group psychotherapy field. His book, *Interactive Group Therapy: Integrating Interpersonal, Action-Oriented, and Psychodynamic Approaches*, Brunner/Mazel, describes his group therapy method in which people learn interpersonal relationship skills by working directly on their relationships with each other. During his ten years on the east coast, Jay was Director of the Group Therapy Center of Long Island, where he trained group therapists in this method. He has written a number of articles on interactive groups and made numerous presentations at regional and national psychotherapy conferences. He continues to lead interactive therapy groups in the Bay Area.

Jay offers Life Purpose Coaching and Change Agent Coaching, on finding your life purpose and making a difference in the world. He has been writing about and leading workshops on Life Purpose since 1984. He has collected his writings on life purpose into an ebook *Finding Your Life Purpose*.

Jay also has a Ph.D. in computer science from Carnegie-Mellon University and was formerly on the U.C. Berkeley faculty, where he published 12 computer science papers, one of which was voted one of the best 25 papers of the quarter century by the Communications of the A.C.M.

Overview of Earley's Algorithm

- Finds constituents and partial constituents in input
 - $A \rightarrow B C . D E$ is partial: only the first half of the A



Overview of Earley's Algorithm

- Proceeds incrementally, left-to-right
 - Before it reads word 5, it has already built all hypotheses that are consistent with first 4 words
 - Reads word 5 & attaches it to immediately preceding hypotheses. Might yield new constituents that are then attached to hypotheses immediately preceding *them* ...
 - E.g., attaching D to $A \rightarrow B C . D E$ gives $A \rightarrow B C D . E$
 - Attaching E to that gives $A \rightarrow B C D E$.
 - Now we have a complete A that we can attach to hypotheses immediately preceding the A , etc.

Our Usual Example Grammar

$ROOT \rightarrow S$
 $S \rightarrow NP VP$ $NP \rightarrow Papa$
 $NP \rightarrow Det N$ $N \rightarrow caviar$
 $NP \rightarrow NP PP$ $N \rightarrow spoon$
 $VP \rightarrow VP PP$ $V \rightarrow ate$
 $VP \rightarrow V NP$ $P \rightarrow with$
 $PP \rightarrow P NP$ $Det \rightarrow the$
 $Det \rightarrow a$

0 Papa 1 ate 2 the 3 caviar 4 with 5 a 6 spoon 7

First Try: Recursive Descent

$ROOT \rightarrow S$ $VP \rightarrow VP PP$ $NP \rightarrow Papa$ $V \rightarrow ate$
 $S \rightarrow NP VP$ $VP \rightarrow V NP$ $N \rightarrow caviar$ $P \rightarrow with$
 $NP \rightarrow Det N$ $PP \rightarrow P NP$ $N \rightarrow spoon$ $Det \rightarrow the$
 $NP \rightarrow NP PP$ $Det \rightarrow a$

- 0 $ROOT \rightarrow . S$ 0
 - 0 $S \rightarrow . NP VP$ 0
 - 0 $NP \rightarrow . Papa$ 0
 - 0 $NP \rightarrow Papa .$ 1
 - 0 $S \rightarrow NP . VP$ 1
 - 1 $VP \rightarrow . VP PP$ 1
 - 1 $VP \rightarrow . VP PP$ 1
 - 1 $VP \rightarrow . VP PP$ 1
 - 1 $VP \rightarrow . VP PP$ 1
 - oops, stack overflowed
- OK, let's pretend that didn't happen.
- Let's suppose we didn't see $VP \rightarrow VP PP$, and used $VP \rightarrow V NP$ instead.

0 Papa 1 ate 2 the 3 caviar 4 with 5 a 6 spoon 7

First Try: Recursive Descent

ROOT → S	VP → V NP	NP → Papa	V → ate
S → NP VP	VP → VP PP	N → caviar	P → with
NP → Det N	PP → P NP	N → spoon	Det → the
NP → NP PP			Det → a

- 0 ROOT → . S 0
 - 0 S → . NP VP 0
 - 0 NP → . Papa 0
 - 0 NP → Papa . 1
 - 0 S → NP . VP 1
 - 1 VP → . V NP 1
 - 1 V → . ate 1
 - 1 V → ate . 2
 - 1 VP → V . NP 2
 - 2 NP → 2
 - 2 NP → 7
 - 1 VP → V NP . 7
 - 1 VP → V NP . 7
- 0 S → NP VP . 7
 - 0 S → NP VP . 7

600.465 - Intro to NLP - J. Eisner

7

0 Papa 1 ate 2 the 3 caviar 4 with 5 a 6 spoon 7

First Try: Recursive Descent

ROOT → S	VP → V NP	NP → Papa	V → ate
S → NP VP	VP → VP PP	N → caviar	P → with
NP → Det N	PP → P NP	N → spoon	Det → the
NP → NP PP			Det → a

- 0 ROOT → . S 0
 - 0 S → . NP VP 0
 - 0 NP → . Papa 0
 - 0 NP → Papa . 1
 - 0 S → NP . VP 1
 - 1 VP → . V NP 1
 - 1 V → . ate 1
 - 1 V → ate . 2
 - 1 VP → V . NP 2
 - 2 NP → 2
 - 2 NP → 7
 - 1 VP → V NP . 7
 - 1 VP → V NP . 7
- 0 S → NP VP . 7
 - 0 S → NP VP . 7

600.465 - Intro to NLP - J. Eisner

8

0 Papa 1 ate 2 the 3 caviar 4 with 5 a 6 spoon 7

First Try: Recursive Descent

ROOT → S	VP → V NP	NP → Papa	V → ate
S → NP VP	VP → VP PP	N → caviar	P → with
NP → Det N	PP → P NP	N → spoon	Det → the
NP → NP PP			Det → a

- 0 ROOT → . S 0
 - 0 S → . NP VP 0
 - 0 NP → . Papa 0
 - 0 NP → Papa . 1
 - 0 S → NP . VP 1
 - 1 VP → . VP PP 1
 - 1 VP → . V NP 1
 - 1 V → . ate 1
 - 1 V → ate . 2
 - 1 VP → V . NP 2
 - 2 NP → 2
 - 2 NP → 4

600.465 - Intro to NLP - J. Eisner

9

0 Papa 1 ate 2 the 3 caviar 4 with 5 a 6 spoon 7

First Try: Recursive Descent

ROOT → S	VP → V NP	NP → Papa	V → ate
S → NP VP	VP → VP PP	N → caviar	P → with
NP → Det N	PP → P NP	N → spoon	Det → the
NP → NP PP			Det → a

- 0 ROOT → . S 0
 - 0 S → . NP VP 0
 - 0 NP → . Papa 0
 - 0 NP → Papa . 1
 - 0 S → NP . VP 1
 - 1 VP → . VP PP 1
 - 1 VP → . VP PP 1
 - 1 VP → . VP PP 1
 - 1 VP → . VP PP 1

600.465 - Intro to NLP - J. Eisner

10

Use a Parse Table

- Earley's algorithm resembles recursive descent, but solves the left-recursion problem. No recursive function calls.
- Use a parse table as we did in CKY, so we can look up anything we've discovered so far. "Dynamic programming."
- Entries in column 5 look like (3, S → NP . VP)
 - (but we'll omit the → etc. to save space)
 - Built while processing word 5
 - Means that the input substring from 3 to 5 matches the initial NP portion of a S → NP VP rule
 - Dot shows how much we've matched as of column 5
 - Perfectly fine to have entries like (3, S → is it . true that S)

600.465 - Intro to NLP - J. Eisner

11

Use a Parse Table

- Entries in column 5 look like (3, S → NP . VP)
- What does it mean if we have this entry?
 - Unknown right context: Doesn't mean we'll necessarily be able to find a VP starting at column 5 to complete the S.
 - Known left context: Does mean that some dotted rule back in column 3 is looking for an S that starts at 3.
 - So if we actually do find a VP starting at column 5, allowing us to complete the S, then we'll be able to attach the S to something.
 - And when that something is complete, it too will have a customer to its left ... just as in recursive descent!
 - In short, a top-down (i.e., goal-directed) parser: it chooses to start building a constituent not because of the input but because that's what the left context needs. In the spoon, won't build spoon as a verb because there's no way to use a verb there.
 - So any hypothesis in column 5 could get used in the correct parse, if words 1-5 are continued in just the right way by words 6-n.

600.465 - Intro to NLP - J. Eisner

12

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP		
0 NP . Det N		
0 NP . NP PP		
0 NP . Papa		
0 Det . the		
0 Det . a		

scan: the desired word is in the input!

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP		
0 NP . Det N		
0 NP . NP PP		
0 NP . Papa		
0 Det . the		
0 Det . a		

scan: failure

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP		
0 NP . Det N		
0 NP . NP PP		
0 NP . Papa		
0 Det . the		
0 Det . a		

scan: failure

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP		
0 NP . Papa		
0 Det . the		
0 Det . a		

attach the newly created NP (which starts at 0) to its customers (incomplete constituents that end at 0 and have NP after the dot)

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the		
0 Det . a		

predict

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a		

predict

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a	1 V . ate	

predict

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a	1 V . ate	

predict

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a	1 V . ate	
	1 P . with	

predict

0	Papa	1	ate	2
0 ROOT . S	0 NP Papa .		1 V ate .	
0 S . NP VP	0 S NP . VP			
0 NP . Det N	0 NP NP . PP			
0 NP . NP PP	1 VP . V NP			
0 NP . Papa	1 VP . VP PP			
0 Det . the	1 PP . P NP			
0 Det . a	1 V . ate			
	1 P . with			

scan: success!

0	Papa	1	ate	2
0 ROOT . S	0 NP Papa .		1 V ate .	
0 S . NP VP	0 S NP . VP			
0 NP . Det N	0 NP NP . PP			
0 NP . NP PP	1 VP . V NP			
0 NP . Papa	1 VP . VP PP			
0 Det . the	1 PP . P NP			
0 Det . a	1 V . ate			
	1 P . with			

scan: failure

0	Papa	1	ate	2
0 ROOT . S	0 NP Papa .		1 V ate .	
0 S . NP VP	0 S NP . VP		1 VP V . NP	
0 NP . Det N	0 NP NP . PP			
0 NP . NP PP	1 VP . V NP			
0 NP . Papa	1 VP . VP PP			
0 Det . the	1 PP . P NP			
0 Det . a	1 V . ate			
	1 P . with			

attach

0	Papa	1	ate	2	the	3
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .			
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N			
0 NP . Det N	0 NP NP . PP	2 NP . Det N				
0 NP . NP PP	1 VP . V NP	2 NP . NP PP				
0 NP . Papa	1 VP . VP PP	2 NP . Papa				
0 Det . the	1 PP . P NP	2 Det . the				
0 Det . a	1 V . ate	2 Det . a				
	1 P . with					

0	Papa	1	ate	2	the	3
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .			
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N			
0 NP . Det N	0 NP NP . PP	2 NP . Det N	3 N . caviar			
0 NP . NP PP	1 VP . V NP	2 NP . NP PP	3 N . spoon			
0 NP . Papa	1 VP . VP PP	2 NP . Papa				
0 Det . the	1 PP . P NP	2 Det . the				
0 Det . a	1 V . ate	2 Det . a				
	1 P . with					

0	Papa	1	ate	2	the	3	caviar	4
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .	3 N caviar .				
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N					
0 NP . Det N	0 NP NP . PP	2 NP . Det N	3 N . caviar					
0 NP . NP PP	1 VP . V NP	2 NP . NP PP	3 N . spoon					
0 NP . Papa	1 VP . VP PP	2 NP . Papa						
0 Det . the	1 PP . P NP	2 Det . the						
0 Det . a	1 V . ate	2 Det . a						
	1 P . with							

0	Papa	1	ate	2	the	3	caviar	4
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .	3 N caviar .				
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N					
0 NP . Det N	0 NP NP . PP	2 NP . Det N	3 N . caviar					
0 NP . NP PP	1 VP . V NP	2 NP . NP PP	3 N . spoon					
0 NP . Papa	1 VP . VP PP	2 NP . Papa						
0 Det . the	1 PP . P NP	2 Det . the						
0 Det . a	1 V . ate	2 Det . a						
	1 P . with							

0	Papa	1	ate	2	the	3	caviar	4
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .	3 N caviar .				
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N	2 NP Det . N				
0 NP . Det N	0 NP NP . PP	2 NP . Det N	3 N . caviar					
0 NP . NP PP	1 VP . V NP	2 NP . NP PP	3 N . spoon					
0 NP . Papa	1 VP . VP PP	2 NP . Papa						
0 Det . the	1 PP . P NP	2 Det . the						
0 Det . a	1 V . ate	2 Det . a						
	1 P . with							

attach

0	Papa	1	ate	2	the	3	caviar	4
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .	3 N caviar .				
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N	2 NP Det . N				
0 NP . Det N	0 NP NP . PP	2 NP . Det N	3 N . caviar	1 VP V NP .				
0 NP . NP PP	1 VP . V NP	2 NP . NP PP	3 N . spoon	2 NP NP . PP				
0 NP . Papa	1 VP . VP PP	2 NP . Papa						
0 Det . the	1 PP . P NP	2 Det . the						
0 Det . a	1 V . ate	2 Det . a						
	1 P . with							

attach
(again!)

0	Papa	1	ate	2	the	3	caviar	4	with a spoon	7
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .	3 N caviar	6 N spoon .				
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N	2 NP Det N .		5 NP Det N .				
0 NP . Det N	0 NP NP . PP	2 NP . Det N	3 N . caviar	1 VP V NP .		4 PP P NP .				
0 NP . NP PP	1 VP . V NP	2 NP . NP PP	3 N . spoon	2 NP NP . PP		5 NP NP . PP				
0 NP . Papa	1 VP . VP PP	2 NP . Papa		0 S NP VP .		2 NP NP PP .				
0 Det . the	1 PP . P NP	2 Det . the		1 VP VP . PP		1 VP VP PP .				
0 Det . a	1 V . ate	2 Det . a		4 PP . P NP		7 PP . P NP				
	1 P . with			0 ROOT S .		1 VP V NP .				
				4 P . with		2 NP NP . PP				
						0 S NP VP .				
						1 VP VP . PP				
						7 P . with				
						0 ROOT S .				

0	Papa	1	ate	2	the	3	caviar	4	with a spoon	7
0 ROOT . S	0 NP Papa .	1 V ate .	2 Det the .	3 N caviar	6 N spoon .				
0 S . NP VP	0 S NP . VP	1 VP V . NP	2 NP Det . N	2 NP Det N .		5 NP Det N .				
0 NP . Det N	0 NP NP . PP	2 NP . Det N	3 N . caviar	1 VP V NP .		4 PP P NP .				
0 NP . NP PP	1 VP . V NP	2 NP . NP PP	3 N . spoon	2 NP NP . PP		5 NP NP . PP				
0 NP . Papa	1 VP . VP PP	2 NP . Papa		0 S NP VP .		2 NP NP PP .				
0 Det . the	1 PP . P NP	2 Det . the		1 VP VP . PP		1 VP VP PP .				
0 Det . a	1 V . ate	2 Det . a		4 PP . P NP		7 PP . P NP				
	1 P . with			0 ROOT S .		1 VP V NP .				
				4 P . with		2 NP NP . PP				
						0 S NP VP .				
						1 VP VP . PP				
						7 P . with				
						0 ROOT S .				

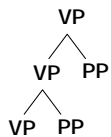
Left Recursion Kills Pure Top-Down Parsing ...

VP

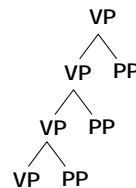
Left Recursion Kills Pure Top-Down Parsing ...



Left Recursion Kills Pure Top-Down Parsing ...



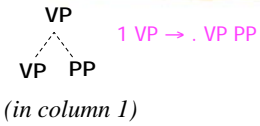
Left Recursion Kills Pure Top-Down Parsing ...



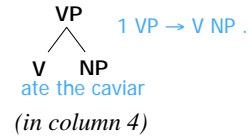
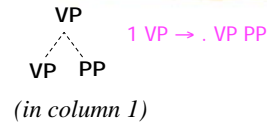
makes new hypotheses ad infinitum before we've seen the PPs at all

hypotheses try to predict in advance how many PP's will arrive in input

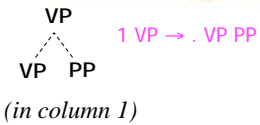
... but Earley's Alg is Okay!



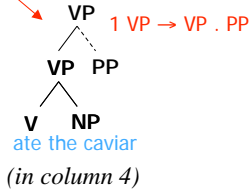
... but Earley's Alg is Okay!



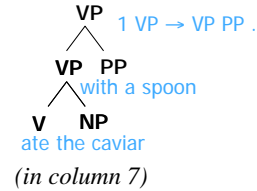
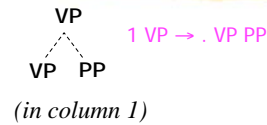
... but Earley's Alg is Okay!



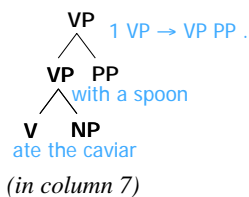
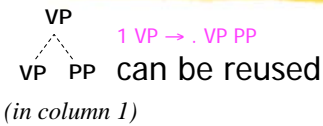
attach



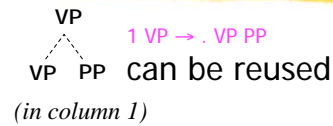
... but Earley's Alg is Okay!



... but Earley's Alg is Okay!



... but Earley's Alg is Okay!



attach

