On the Practice of B-ing Earley

Dan Zingaro
zingard@mcmaster.ca

McMaster University
August 2007
What’s With the Title?

- Earley: algorithm for testing whether a sentence can be recognized via an arbitrary context-free grammar in worst-case $O(n^3)$

- Other algorithms require special normal form (CYK, 1965), have more complicated structures (Unger, 1968), or parse only some grammars ($LL$, $LR$)

- Applications: probabilistic language processing (Stolcke, 1995), compiler-compilers (Accent)

- Other formalisms: Earley’s thesis (1968, informal), Sikkel (1998, parsing schemata), Jones (1972, origins of refinement)
Current Interest in Earley

- Previously thought Earley was too slow compared to \textit{LL} and \textit{LR}
- Aycock and Horspool (2002) showed \textit{LR} is only 1.9x faster than Earley
- Claim: time difference is not noticeable, might as well support full power of CFGs
What’s With the Title? ...

- B: formal method for specifying and refining software
- Expressive: sets, sequences, relations, functions
- Want to prove properties of algorithm but also derive it from specification
- Modularization: separate development and understanding of components
- Typecheckers, provers, model checkers available
What is aRecognizer?

- Derivable: $\Rightarrow^*$
- Recognizer returns true if $Root \Rightarrow^* sentence$

$$Nonterminals = set \text{ of } nonterminal;$$
$$Terminals = set \text{ of } terminal;$$
$$Symbols = Terminals \cup Nonterminals;$$
$$Root \in Nonterminals;$$
$$productions \in Nonterminals \leftrightarrow \text{seq}(Symbols)$$
Derivability

MACHINE recm

SEES gram, sent

DEFINITIONS

\(\text{directlyDerivable} \quad == \quad \{ x, y \mid \exists \mu, \sigma, \nu, \tau. ( x = (\mu \, [\sigma] \, \nu) \land ( y = \mu \, \tau \, \nu) \land (\sigma \rightarrow \tau \in \text{productions}) )\};\)

\(\text{derivable} \quad == \quad \text{closure}(\text{directlyDerivable})\)

OPERATIONS

\(\text{ans} \leftarrow \text{isSentence} = \)

\(\text{ans} := \text{bool} ([\text{Root}] \rightarrow \text{sentence} \in \text{derivable})\)
Earley Ingredients

- Length of sentence: $n$
- Create $n + 1$ state sets; one per sentence position + initial set
- State: $[A \rightarrow \alpha \beta, j]$

\[
\text{state} \equiv \text{Nonterminals} \times \text{seq(Symbol)} \times \text{seq(Symbol)} \times \mathbb{N};
\]
\[
s \in (0 \ldots \text{size(sentence)}) \rightarrow \mathcal{P} \text{(state)}
\]
Earley Execution

Grammar Rules: $S \rightarrow A, A \rightarrow a, A \rightarrow b$
Sentence: $a$
$s(0)$:
Earley Execution

Grammar Rules: $S \rightarrow A$, $A \rightarrow a$, $A \rightarrow b$

Sentence: $a$

$s(0)$:

- 1. $[S \rightarrow \bullet A, 0]$ (basis)
Earley Execution

Grammar Rules: $S \rightarrow A$, $A \rightarrow a$, $A \rightarrow b$
Sentence: $a$

$s(0)$:

- 1. $[S \rightarrow \bullet A, 0]$ (basis)
- 2. $[A \rightarrow \bullet a, 0]$ (predict from 1)
- 3. $[A \rightarrow \bullet b, 0]$ (predict from 1)

$s(1)$:
Earley Execution

Grammar Rules: \( S \rightarrow A, A \rightarrow a, A \rightarrow b \)
Sentence: \( a \)

\( s(0) : \)

- 1. \([S \rightarrow \bullet A, 0]\) (basis)
- 2. \([A \rightarrow \bullet a, 0]\) (predict from 1)
- 3. \([A \rightarrow \bullet b, 0]\) (predict from 1)

\( s(1) : \)

- 4. \([A \rightarrow a\bullet, 0]\) (scan from 2)
Earley Execution

Grammar Rules: \( S \rightarrow A, \ A \rightarrow a, \ A \rightarrow b \)
Sentence: \( a \)

\( s(0): \)
- 1. \([S \rightarrow \bullet A, 0]\) (basis)
- 2. \([A \rightarrow \bullet a, 0]\) (predict from 1)
- 3. \([A \rightarrow \bullet b, 0]\) (predict from 1)

\( s(1): \)
- 4. \([A \rightarrow a\bullet, 0]\) (scan from 2)
- 5. \([S \rightarrow A\bullet, 0]\) (complete from 4)
Key Invariants

- \( l \) recognizes \( \text{sentence}(f+1 .. \text{ind}) \)

\[
(\forall a1, l1, r1, f1, \text{ind}). \ (\text{ind} < ii \ \land (a1 \leftrightarrow l1 \leftrightarrow r1 \leftrightarrow f1) \in \text{state} \ \land \\
(a1 \leftrightarrow l1 \leftrightarrow r1 \leftrightarrow f1) \in s(\text{ind}) \ \Rightarrow \\
(l1 \leftrightarrow \text{sentence}(f+1 .. \text{ind})) \in \text{derivable} \ ) \land
\]

- If next symbol recognizes \( \text{sentence}(j+1 .. z) \), item with dot moved is in state set \( z \)

\[
(\forall z, j, a1, l1, r1, f1). \ (z < ii \ \land \\
j \leq z \ \land (a1 \leftrightarrow l1 \leftrightarrow r1 \leftrightarrow f1) \in \text{state} \ \land (a1 \leftrightarrow l1 \leftrightarrow r1 \leftrightarrow f1) \in s(j) \ \land \\
([ \text{first}(r1)] \leftrightarrow \text{sentence}(j+1 .. z)) \in \text{derivable} \ \Rightarrow \\
((a1 \leftrightarrow (l1 \ ^{\text{first}(r1)}) \leftrightarrow \text{tail}(r1) \leftrightarrow f1)) \in s(z))
\]
Refining to Lists

- Specification of algorithm uses transitive closure of Earley operations to add all states
- Implementation: use something more concrete
- Idea: move linearly through the list; apply Earley operation to items as we go
- Works when there are no $\epsilon$-productions, but we run into trouble when there are
- Problem: completer misses states
Refining to Lists...

- Keep traversing list over and over until nothing new (inefficient)
- Keep track of all previous completers; run them again after processing (yuck, dynamic data structure)
- Aycock, Horspool 2002: traverse once, modify predictor to ensure no item is missed
- Linking invariant: states and sets are equal
Conclusion

• First formal verification of context-free recognizer
• Tools: ProB (for model checking), B4free (typechecking, automatic proving), hand-proof of correctness
• Invariants also lead to discovery of an optimization of the list refinement,
• Development synthesizes Earley and Aycock-Horspool into a single framework
Future Work

- Formalize parser version of recognizer
  - Purpose: investigate when and why Earley produces incorrect parse trees (Tomita, 1986)

- Develop a parser based on Parsing Expression Grammars
  - Purpose: inherently unambiguous (does this simplify things?), characterize the languages they can deal with

- CNF-based algorithms like CYK: do restrictions lead to more direct formal proofs?

- Any (simple) questions?
Epsilon Example

Grammar rules:  \( S \rightarrow AA, A \rightarrow \epsilon \)

- \( S \rightarrow \bullet AA \), 0  Basis
- \( A \rightarrow \epsilon \), 0  Predictor
- \( S \rightarrow A \bullet A \), 0  Completer
- \( S \rightarrow AA\bullet \), 0  MISSED!
Non-LR Java Fragment

FieldDecl → [FieldMods] Type Vars
FieldMods → FieldMod | FieldMods FieldMod
MethodHeader → [MethodMods] ResultType MethodDeclarator
Try parsing public static int...