Parsing beyond context-free grammar:

Simple RCG:
Incremental Earley Parsing
Laura Kallmeyer, Wolfgang Maier
Sommersemester 2009

Overview
1. Incremental Earley Parsing
   (a) Deduction Rules
   (b) Filters
2. Treebank Grammar Extraction
3. Demo

Incremental Earley Parsing
- Modification of Incremental CYK (MCFG) for ordered simple RCG
- Implemented in the TuLiPA system (http://sourcesup.cru.fr/tulipa)
- Strategy:
  - Process LHS arguments incrementally, starting from an S-rule
  - Whenever we reach a variable, move into rule of corresponding rhs predicate (predict or resume).
  - Whenever we reach the end of an argument, suspend the rule and move into calling parent rule.
  - Whenever we reach the end of the last argument convert item into a passive one and complete parent item.

Incremental Earley Parsing: Deduction Rules (1)
- Passive items: \([A, \vec{\rho}]\) where \(A\) is a predicate of arity \(k\) and \(\vec{\rho}\) is a range vector of arity \(k\)
- Active items:
  \([A(\vec{\phi}) \rightarrow A_1(\vec{\phi}_1) \ldots A_m(\vec{\phi}_m), pos, \langle i, j \rangle, \vec{\rho}]\)
  where
  - \(A(\vec{\phi}) \rightarrow A_1(\vec{\phi}_1) \ldots A_m(\vec{\phi}_m) \in P;\)
  - \(pos \in \{0, \ldots, n\};\) We have reached input position \(pos;\)
  - \(\langle i, j \rangle \in \mathbb{N}^2;\) We have reached the \(j\)th element of \(i\)th argument (dot position);\)
  - \(\vec{\rho}\) is a range vector containing variable and terminal bindings. All elements are initialized to "?", an initialized vector is called \(\vec{\rho}_{\text{init}}.\)
Incremental Earley Parsing: Deduction Rules (2)

- Notation:
  - $\vec{\rho}(X)$: range bound to variable $X$.
  - $\vec{\rho}((i, j))$: range bound to $j$th element of $i$th argument on LHS.

- Applying a range vector $\vec{\rho}$ containing variable bindings for given rule $c$ to the argument vector of the lefthand side of $c$ means mapping the $i$th element in the arguments to $\vec{\rho}(i)$ and concatenating adjacent ranges. The result is defined iff every argument is thereby mapped to a range.

Incremental Earley Parsing: Initialize, Goal item

Initialize: $[S(\vec{\phi}) \rightarrow \vec{\Phi}, 0, (1, 0), \vec{\rho}_{init}] 
\Rightarrow S(\vec{\phi}) \rightarrow \vec{\Phi} \in P$

Goal Item: $[S(\vec{\phi}) \rightarrow \vec{\Phi}, n, (1, j), \psi]$ with $|\vec{\phi}(1)| = j$ (i.e., dot at the end of lhs argument).

Incremental Earley Parsing: Scan

If next symbol after dot is next terminal in input, scan it.

Scan: $[A(\vec{\phi}) \rightarrow \vec{\Phi}, pos, (i, j), \vec{\rho}] 
\Rightarrow [A(\vec{\phi}) \rightarrow \vec{\Phi}, pos + 1, (i, j + 1), \vec{\rho'}]$ where $\vec{\rho'}$ is $\vec{\rho}$ updated with $\vec{\rho}((i, j + 1)) = (pos, pos + 1)$.

Incremental Earley Parsing: Predict

Whenever our dot is left of a variable that is the first argument of some rhs predicate $B$, we predict new $B$-rules:

Predict: $[A(\vec{\phi}) \rightarrow \ldots B(X, \ldots \ldots \ldots pos, (i, j), \vec{\rho}_{init}] 
\Rightarrow \vec{\phi}(i, j + 1) = X, B(\vec{\psi}) \rightarrow \vec{\Psi} \in P$

where $\vec{\phi}(i, j + 1) = X, B(\vec{\psi}) \rightarrow \vec{\Psi} \in P$.
Incremental Earley Parsing: Suspend

Suspend:
\[ [B(\psi) \rightarrow \Psi, \text{pos}', (i, j), \tilde{\rho}_B], [A(\tilde{\phi}) \rightarrow \ldots B(\xi), \ldots, \text{pos}, (k, l), \tilde{\rho}_A] \]
\[ [A(\tilde{\phi}) \rightarrow \ldots B(\xi), \ldots, \text{pos}', (k, l + 1), \tilde{\rho}] \]

where
- the dot in the antecedent A-item precedes the variable \( \xi(i) \),
- \( |\psi(i)| = j \) (ith argument has length \( j \), i.e., is completely processed),
- \( i < j \) (ith argument is not the last argument of \( B \)),
- \( \tilde{\rho}_B(\tilde{\psi}(i)) = \text{pos, pos}' \),
- and for all \( 1 \leq m < i \): \( \tilde{\rho}_B(\tilde{\psi}(m)) = \tilde{\rho}_A(\xi(m)) \).

\( \tilde{\rho} \) is updated with \( \tilde{\rho}_A(\xi(|\tilde{\rho}_B|)) = \text{pos, pos}' \).

Incremental Earley Parsing: Complete

Whenever we have a passive \( B \) item we can use it to move the dot over the variable of the last argument of \( B \) in a parent \( A \)-rule:

Complete:
\[ [B, \tilde{\rho}_B], [A(\tilde{\phi}) \rightarrow \ldots B(\xi), \ldots, \text{pos}, (k, l), \tilde{\rho}_A] \]
\[ [A(\tilde{\phi}) \rightarrow \ldots B(\xi), \ldots, \text{pos}', (k, l + 1), \tilde{\rho}] \]

where
- the dot in the antecedent A-item precedes the variable \( \xi(|\tilde{\rho}_B|) \),
- the last range in \( \tilde{\rho}_B \) is \( \text{pos, pos}' \),
- and for all \( 1 \leq m < |\tilde{\rho}_B| \): \( \tilde{\rho}_B(m) = \tilde{\rho}_A(\xi(m)) \).

\( \tilde{\rho} \) is updated with \( \tilde{\rho}_A(\xi(|\tilde{\rho}_B|)) = \text{pos, pos}' \).

Incremental Earley Parsing: Resume

Whenever we are left of a variable that is not the first argument of one of the rhs predicates, we resume the rule of the rhs predicate.

Resume:
\[ [A(\tilde{\phi}) \rightarrow \ldots B(\xi), \ldots, \text{pos}, (i, j), \tilde{\rho}_A] \]
\[ [B(\psi) \rightarrow \Psi, \text{pos}', (k - 1, l), \tilde{\rho}_B] \]
\[ [B(\psi) \rightarrow \Psi, \text{pos}, (k, 0), \tilde{\rho}_B] \]

where
- \( \tilde{\phi}(i, j + 1) = \xi(k), k > 1 \) (the next element is a variable that is the \( k \)th element in \( \xi \), i.e., the \( k \)th argument of \( B \)),
- \( |\psi(k - 1)| = l \), and
- \( \tilde{\rho}_A(\xi(m)) = \tilde{\rho}_B(\psi(m)) \) for all \( 1 \leq m \leq k - 1 \).
Incremental Earley Parsing: Filters

- Filters can be applied to decrease the number of items in the chart.
- A filter is an additional condition on the form of items.
- E.g., in a ε-free grammar, the number of variables in the part of the lefthand side arguments of a rule that has not been processed yet must be lower or equal to the length of the remaining input.

Incremental Earley Parsing: Remaining Input Length Filter

- In ε-free grammars each variable must cover at least one input symbol.
- i input symbols left implies no prediction of a clause with more than i variables or terminals on LHS since no instantiation is possible.
- Condition on active items, can be applied with predict, resume, suspend and complete.

An item \( [A(\vec{\phi}) \rightarrow A_1(\vec{\phi}_1) \ldots A_m(\vec{\phi}_m), pos, \langle i, j \rangle, \vec{\rho}] \) satisfies the length filter iff
\[
(n - pos) \geq (|\vec{\phi}(i)| - j) + \sum_{k=i+1}^{\text{term}(A)} |\vec{\phi}(k)|
\]

Incremental Earley Parsing: Preterminal Filter (1)

- Check for the presence of (pre)terminals in the predicted part of a clause in the remaining input, and
- check that terminals appear in the predicted order and that distance between two of them is at least the number of variables/terminals in between.

In other words, an active item \( [A(\vec{\phi}) \rightarrow A_1(\vec{\phi}_1) \ldots A_m(\vec{\phi}_m), pos, \langle i, j \rangle, \vec{\rho}] \) satisfies the preterminal filter iff we can find an injective mapping \( f_T : \text{Term} = \{ \langle k, l \rangle \mid \vec{\phi}(k, l) \in T \} \) such that
1. \( w_{f_T(k, l)}(\vec{\phi}) = k(l) \) for all \( \langle k, l \rangle \in \text{Term} \);
2. for all \( \langle k_1, l_1 \rangle, \langle k_2, l_2 \rangle \in \text{Term} \) with \( k_1 = k_2 \) and \( l_1 < l_2 \):
   \( f_T((k_2, l_2)) \geq f_T((k_1, l_1)) + (l_2 - l_1) \);
3. for all \( \langle k_1, l_1 \rangle, \langle k_2, l_2 \rangle \in \text{Term} \) with \( k_1 < k_2 \):
   \( f_T((k_2, l_2)) \geq f_T((k_1, l_1)) + (|\vec{\phi}(k_1)| - l_1) + \sum_{k=k_1+1}^{k_2-1} |\vec{\phi}(k)| + l_2 \).
Treebank Grammar Extraction

- (P)CFGs can only be extracted from projective dependency trees (constituent trees without crossing branches).
- Dependency treebanks generally contain non-projective dependencies (e.g., PDT); Constituent treebanks generally do not contain crossing branches annotation, however there are counterexamples: TIGER, NeGra, BulTreebank.
- LCFRS/simple RCG can handle non-projective dependencies (crossing branches) as well, extraction algorithms can deliver ordered ε-free simple RCGs without useless rules.

Example: A dependency tree and the set of obtained clauses.

```
Darüber muß nachgedacht werden
```

```
root(Darüber) → ε
pp(Darüber) → ε
root(X1,muß,X3) → aux(X1,X3)
aux(X1,nachgedacht) → pp(X1)
aux(X1,X2,werden) → aux(X1,X2)
```

Example: A constituent tree and the set of obtained clauses.

```
PROAV1(Darüber) → ε
VMFIN1(muß) → ε
VVPP1(nachgedacht) → ε
VAINF1(werden) → ε
S1(X1,X2,X3) → VP2(X1,X3) VMFIN1(X3)
VP2(X1,X2,werden) → VP2(X1,X2) VAINF1(X2)
VP2(X1,X2) → PROAV1(X1) VVPP1(X2)
```

Demo
Conclusion

- Incremental Earley Algorithm for simple RCG takes advantage of variable ordering
- Filters can drastically reduce the number of items in the chart
- Grammar extraction algorithms extract simple RCG for probabilistic parsing