Earley Parsing in Prolog

Aleksandar L. Dimitrov

Universität Tübingen

December 19, 2007

Outline

Use of assert

- Can Prolog have only store a limited set of predicates?

Use of assert

- Can Prolog have only store a limited set of predicates?
  - So far: Predicates from Source files

Use of assert

- Can Prolog have only store a limited set of predicates?
  - So far: Predicates from Source files
  - New: Predicates dynamically generated from

Use of assert

- Can Prolog have only store a limited set of predicates?
  - So far: Predicates from Source files
  - New: Predicates dynamically generated from
    - User input
    - Creative use of existing predicates

Use of assert

- Can Prolog have only store a limited set of predicates?
  - So far: Predicates from Source files
  - New: Predicates dynamically generated from
    - User input
    - Creative use of existing predicates
- ... resulting in Self-Modifying Programs

Use of `assert`, contd.

- Are you only allowed to add to existing predicates?

Use of `assert`, contd.

- Are you only allowed to add to existing predicates?
- *Workaround:* use of `unknown/2`

Use of `assert`, contd.

- Are you only allowed to add to existing predicates?
- *Workaround:* use of `unknown/2`
- How flexible is your Database?

Use of `assert`, contd.

- Are you only allowed to add to existing predicates?
- *Workaround:* use of `unknown/2`
- How flexible is your Database?
- $\longrightarrow$ use of `abolish/1`, `retract/1`

Use of `assert`, contd.

- Are you only allowed to add to existing predicates?
- *Workaround:* use of `unknown/2`
- How flexible is your Database?
- $\longrightarrow$ use of `abolish/1`, `retract/1`
- Watch out for unification!

Use of `assert`, contd.

- Are you only allowed to add to existing predicates?
- *Workaround:* use of `unknown/2`
- How flexible is your Database?
- $\longrightarrow$ use of `abolish/1`, `retract/1`
- Watch out for unification!
- Use `assertz/1` and `asserta/1`

Use of `assert`, contd.

- Are you only allowed to add to existing predicates?
- *Workaround:* use of `unknown/2`
- How flexible is your Database?
- $\longrightarrow$ use of `abolish/1`, `retract/1`
- Watch out for unification!
- Use `assertz/1` and `asserta/1`
- . . . also for efficiency reasons

Chart Parsing with Prolog

- Chart Parsers operate entirely on their chart

Chart Parsing with Prolog

- Chart Parsers operate entirely on their chart
- Why Chart Parsing?

Chart Parsing with Prolog

- Chart Parsers operate entirely on their chart
- Why Chart Parsing?
- Imagine a grammar:
  `A -> BC(D)`

Chart Parsing with Prolog

- Chart Parsers operate entirely on their chart
- Why Chart Parsing?
- Imagine a grammar:
  `A -> BC(D)`
- Input: `BCD`

Chart Parsing with Prolog

- Chart Parsers operate entirely on their chart
- Why Chart Parsing?
- Imagine a grammar:
  `A -> BC(D)`
- Input: `BCD`

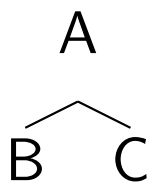
Minimalistic Example

Grammar:

A -> BC

A -> BCD

Input: BCD

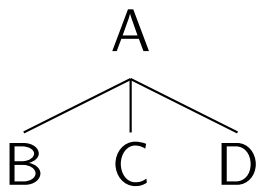


Minimalistic Example

Grammar:

```
A -> BC
A -> BCD
```

Input: BCD



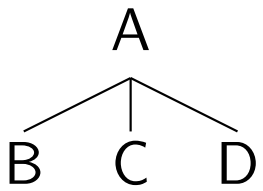
Minimalistic Example

Grammar:

`A -> BC`

`A -> BCD`

Input: `BCD`



→ Solution: Dynamic Programming

Use of assert in this context

Our original TD parser:

```
parse(C,[Word|S],S) :-
word(C,Word).

parse(C,S1,S) :-
rule(C,Cs),
parselist(Cs,S1,S).
```

Use of assert in this context

Our new TD chart parser:

```
parse(C,[Word|S],S) :-
word(C,Word).

parse(C,S1,S) :- chart(C,S1,S).

parse(C,S1,S) :-
rule(C,Cs),
parselist(Cs,S1,S),
asserta(chart(C,S1,S)).
```

Earley's Algorithm

Short Recap

- Earley is a Chart Parser
- The chart is consulted and modified by three main components:

Short Recap

- Earley is a Chart Parser
- The chart is consulted and modified by three main components:
  - The Predictor (Top-Down element)

Short Recap

- Earley is a Chart Parser
- The chart is consulted and modified by three main components:
  - The Predictor (Top-Down element)
  - The Scanner

Short Recap

- Earley is a Chart Parser
- The chart is consulted and modified by three main components:
  - The Predictor (Top-Down element)
  - The Scanner
  - The Completer (Bottum-Up elment)

Short Recap

- Earley is a Chart Parser
- The chart is consulted and modified by three main components:
  - The Predictor (Top-Down element)
  - The Scanner
  - The Completer (Bottum-Up elment)
- Let's see an example...

A Short Example

The dog chases the cat

- Predict:

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict:

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan:

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete:

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing.

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict:

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing.

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing. Why?

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing. Why?
- Scan:

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing. Why?
- Scan: Accept *dog*

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing. Why?
- Scan: Accept *dog*
- Complete:

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing. Why?
- Scan: Accept *dog*
- Complete: Complete *NP*

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing. Why?
- Scan: Accept *dog*
- Complete: Complete *NP*
- etc...

A Short Example

The dog chases the cat

- Predict: $S \longrightarrow NP\ VP$
- Predict: $NP \longrightarrow D\ N$
- Scan: Accept *the*
- Complete: Nothing. Why?
- Predict: Nothing. Why?
- Scan: Accept *dog*
- Complete: Complete *NP*
- etc...

The Outer Predicates

- We only need to modify `parse` sligthly

The Outer Predicates

- We only need to modify `parse` sligthly
- . . . and introduce `process` which is just a wrapper around the three magicians. . .

The Outer Predicates

- We only need to modify `parse` sligthly
- . . . and introduce `process` which is just a wrapper around the three magicians. . .

The three Magicians

- . . . make an *exhaustive* search on their current domain

The three Magicians

- . . . make an *exhaustive* search on their current domain
- . . . store everything they find on the main chart

The three Magicians

- . . . make an *exhaustive* search on their current domain
- . . . store everything they find on the main chart
- . . . but first operate on what they found themselves

The three Magicians

- . . . make an *exhaustive* search on their current domain
- . . . store everything they find on the main chart
- . . . but first operate on what they found themselves (`asserta`)
- All of them rely on `store/3`

The `store` predicate

- Consider $NP \longrightarrow NP\ Conj\ NP$

The `store` predicate

- Consider $NP \longrightarrow NP\ Conj\ NP$
- How does Earley avoid left recursion?

The `store` predicate

- Consider $NP \longrightarrow NP\ Conj\ NP$
- How does Earley avoid left recursion?
- `store` doesn't store anything that's already there!
- But how do we parse recursive structures such as *The dog and the cat and the elephant*?

What do we do about empty determiners?

- What about them?

What do we do about empty determiners?

- What about them?
- We could: modify the predictor

What do we do about empty determiners?

- What about them?
- We could: modify the predictor
- Or recast $D \longrightarrow \emptyset$ as word and modify the scanner

What do we do about empty determiners?

- What about them?
- We could: modify the predictor
- Or recast $D \rightarrow \emptyset$ as word and modify the scanner
- Method 1: modifying the predictor...

What do we do about empty determiners?

- What about them?
- We could: modify the predictor
- Or recast $D \rightarrow \emptyset$ as word and modify the scanner
- Method 1: modifying the predictor...
- What about Method 2?