Application Programming Interface
for Chart-based Parser Framework
for Lexicalized Grammar Formalisms

Prachya Boonkwan
Human Language Technology Laboratory (HLT)
National Electronics and Computer Technology Center
Pathumthani, Thailand 12120

1 Version History
Version 0.1 August 10, 2006
  - First released
Version 0.2 October 23, 2006
  - Feature structure support added
  - Grammar tester (parse and print the results)
Version 0.3 November 4, 2006
  - The parser yields out each parse instead of first finding all solutions

2 Available Predicates
- load_rules/1: This predicate loads all lexicon definition rules from the specified file and compiles them into internal representation.
- compile_rules/0: This predicate compiles all loaded lexicon definition rules.
- show_rules/0: Show all rules in the knowledge base.
- clear_rules/0: Clear all rules from the knowledge base.
- show_chart/0: Show the parsing chart.
- parse/2: Parse a sequence of tokens and yield out a parsing result in the list.
- test/2: Parse a sentence to test a grammar. All possible parses will be displayed.

3 Grammar Rule Manipulation

3.1 Predicate: load_rules(+RuleFile)
This predicate loads all lexicon definition rules from the specified file RuleFile and compiles them into internal representation.

The algorithm of this predicate is described as follows.
1. Remove all existing lexicon definition rules from the knowledge base.
2. Load lexicon definition rules from the specified file.
3. Compile them into internal representation by calling compile_rules/0.

In this program, a lexicon definition rule is represented as $W := C$, where $W$ is a lexicon's surface form and $C$ is a CCG syntactic category. Compilation of the rule is in the form of $\text{rule}(W, R)$, where $R$ is an internal representation of $C$ provided by calling $\text{ccg:conv_expr_repr}(C, R)$.

3.2 Predicate: compile_rules

This predicate compiles all loaded lexicon definition rules.

In this program, a lexicon definition rule is represented as $W := C$, where $W$ is a lexicon's surface form and $C$ is a CCG syntactic category. Compilation of the rule is in the form of $\text{rule}(W, R)$, where $R$ is an internal representation of $C$ provided by calling $\text{ccg:conv_expr_repr}(C, R)$.

3.3 Predicate: compile_rule(+NumericFlag, +Category, –Result)

This predicate converts a syntactic category $\text{Category}$ into its corresponding internal representation $\text{Result}$. The flag $\text{NumericFlag}$ signifies the rule compiler to produce a special form for number handling.

3.4 Predicate: show_rules

Show all rules in the knowledge base.

3.5 Predicate: clear_rules

Clear all rules from the knowledge base.

4 Chart Parsing

4.1 Predicate: init_chart(–Tokens, –NoTokens)

This predicate initiates the parsing chart with a list of tokens $\text{Tokens}$ whose number of tokens is $\text{NoTokens}$.

The initiation of the chart is as follows. For each token at the position $i$, this predicate asserts a new edge $(i, i+1, t_i)$ to the chart. If the token is a number, its specialized representation will be instantiated before assertion.

4.2 Predicate: iterate_chart(+Len, +NoTokens)

Iterate all edges in the chart, merge them, and combine their category with the specified length $\text{Len}$ and repeat iteration with increased lengths until the length equates the specified limit $\text{NoTokens}$.

Development details: This predicate call two sub-predicates; namely, $\text{find_mergeable_edges}/4$ to find all consecutive edges with combinable categories, and $\text{find_conjoinable_edges}/4$ to find all consecutive edges conjoined with the conjunction.
4.3 Predicate: `find_mergeable_edges(+Len, +Src, +Dst, -DerivedRepr)`

Find all mergeable edge pairs; one starts at the specified position `Src` and the other one ends at the specified position `Dst`. Two edges, when merged, must be of the length `Len`. The combination of their categories yields `DerivedRepr`.

4.4 Predicate: `find_conjoinable_edges(+Len, +Src, +Dst, -DerivedRepr)`

Find all edges conjoined with a conjunction; one starts at the specified position `Src`, the other one ends at the specified position `Dst`, and these two must be joined with a conjunction. Two edges including the conjunction edge must be of the specified length `Len`. The conjunction of the two edges' categories yields `DerivedRepr`.

4.5 Predicate: `show_chart`

Show the parsing chart. This predicate lists all edges in the parsing chart.

4.6 Predicate: `parse(+Tokens, -Parse)`

Parse a sequence of tokens `Tokens` and yield out a parsing result in the list `Parse`.

The algorithm of this predicate is as follows.

1. Initiate the parsing chart with `init_chart/2`.
2. Iterate each edge of the chart with `iterate_chart/2`.
3. Yield out a parsing result.

5 Grammar Testing

5.1 Predicate: `test(+Id, +Sentence)`

Parse a sentence `Sentence` whose ID is `Id`. The parses are listed on the screen.

6 Example Usage

First, load a grammar formalism library to the memory. In this example, CCG is used for demonstration.

```prolog
1 ?- [ccg].
% ccg compiled into ccg 0.00 sec, 27,984 bytes
Yes
```

Then load the parser module.
At this step, we can load up the grammar rules from a file. In this example, we load up a dependency grammar for English.

![Code snippet](image)

After loading, we can list available grammar rules.

![Code snippet](image)

We can parse a sentence with the predicate `parser:parse/2`.

![Code snippet](image)

After parsing, we can trace back parsing steps by investigating the chart.
6 ?- show_chart.
Edge (0, 1): n(i)
Edge (1, 2): s(_G374 > (kiss < _G378)) \# n(_G374) /\ n(_G378)
Edge (2, 3): conj and
Edge (3, 4): n(they)
Edge (4, 5): s(_G374 > (await < _G378)) \# n(_G374) /\ n(_G378)
Edge (5, 6): n(you)
Edge (0, 2): s(i > (kiss < _G378)) /\ n(_G378)
Edge (3, 5): s(they > (await < _G378)) /\ n(_G378)
Edge (4, 6): s(_G374 > (await < you)) \# n(_G374)
Edge (3, 6): s(they > (await < you))
Edge (0, 5): s(coor(and, s, [[i > (kiss < _G390)], [they > (await < _G390)]], [\], [n(_G390)])) /\ n(_G390)
Edge (0, 6): s(coor(and, s, [[i > (kiss < you)], [they > (await < you)]], [\], [n(you)]))
Yes