Spejd

Adam Przepiórkowski, Aleksander Buczyński (formalism, basic implementation)
Bartosz Zaborowski (re-implementation)
{**adamp;b.zaborowski**}**@ipipan.waw.pl** Institute of Computer Science, PAS

Introduction

Tagging and (partial) parsing are usually done as separate processes. Spejd combines them into one parallel process: allows to simultaneously disambiguate and build syntactic structures within a single rule.

Formalism

Main ideas:

- rules = cascade of regular grammars
- each rule has 2 parts:
  - **match specification** (describes desired phrase and its contexts at various levels of annotation)
  - **list of operations** (modifying morphosyntactic information, creating and altering syntactic structures and others)
- the formalism is language and tagset independent (of course specific rules are not:)

An example rule, which at the same time unifies morphosyntactic information of words and builds a syntactic structure:

```
Rule "NG: Adj + Noun"
Match: [pos~"Adj|Pact|Ppas"]
     ([pos~"Noun"] | [type="NG.*"]);
Eval: unify(case number gender,1,2);
      group(NG,2,2);
```

In comparison with 0.8x, current version provides:

- **simplifications of syntax**
- **larger number of eval operations**
- **numeric variables/attributes support**

New extensions in an example rule:

```
Rule "numeric example"
Match: A[orth~nie/i] B[base~być]
      C[abs(sen)>5 && revers~"rev"];
Eval: alter(B, nrefl, B.base);
    word(C, sen=-sen*@some_var:neg,
                      A.base C.base);
    assign(@some_var=C.sen*0.8);
```

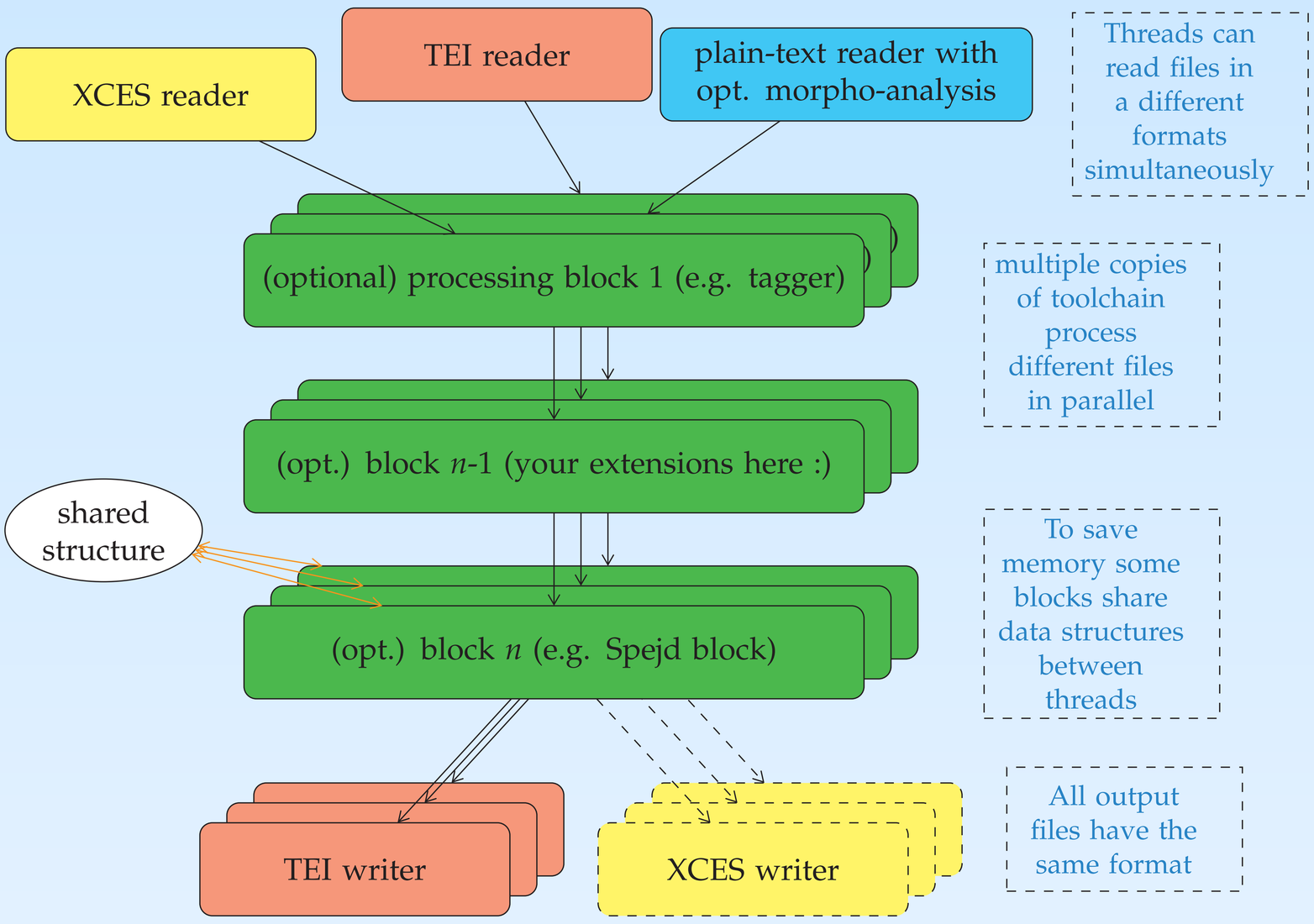
Main features

- bases on handwritten rules
- supports for various input/output formats (XCES, TEI, plain-text)
- built-in sentencer, tokenizer, morphological analyzer, tagger (for Polish)
- has easy interface to external morphological dictionaries
- fast, stable, extensible
- open-source (GPL license), homepage: **http://zil.ipipan.waw.pl/Spejd/**

Implementation

- Spejd formalism is implemented as a one block in a toolchain.
- the code is highly optimized, written in C++
- available as a library for Linux and Windows, to make integration with other tools easier

The only required nodes in toolchain are reader and writer. Each thread uses a separate copy of chain (blocks may share some data).



The toolchain is easily extensible, e.g. blocks allowing to use external tools like taggers can be written in just few hours. It is also possible to add support for different input/output formats. Documentation on writing extensions is included in the source package.

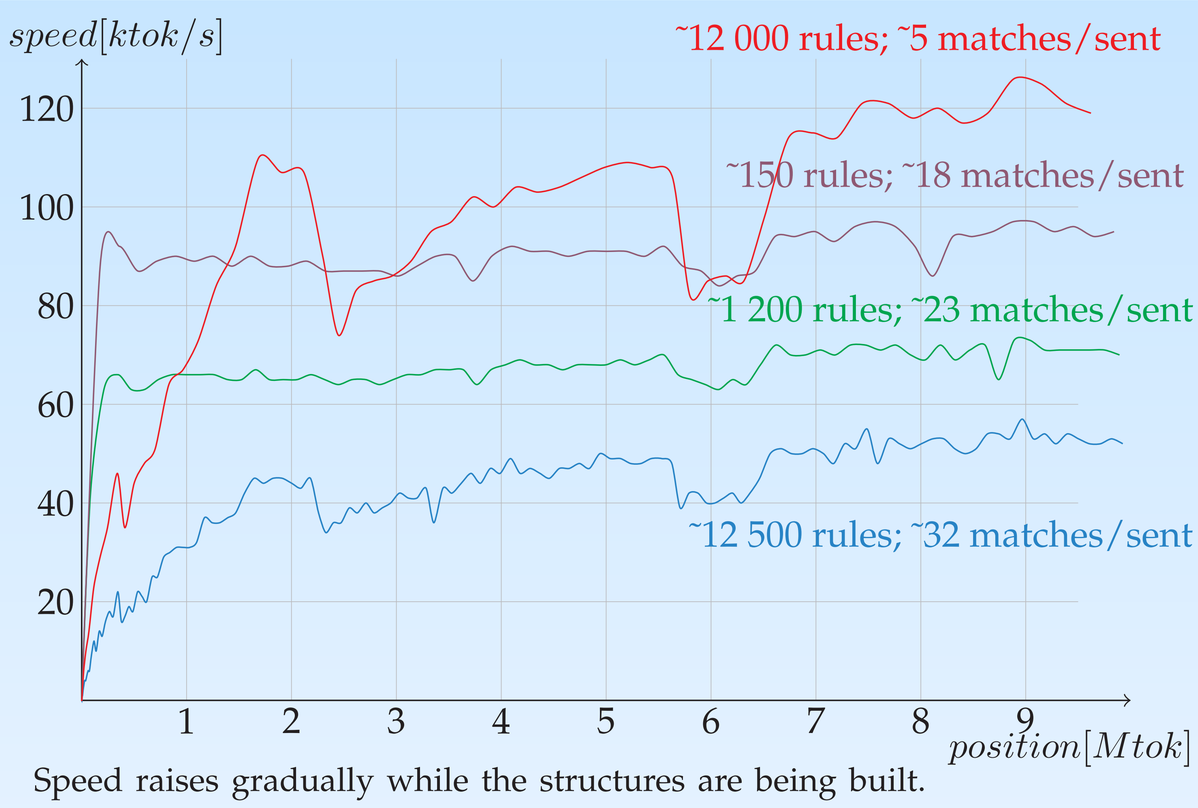
Performance

The toolchain is highly optimized, some major optimizations are:

- custom fast parsers for XML formats
- full parallelisation
- Spejd formalism uses advanced finite-state techniques: its performance is hardly related to the total number of rules

At the moment, Spejd toolchain:

- was tested on grammars with over 12.000 of rules
- was tested on input data consisting of over a billion of words
- on a modern computer with 4-core CPU reaches performance of **60k rules applied per second** and for less complex grammars reaches speed of **over 200k words per second**
- scales well on systems with dozens of CPUs (only for small grammars or relatively large corpora, because of gradual gaining of speed which is visible on the plot)



Speed raises gradually while the structures are being built.
It depends mainly on rule matches rate, not on the total number of rules.