

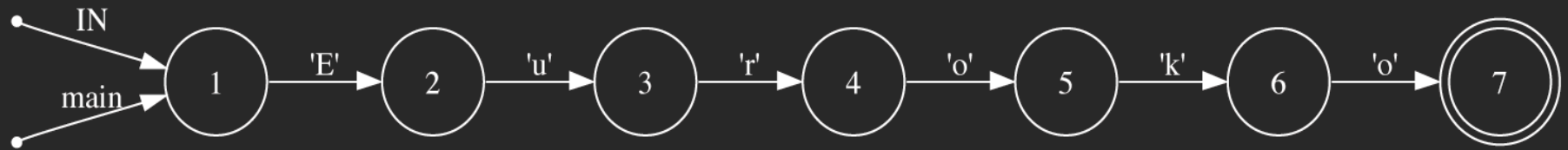


Ragel & Ruby

Ry Dahl ([ry@tinyclouds.org](mailto:ry@tinyclouds.org))

Euruko, November 2007

# What is Ragel?



- A parser generator
- Created by Adrian Thurston (Queens University)
- Ragel compiles to a host language: C, D, Java, Ruby.
- Compiled code has no dependencies on to Ragel

# What is Ragel useful for?



- Parsing protocols and data formats.
  - EG: HTTP, XML, JSON, CSS

You should use it when you need...

- More than a single regular expression
- Speed
- Less than a full LALR parser.

# Ruby projects using Ragel



- Mongrel (HTTP request parsing)
- Ruby JSON
- SuperRedCloth (Textile markup)
- Hpricot (HTML parser)

Example: Parsing  $/(ab|cd)^*/$

```
%%{  
  machine ex2;  
  action foo { f += 1 }  
  action bar { b += 1 }  
  x = "ab" %foo;  
  y = "cd" %bar;  
  main := "_" (x|y)* "_";  
}%%
```

# Actions and Non-Determinism

- Actions can be executed midstream.
  - `%action_name` leaving action
  - `$action_name` all transition action
  - `>action_name` starting action
  - `@action_name` finishing action
- Non-Determinism can be controlled by setting priorities

A decorative header consisting of five blue circles. The first, second, and fifth circles are solid blue, while the third and fourth circles are hollow with a blue outline. The text 'Actions have the ability to' is centered across these circles.

Actions have the ability to

- Change the state (node)
- Move the data pointer
- Exit the parser
- Anything really...

Actions are written in the host language but can include special Ragel functions (`fcall`, `fgoto`, ...)

# Interfacing Ruby & Ragel



- Compile into Ruby
  - SLOW
  - Even if you're willing to take the speed hit, might be easier to use adhoc code instead of Ragel.
  - Useful for prototyping and testing a Ragel machine.
- Write C extension
  - FAST

# Writing a Ruby-Ragel C extension

- Most important task is to extract string
  - Mark the beginning with `>mark_action`
  - At the end save to a `VALUE` variable with `%save_action`
  - In the save action use

```
VALUE rb_str_new(const char* str, long length);
```

# Writing a Ruby-Ragel C extension

## An example from Hpricot

```
/* Adapted from hpricot_scan.rl */
VALUE hpricot_scan(VALUE self, VALUE data)
{
    int cs = 0; # current state
    char *p = RSTRING_PTR(data); # data pointer
    char *pe = p + RSTRING_LEN(data); # data end pointer

    VALUE tag = Qnil;
    char *mark_tag = 0;

    %% write init;
    %% write exec;

    /* ... */
}
```

# Writing a Ruby-Ragel C extension

## An example from Hpricot

```
%%{  
  # Adapted from hpricot_scan.rl  
  machine hpricot_common;  
  
  action _tag { mark_tag = p; }  
  action tag { tag = rb_str_new(mark_tag, p-mark_tag); }  
  
  NameChar = [\-A-Za-z0-9._:~] ;  
  Name = [A-Za-z_:] NameChar* ;  
  NameCap = Name >_tag %tag; ←  
  StartTag = "<" NameCap ">";  
  EndTag = "</" NameCap ">";  
  # ...  
}%}
```

# What if the host language is Ruby?

```
action _tag { mark_tag = p }  
action tag { tag = data[mark_tag..p-1] }
```

# Writing a Ruby-Ragel C extension

- Sometimes desirable to have a Ruby independent interface
  - Not much more work
  - Clean interface
  - Zed Shaw's `http11` is a good example. He uses
    - A `struct` with callbacks for each token encountered.
    - The `struct` contains `void*` pointer for the Ruby data structure that is being built up during the parsing

```
typedef void (*element_cb)(void *data, const char *at,  
                           size_t length);
```

```
typedef struct http_parser {  
    int cs;  
    size_t body_start;  
    int content_len;  
    size_t nread;  
    size_t mark;  
    size_t field_start;  
    size_t field_len;  
    size_t query_start;
```

```
    void *data;
```

```
    element_cb request_method;
```

```
    element_cb request_uri;
```

```
    element_cb fragment;
```

```
    element_cb request_path;
```

```
    element_cb query_string;
```

```
    element_cb http_version;
```

```
    element_cb header_done;
```

```
} http_parser;
```

# What to take away from this

- Ragel is a parser generator being used in a few Ruby projects
- You should read the code of Zed Shaw and why the lucky stiff.