

GENERATING STATE-BASED SYNTAX HIGHLIGHTERS FROM RASCAL'S CONTEXT-FREE GRAMMARS

By Edser Apperloo

Supervisors:
Tijs van der Storm
&
Mircea Lungu



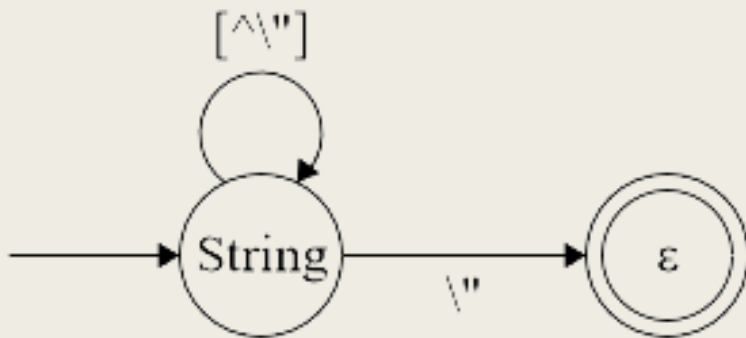
university of
 groningen

State-based syntax highlighters

- Similar to state machines
- Contexts
- Matches
- Scopes

State-based syntax highlighters

- Contexts -> States
- Matches -> Transitions
- Scopes -> Colouring



String:

```
- meta_scope: string.quoted.double
- match: '[^"]'
- match: '\"'
  pop: true
```

The designed algorithm in Rascal

- How to embed highlighter information
- Simplify grammars
- Transform to strongly regular grammars
- To component machines
- Translation to a syntax highlighter

The original grammar

- Example sentences:

- “bbddbdb”
- “bdbdbd”

```
lexical A = (B | D)+;
```

```
lexical B
```

```
= @Context="keyword.control.flow" "b"  
| @Context="null keyword.control.flow" D "b"  
;
```

```
lexical D
```

```
= @Context="storage.type" "d"  
| @Context="storage.type null" "d" B  
;|
```

- @Context=“storage.type”
- @Context=“null storage.type”

The plain version

- Rewritten regular tokens
- Reduced to $\backslash prod(_,_,_)$

```
start syntax S = A;
```

```
lexical A = B_D_ALT_PLUS;
```

```
lexical B_D_ALT_PLUS  
  = B_D_ALT B_D_ALT_PLUS  
  | B_D_ALT  
  ;
```

```
syntax B_D_ALT  
  = B  
  | D  
  ;
```

```
lexical B  
  = @Context="keyword.control.flow" "b"  
  | @Context="null keyword.control.flow" D "b"  
  ;
```

```
lexical D  
  = @Context="storage.type null" "d" B  
  | @Context="storage.type" "d"  
  ;
```

Strongly regular version

- Mohri and Nederhof
- Wants right- or left-linear rules
- Mutually recursive non-terminals
- Transforms any CFG into a strongly regular approximation

```
start syntax S = A;
```

```
lexical A = B_D_ALT_PLUS;
```

```
lexical B_D_ALT_PLUS  
  = B_D_ALT B_D_ALT_PLUS  
  | B_D_ALT  
  ;
```

```
syntax B_D_ALT  
  = B  
  | D  
  ;
```

```
lexical B  
  = "b" B_end  
  | D  
  ;
```

```
lexical B_end  
  = D_end  
  |  
  ;
```

```
lexical D  
  = "d" D_end  
  | "d" B  
  ;
```

```
lexical D_end  
  = "b" B_end  
  |  
  ;
```

```
start syntax S = A;
```

```
lexical A = B_D_ALT_PLUS;
```

```
lexical B_D_ALT_PLUS  
    = B_D_ALT B_D_ALT_PLUS  
    | B_D_ALT  
    ;
```

```
syntax B_D_ALT  
    = B  
    | D  
    ;
```

```
lexical B  
    = "b" B_end  
    | D  
    ;
```

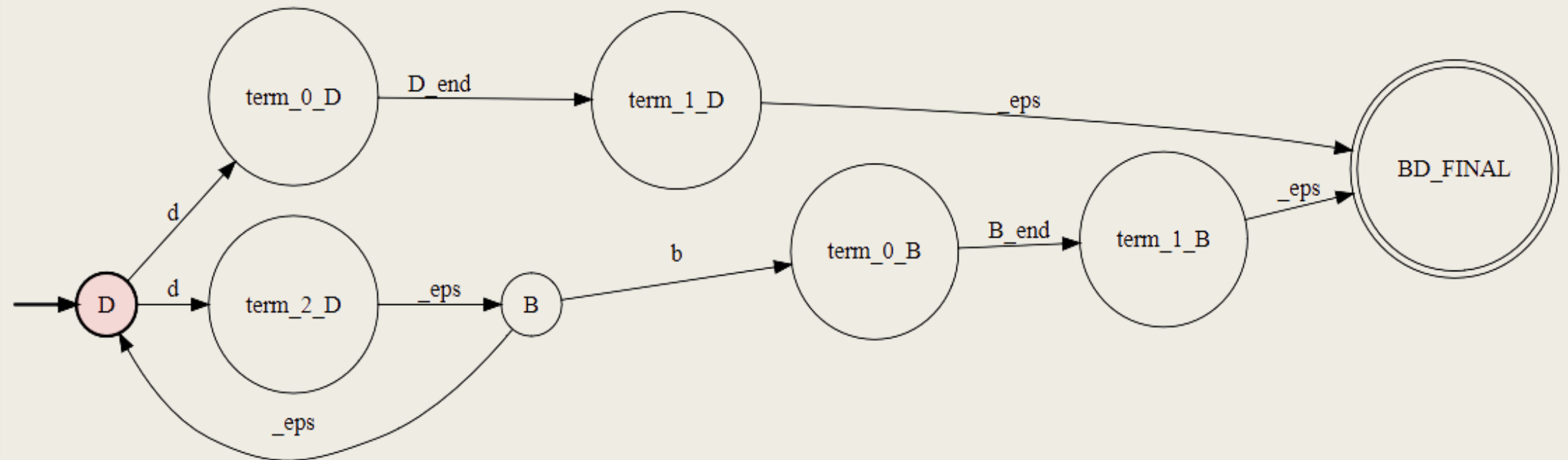
```
lexical B_end  
    = D_end  
    |  
    ;
```

```
lexical D  
    = "d" D_end  
    | "d" B  
    ;
```

```
lexical D_end  
    = "b" B_end  
    |  
    ;
```

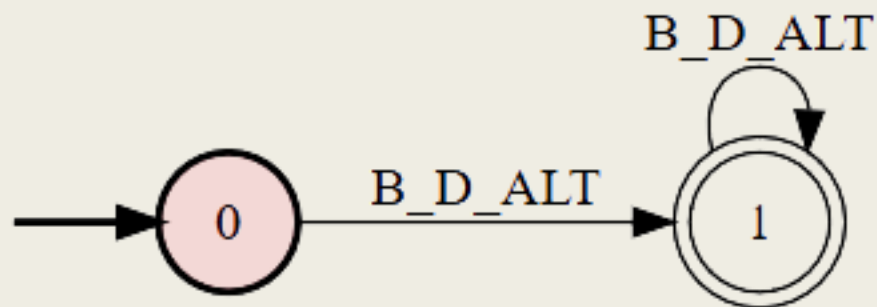

Component machines

- One machine per strongly connected component
- Compact representation
- NFA for “*lexical D*”



Mapping to Highlighter

- Almost direct mapping
- Final states
 - *Sink*
 - *Non-Sink*



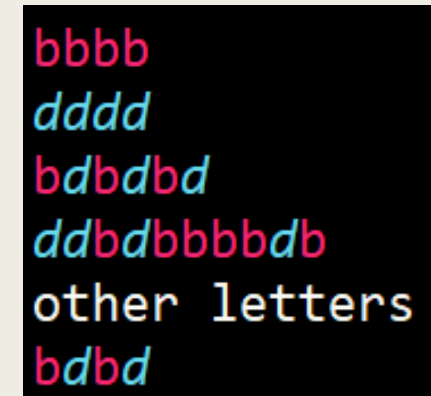
```
B_D_ALT_PLUS_0:  
- meta_scope: B_D_ALT_PLUS_0  
- match: '(?=(b|d))'  
  set: [B_D_ALT_PLUS_1, B_D_ALT_0]
```

```
B_D_ALT_PLUS_1:  
- meta_scope: B_D_ALT_PLUS_1  
- match: '(?=(b|d))'  
  set: [B_D_ALT_PLUS_1, B_D_ALT_0]  
- match: '(?!((?=(b|d))))'  
  pop: true
```

Results & Evaluation

- Performance and resulting highlighter

- *11 contexts*
- *~100 SLOC*

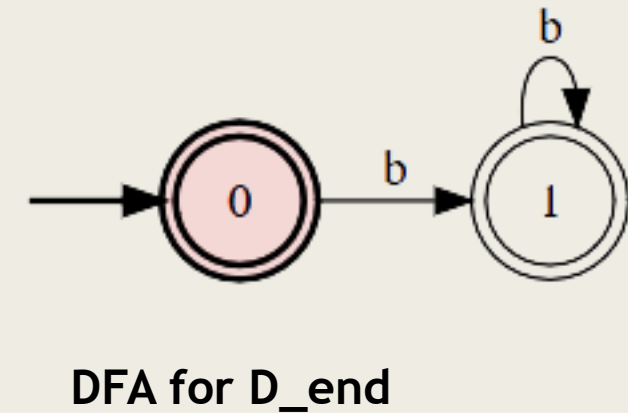
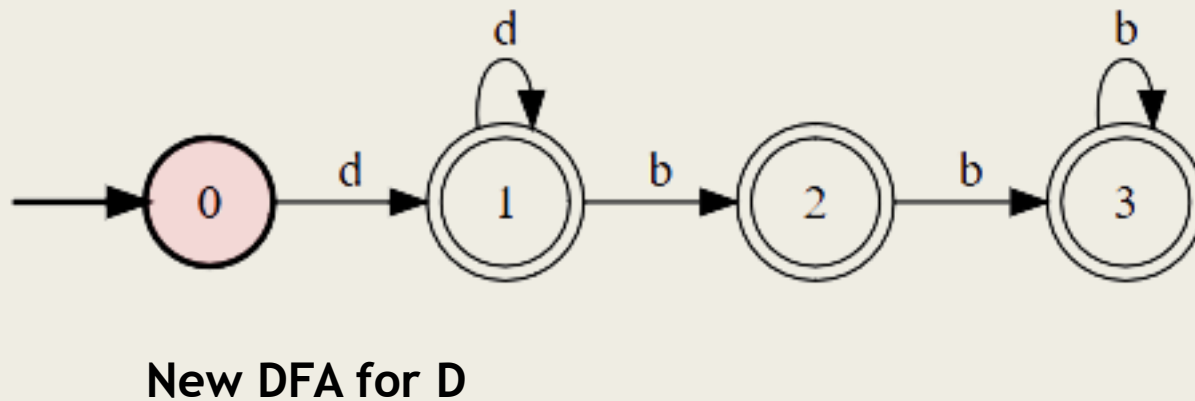
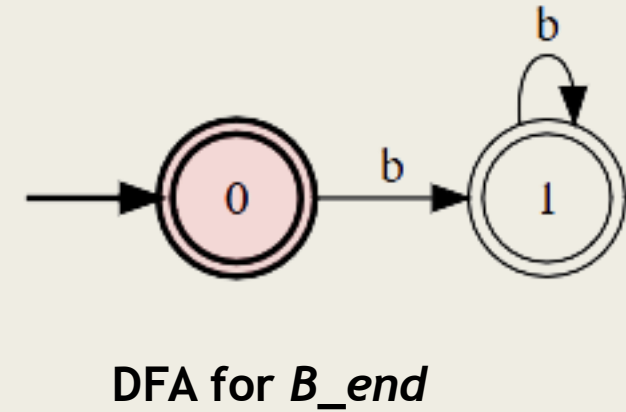
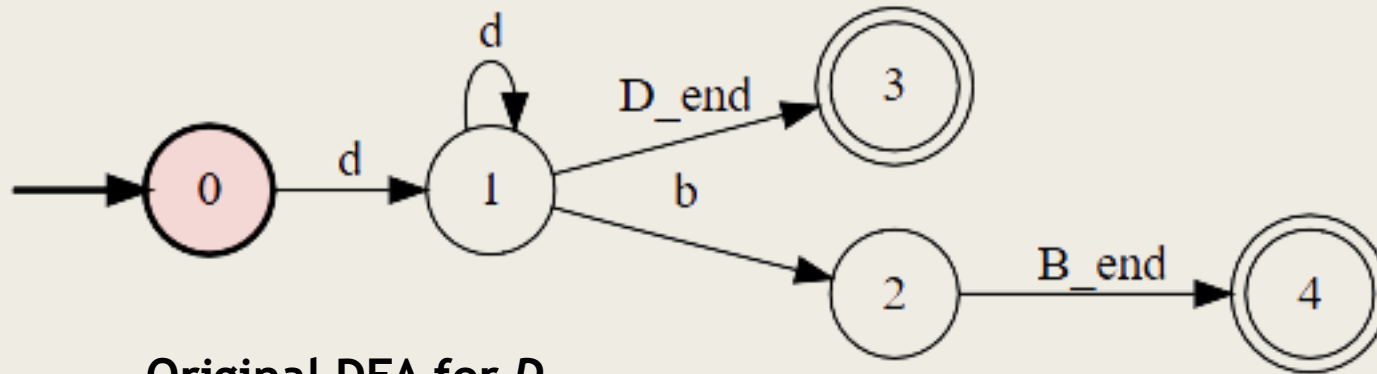


```
bbbb  
dddd  
bdbdbd  
ddbdbbbbdb  
other letters  
bdbd
```

- Reasons for errors:

- *Conflicts*
- *End-of-line characters (\$)*
- *Parser like behavior*
- *Lost possibility to handle nested structures like comments*

Conflict resolution



Results & Evaluation

- Reasons for errors:
 - *Conflicts and how these are solved*
 - *End-of-line characters (\$)*
 - *Parser like behavior*
 - *Lost possibility to handle nested structures like comments*

Results & Evaluation

```
begin
  declare
    i2d : string, nat : natural,
    noHighlightingOnTheNextLine : nil-type
  ;

  %disfunctional multiline
  comment%
  %If is seen as identifier and therefore not highlighted%
  %Once it picks up on colouring the rest of the line colours quite okay%
  %If this was divided over multiple lines it would fail because of $-problem%

  if id then nat := 42 else nat := 24 fi; id := "some %% with comment token literal string";

  %%fails because the rule for line-comment is lower than multiline comment%
  %so its never matched and always opens a multiline comment%

  id := "should not be comment coloured which is good, but string coloured, which fails"
end
```

Results & Evaluation

- Reasons for errors:

- *Conflicts and how these are solved*
- *End-of-line characters (\$)*
- *Parser like behavior*
- *Lost possibility to handle nested structures like comments*

Conclusion

- Promising at first
- Severity of errors
- Wrong approach
- Suggestions for future work
 - *Different information embedding*
 - *Retain less of the structure*