



KTH ROYAL INSTITUTE
OF TECHNOLOGY

Miking Tutorial

Making Your Own DSL

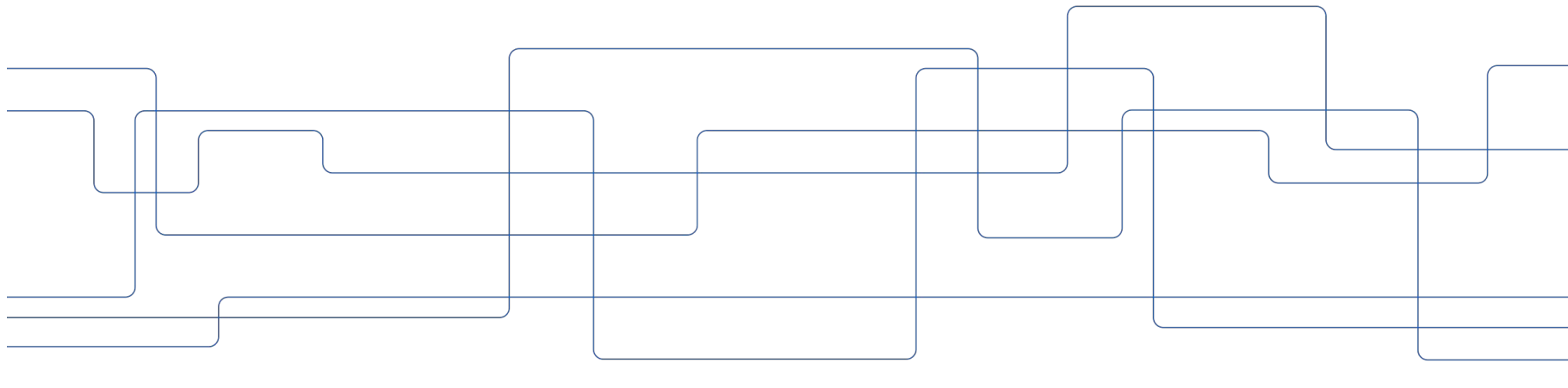


digital futures



Financially supported
by the Swedish Foundation
for Strategic Research.

**Vetenskapsrådet
(VR)**





Tutorial Overview

The components of a language

type, prod, precedence, ...

Syntax for `calc` (code-along)

writing a `.syn`, parsing

Ordinary differential equations

a recap

Tasks

making a DSL for ODEs



Tokens

Named token kinds

Ex: UIdent, Integer, String

Literals (keywords, symbols, etc.)

Ex: "let", "(", "-"

Manually implemented in MCore

token UName {...}

Already implemented and available
in tutorial:

LIdent, UIdent, Float,
Integer, String



Types

Syntactic types (“non-terminals”)

Example

Generates language fragments

```
Expr, Stmt, Type, Pat
```

```
type Expr {  
  grouping = "(" " )",  
}
```

```
lang ExampleBase =  
  syn Expr =  
    ...  
end
```



Productions (Essentials)

Semantic units of your language

If, Plus, Int

Example

```
prod If: Expr =  
  "if" c:Expr "then" t:Expr  
  ("else" e:Expr)? "end"
```

Generates a language fragment

```
syn Expr =  
  | IfExpr  
    {info:Info, c:Expr,  
      t:Expr, e:Option Expr}
```



Productions (Good to Know, pt 1)

Regex-like

|, ?, +, *, (), empty

Automatic AST based on types

field_name:T

```
type T      -> T
token T     -> {v:T,i:Info}
literal "." -> Info
```

...and possible appearances

```
0-1      -> Option T
1        -> T
2+       -> [T]
```



Productions (Good to Know, pt 2)

Nested structure with records

```
field:{inner:Expr ...}
```

Example

Resulting record

```
{ info:Info
, f:Expr
, args:
  [{v:Name,t:Type}]
}
```

```
prod FnDecl: Expr = f:LIdent "("
  (args:{v:LName ":" t:Type}
  ("," args:{v:LName ":" t:Type}
  )*)
)?
")" ";"
```



Precedence et al. (Essentials)

Syntactic sugar

```
prod  Add: Expr = left:Expr "+" right:Expr
```

```
infix Add: Expr = "+"
```

Associativity

```
infix left Add: Expr = "+"
```

Precedence

```
precedence {  
  Add Sub;  
  Equal NotEqual;  
}
```


Precedence et al. (Good to Know)

```
precedence {  
  Add Sub;  
  Equal NotEqual;  
} except {  
  Equal ? NotEqual;  
}
```

```
precedence {  
  Add Sub;  
  ~Equal NotEqual;  
}
```

$a + b == c$

Unambiguous

$(a + b) == c$

$a == b != c$

Ambiguous

$(a == b) != c$

$a == (b != c)$

Live-coding!

