Spoofax:
A Development Environment for Software Transformations

Karl Trygve Kalleberg
University of Bergen

UNIVERSITETET I BERGEN

Prerequisites

- Rudimentary programming skills
- Some familiarity with non-linear storytelling
- Basic knowledge of compilers is a plus

Act I

– *wherein introductions are made*

What is Spoofax?

- An interactive development environment
  - supports development of program transformations
  - based on the Stratego transformation language
- A set of plugins for Eclipse
  - editor – for writing Stratego programs
  - help system – for reading the Stratego manual (!)
  - interpreter – for executing Stratego programs
  - parser – for parsing any language
- An extensible system
  - Stratego scripts may extend the environment

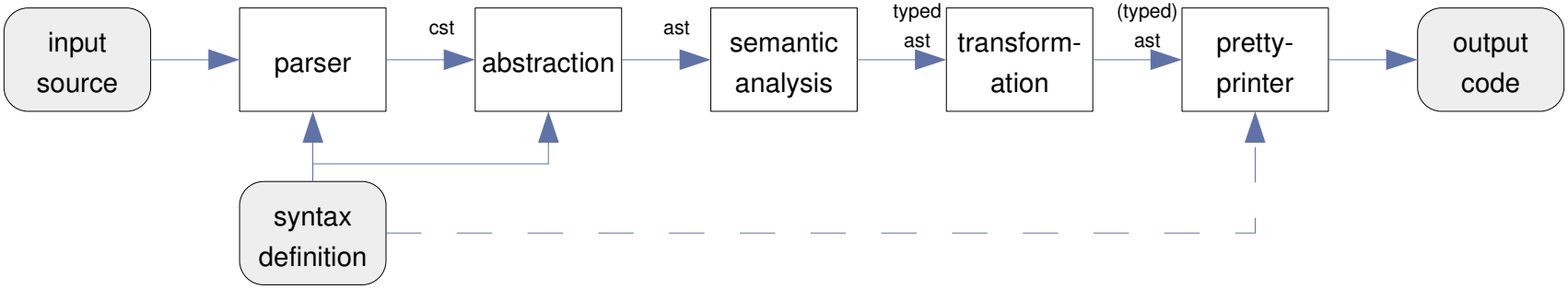
What is Eclipse?

- A framework for application development
  - based on Java
  - open-source
- A development platform
  - for Java, C/C++, Python, PHP, ...
- A pluggable framework
  - everything is a plugin
    - (including the plugin loader)
- Heavily sponsered
  - BEA, Borland, IBM, Oracle, Nokia, Sybase, Intel, Motorola, ...

What is program transformation?

- *"The act of changing one program into another"*
  - using *transformation programs*
  - interactively or (fully) automatically
  - often organised as a compiler pipeline
- Useful for
  - language implementation
    - compilers, interpreters, extensible languages, embedded languages
  - program analysis
    - defect checking, code smell sniffing, security analysis, program validation
  - program transformation
    - refactoring, reengineering, language translation, optimisation
  - code generation
    - from UML models, from specifications, in software factories

A transformation pipeline



Why abstraction?

```
x := 1 + 2
```

⇓ *Concrete Syntax Tree*

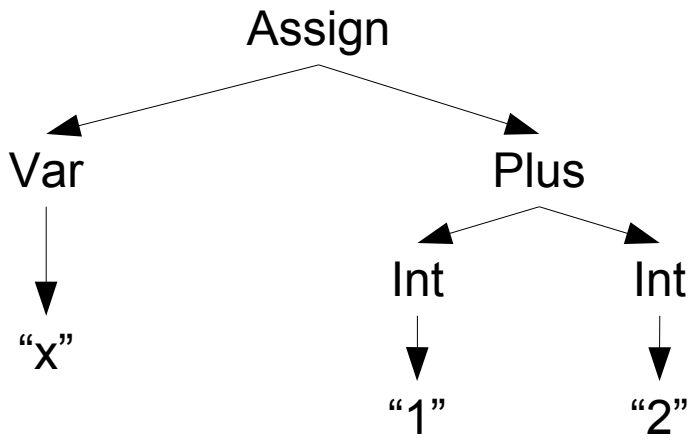
```
parsetree(appl(prod([cf(opt(layout)),cf(sort("Program")),cf(opt(layout))],sort("<START>"),no-at
trs),[appl(prod([],cf(opt(layout)),no-attrs),[]),appl(prod([cf(iter-star(sort("Stat")))],cf(sor
t("Program")),attrs([term(cons("Program"))])),[appl(list(cf(iter-star(sort("Stat")))),[appl(pro
d([lit("var"),cf(opt(layout)),cf(sort("Id")),cf(opt(layout)),lit(";")],cf(sort("Stat")),attrs([
term(cons("Declaration"))])),[lit("var"),appl(prod([cf(layout)],cf(opt(layout)),no-attrs),[32])
,appl(prod([lex(sort("Id"))],cf(sort("Id")),no-attrs),[appl(list(iter-star(char-class([range(0,
255)]))),[120])]),appl(prod([],cf(opt(layout)),no-attrs),[]),lit(";")]),appl(prod([cf(layout)],
cf(opt(layout)),no-attrs),[10]),appl(prod([cf(sort("Id")),cf(opt(layout)),lit(":="),cf(opt(layo
ut)),cf(sort("Exp")),cf(opt(layout)),lit(";")],cf(sort("Stat")),attrs([term(cons("Assign"))])),
[appl(prod([lex(sort("Id"))],cf(sort("Id")),no-attrs),[appl(list(iter-star(char-class([range(0,
255)]))),[120])]),appl(prod([cf(layout)],cf(opt(layout)),no-attrs),[32]),lit(":="),appl(prod([c
f(layout)],cf(opt(layout)),no-attrs),[32]),appl(prod([cf(sort("Exp")),cf(opt(layout)),lit("+"),
cf(opt(layout)),cf(sort("Exp"))],cf(sort("Exp")),attrs([term(cons("Add")),assoc(assoc)])),[appl
(prod([cf(sort("Int"))],cf(sort("Exp")),attrs([term(cons("Int"))])),[appl(prod([lex(sort("Int")
)],cf(sort("Int")),no-attrs),[appl(list(iter-star(char-class([range(0,255)]))),[49])])]),appl(p
rod([cf(layout)],cf(opt(layout)),no-attrs),[32]),lit("+"),appl(prod([cf(layout)],cf(opt(layout)
),no-attrs),[32]),appl(prod([cf(sort("Int"))],cf(sort("Exp")),attrs([term(cons("Int"))])),[appl
(prod([lex(sort("Int"))],cf(sort("Int")),no-attrs),[appl(list(iter-star(char-class([range(0,255
)]))),[50])])])]),appl(prod([],cf(opt(layout)),no-attrs),[]),lit(";")])])]),appl(prod([cf(layou
t)],cf(opt(layout)),no-attrs),[10])]),0)
```

Why abstraction?

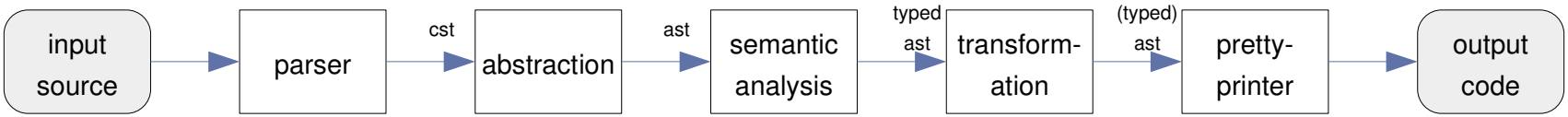
```
x := 1 + 2
```

⇓ *Abstract Syntax Tree*

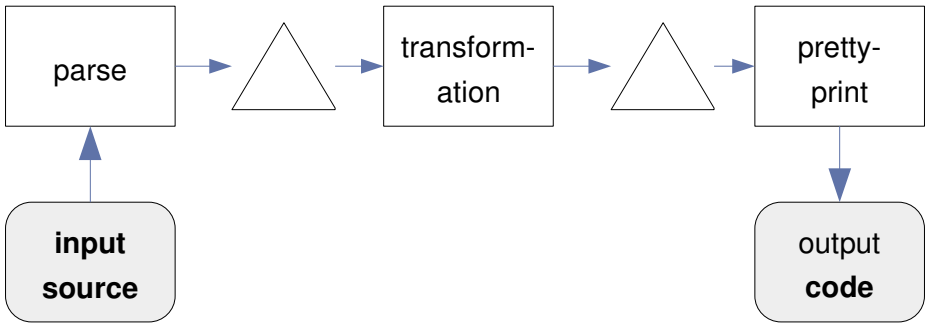
```
Assign(Var("x"), Plus(Int("1"),Int("2"))
```



A transformation pipeline



=



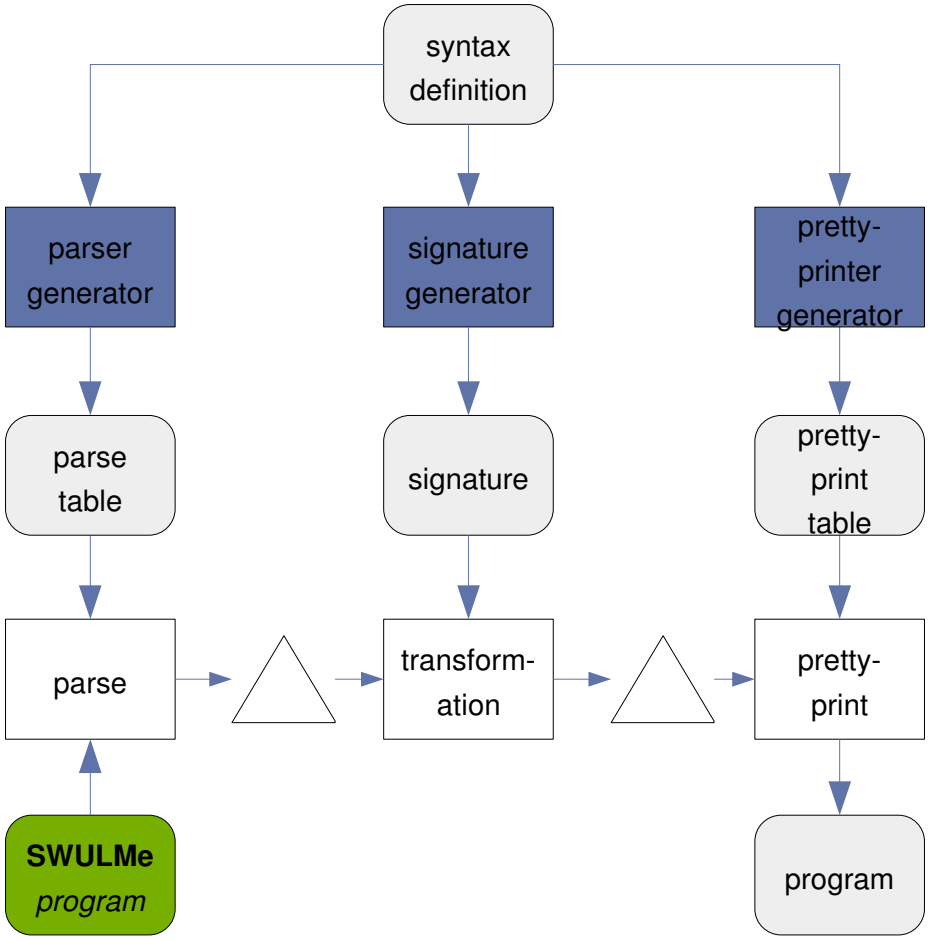
Interlude

– an example application

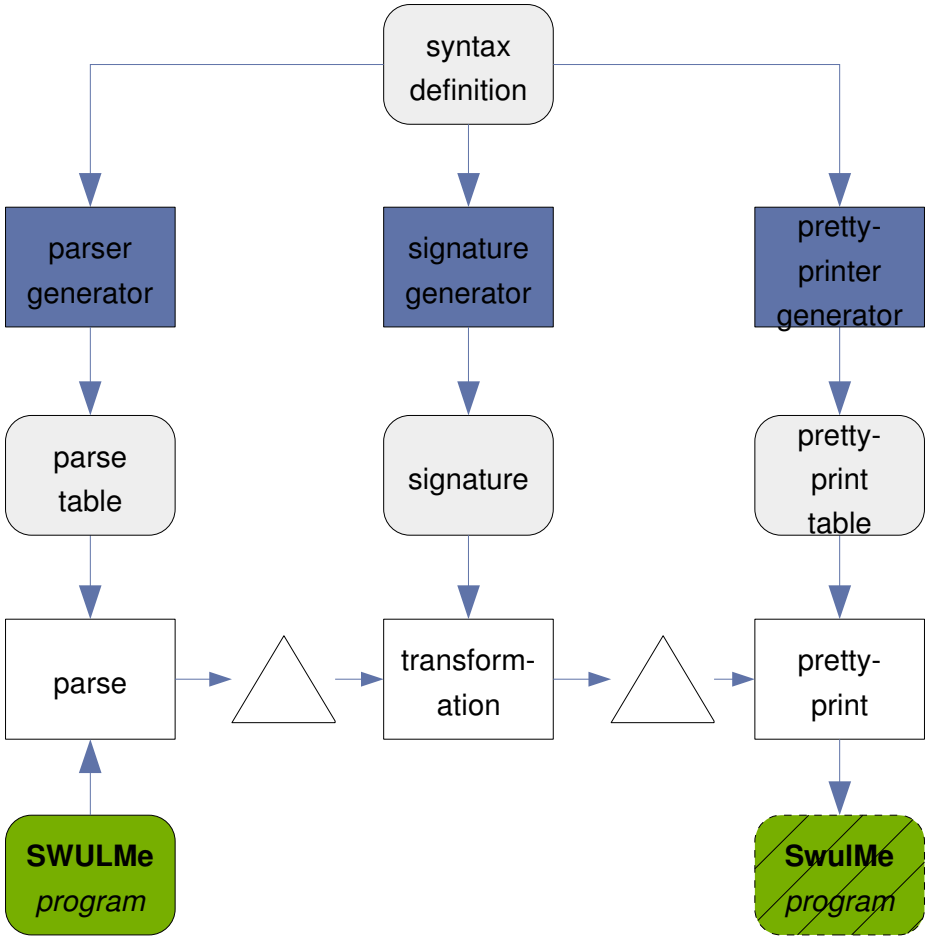
Example language extension: JSwul

```
class SWULMe {
  public static void main(String[] args) {
    JFrame frame = frame {
      title = "Welcome!"
      content = panel of border layout {
        center = label { text = "Hello World" }
        south = panel of grid layout {
          row = {
            button { text = "cancel" }
            button { text = "ok" }
          }
        }
      }
    }
  }
}
```

Step 0: What we have



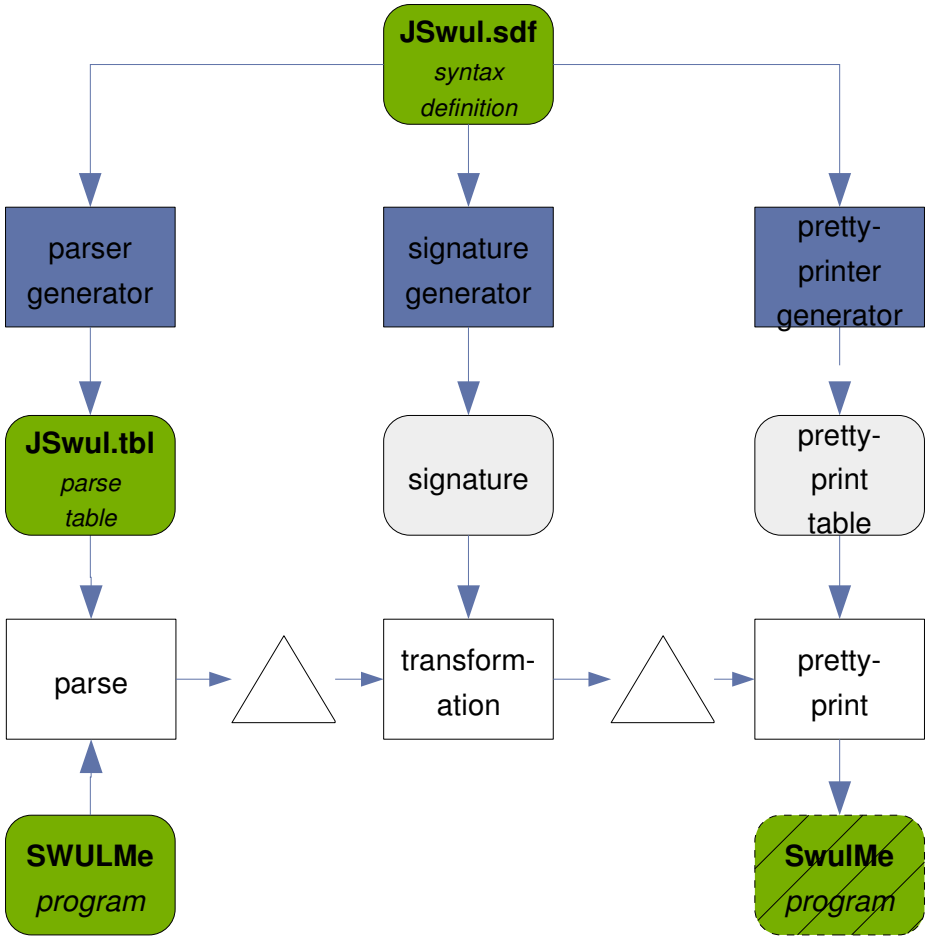
Step 1: What we want



A pure Java program

```
class SWULMe {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Welcome!");
    JPanel mainPanel = new JPanel();
    BorderLayout borderLayout = new BorderLayout();
    JLabel helloLabel = new JLabel("Hello World");
    JPanel southPanel = new JPanel();
    JButton cancelButton = new JButton("cancel");
    JButton okButton = new JButton("ok");

    southPanel.setLayout(new GridLayout(1,2));
    southPanel.add(cancelButton);
    southPanel.add(okButton);

    mainPanel.setLayout(borderLayout);
    mainPanel.add(southPanel, BorderLayout.SOUTH);
    mainPanel.add(helloLabel, BorderLayout.CENTER);

    frame.setContentPane(mainPanel);
  }
}
```

Step 2: We need a JSwul grammar



SWUL grammar definition

```
module Swul
exports context-free start-symbols Component

sorts Component ComponentType ComponentProps ComponentPropValues ComponentPropType
context-free syntax

  ComponentType ComponentProps? -> Component

  "panel"                  -> ComponentType
  "button"                 -> ComponentType
  "border" "layout"        -> ComponentType
  "grid" "layout"          -> ComponentType
  "frame"                  -> ComponentType

  "{" ComponentProp* "}"   -> ComponentProps
  "of" Component           -> ComponentProps

  ComponentPropType "=" ComponentPropValues -> ComponentProp
  "{" Component* "}"                        -> ComponentPropValues

  "content"                -> ComponentPropType
  "title"                  -> ComponentPropType
  "row"                    -> ComponentPropType
  "south"                  -> ComponentPropType
  "center"                 -> ComponentPropType
  "border"                 -> ComponentPropType
```

Java + SWUL = JSwul

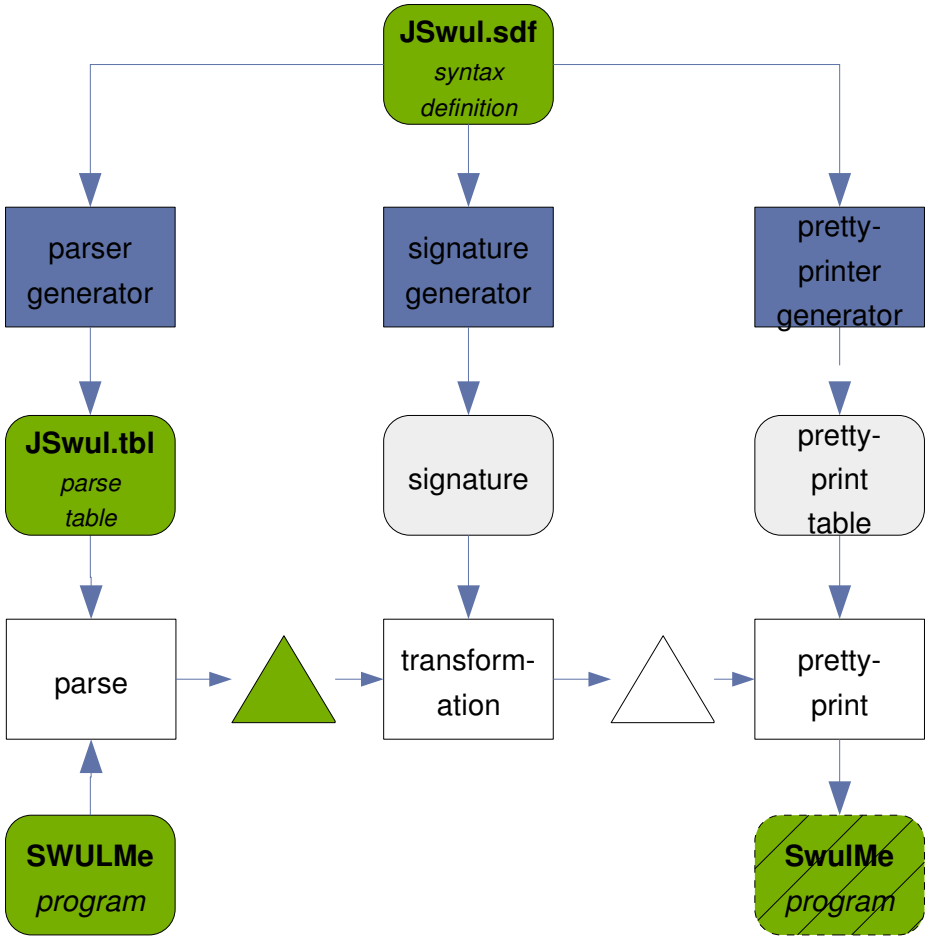
```
module JSwul
imports Java-15-Prefixed Swul-Prefixed

exports
  sorts JavaExpr SwulComponent JavaBlock
  context-free syntax

    SwulComponent -> JavaExpr
```

Java + SWUL = JSwul

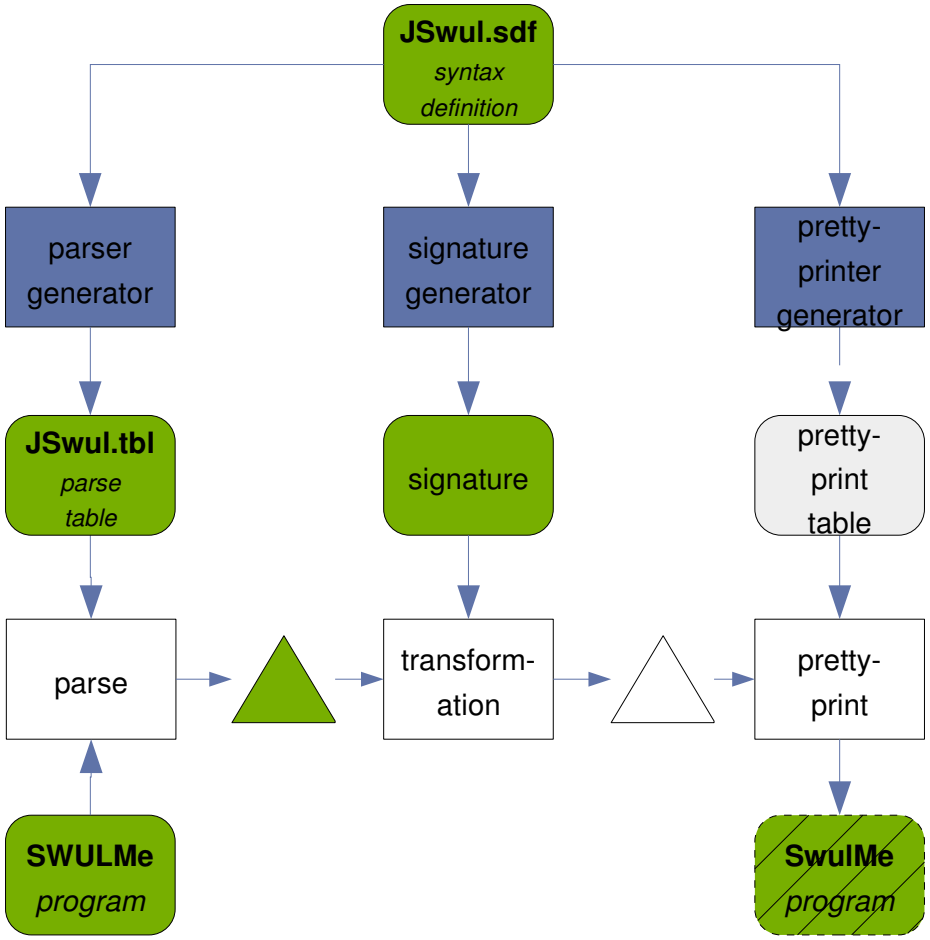
```
module JSwul
imports Java-15-Prefixed Swul-Prefixed

exports
  sorts JavaExpr SwulComponent JavaBlock
  context-free syntax

    SwulComponent -> JavaExpr
```

```
    JavaExpr       -> SwulComponent
    JavaBlock      -> SwulComponent
```

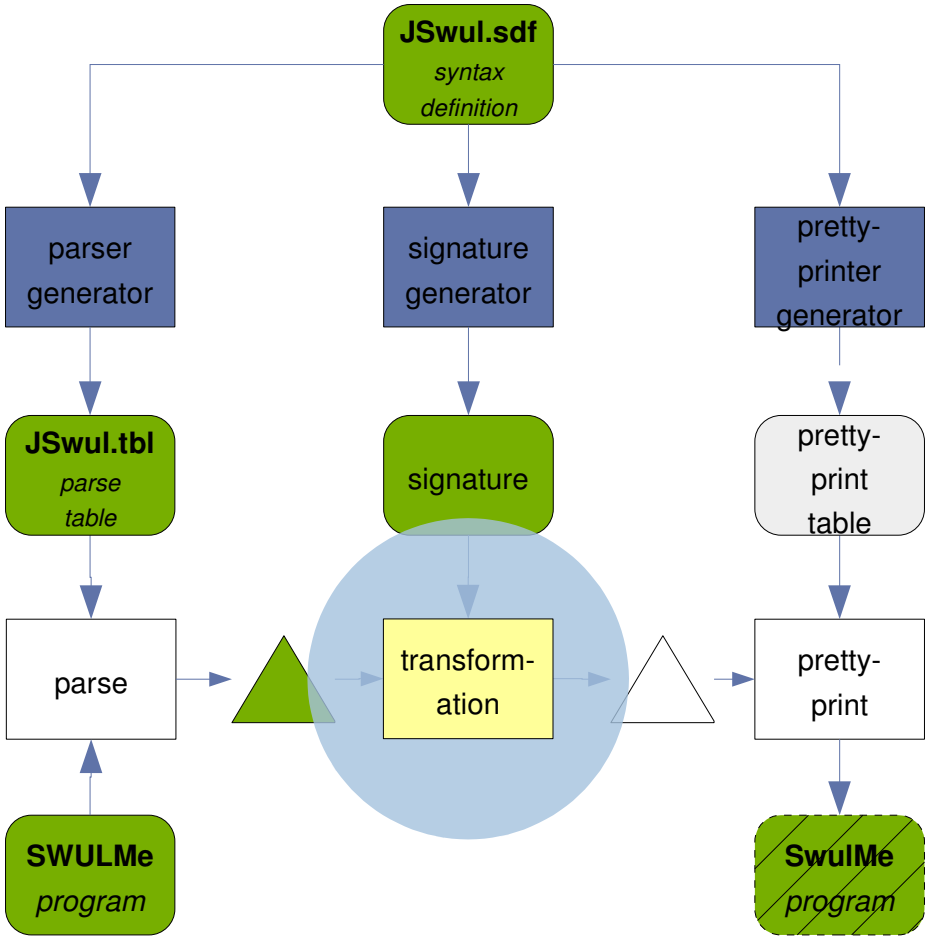
Step 3: Parsing the SWUL language



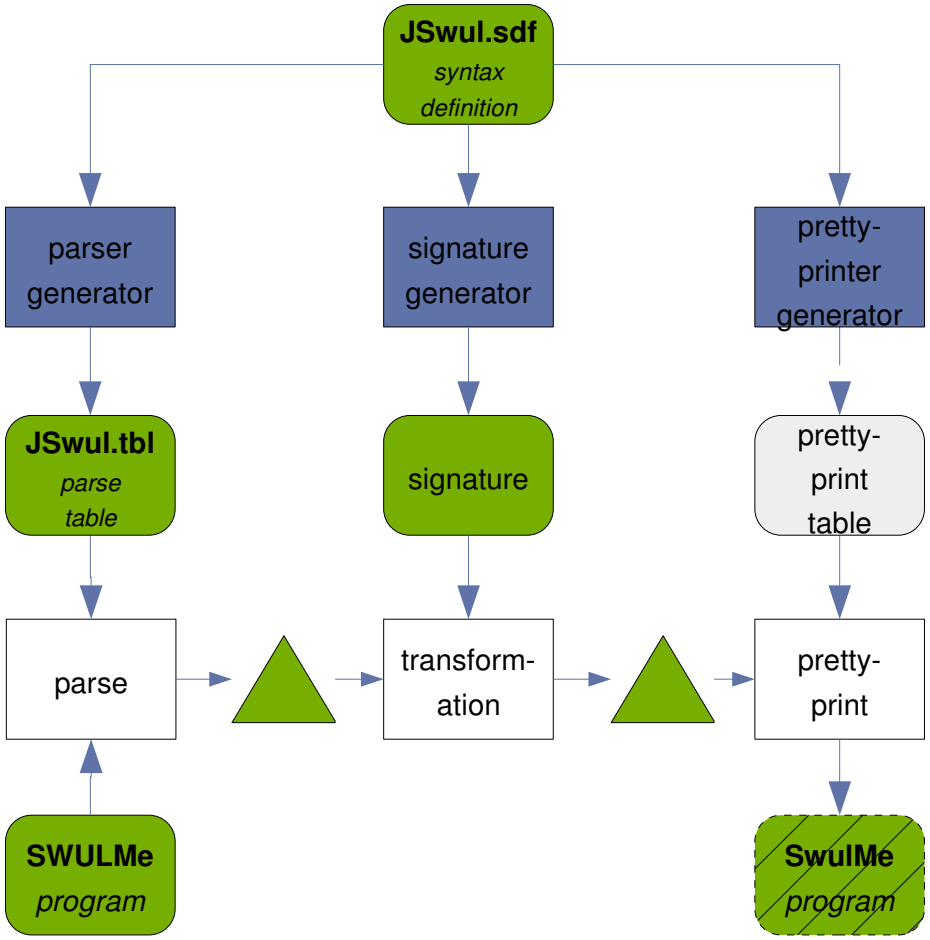
Step 4: Defining JSwul to Stratego



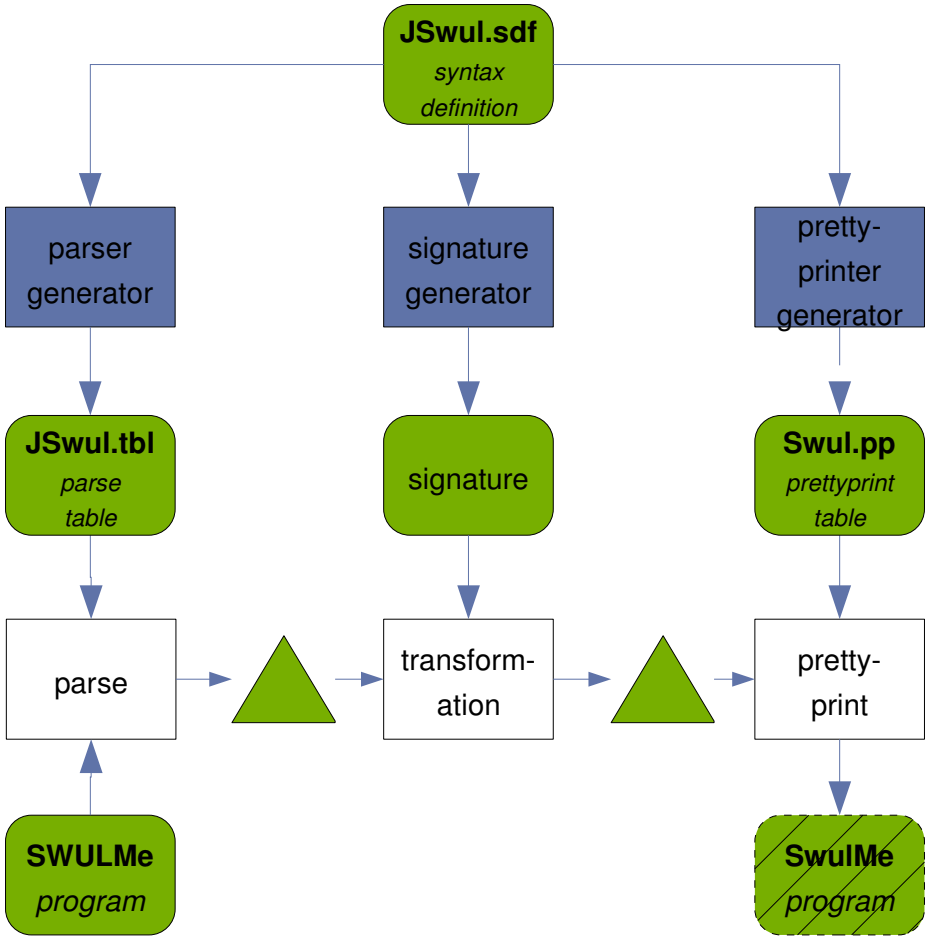
Step 5: Transformation!



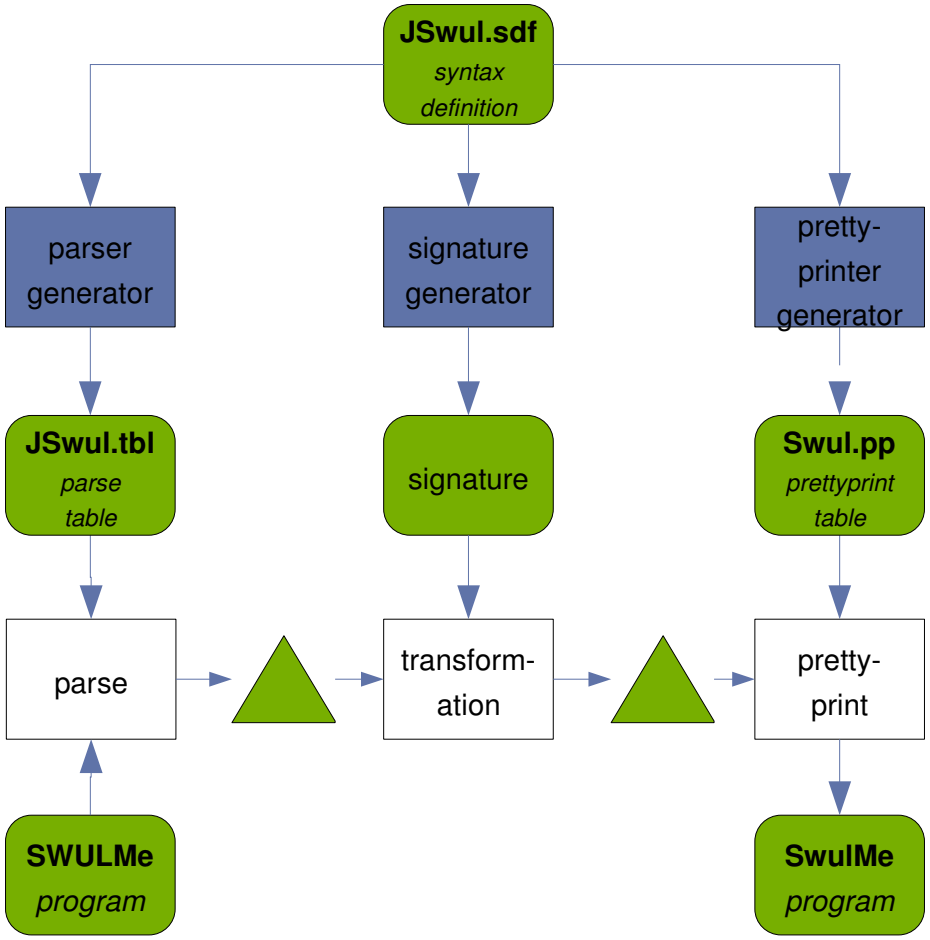
Step 6: Transformation!



Step 7: Deriving a pretty printer



Step 8: Are we there yet? – Yes!



```
class SWULMe {
  public static void main(String[] args) {
    JFrame frame = frame {
      title = "Welcome!"
      content = panel of border layout {
        center = label { text = "Hello World" }
        south = panel of grid layout {
          row = {
            button { text = "cancel" }
            button { text = "ok" }}}}}}}
```

program transformation

```
class SWULMe {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Welcome!");
    JPanel mainPanel = new JPanel();
    BorderLayout borderLayout = new BorderLayout();
    JLabel helloLabel = new JLabel("Hello World");
    JPanel southPanel = new JPanel();
    JButton cancelButton = new JButton("cancel");
    JButton okButton = new JButton("ok");

    southPanel.setLayout(new GridLayout(1,2));
    southPanel.add(cancelButton);
    southPanel.add(okButton);

    mainPanel.setLayout(borderLayout);
    mainPanel.add(southPanel, BorderLayout.SOUTH);
    mainPanel.add(helloLabel, BorderLayout.CENTER);

    frame.setContentPane(mainPanel);
  }
}
```

compilation

Meanwhilst, back at the ranch...

What is Stratego?

- A scripting language for program transformation
  - based on term rewriting...
    - express small, incremental changes to the AST
  - .. and strategic programming
    - separate data processing from data traversal
- Provides powerful domain-specific features
  - rewrite rules
  - application strategies
  - concrete syntax patterns
  - XT – a collection of reusable transformation components

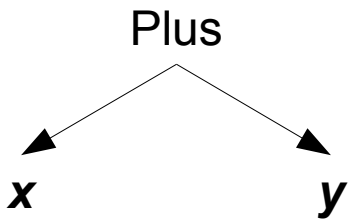
Rewrite rules

```
rules

  Simplify:
    If(Boolean(True), th, ls) -> th

  Simplify:
    Plus(e, Int("0")) -> e

  Simplify:
    Plus(Int(x), Int(y)) -> Int(z)
  where <addS> (x, y) => z

  Simplify:
    Plus(Int(x), Int(y)) -> Int(<addS> (x, y))
```

Matching of patterns

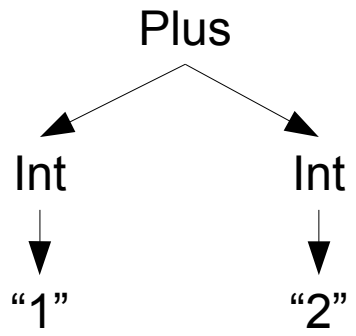
- Pattern

```
?Plus(x,y)
```



- Tree (term)

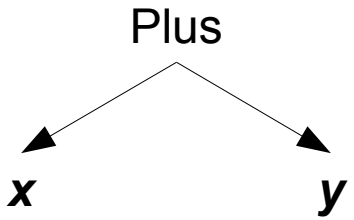
```
Plus(Int("1"),Int("2"))
```



Matching of patterns

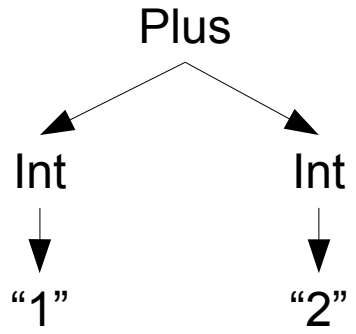
- Pattern

```
?Plus(x,y)
```



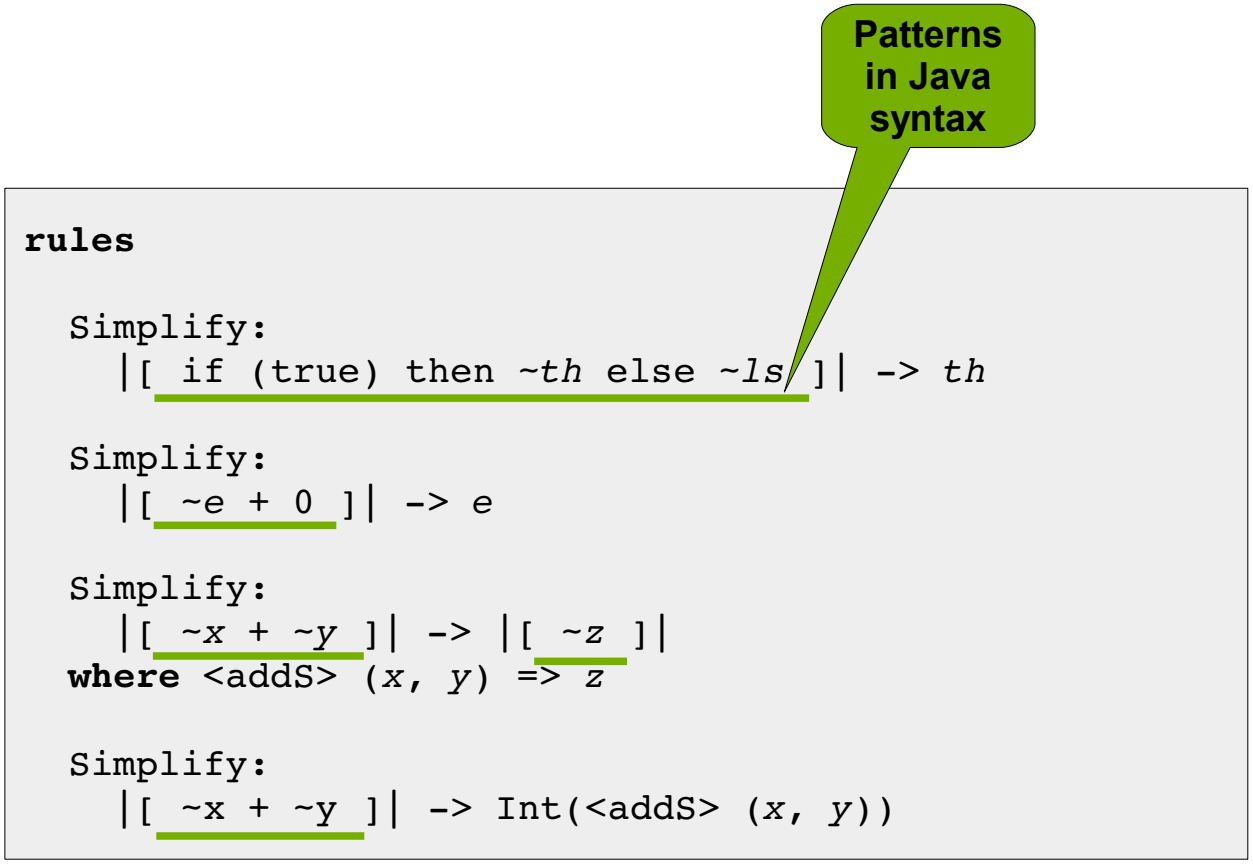
- Tree (term)

```
Plus(Int("1"),Int("2"))
```




```
x = Int("1"), y = Int("2")
```

Rewrite rules with concrete syntax



Application Strategies

- Apply rule to tree (term)
  - `<R> t`
- Left choice
  - `s1 <+ s2`
- Guarded choice
  - `s1 < s2 + s3`
- Sequence
  - `s1 ; s2`
- Subtree(term) traversal
  - `all(s)` – all subterms
  - `one(s)` – one subterm

- Generic traverals

```
bottomup(s) = all(bottomup(s)); s
topdown(s)  = s; all(topdown(s))
downup(s)   = s; all(downup(s)); s
alltd(s)    = s <+ all(alltd(s))
```

Assimilation of SWUL

```
swul-assimilate =
     class-declaration
  <+ class-initializer
  <+ class-method
  <+ swul-expression
  <+ all(swul-assimilate)
```

Assimilation of SWUL

```
swul-assimilate =
     class-declaration
  <+ class-initializer
  <+ class-method
  <+ swul-expression
  <+ all(swul-assimilate)
```

```
swul-expression = ... ; SwulAs-Component ... ; ...
```

Assimilation of SWUL

```
swul-assimilate =
     class-declaration
  <+ class-initializer
  <+ class-method
  <+ swul-expression
  <+ all(swul-assimilate)
```

```
swul-expression = ... ; SwulAs-Component ... ; ...
```

```
SwulAs-Component   = SwulAs-Container <+ SwulAs-JavaExpr
SwulAs-Container   = SwulAs-JComponent
SwulAs-JComponent = SwulAs-JPanel <+ SwulAs-JToolBar <+ SwulAs-JSplitPane <+ ..

SwulAs-JPanel:
  swul c |[ panel { ps* } ]|{x} -> expr |[ {| x = new JPanel(); bstm* | x |} ]|
  where <map(SwulAs-JPanelProp(|x)> ps* => bstm*

SwulAs-JPanelProp(|x) = ... <+ SwulAs-ContainerProp(|x) <+ ...

SwulAs-ContainerProp(|x) :
  swul cp |[ layout = c ]| -> bstm |[  x.setLayout( e ) ; ]|
  where <SwulAs-LayoutManager(|x)> c => e
```

Act II

– *the showdown*

Development aids for Stratego

- Syntax highlighting
  - SDF and Stratego
- Content completion
  - rules, strategies, constructors, modules
- Navigation
  - to rule, strategy, constructor, module definitions
- Project building
  - incremental, automatic
- Code outline
  - definitions in module
- Help
  - bundled Stratego/XT manual
- Tasks
  - FIXME, TODO
- Extensible
  - Stratego scripts

Demo

Epilogue

Conclusion

- Program transformation techniques are widely applicable
  - software processing
  - language development
  - program analysis
- Program transformation has never been less difficult!
  - Spoofax + Stratego/XT = transformation development platform
  - `www.spoofax.org`
  - `www.strategoxt.org`
    - Works on and for Linux, Unix, OSX and Windows
    - Licensed under the LGPL
    - Tutorials, examples and API docs online

The End.

SWUL grammar definition (with cons)

```
module Swul
exports context-free start-symbols Component

sorts Component ComponentType ComponentProps ComponentPropValues ComponentPropType
context-free syntax

  ComponentType ComponentProps? -> Component {cons("Component")}

  "panel"                 -> ComponentType {cons("JPanel")}
  "button"                -> ComponentType {cons("JButton")}
  "border" "layout"       -> ComponentType {cons("BorderLayout")}
  "grid" "layout"         -> ComponentType {cons("GridLayout")}
  "frame"                 -> ComponentType {cons("JFrame")}

  "{" ComponentProp* "}"  -> ComponentProps {cons("ComponentProps")}
  "of" Component          -> ComponentProps {cons("DefaultPropsWithComponent")}

  ComponentPropType "=" ComponentPropValues -> ComponentProp {cons("ComponentProp")}
  "{" Component* "}"              -> ComponentPropValues {cons("ComponentPropMultiValue")}

  "content"               -> ComponentPropType {cons("Content")}
  "title"                 -> ComponentPropType {cons("Title")}
  "row"                   -> ComponentPropType {cons("Row")}
  "south"                 -> ComponentPropType {cons("South")}
  "center"                -> ComponentPropType {cons("Center")}
  "border"                -> ComponentPropType {cons("Border")}
```

Spoofax Architecture