UNIKASSEL
VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

Software Tool Construction

Wintersemester 2023/24

Adrian Kunz

Vorlesung 1

Organisatorisches

- 3 Meilensteine
  - je 2-3 Wochen
  - Prüfungsform TBD (*)
- 1 Abschlussprojekt
  - 2-3 Wochen
- Mündliche Prüfung (*)
  - 10 min Sprache vorstellen
  - 5-10 min Fragen zur Veranstaltung
- Note ergibt sich aus Punkten aus Meilensteinen und Prüfung

(*) Abhängig von Teilnehmeranzahl

- Aufnahmen auf YouTube
- Folien auf dem Blog
- Fragen und Ankündigungen auf Discord
- Anmeldung bis Montag 06.11. 16:00 über Google Forms

Ablauf

- 1. Meilenstein: **Basics**
  - Nx Workspace
  - ANTLR Grammatik
  - Compiler in TypeScript
- 2. Meilenstein: **LSP**
  - Language Server
  - VSCode Extension
  - TextMate Syntax
- 3. Meilenstein: **Monaco**
  - Monarch Sprache
  - Monaco LSP Client
  - Angular Web App
- Evtl. Bonus: **Compiler Backend**
- Evtl. Bonus: **IntelliJ**
- Bonus: **MPS**
- **Projekt**

Projekt: Anforderungen

Generell

- Alle Features aus Meilensteinen
- *n* (*) weitere LSP Features
- *k* (**) weitere Sprach-Features
- Extension auf open-vsx.org publiziert und auf fulib.org/projects verwendbar
- Web App auf sekassel.github.io/<dsl> oder stc.uniks.de/<dsl> verfügbar

(*) Abhängig vom VL-Fortschritt

(**) Abhängig von Fortschritt nach 3. Meilenstein

Sprache

- Eigene DSL oder Programmiersprache
- Verschiedene Tokens
  - z.B. Zahlen, Strings, Bezeichner
- Referenzen
  - z.B. Variablen, Funktionen, Typen
- Validierung
  - z.B. Warnings, Errors, Typechecks
- Semantik
  - kompilier- oder ausführbar

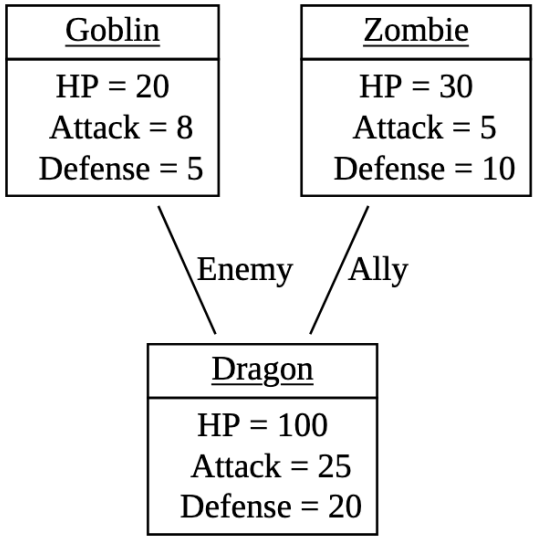
Grundlagen: DSLs

Notationen

```
{"monsters": [
  {"name": "Goblin", "hp": 20,
"attack": 8, "defense": 5},
  {"name": "Dragon", "hp":
100, "attack": 25, "defense":
20},
  {"name": "Zombie", "hp": 30,
"attack": 5, "defense": 10}
],
"relations": [
  {"source": "Goblin",
"target": "Dragon",
"relation": "enemy"},
  {"source": "Zombie",
"target": "Dragon",
"relation": "ally"}
]}
```

Name	HP	Attack	Defense
Goblin	20	8	5
Dragon	100	25	20
Zombie	30	5	10

Source	Target	Relation
Goblin	Dragon	Enemy
Zombie	Dragon	Ally




```
monster Goblin hp: 20 attack: 8 defense: 5
monster Dragon hp: 100 attack: 25 defense: 20
monster Zombie hp: 30 attack: 5 defense: 10

relation enemy between Goblin and Dragon
relation ally between Zombie and Dragon
```

DSLs

- Eigenständige DSLs (z.B. als Dateien)
  - LOOP, Prisma, Graphviz/Dot, YACC, ANTLR, Protobuf
- Macro-DSLs
  - Scala, Rust
- Eingebettete DSLs (eDSLs, auch: Interne DSLs)
  - jQuery, LINQ, React/JSX, numpy
- Laufzeit-DSLs
  - Format-Strings, RegExp
- DSLs für Benutzer von Anwendungen
  - Excel Formeln, Suchfeld-Syntax

Diomidis Spinellis: *Notable design patterns for domain specific languages* https://www.spinellis.gr/pubs/jrnl/2000-JSS-DSLPatterns/html/dslpat.html
Felleisen et al: *A Programmable Programming Language* https://cacm.acm.org/magazines/2018/3/225475-a-programmable-programming-language/fulltext

Eigenständige DSLs

Graphviz

```
digraph Monsters {
  node [shape=plaintext];

  Goblin [label=<
  <table border="0" cellborder="1" cellspacing="0">
    <tr><td port="name"><u>Goblin</u></td></tr>
    <tr><td>HP = 20<br/>Attack = 8<br/>Defense =
5</td></tr>
  </table>>];

  Dragon [label=<...>];

  Zombie [label=<...>];

  Goblin -> Dragon [label="Enemy",arrowhead=none];
  Zombie -> Dragon [label="Ally",arrowhead=none];
}
```

ANTLR [1]

```
grammar Expr;

prog: expr EOF ;
expr: expr ('*'|'/') expr
    | expr ('+'|'-') expr
    | INT
    | '(' expr ')' ;

NEWLINE : [\r\n]+ -> skip;
INT : [0-9]+ ;
```

[1]: https://github.com/antlr/antlr4/blob/master/doc/getting-started.md

Macro-DSLs

Scala [1]

```
def people[A](block: A): People = macro
Macros.impl[A]

// ...

people {
        introduce John please
        introduce Frank and Lilly please
}

// =>

new People {
        val john = new Person("John")
        val frank = new Person("Frank")
        val lilly = new Person("Lilly")
}
```

[1]: https://stackoverflow.com/a/29269065/4138801
[2]: https://doc.rust-lang.org/stable/rust-by-example/macros/dsl.html

Rust [2]

```
macro_rules! calculate {
  (eval $e:expr) => {
    {
      let val: usize = $e;
      println!("{} = {}",
               stringify!{$e}, val);
    }
  };
}

fn main() {
  calculate! {
    eval 1 + 2 // output: 1 + 2 = 3
  }
  calculate! {
    eval (1 + 2) * (3 / 4)
  }
}
```

Eingebettete DSLs

JQuery (JavaScript)

```
$(document).ready(function() {

  $('h1').css('color', 'blue');

  $('button').click(function() {
    alert('Button clicked!');
  });

  $('#box').animate({
    left: '250px',
    opacity: '0.5',
    height: '150px',
    width: '150px'
  }, 1000);
});
```

LINQ (C#) [1]

```
List<int> numbers = new(){ 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

IEnumerable<int> filteringQuery =
    from num in numbers
    where num < 3 || num > 7
    select num; // => 1, 9, 8, 2, 0

IEnumerable<int> orderingQuery =
    from num in numbers
    where num < 3 || num > 7
    orderby num ascending
    select num; // => 0, 1, 2, 8, 9

string[] groupingQuery = { "carrots", "cabbage",
"broccoli", "beans", "barley" };
IEnumerable<IGrouping<char, string>> queryFoodGroups =
    from item in groupingQuery
    group item by item[0];
// => {c: [carrots, cabbage], b: [broccoli, beans,
barley]}
```

[1]: https://learn.microsoft.com/en-us/dotnet/csharp/linq/write-linq-queries

Laufzeit-DSLs

Java Format Strings [1]

```
System.out.printf("%10.3f%n", pi);   // => "      3.142"

Calendar c = Calendar.getInstance();
System.out.printf("%tB %te, %tY%n", c, c, c); // => "May 29, 2006"
System.out.printf("%tl:%tM %tp%n", c, c, c);  // => "2:34 am"
```

JavaScript RegExp

```
const pattern = /([a-zA-Z0-9_]+)\([^)]*\)\s*\{/gi;
pattern.exec('void main() {}') // => ['void main() {', 'main']
```

[1]: https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Benutzer-DSLs

Excel Formeln

```
=IF(AC5;((SUM(I5:L5;N5:Q5;S5:V5;X5:AA5;AC5)-MAX(I5:L5;N5:Q5;S5:V5;X5:AA5;AC5))/(COUNT(I5:L5;N5:Q5;S5:V5;X5:AA5;AC5)-1));(SUM(I5:L5;N5:Q5;S5:V5;X5:AA5))/(COUNT(I5:L5;N5:Q5;S5:V5;X5:AA5)))
```

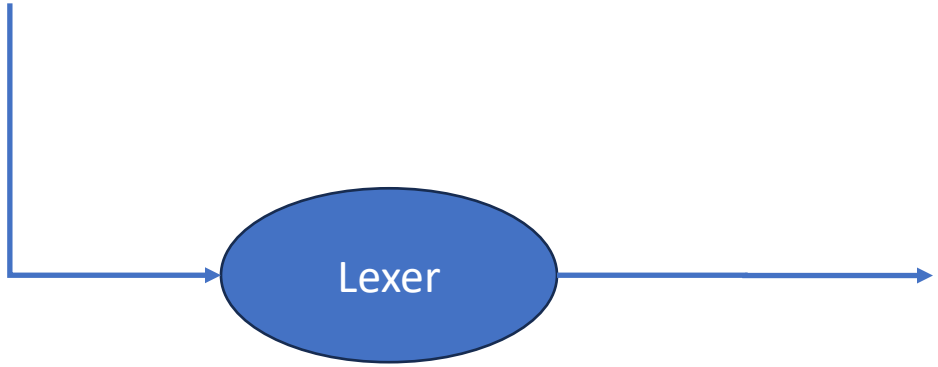
GitHub Suchanfragen

org:fujaba error (language:JavaScript OR language:Java) NOT path:test

Grundlagen: Compilerbau

Tokens

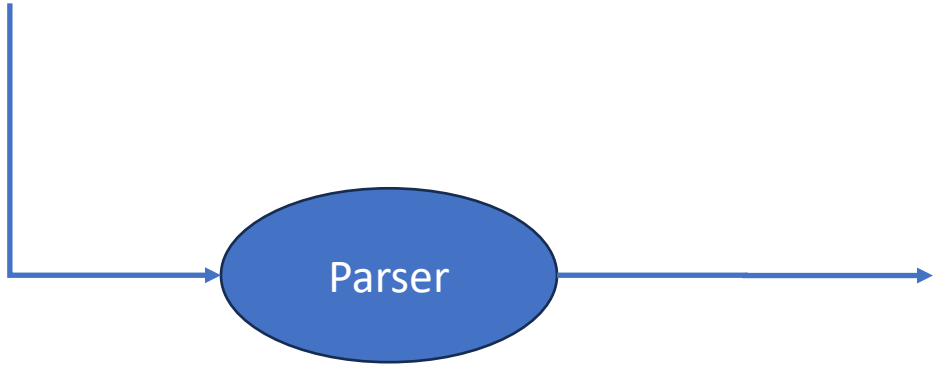
```
while (true) {
  print("Hello World");
}
```

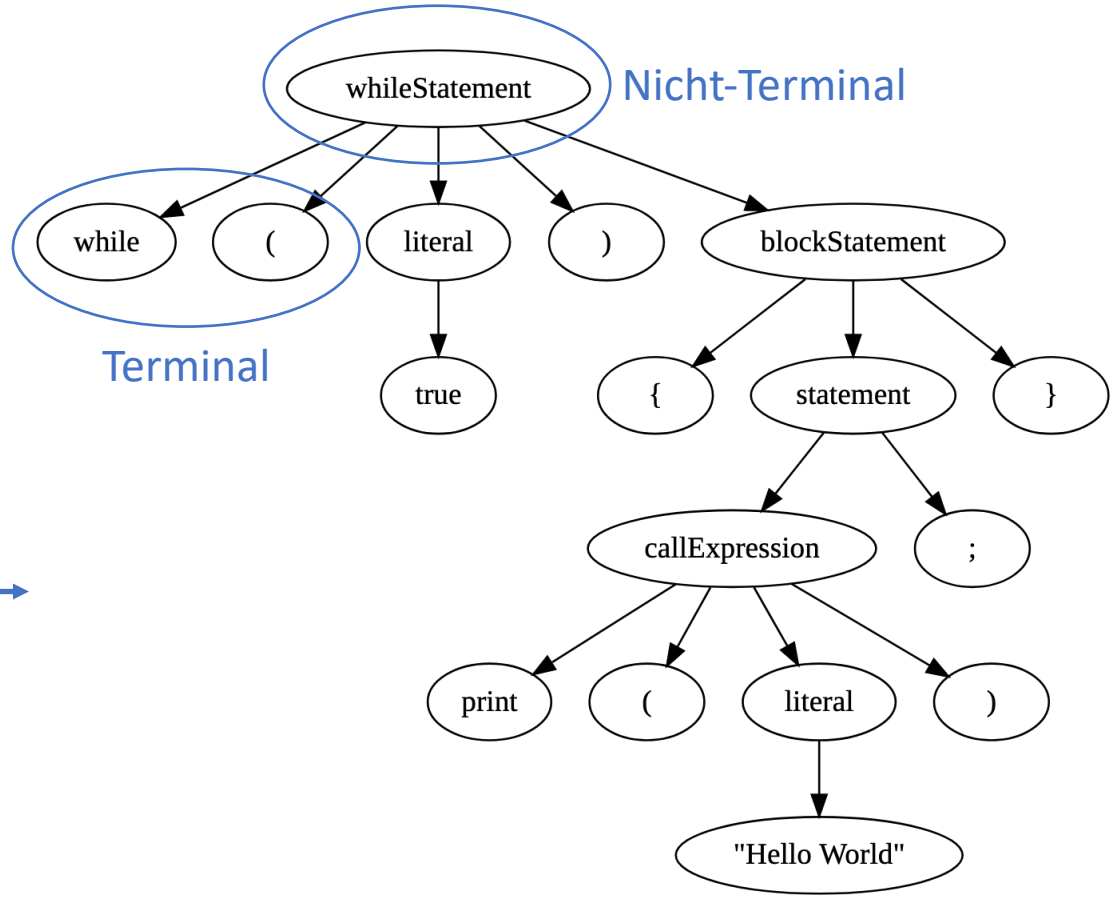


- `while` WHILE @1:1
- `(` OPEN_PAREN @1:7
- `true` TRUE @1:8
- `)` CLOSE_PAREN @1:12
- `{` OPEN_BRACE @1:14
- `print` ID @2:3
- `(` OPEN_PAREN @ …
- `"Hello World"` STRING
- `)` CLOSE_PAREN
- `;` SEMICOLON
- `}` CLOSE_BRACE

Konkreter Syntaxbaum (CST)

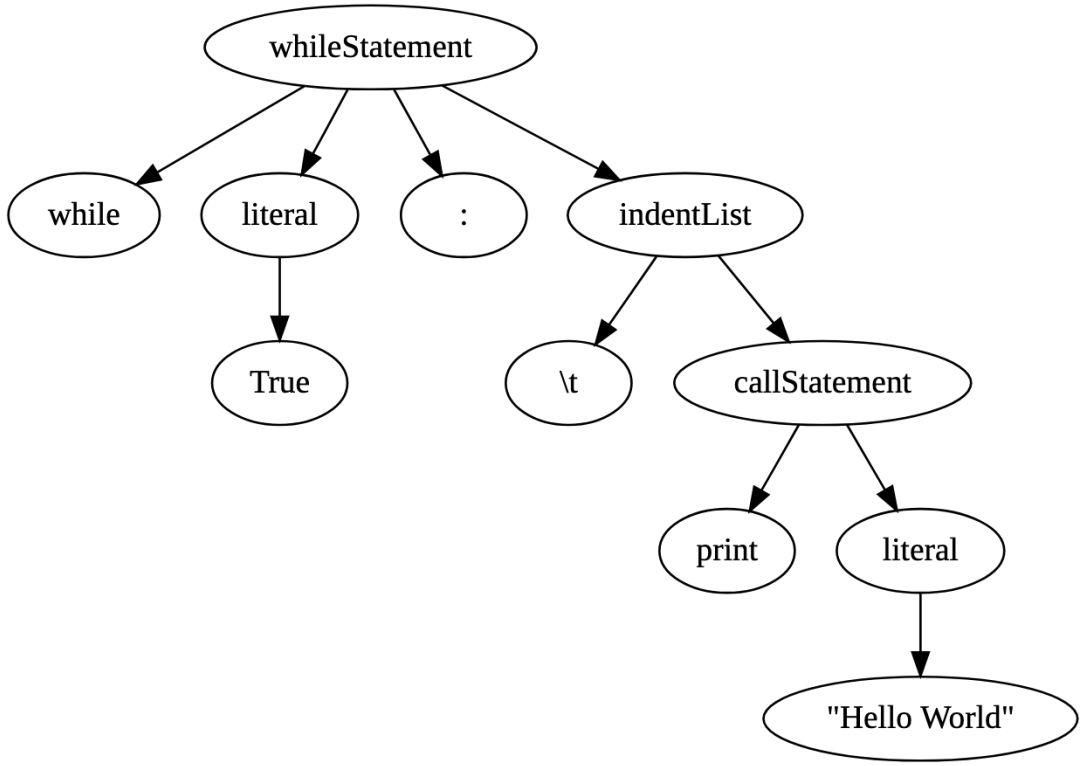
```
while ( true ) {
  print ( "Hello World" ) ;
}
```



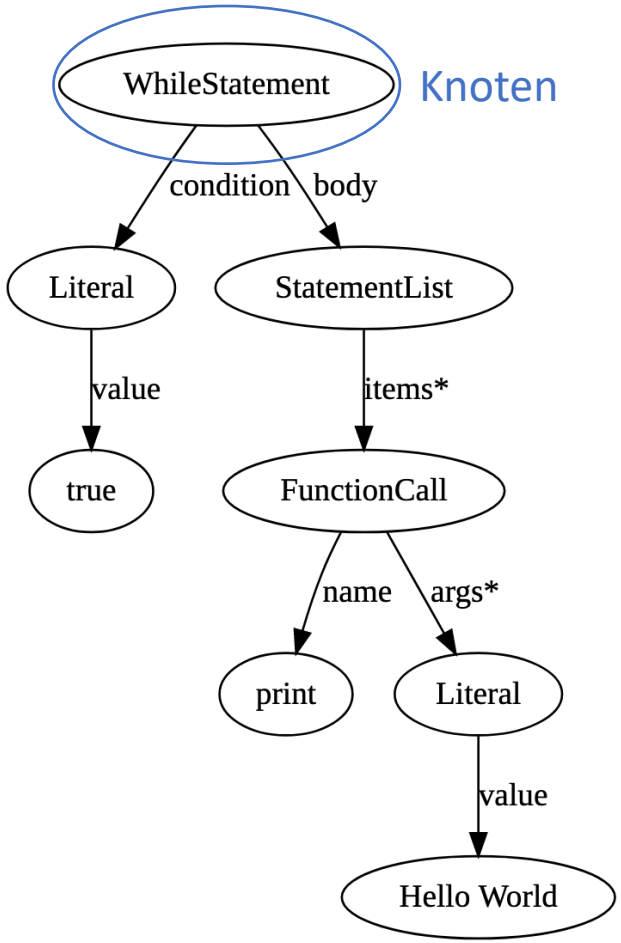


Ein anderer CST

```
while True:
    print "Hello World"
```



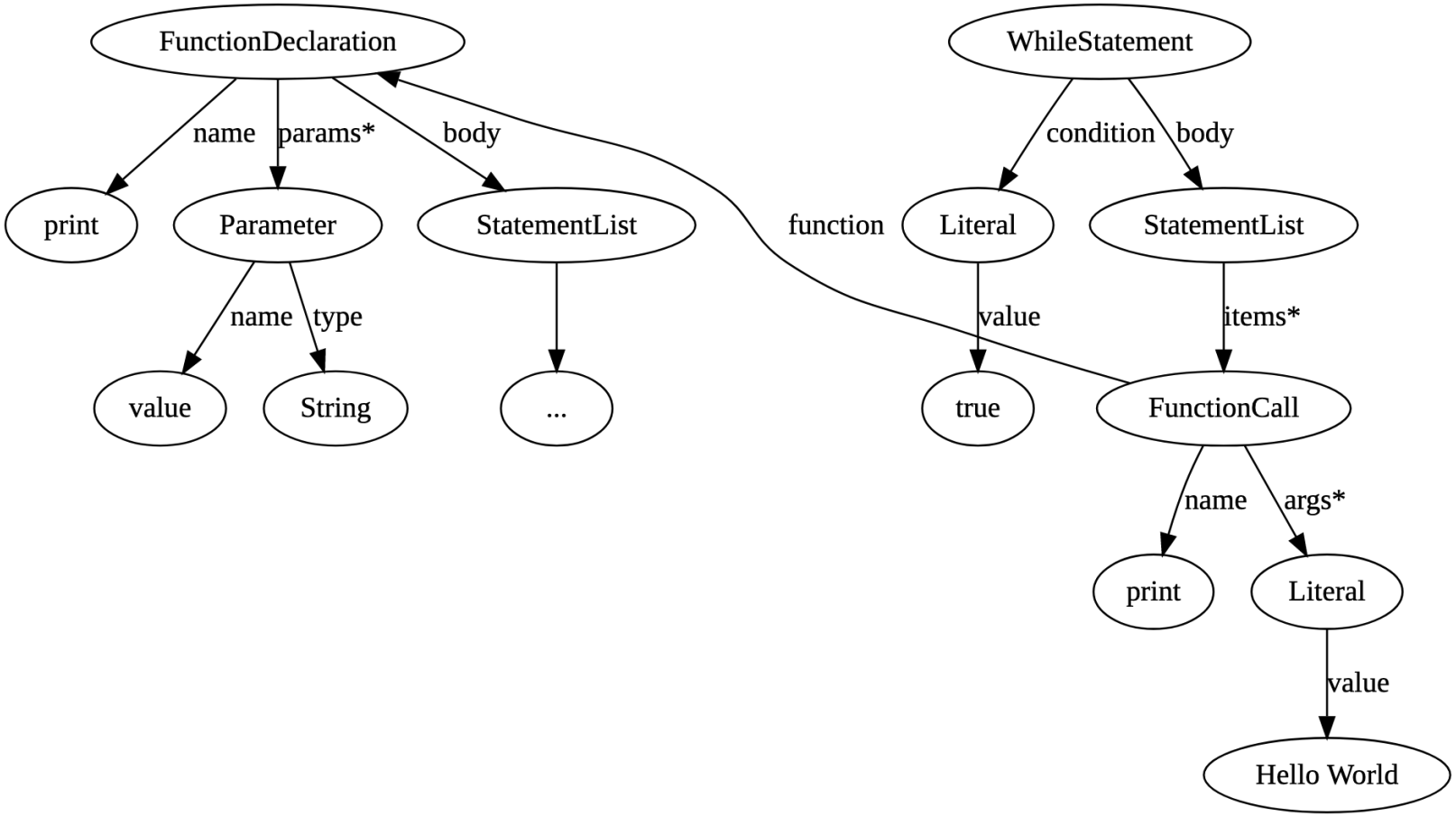
Abstrakter Syntaxbaum (AST)



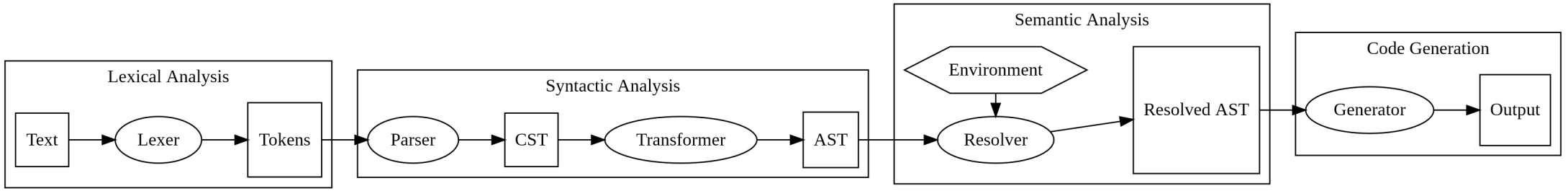

```
const ws = 
new WhileStatement(
  new Literal(true),
  new StatementList([
    new FunctionCall(
      "print",
      new Literal(
        "Hello World"),
    ),
  ]),
);

(ws.condition as
Literal).value = false
```

Konzept

Aufgelöster AST



Unser Compiler



Unsere Sprache

```
class Greeter {
  var name: string = "World"
  var count: int = 0

  init(name: string) {
    this.name = name
  }

  static func main(): void {
    var greeter = Greeter("World")
    greeter.greet()
  }

  func greet(): void {
    var greeting
      = "Hello, " + this.name + "!"
    this.greet(greeting)
  }

  /**
   * Sends a greeting
   */
  func greet(greeting: string) {
    println(greeting)
    this.count = this.count + 1
  }
}
```

https://sekassel-research.github.io/stc-23/editor

Nächste Schritte

- Bis zur nächsten VL (30.10.): **DSL** überlegen
  - Name und ggf. Link oder Beschreibung in Discord posten
  - Jeder braucht eigene Sprache, frühere Meldung hat Vorrang!
  - Sprachen müssen die Kriterien erfüllen und werden abgesegnet
  - Beispielsprachen: LOOP, SQL, Prisma, Graphviz/Dot, ANTLR, Protobuf, Java++, Dreamberd
- Bis zur übernächsten VL (06.11.): **Anmeldungsformular** ausfüllen
  - https://forms.gle/v7nJZ4xqTma14kyq8
- Bis in einigen Wochen: **Prüfung** im HIS anmelden
  - Termin wird noch bekanntgegeben