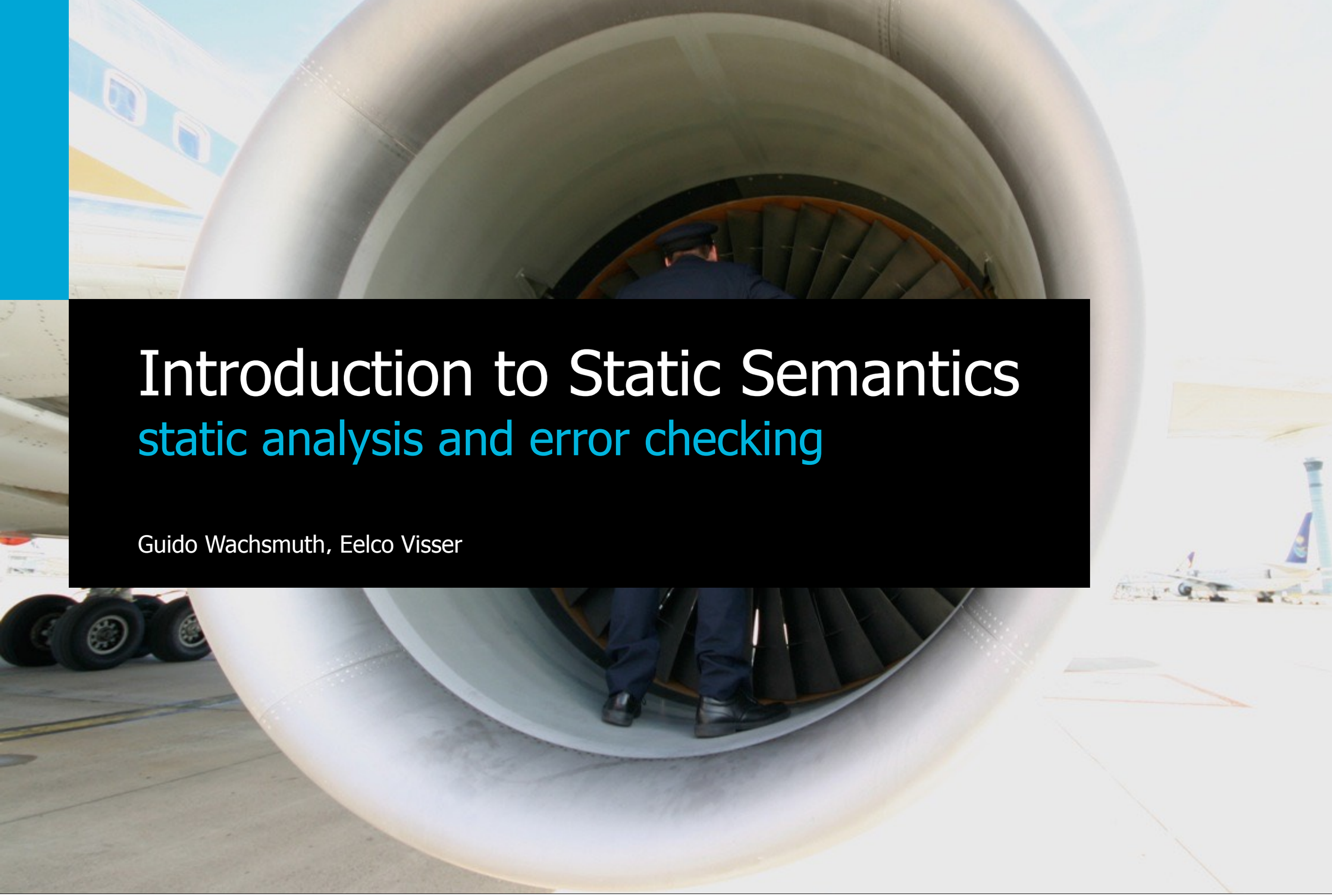
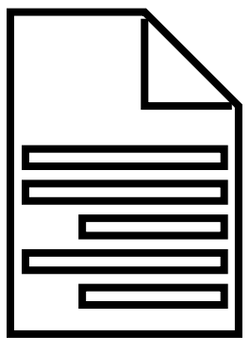


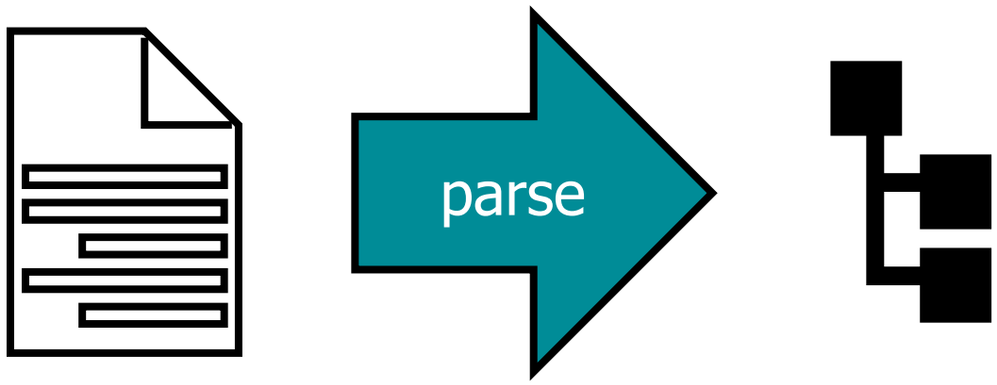
Introduction to Static Semantics

static analysis and error checking

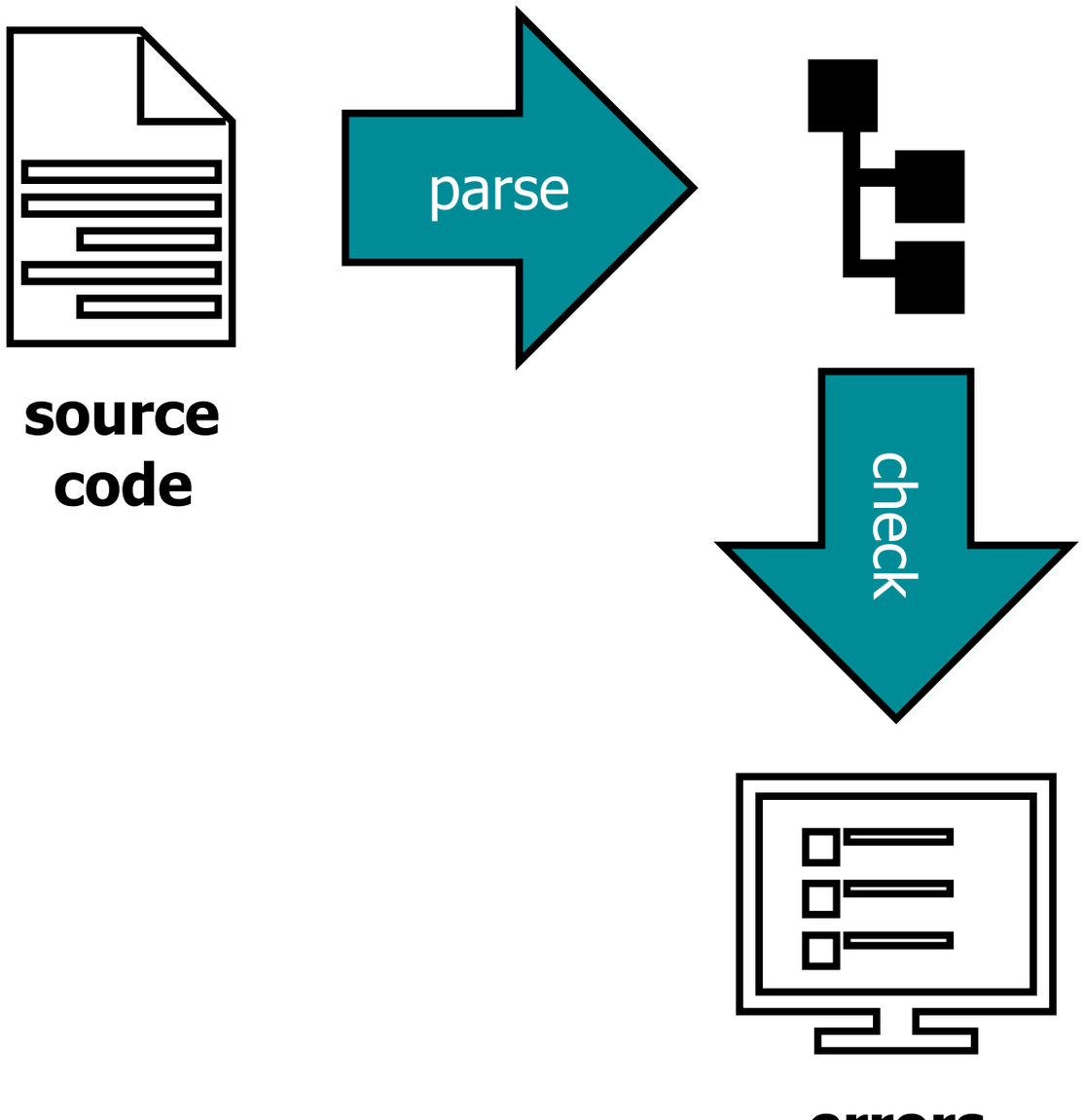
Guido Wachsmuth, Eelco Visser



source code

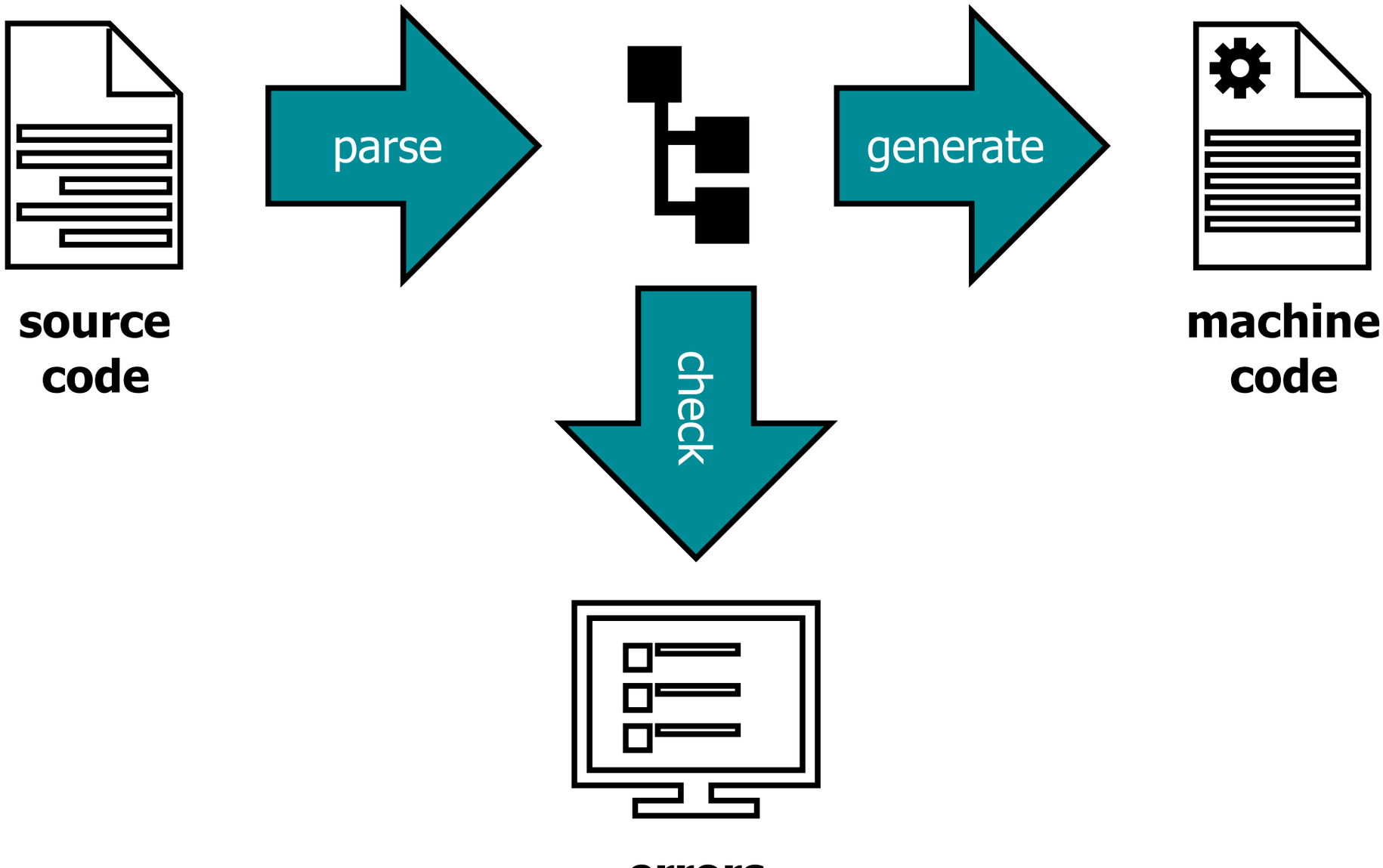


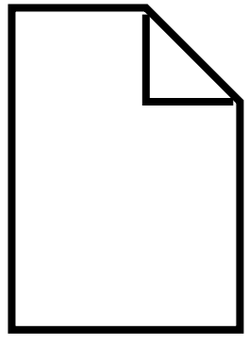
source code



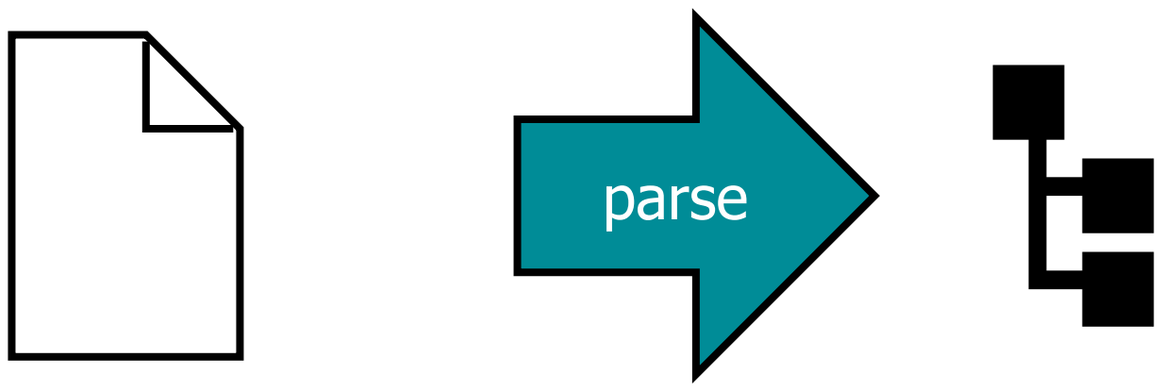
source code

errors

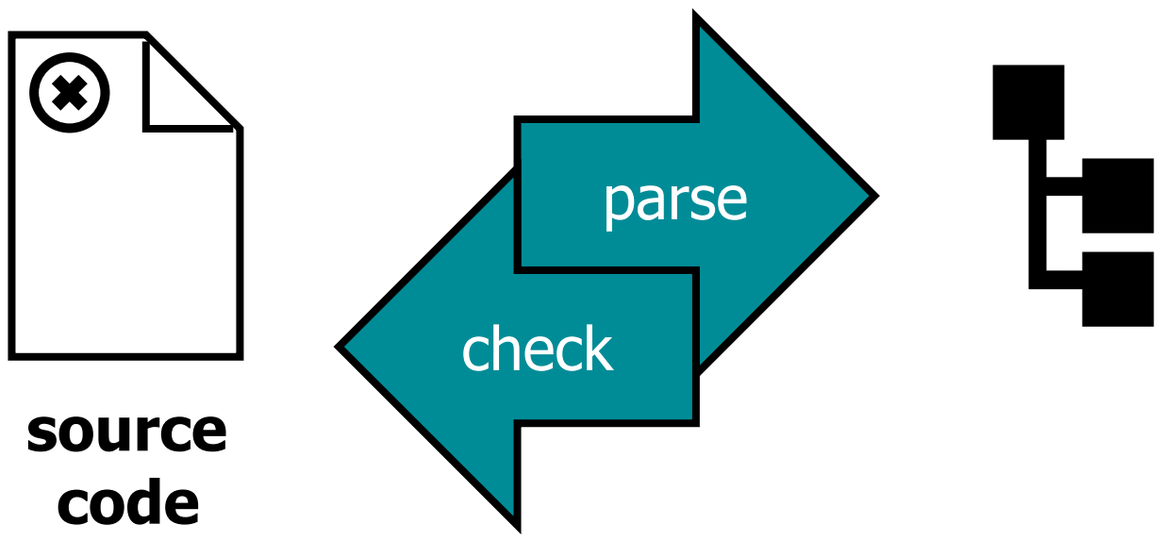

parse
generate
check
source code
machine code
errors

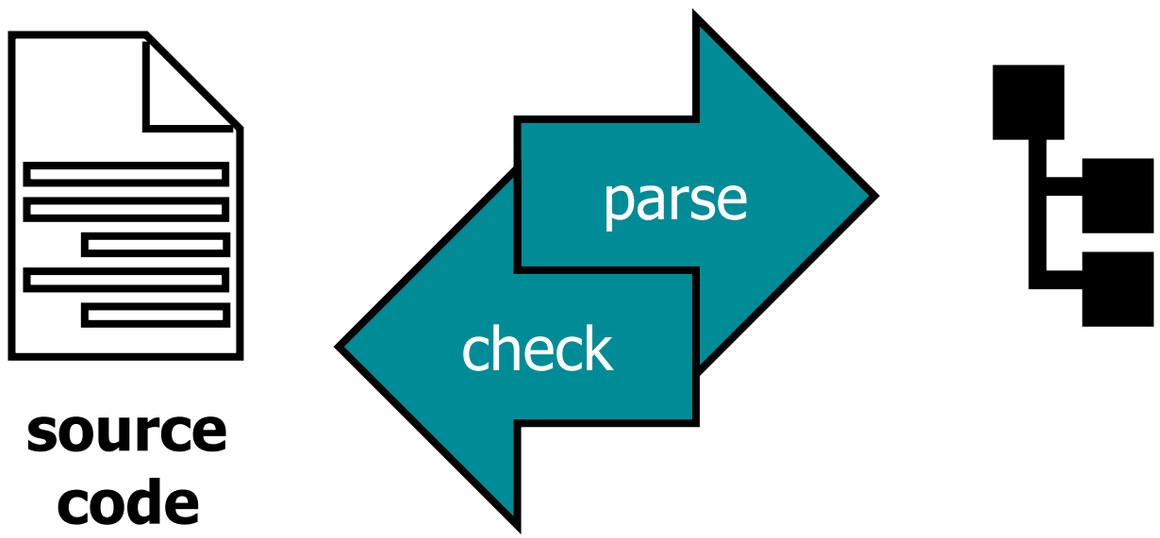


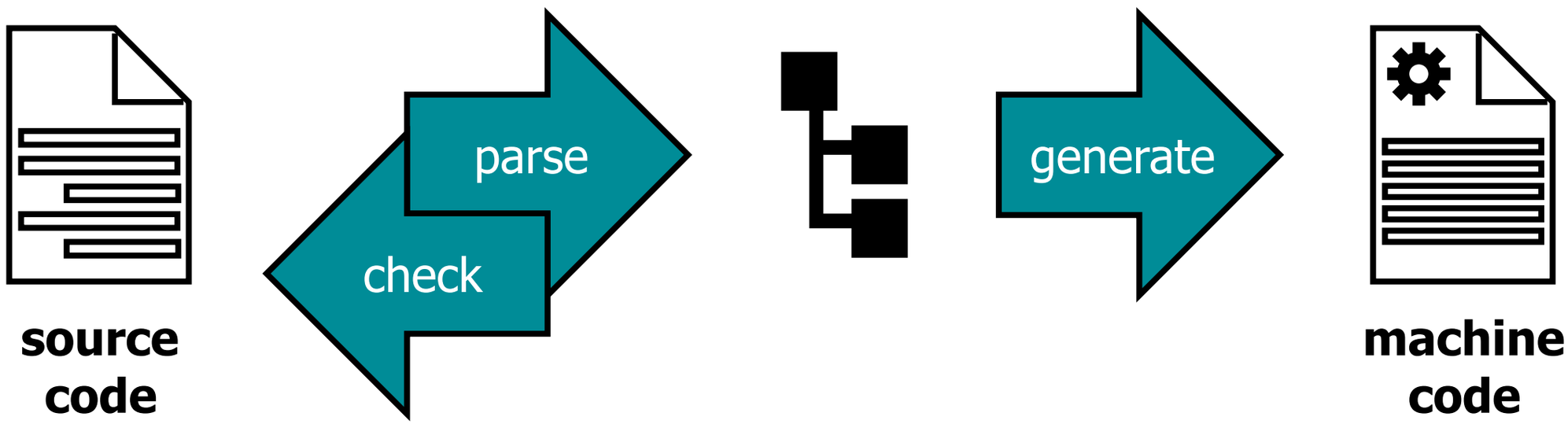
source code



source code


parse
check
source
code


parse
check
source
code


parse
check
generate
source code
machine code

Name Analysis

name binding and scope

static checking

```
1  class User {
2    string name;
3  }
4  class Blog {
5    string post(User user, string message) {
6      posterName = "name";
7      string posterName;
8      posterName = user.nam;
9      string posterName = user.name;
10     return posterName;
11   }
12 }
```

Name Analysis

name binding and scope

static checking

editor services

```
1  class User {
2    string name;
3  }
4  class Blog {
5    string post(User user, string message) {
6      posterName = "name";
7      string posterName;
8      posterName = user.nam;
9      string posterName = user.name;
10     return posterName;
11   }
12 }
```

Name Analysis

name binding and scope

static checking

editor services

transformation

```
1  class User {
2    string name;
3  }
4  class Blog {
5    string post(User user, string message) {
6      posterName = "name";
7      string posterName;
8      posterName = user.nam;
9      string posterName = user.name;
10     return posterName;
11   }
12 }
```

Name Analysis

name binding and scope

static checking

editor services

transformation

refactoring

```
1  class User {
2    string name;
3  }
4  class Blog {
5    string post(User user, string message) {
6      posterName = "name";
7      string posterName;
8      posterName = user.nam;
9      string posterName = user.name;
10     return posterName;
11   }
12 }
```

Name Analysis

name binding and scope

static checking

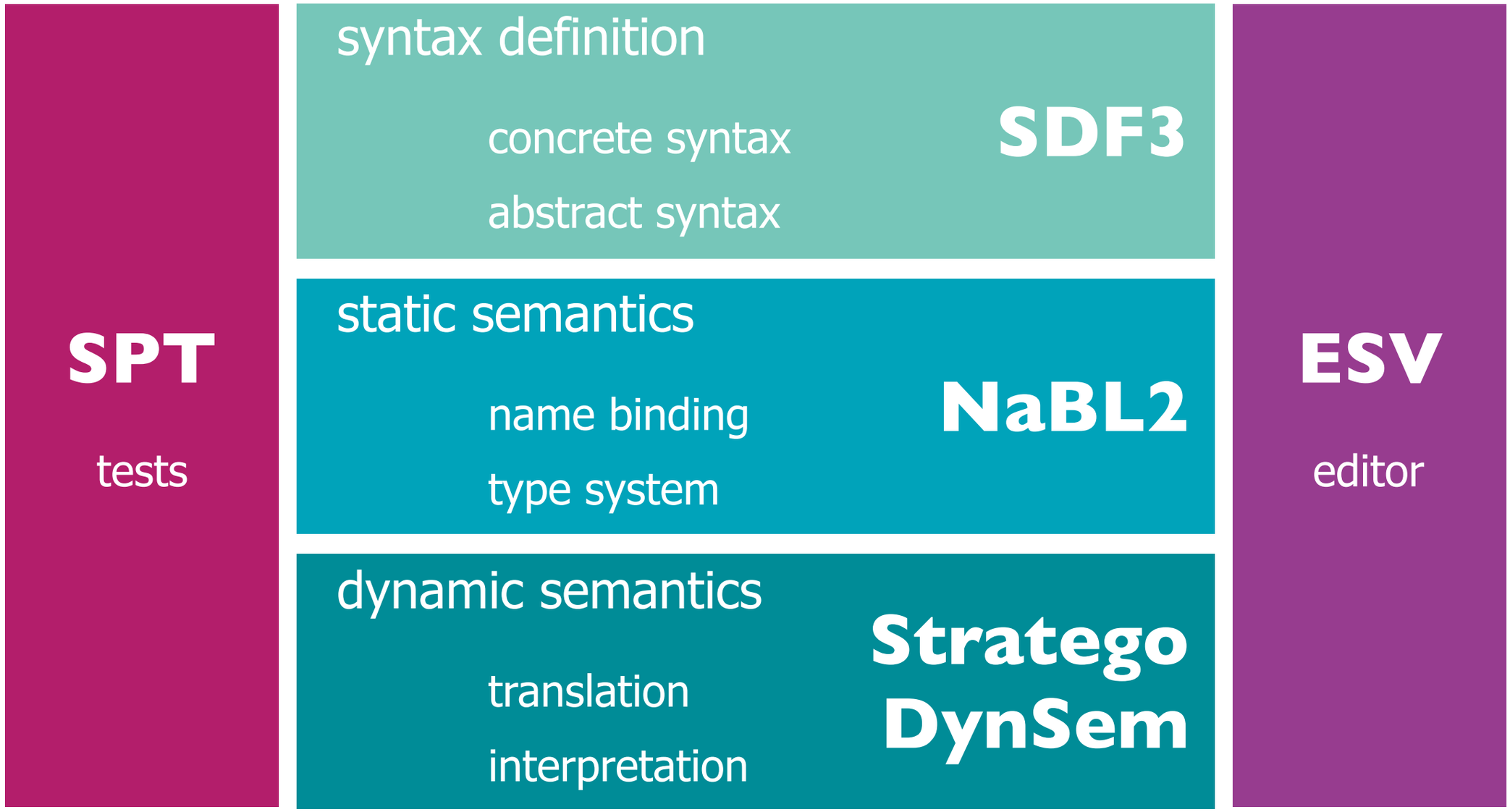
editor services

transformation

refactoring

code generation

```
1  class User {
2    string name;
3  }
4  class Blog {
5    string post(User user, string message) {
6      posterName = "name";
7      string posterName;
8      posterName = user.nam;
9      string posterName = user.name;
10     return posterName;
11   }
12 }
```


SPT
tests
syntax definition
concrete syntax
abstract syntax
SDF3
static semantics
name binding
type system
NaBL2
dynamic semantics
translation
interpretation
Stratego
DynSem
ESV
editor

SPT

tests

syntax definition

concrete syntax

abstract syntax

SDF3

static semantics

name binding

type system

NaBL2

dynamic semantics

translation

interpretation

Stratego DynSem

ESV

editor

formal semantics

type system

name binding

testing

name binding

type system

constraints

specification

name binding

type system

constraints

Formal Semantics

static semantics

Theoretical Computer Science

decidability & complexity

word problem χ_L: $\Sigma^* \rightarrow \{0,1\}$

$w \rightarrow 1$, if $w \in L$

$w \rightarrow 0$, else

Theoretical Computer Science

decidability & complexity

word problem χ_L: $\Sigma^* \rightarrow \{0,1\}$

$w \rightarrow 1$, if $w \in L$

$w \rightarrow 0$, else

decidability

type-0: semi-decidable

type-1, type-2, type-3: decidable

Theoretical Computer Science

decidability & complexity

word problem χ_L: $\Sigma^* \rightarrow \{0,1\}$

- $w \rightarrow 1$, if $w \in L$
- $w \rightarrow 0$, else

decidability

- type-0: semi-decidable
- type-1, type-2, type-3: decidable

complexity

- type-1: PSPACE-complete
- type-2, type-3: P

Theoretical Computer Science

decidability & complexity

word problem χ_L: $\Sigma^* \rightarrow \{0,1\}$

- $w \rightarrow 1$, if $w \in L$
- $w \rightarrow 0$, else

decidability

- type-0: semi-decidable
- type-1, type-2, type-3: decidable

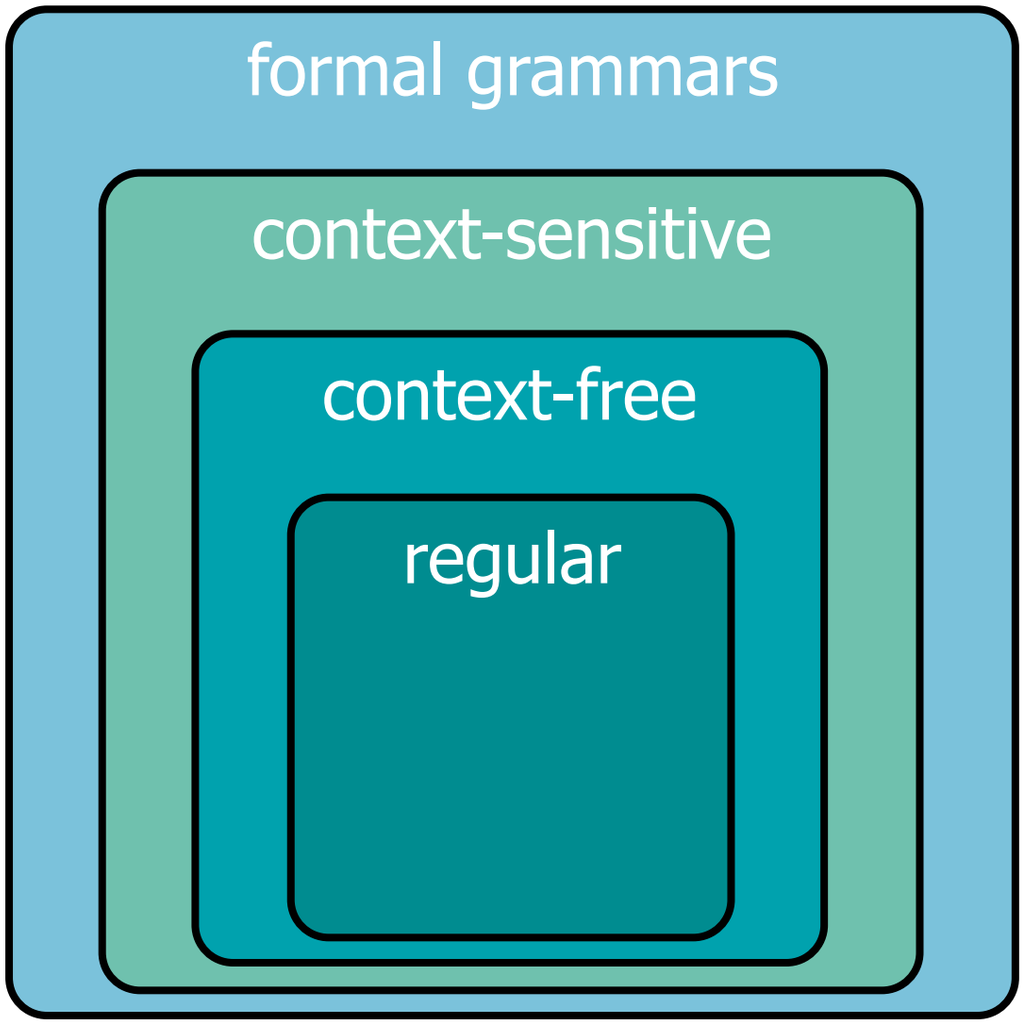
complexity

- type-1: PSPACE-complete
- type-2, type-3: P

$PSPACE \supseteq NP \supseteq P$

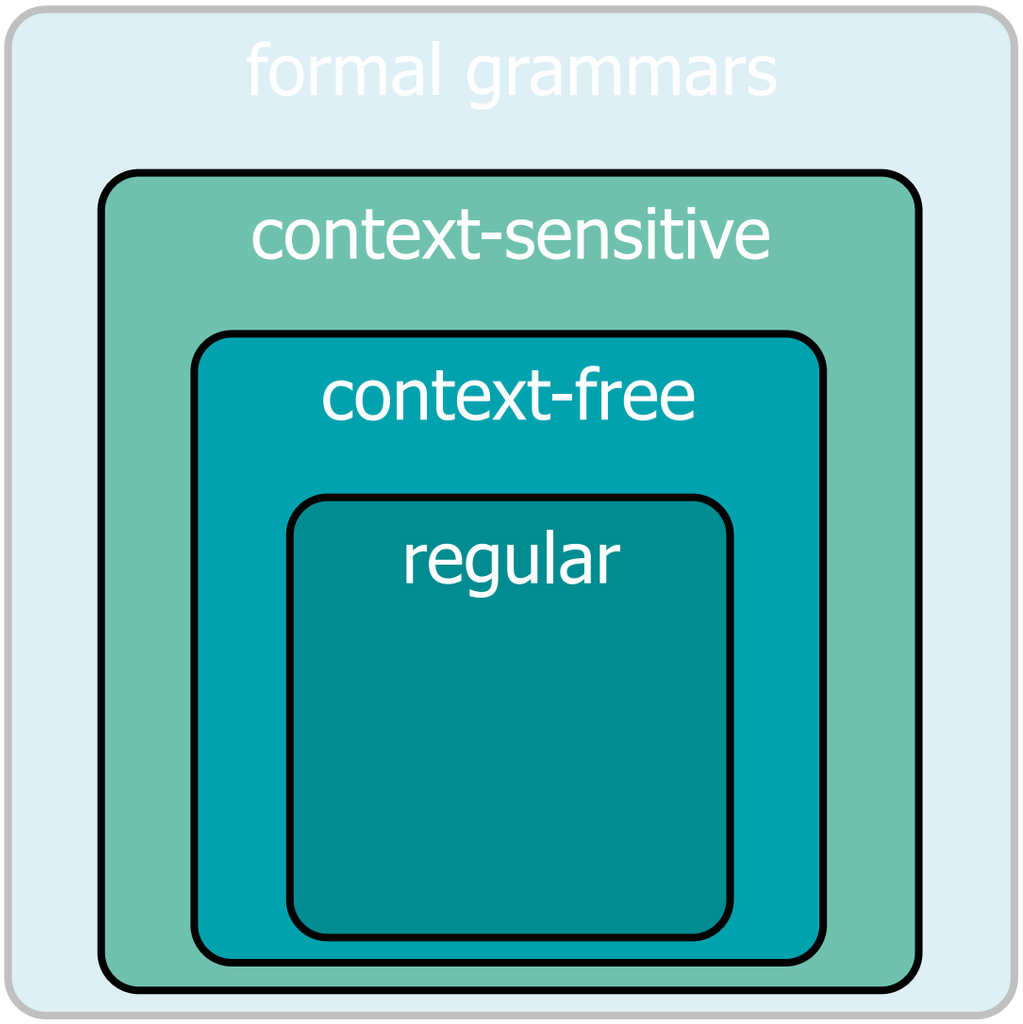
theoretical computer science

decidability & complexity



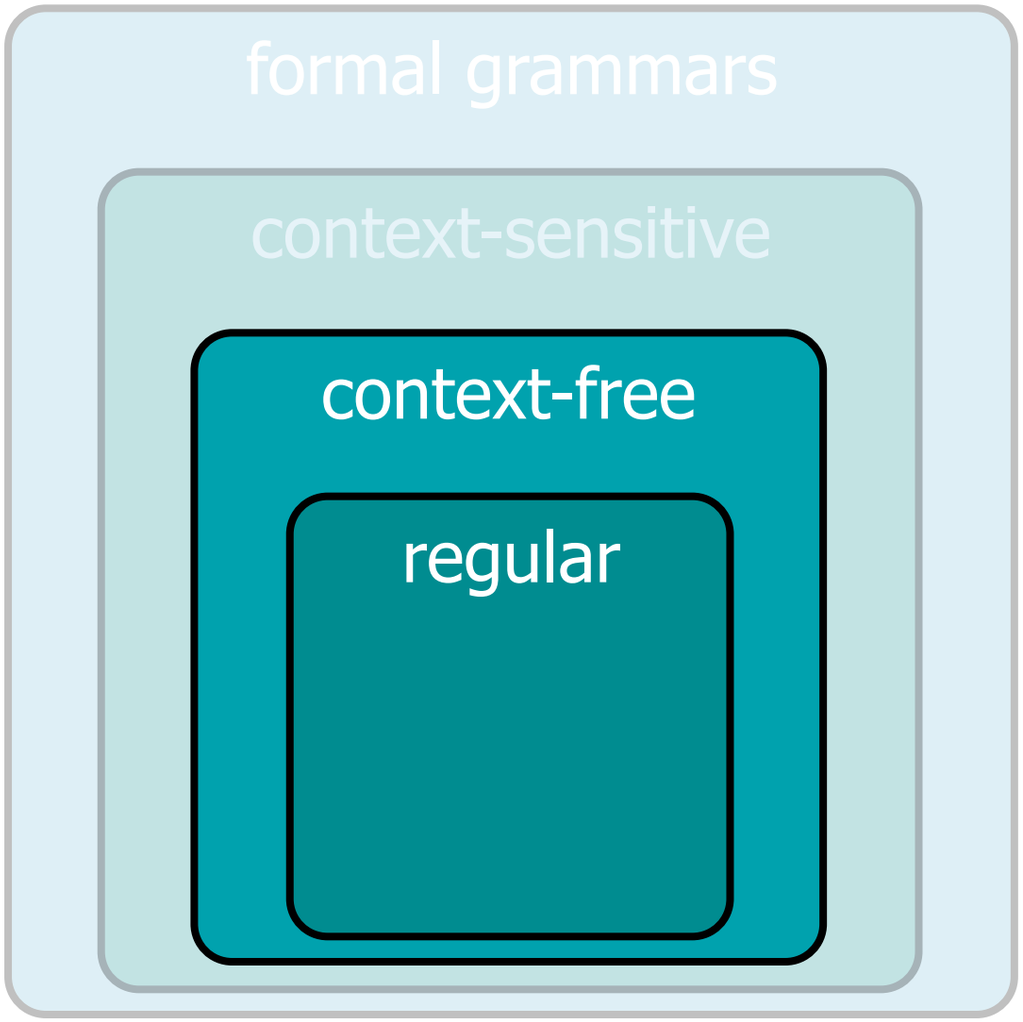
theoretical computer science

decidability & complexity

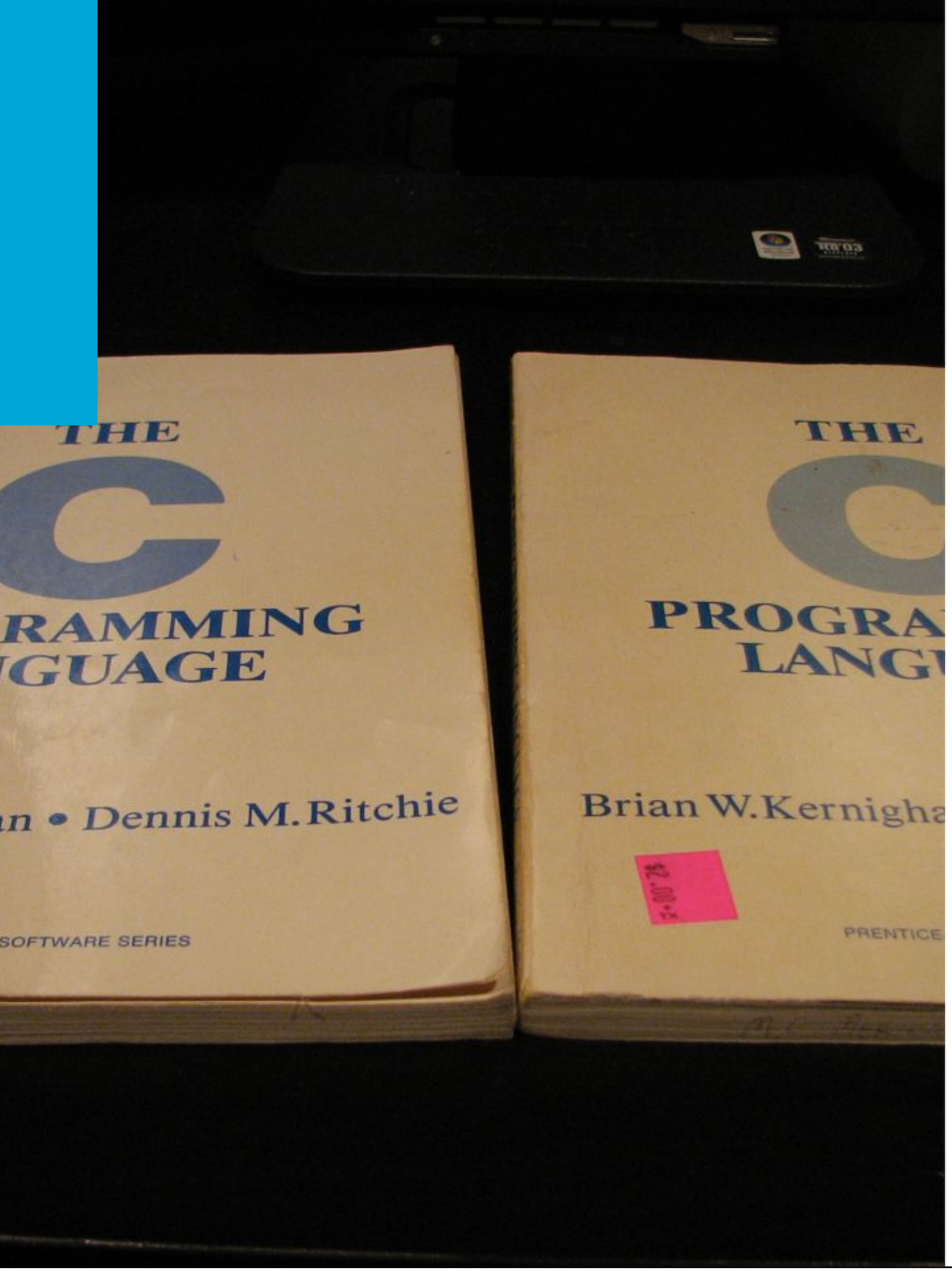


theoretical computer science

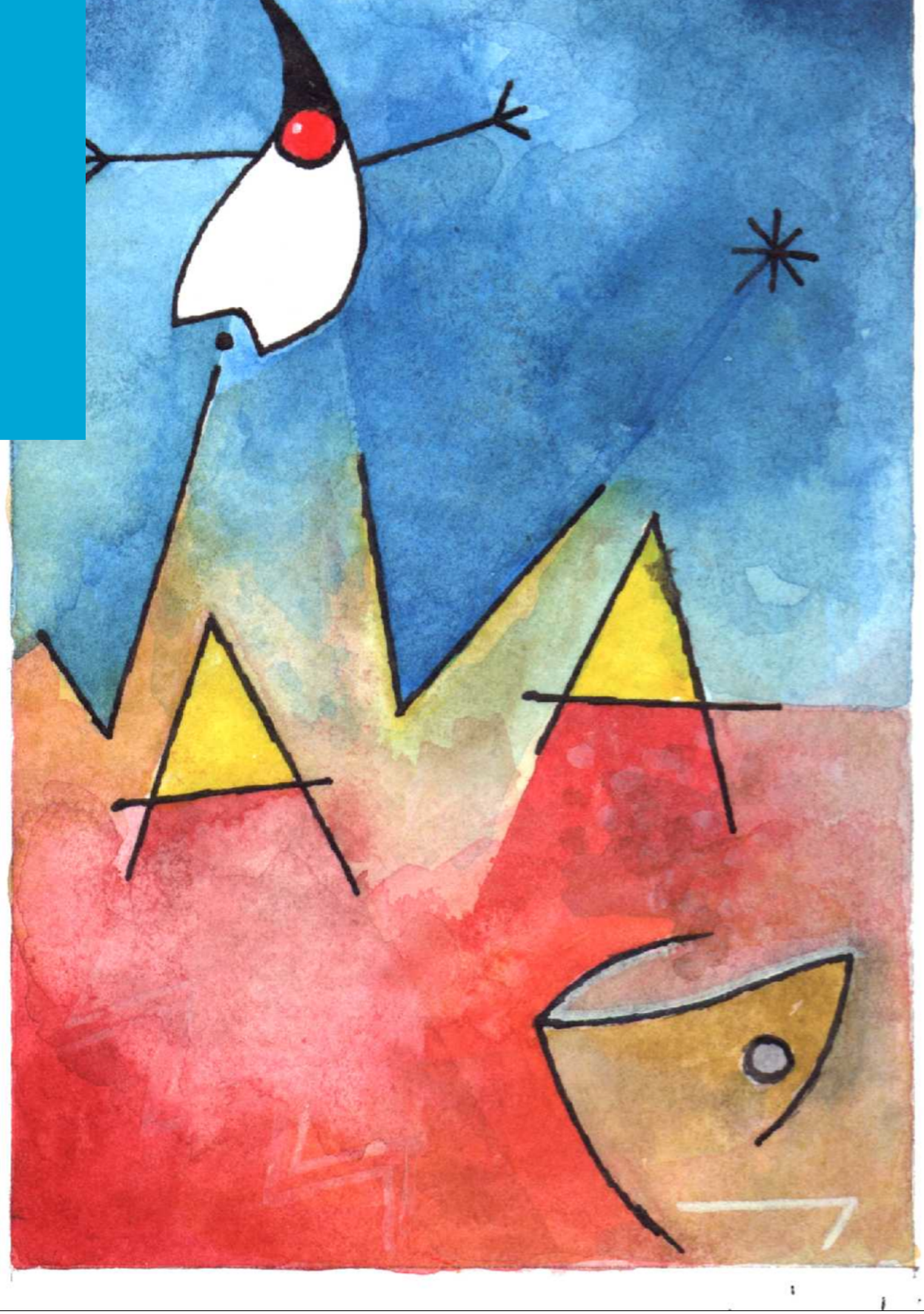
decidability & complexity



```
/* factorial function */

let

  var x := 0

  function fact(n : int) : int =
    if n < 1 then 1 else (n * fact(n - 1))

in

  for i := 1 to 3 do (
    x := x + fact(i);
    printint(x);
    print(" ")
  )

end
```




```c
#include <stio.h>

/* factorial function */

int fac(int num) {
  if (num < 1)
     return 1;
  else
    return num * fac(num - 1);
}

int main() {
  printf("%d! = %d\n", 10, fac(10));
  return 0;
}
```



```
class Main {

  public static void main(String[] args) {
    System.out.println(new Fac().fac(10));
  }
}

class Fac {

  public int fac(int num) {
    int num_aux;
    if (num < 1)
      num_aux = 1;
    else
      num_aux = num * this.fac(num - 1);
    return num_aux;
  }
}
```

static semantics

restricting context-free languages

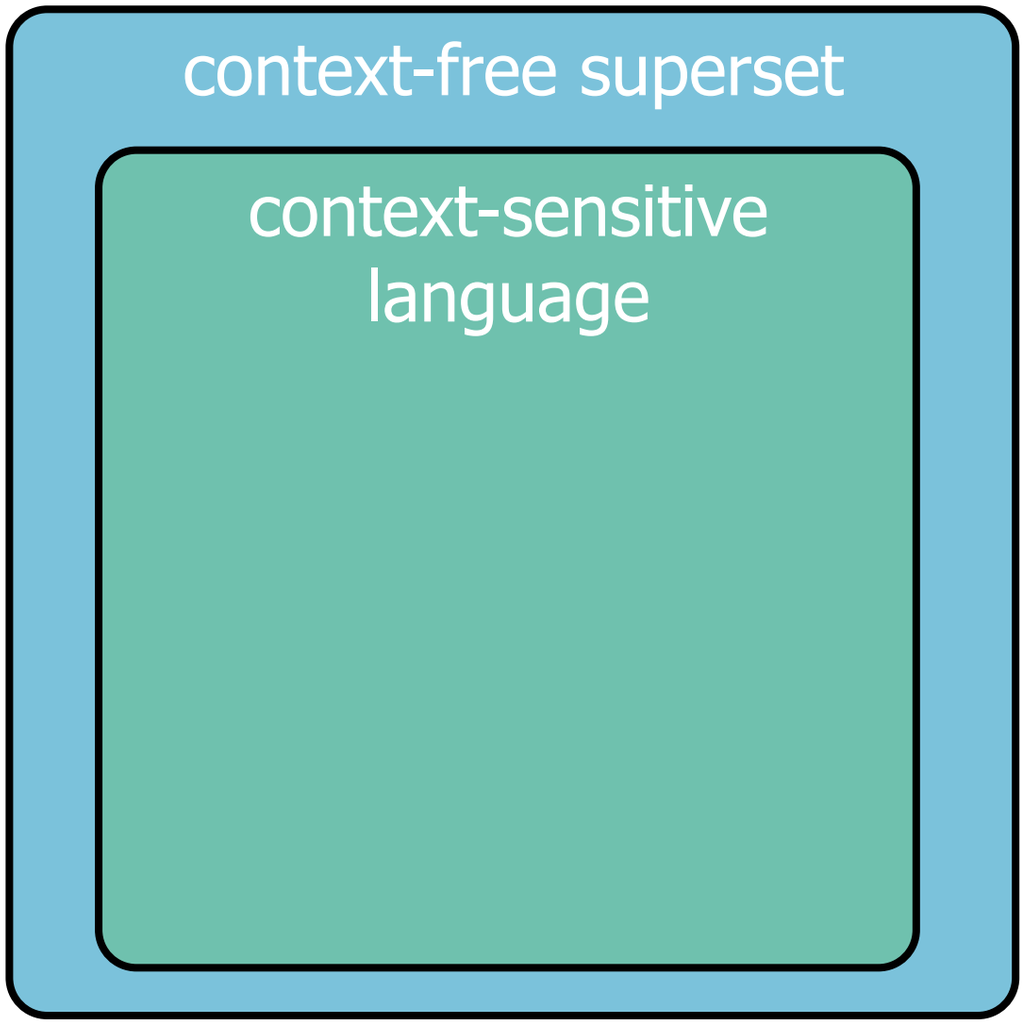
context-free grammar

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$

context-sensitive language

static semantics

restricting context-free languages



context-free grammar

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$

static semantics

restricting context-free languages

context-free grammar

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$

static semantics

restricting context-free languages

context-free grammar

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$

static semantics

$L = \{w \in L(G) \mid \vdash w\}$

static semantics

restricting context-free languages

context-free grammar

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$

static semantics

$L = \{w \in L(G) \mid \vdash w\}$

judgements

well-formed $\vdash w$

well-typed $E \vdash e : t$

Formal Semantics

type systems

Tiger

type system

$E \vdash i : \textbf{int}$

$E \vdash s : \textbf{string}$

$E \vdash \textbf{nil} : \bot$

Tiger

type system

$$E \vdash () : \varnothing$$

$$\frac{E \vdash e_1 : t_1 \qquad E \vdash e_2 : t_2}{E \vdash e_1 \,;\, e_2 : t_2}$$

Tiger

type system

$$\frac{E \vdash e_1 : \textbf{int} \quad E \vdash e_2 : \textbf{int}}{E \vdash e_1 \textbf{ + } e_2 : \textbf{int}}$$

$$\frac{E \vdash e_1 : \textbf{array of } t \quad E \vdash e_2 : \textbf{int}}{E \vdash e_1\textbf{[}e_2\textbf{]} : t}$$

$$\frac{E \vdash e_1 : \textbf{int} \quad E \vdash e_2 : \textbf{int}}{E \vdash e_1 \textbf{ < } e_2 : \textbf{int}}$$

$$\frac{E \vdash e_1 : \textbf{string} \quad E \vdash e_2 : \textbf{string}}{E \vdash e_1 \textbf{ < } e_2 : \textbf{int}}$$

Tiger

type system

$$\frac{E \vdash e_1 : t_1 \quad E \vdash e_2 : t_2 \quad t_1 \cong t_2}{E \vdash e_1 = e_2 : \textbf{int}}$$

$$\frac{t_1 <: t_2}{t_1 \cong t_2} \qquad \frac{t_2 <: t_1}{t_1 \cong t_2} \qquad \frac{t \neq \varnothing}{t \cong t}$$

$$\bot <: \{f_1, \ldots, f_n\}$$

$$\bot <: \textbf{array of } t$$

Tiger

type system

$$\frac{E \vdash e_1 : t_1 \quad E \vdash e_2 : t_2 \quad t_1 \cong t_2}{E \vdash e_1 \texttt{ := } e_2 : \varnothing}$$

$$\frac{E \vdash e_1 : \textbf{int} \quad E \vdash e_2 : t_1 \quad E \vdash e_3 : t_2 \quad t_1 \cong t_2}{E \vdash \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : t_2}$$

Formal Semantics

name binding

Tiger

scoping

```
let
    type t = u
    type u = int
    var x: u := 0
in
    x := 42 ;
    let
        type u = t
        var y: u := 0
    in
        y := 42
    end
end
```

Tiger

scoping

```
let
    type t = u
    type u = int
    var x: u := 0
in
    x := 42 ;
    let
        type u = t
        var y: u := 0
    in
        y := 42
    end
end
```

Tiger

scoping

```
let
    type t = u
    type u = int
    var x: u := 0
in
    x := 42 ;
    let
        type u = t
        var y: u := 0
    in
        y := 42
    end
end
```

Tiger

scoping

```
let
    type t = u
    type u = int
    var x: u := 0
in
    x := 42 ;
    let
        type u = t
        var y: u := 0
    in
        y := 42
    end
end
```

Tiger

scoping

```
let
    type t = u
    type u = int
    var x: u := 0
in
    x := 42 ;
    let
        type u = t
        var y: u := 0
    in
        y := 42
    end
end
```

Tiger

variable names

$$\frac{E \vdash e_1 : t_1 \quad t \cong t_1 \quad E \oplus v \mapsto t \vdash e_2 : t_2}{E \vdash \textbf{let var } v : t = e_1 \textbf{ in } e_2 : t_2}$$

$$\frac{E(v) = t}{E \vdash v : t}$$

Tiger

function names

$$\frac{E \oplus v_1 \mapsto t_1 \,,\ldots,\, v_n \mapsto t_n \vdash e_1 : t_f \qquad E \oplus f \mapsto t_1 \times \ldots \times t_n \rightarrow t_f \vdash e_2 : t}{E \vdash \textbf{let}\ \textbf{function}\ f\ (v_1 : t_1, \ldots, v_n : t_n) = e_1\ \textbf{in}\ e_2 : t}$$

$$\frac{E(f) = t_1 \times \ldots \times t_n \rightarrow t_f \qquad e_1 : t_1 \qquad \ldots \qquad e_n : t_n}{E \vdash f\ (e_1, \ldots, e_n) : t}$$

Testing Static Analysis

Testing

name binding

```
test outer name [[
    let type t = u
        type [[u]] = int
        var x: [[u]] := 0
    in
        x := 42 ;
        let type u = t
            var y: u := 0
        in
            y := 42
        end
end
]] resolve #2 to #1
```

```
test inner name [[
    let type t = u
        type u = int
        var x: u := 0
    in
        x := 42 ;
        let type [[u]] = t
            var y: [[u]] := 0
        in
            y := 42
        end
end
]] resolve #2 to #1
```

Testing

type system

```
test integer constant [[
    let type t = u
        type u = int
        var x: u := 0
    in
        x := 42 ;
        let type u = t
            var y: u := 0
        in
            y := [[42]]
        end
end
]] run get-type to IntTy()
```

```
test variable reference [[
    let type t = u
        type u = int
        var x: u := 0
    in
        x := 42 ;
        let type u = t
            var y: u := 0
        in
            y := [[x]]
        end
end
]] run get-type to IntTy()
```

Testing

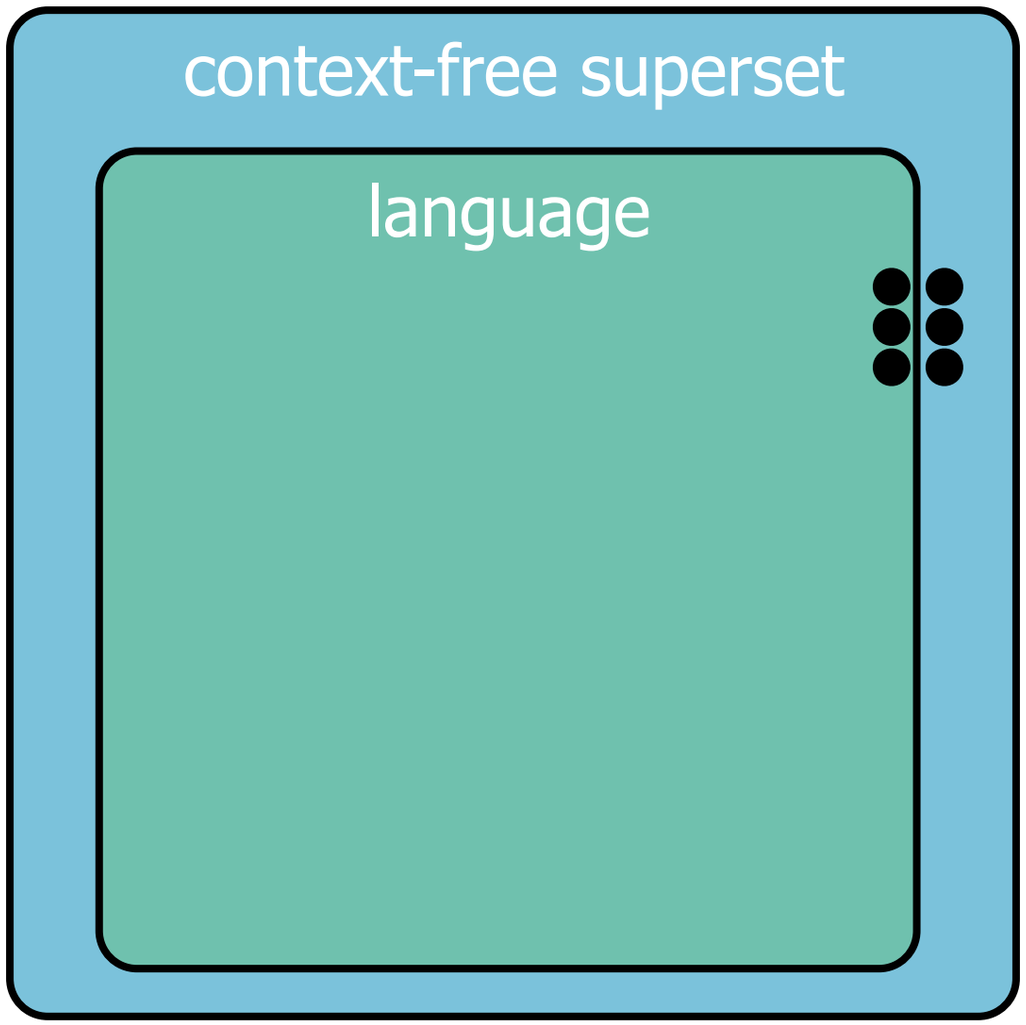
constraints

```
test undefined variable [[
    let type t = u
        type u = int
        var x: u := 0
    in
        x := 42 ;
        let type u = t
            var y: u := 0
        in
            y := [[z]]
        end
end
]] 1 error
```

```
test type error [[
    let type t = u
        type u = string
        var x: u := 0
    in
        x := 42 ;
        let type u = t
            var y: u := 0
        in
            y := [[x]]
        end
end
]] 1 error
```

Testing

static semantics



Next

Next

Week 5

- lexical analysis: from regular grammar to DFA

Next

Week 5

- lexical analysis: from regular grammar to DFA

Week 6

- name binding rules
- name resolution with scope graphs

Next

Week 5

- lexical analysis: from regular grammar to DFA

Week 6

- name binding rules
- name resolution with scope graphs

Week 8

- constraint-based type analysis

Next

Week 5

- lexical analysis: from regular grammar to DFA

Week 6

- name binding rules
- name resolution with scope graphs

Week 8

- constraint-based type analysis

Q2

- dynamic semantics, code generation, optimization
- parsing algorithms

Spoofax

annotated terms

```
t{t1, ..., tn}
```

add additional information to a term but preserve its signature

attribution

slide	title	author	license
1	Inspection	Kent Wien	CC BY-NC 2.0
2, 3	PICOL icons	Melih Bilgil	CC BY 3.0
10	Noam Chomsky	Maria Castelló Solbes	CC BY-NC-SA 2.0
11, 16-20, 22-24	Tiger	Bernard Landgraf	CC BY-SA 3.0
12	The C Programming Language	Bill Bradford	CC BY 2.0
13	Italian Java book cover		