# Jezički invarijantna provera semantičke ekvivalentnosti strukturno sličnih segmenata imperativnog koda

Ivan Ristović

## Teme

- Motivacija i uvod
- ANTLR
- Opšti AST
- Poređenje opštih AST
- LICC — Language Invariant Code Comparer

## Motivacija i uvod

```
void array_sum(int[] arr, int n) {
    int sum = 0, i = 0;
    while (i < n) {
        int v = arr[i]
        sum += v;
        i++;
    }
    return sum;
}
```

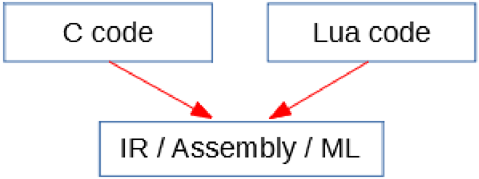
```
function array_sum(arr, n)
    local sum = 0
    for i,v in ipairs(arr) do
        sum = sum + v
    end
    return sum
end
```

## Uvod i motivacija

- Pristup?
  - "Niski"pristup
  - "Visoki"pristup
- Razlika je u reprezentaciji na koju se dovode segmenti koda pre procesa poređenja
- Ako je reprezentacija uvek ista, analiza je lakša

## Uvod i motivacija

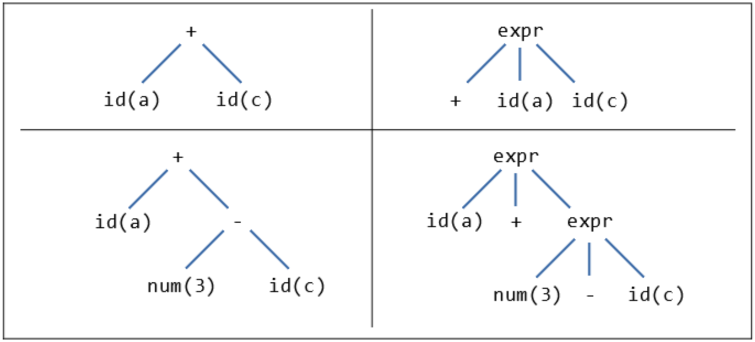
- Niski pristup



- Prednosti: ista reprezentacija (?)
- Mane: vezanost sa specifičnom arhitekturom procesora, potrebno prevoditi kod, JVM/CLR, razliciti programski jezici?

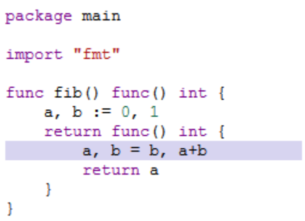
## Uvod i motivacija

- AST - Abstract Syntax Tree

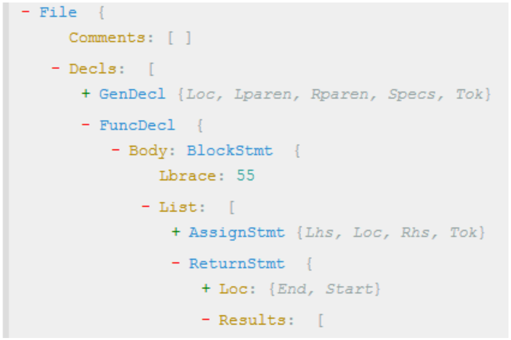


## Uvod i motivacija

- Go AST



```go
package main

import "fmt"

func fib() func() int {
    a, b := 0, 1
    return func() int {
        a, b = b, a+b
        return a
    }
}
```

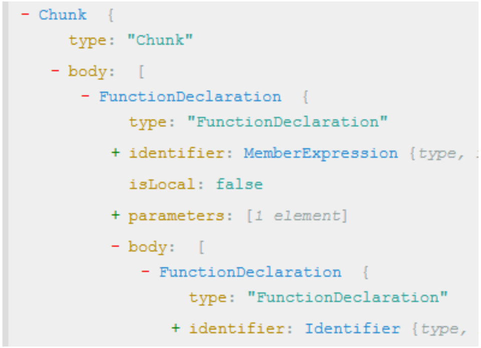


```
- File  {
     Comments: [ ]
   - Decls:  [
      + GenDecl {Loc, Lparen, Rparen, Specs, Tok}
      - FuncDecl  {
         - Body: BlockStmt  {
              Lbrace: 55
            - List:  [
               + AssignStmt {Lhs, Loc, Rhs, Tok}
               - ReturnStmt  {
                  + Loc: {End, Start}
                  - Results:  [
```

## Uvod i motivacija

- Lua AST

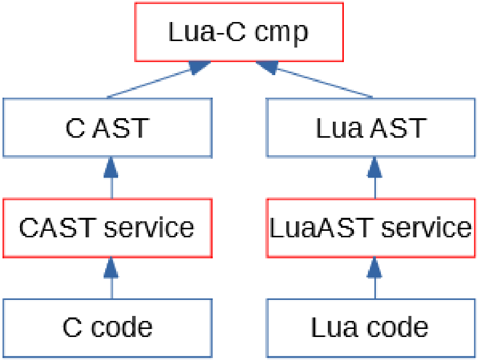
```
function Fibonacci.naive(n)
  local function inner(m)
    if m < 2 then
      return m
    end
    return inner(m-1) + inner(m-2)
  end
  return inner(n)
end
```



```
- Chunk  {
     type: "Chunk"
   - body:  [
      - FunctionDeclaration  {
           type: "FunctionDeclaration"
         + identifier: MemberExpression {type,
           isLocal: false
         + parameters: [1 element]
         - body:  [
            - FunctionDeclaration  {
                 type: "FunctionDeclaration"
               + identifier: Identifier {type,
```

## Uvod i motivacija

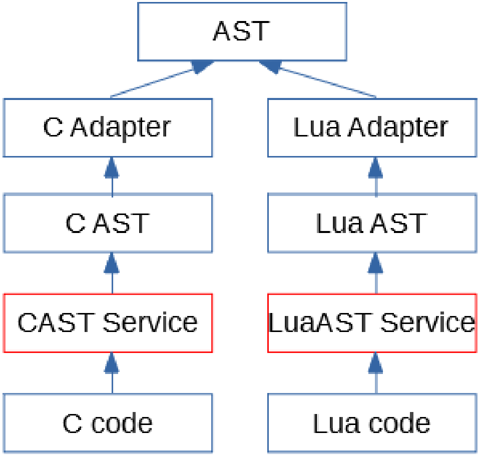
- Visoki pristup (varijanta 1)



- Prednosti: ista reprezentacija (?), nije potrebno prevoditi kod, kompatibilno sa bilo kojim programskim jezikom, moguće koristiti algoritme za poređenje stabala
- Mane: zavisnost od eksternih servisa, skaliranje

## Uvod i motivacija

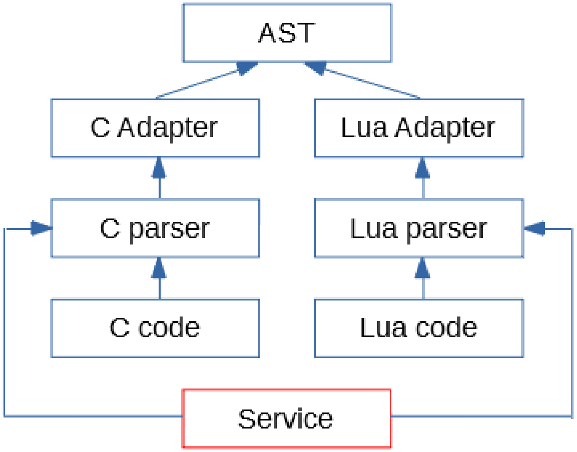
- Visoki pristup (varijanta 2)



- Prednosti: ista reprezentacija (!), nema prevođenja, moguće koristiti algoritme za poređenje stabala, skalabilno (?)

## Uvod i motivacija

- Visoki pristup (finalna varijanta)



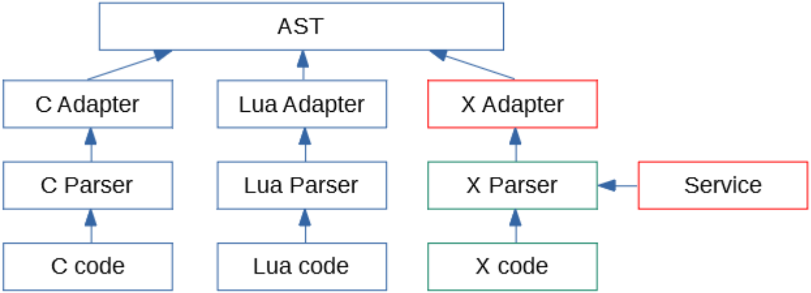
- Prednosti: isto kao pre uz to da postoji samo **jedan** servis i moguće je kod **proizvoljnog** programskog jezika prevesti u AST

## Uvod i motivacija

- Pošto nema prevođenja, može se analizirati kod samo na osnovu gramatike njegovog jezika
- AST se dobija od stabla parsiranja izvornog koda korišcenjem adaptera
- Adapteri se moraju razlikovati zbog razlika u AST

## Uvod i motivacija

- Kako proširiti?



## Dobijanje AST - ANTLR

- Neophodan je parser!
- AST nastaje apstrahovanjem stabla parsiranja
- Dosta alata: Yacc, BYACC, GNU Bison, ANTLR
- Svi ovi alati mogu generisati parsere za proizvoljne gramatike

## Dobijanje AST - ANTLR

- *ANother Tool for Language Recognition*
- ANTLR v4 izabran zbog:
  - Mogućnosti generisanja parsera u raznim jezicima (uključujući C#)
  - Trivijalno definisati gramatike (dosta poznatih jezika već podržano)
  - Mogu se generisati i klase koje pružaju interfejs za obilazak stabla parsiranja

## Dobijanje AST - ANTLR

- Prvi korak: definicija gramatike

```
grammar Lua;
chunk : block EOF ;
block : stat* retstat? ;
stat
     : ';'
     | varlist '=' explist
     | functioncall
     | label
     | 'break'
     | 'do' block 'end'
     | 'while' exp 'do' block 'end'
     | 'if' exp 'then' block ('elseif' exp 'then'
         block)* ('else' block)? 'end'
     | 'for' NAME '=' exp ',' exp (',' exp)? 'do'
         block 'end'
     | 'function' funcname funcbody
     ...
```

## Dobijanje AST - ANTLR

- Prvi korak: definicija gramatike

```
NAME
        : [a-zA-Z_][a-zA-Z_0-9]*
        ;

NORMALSTRING
        : '"' ( EscapeSequence | ~('\\'|'"') )* '"'
        ;

WS
        : [ \t\u000C\r\n]+ -> skip
        ;
```

## Dobijanje AST - ANTLR

- Drugi korak: generisanje parsera

```
$ antlr4 Lua.g4 -Dlanguage=CSharp --visitor
```

- Generisane `LuaLexer` i `LuaParser` klase
- Generisani interfejsi `LuaListener` i `LuaVisitor`

## Dobijanje AST - ANTLR

- Treći korak: obići stablo parsiranja i kreirati AST

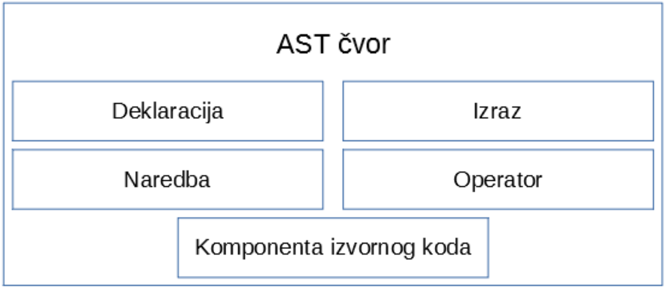
```
public interface ILuaVisitor<T> :
    IParseTreeVisitor<T>
{
    T VisitChunk([NotNull]
        LuaParser.ChunkContext context);
    T VisitBlock([NotNull]
        LuaParser.BlockContext context);
    T VisitStat([NotNull] LuaParser.StatContext
        context);

    ...
}
```

## Opšti AST

- Želimo opšti AST, koji će podržavati koncepte raznih imperativnih jezika
- Koncepti: literali, izrazi, naredbe, ...
- Kreirati dovoljno (ali ne previše) apstraktne tipove čvora za ove koncepte
- Specifičnosti svesti na ”već viđeno”
- Ako svođenje nema smisla, uvesti novi tip AST čvora
- Izgubiti što manje informacija!!!

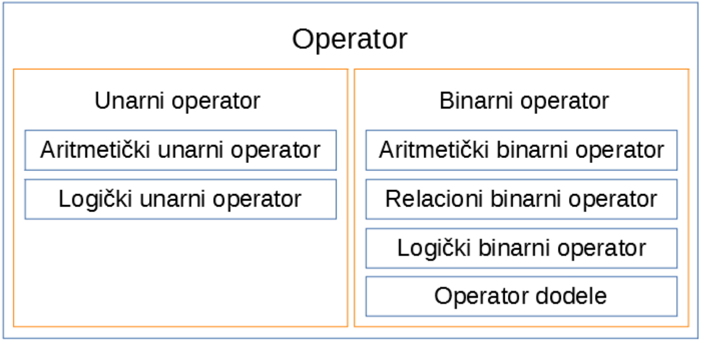
## Opšti AST

- Bazna hijerarhija



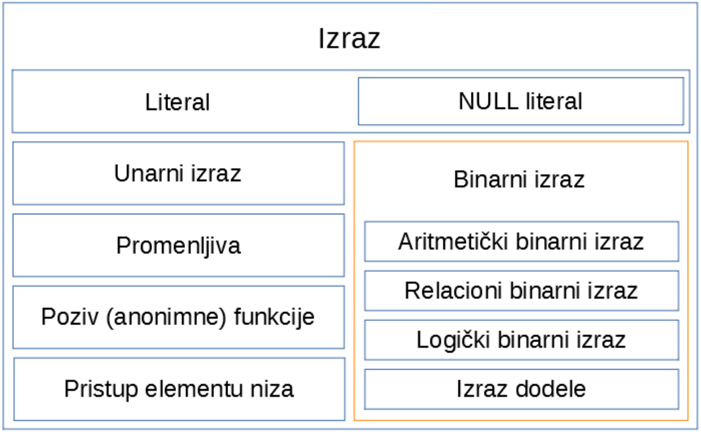
## Opšti AST

- Operatori



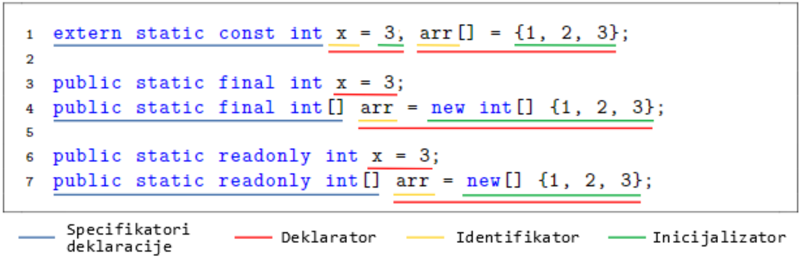
## Opšti AST

- Izrazi



# Opšti AST

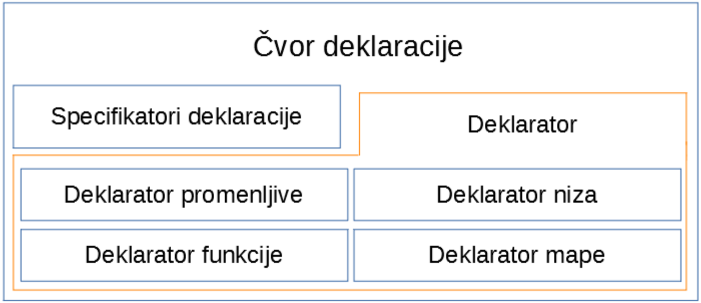
- Deklaracije



```
extern static const int x = 3, arr[] = {1, 2, 3};

public static final int x = 3;
public static final int[] arr = new int[] {1, 2, 3};

public static readonly int x = 3;
public static readonly int[] arr = new[] {1, 2, 3};
```

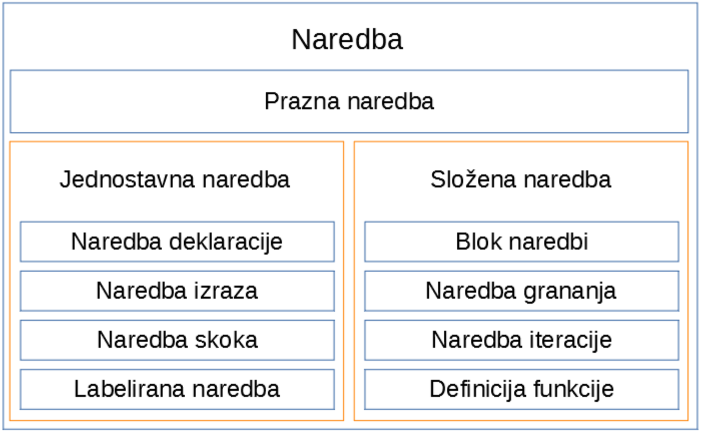
## Opšti AST

- Deklaracije



## Opšti AST

- Naredbe



## Opšti AST

- Primer - *swap*

```
int tmp = x;
x = y;
y = tmp;
```

```
x, y = y, x
```

- Paziti na nove konstrukte
- U slučaju skript jezika, deklarisati promenljive pre korišćenja

## Gde smo sada?

- x Motivacija
- x AST zasnovan pristup
- x Dobijanje AST - ANTLR
- x Opšti AST
- Poređenje opštih AST
- LICC — Language Invariant Code Comparer

## Poređenje opštih AST

- Cilj: Napraviti proširiv sistem
- Poređenje treba da radi nad bilo koja dva čvora
- Ima smisla porediti samo čvorove istog tipa, ali u nekim slučajevima možda ima smisla porediti i različite tipove (rečnik — objekat)
- Potrebno je voditi računa o vrednostima promenljivih
- Naivno porediti čvorove stabla po jednakosti atributa i rekurzivno po jednakosti dece?

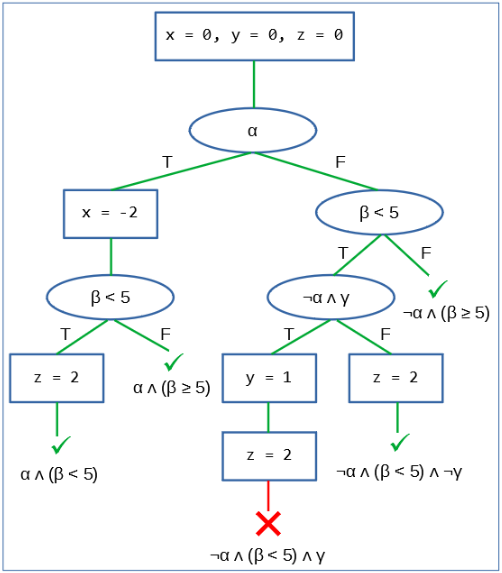
## Poređenje opštih AST

- Simboličko izvršavanje

```
int a, b, c;

// ...

int x = 0, y = 0, z = 0;
if (a)
    x = -2;
if (b < 5)  {
    if (!a && c)
        y = 1;
    z = 2;
}

assert(x + y + z != 3);
```

## Poređenje opštih AST

- Simboličko izvršavanje



## Poređenje opštih AST

- Analizirati simboličke promenljive na kraju svakog bloka

```
// x: 4, y: Y;
if (x > 3) {
    x = 1 + y;
}                   // x: 1 + y | y: Y
y = 1 + x;
                    // x: 1 + y | y: 1 + x
```

```
// x: 4, y: Y;
if (x > 3) {
    x = 1 - y;
}                   // x: 1 - y | y: Y
y = x;
y++;
                    // x: 1 - y | y: 1 + x
```

## Poređenje opštih AST

```
1: procedure UporediBlokove($b_1$, $b_2$)
2:     $gds_1 \leftarrow$ simboli iz svih predaka bloka $b_1$
3:     $gds_2 \leftarrow$ simboli iz svih predaka bloka $b_2$
4:     $lds_1 \leftarrow$ lokalni simboli za blok $b_1$
5:     $lds_2 \leftarrow$ lokalni simboli za blok $b_2$
6:     UporediSim($lds_1, lds_2$)
7:     IzvrsiNaredbe($b_1, b_2, lds_1, lds_2, gds_1, gds_2$)
8:     return UporediSim($lds_1, lds_2$) $\wedge$ UporediSim($gds_1, gds_2$)
```

## Poređenje opštih AST

```
procedure IzvrsiNaredbe($b_1$, $b_2$, $lds_1$, $lds_2$, $gds_1$, $gds_2$)
    $n_1 \leftarrow$ niz naredbi bloka $b_1$
    $n_2 \leftarrow$ niz naredbi bloka $b_2$
    $i \leftarrow j \leftarrow 0$
    $ni \leftarrow nj \leftarrow 0$
    $eq \leftarrow$ True
    while True do
        $ni \leftarrow$ indeks prve blok-naredbe u $n_1$ počev od $ni$
        $nj \leftarrow$ indeks prve blok-naredbe u $n_2$ počev od $nj$
        for $naredba \in \{n_1[x] \mid x \in [i..ni]\}$ do
            IzvrsiNaredbu($naredba, lds_1, gds_1$)
        $i \leftarrow i + ni$
        for $naredba \in \{n_2[x] \mid x \in [j..nj]\}$ do
            IzvrsiNaredbu($naredba, lds_2, gds_2$)
        $j \leftarrow j + nj$
        if $i >$ Duzina($n_1$) $\vee\ j >$ Duzina($n_2$) then
            prekini petlju
        $nb_1 \leftarrow$ izvuci blok iz naredbe $n_1[i]$
        $nb_2 \leftarrow$ izvuci blok iz naredbe $n_2[j]$
        $eq \leftarrow eq \wedge$ UporediBlokove($nb_1, nb_2$)
        $i \leftarrow i + 1$
        $j \leftarrow j + 1$
    return $eq$
```

## LICC — Language Invariant Code Comparer

- `LICC`
- `LICC.AST`
  - `LICC.AST.Builders`
  - `LICC.AST.Nodes`
  - `LICC.AST.Visitors`
- `LICC.Core`
  - `LICC.Core.Comparers`
  - `LICC.Core.Issues`
- `LICC.Visualizer`
- `LICC.Tests`
  - `LICC.Tests.AST`
  - `LICC.Tests.Core`

## LICC — Language Invariant Code Comparer

```
$ ./LICC ast [-v -c -t] source-path [-o output-path]
```

```c
int gl_y = 2;
void f(int x);
int gl_x = 3;

int main()
{
	int x = 1;
	//printf("Hello world!% d\n", x);
	return 0;
}

static int gl_z;
```

# LICC — Language Invariant Code Comparer

```
{
  "Name": null,
  "NodeType": "SourceNode",
  "Line": 1,
  "Children": [
    {
      "NodeType": "DeclStatNode",
      "Line": 1,
      "Children": [
        {
          "Modifiers": {
            "AccessModifiers": "Unspecified",
            "QualifierFlags": "None"
          },
          "TypeName": "int",
          "NodeType": "DeclSpecsNode",
          "Line": 1,
          "Children": []
        },
        {
          "NodeType": "DeclListNode",
          "Line": 1,
          "Children": [
            {
              "Pointer": false,
              "NodeType": "VarDeclNode",
              "Line": 1,
              "Children": [
                {
                  "Identifier": "gl_y",
                  "NodeType": "IdNode",
```

## LICC — Language Invariant Code Comparer

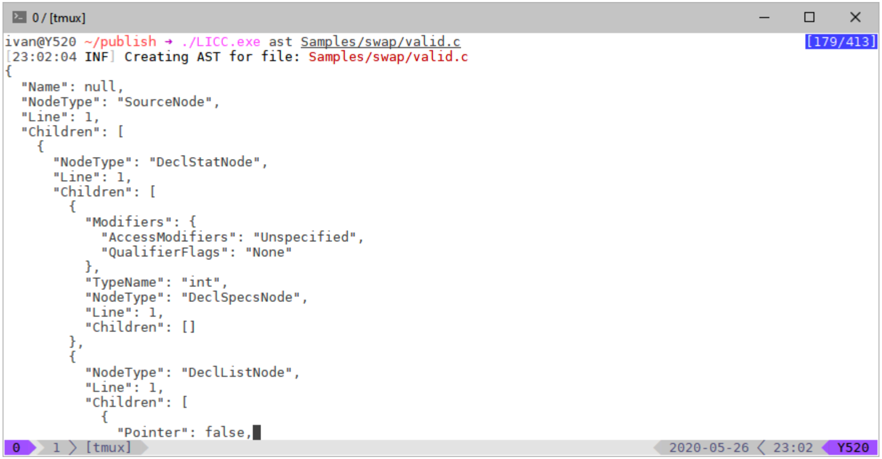
```
$ ./LICC ast [-v -c -t] source-path [-o output-path]
```

```
function fact (n)
  if n == 0 then
    return 1
  else
    return n * fact(n-1)
  end
end
```

# LICC — Language Invariant Code Comparer

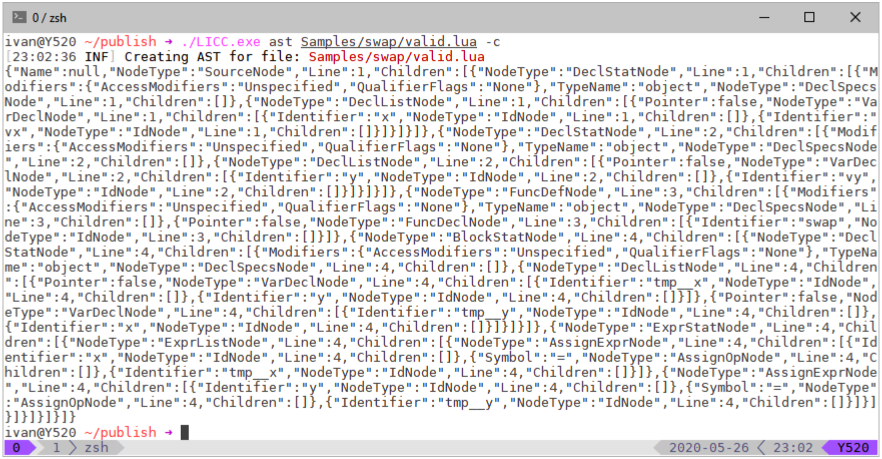
```
{"Name":null,"NodeType":"SourceNode","Line":1,
"Children":[{"NodeType":"FuncDefNode","Line":1,
"Children":[{"Modifiers":{"AccessModifiers":
"Unspecified","QualifierFlags":"None"},"TypeName":
"object","NodeType":"DeclSpecsNode","Line":1,
"Children":[]},{"Pointer":false,"NodeType":
"FuncDeclNode","Line":1,"Children":[{"Identifier":
"fact","NodeType":"IdNode","Line":1,"Children":[]},
{"IsVariadic":false,"NodeType":"FuncParamsNode",
"Line":1,"Children":[{"NodeType":"FuncParamNode",
"Line":1,"Children":[{"Modifiers":{"AccessModifiers":

...
```

## LICC — Language Invariant Code Comparer



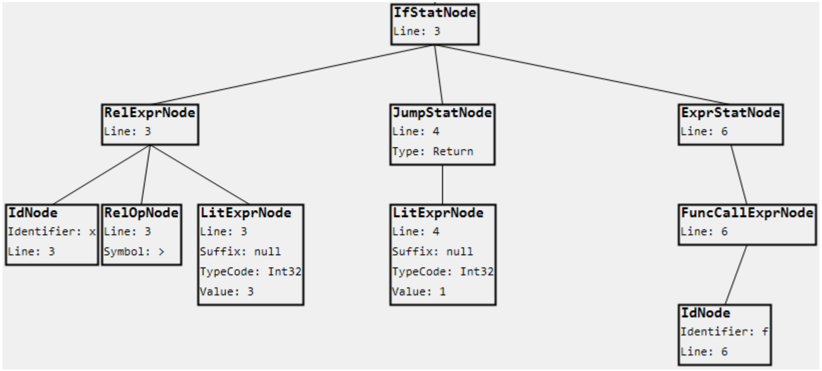
```
ivan@Y520 ~/publish ➜ ./LICC.exe ast Samples/swap/valid.c                                  [179/413]
[23:02:04 INF] Creating AST for file: Samples/swap/valid.c
{
  "Name": null,
  "NodeType": "SourceNode",
  "Line": 1,
  "Children": [
    {
      "NodeType": "DeclStatNode",
      "Line": 1,
      "Children": [
        {
          "Modifiers": {
            "AccessModifiers": "Unspecified",
            "QualifierFlags": "None"
          },
          "TypeName": "int",
          "NodeType": "DeclSpecsNode",
          "Line": 1,
          "Children": []
        },
        {
          "NodeType": "DeclListNode",
          "Line": 1,
          "Children": [
            {
              "Pointer": false,
 0  1 > [tmux]                                                          2020-05-26 < 23:02  Y520
```

## LICC — Language Invariant Code Comparer



```
0 / zsh
ivan@Y520 ~/publish ➜ ./LICC.exe ast Samples/swap/valid.lua -c
[23:02:36 INF] Creating AST for file: Samples/swap/valid.lua
{"Name":null,"NodeType":"SourceNode","Line":1,"Children":[{"NodeType":"DeclStatNode","Line":1,"Children":[{"M
odifiers":{"AccessModifiers":"Unspecified","QualifierFlags":"None"},"TypeName":"object","NodeType":"DeclSpecs
Node","Line":1,"Children":[]},{"NodeType":"DeclListNode","Line":1,"Children":[{"Pointer":false,"NodeType":"Va
rDeclNode","Line":1,"Children":[{"Identifier":"x","NodeType":"IdNode","Line":1,"Children":[]},{"Identifier":"
vx","NodeType":"IdNode","Line":1,"Children":[]}]}]}]},{"NodeType":"DeclStatNode","Line":2,"Children":[{"Modif
iers":{"AccessModifiers":"Unspecified","QualifierFlags":"None"},"TypeName":"object","NodeType":"DeclSpecsNode
","Line":2,"Children":[]},{"NodeType":"DeclListNode","Line":2,"Children":[{"Pointer":false,"NodeType":"VarDec
lNode","Line":2,"Children":[{"Identifier":"y","NodeType":"IdNode","Line":2,"Children":[]},{"Identifier":"vy",
"NodeType":"IdNode","Line":2,"Children":[]}]}]}]},{"NodeType":"FuncDefNode","Line":3,"Children":[{"Modifiers"
:{"AccessModifiers":"Unspecified","QualifierFlags":"None"},"TypeName":"object","NodeType":"DeclSpecsNode","Li
ne":3,"Children":[]},{"Pointer":false,"NodeType":"FuncDeclNode","Line":3,"Children":[{"Identifier":"swap","No
deType":"IdNode","Line":3,"Children":[]}]},{"NodeType":"BlockStatNode","Line":4,"Children":[{"NodeType":"Decl
StatNode","Line":4,"Children":[{"Modifiers":{"AccessModifiers":"Unspecified","QualifierFlags":"None"},"TypeNa
me":"object","NodeType":"DeclSpecsNode","Line":4,"Children":[]},{"NodeType":"DeclListNode","Line":4,"Children
":[{"Pointer":false,"NodeType":"VarDeclNode","Line":4,"Children":[{"Identifier":"tmp__x","NodeType":"IdNode",
"Line":4,"Children":[]},{"Identifier":"y","NodeType":"IdNode","Line":4,"Children":[]}]},{"Pointer":false,"Nod
eType":"VarDeclNode","Line":4,"Children":[{"Identifier":"tmp__y","NodeType":"IdNode","Line":4,"Children":[]},
{"Identifier":"x","NodeType":"IdNode","Line":4,"Children":[]}]}]}]},{"NodeType":"ExprStatNode","Line":4,"Chil
dren":[{"NodeType":"ExprListNode","Line":4,"Children":[{"NodeType":"AssignExprNode","Line":4,"Children":[{"Id
entifier":"x","NodeType":"IdNode","Line":4,"Children":[]},{"Symbol":"=","NodeType":"AssignOpNode","Line":4,"C
hildren":[]},{"Identifier":"tmp__x","NodeType":"IdNode","Line":4,"Children":[]}]},{"NodeType":"AssignExprNode
","Line":4,"Children":[{"Identifier":"y","NodeType":"IdNode","Line":4,"Children":[]},{"Symbol":"=","NodeType"
:"AssignOpNode","Line":4,"Children":[]},{"Identifier":"tmp__y","NodeType":"IdNode","Line":4,"Children":[]}]}]
}]}]}]}]}
ivan@Y520 ~/publish ➜ █
 0  1 〉zsh                                                                    2020-05-26 〈 23:02  Y520
```

## LICC — Language Invariant Code Comparer



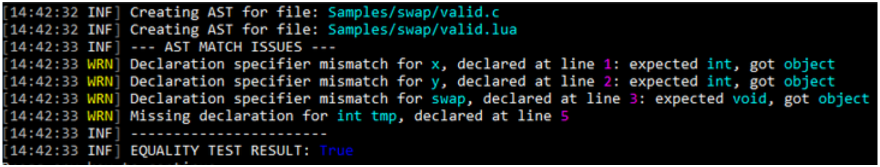
## LICC — Language Invariant Code Comparer

```
$ ./LICC cmp [-v] specification-path test-path
```

```
int x = vx, y = vy;
void swap() { int tmp = y; y = x; x = tmp; }
```

```
x = vx
y = vy
function swap()
    x, y = y, x
end
```

# LICC — Language Invariant Code Comparer



```
[14:42:32 INF] Creating AST for file: Samples/swap/valid.c
[14:42:32 INF] Creating AST for file: Samples/swap/valid.lua
[14:42:33 INF] --- AST MATCH ISSUES ---
[14:42:33 WRN] Declaration specifier mismatch for x, declared at line 1: expected int, got object
[14:42:33 WRN] Declaration specifier mismatch for y, declared at line 2: expected int, got object
[14:42:33 WRN] Declaration specifier mismatch for swap, declared at line 3: expected void, got object
[14:42:33 WRN] Missing declaration for int tmp, declared at line 5
[14:42:33 INF] -----------------------
[14:42:33 INF] EQUALITY TEST RESULT: True
```

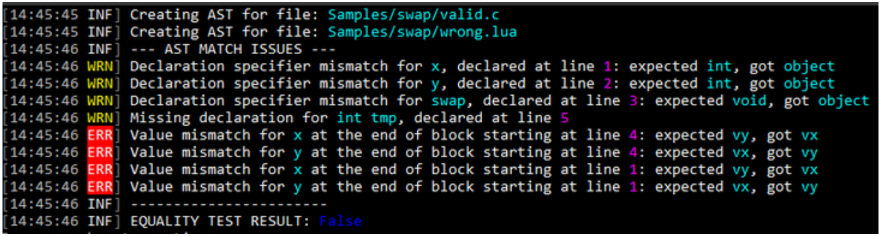
## LICC — Language Invariant Code Comparer

```
$ ./LICC cmp [-v] specification-path test-path
```

```
int x = vx, y = vy;
void swap() { int tmp = y; y = x; x = tmp; }
```

```
x = vx
y = vy
function swap()
    x, y = x, y
end
```

LICC — Language Invariant Code Comparer



```
[14:45:45 INF] Creating AST for file: Samples/swap/valid.c
[14:45:45 INF] Creating AST for file: Samples/swap/wrong.lua
[14:45:46 INF] --- AST MATCH ISSUES ---
[14:45:46 WRN] Declaration specifier mismatch for x, declared at line 1: expected int, got object
[14:45:46 WRN] Declaration specifier mismatch for y, declared at line 2: expected int, got object
[14:45:46 WRN] Declaration specifier mismatch for swap, declared at line 3: expected void, got object
[14:45:46 WRN] Missing declaration for int tmp, declared at line 5
[14:45:46 ERR] Value mismatch for x at the end of block starting at line 4: expected vy, got vx
[14:45:46 ERR] Value mismatch for y at the end of block starting at line 4: expected vx, got vy
[14:45:46 ERR] Value mismatch for x at the end of block starting at line 1: expected vy, got vx
[14:45:46 ERR] Value mismatch for y at the end of block starting at line 1: expected vx, got vy
[14:45:46 INF] -----------------------
[14:45:46 INF] EQUALITY TEST RESULT: False
```

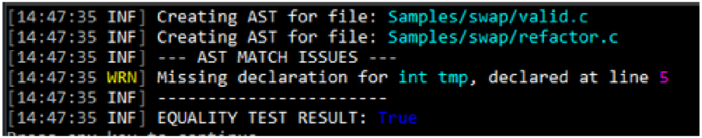
# LICC — Language Invariant Code Comparer

```
$ ./LICC cmp [-v] specification-path test-path
```

```
int x = vx, y = vy;
void swap() { int tmp = y; y = x; x = tmp; }
```

```
int x = vx, y = vy;

void swap()
{
    x = x + y;
    y = x - y;
    x = x - y;
}
```

## LICC — Language Invariant Code Comparer



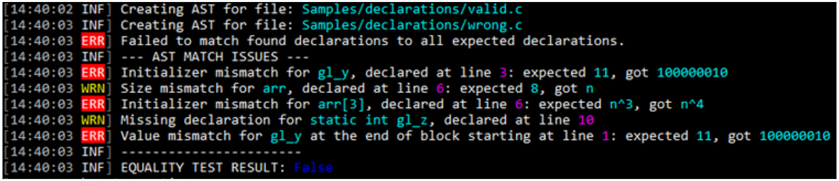
```
[14:47:35 INF] Creating AST for file: Samples/swap/valid.c
[14:47:35 INF] Creating AST for file: Samples/swap/refactor.c
[14:47:35 INF] --- AST MATCH ISSUES ---
[14:47:35 WRN] Missing declaration for int tmp, declared at line 5
[14:47:35 INF] -----------------------
[14:47:35 INF] EQUALITY TEST RESULT: True
```

## LICC — Language Invariant Code Comparer

```c
extern int n = 5;
int gl_y = 1 + 3 - 1 * 0 - 2 + 2 * n - 1;
void f(MyStruct x);
int gl_x = n + n + n;
int arr[8] = { 1, n, n * n, n * n * n };
static int gl_z;
```

```c
extern int n = 5;
int gl_x = 3 * n;
int gl_y = 1 + 3 - 1 * 0 - 2 + 2 * n - 1 + 99999999;
void f(MyStruct x);
// missing gl_z
int arr[n] = { 1, n, n * n, n * n * n * n };
```

## LICC — Language Invariant Code Comparer



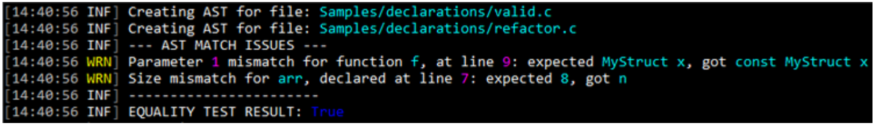
```
[14:40:02 INF] Creating AST for file: Samples/declarations/valid.c
[14:40:03 INF] Creating AST for file: Samples/declarations/wrong.c
[14:40:03 ERR] Failed to match found declarations to all expected declarations.
[14:40:03 INF] --- AST MATCH ISSUES ---
[14:40:03 ERR] Initializer mismatch for gl_y, declared at line 3: expected 11, got 100000010
[14:40:03 WRN] Size mismatch for arr, declared at line 6: expected 8, got n
[14:40:03 ERR] Initializer mismatch for arr[3], declared at line 6: expected n^3, got n^4
[14:40:03 WRN] Missing declaration for static int gl_z, declared at line 10
[14:40:03 ERR] Value mismatch for gl_y at the end of block starting at line 1: expected 11, got 100000010
[14:40:03 INF] -----------------------
[14:40:03 INF] EQUALITY TEST RESULT: False
```

## LICC — Language Invariant Code Comparer

```c
extern int n = 5;
int gl_y = 1 + 3 - 1 * 0 - 2 + 2 * n - 1;
void f(MyStruct x);
int gl_x = n + n + n;
int arr[8] = { 1, n, n * n, n * n * n };
static int gl_z;
```

```c
extern int n = 5;
static int gl_z;
int gl_x = 3*n;
int gl_y = 2*n + 1;
int arr[n] = { 1, n, n * n, n * n * n };
void f(const MyStruct x);
```

## LICC — Language Invariant Code Comparer



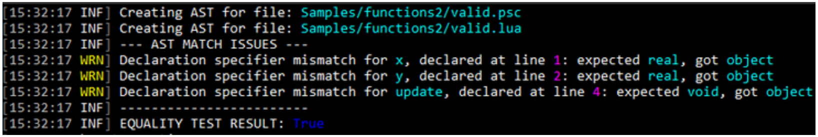
```
[14:40:56 INF] Creating AST for file: Samples/declarations/valid.c
[14:40:56 INF] Creating AST for file: Samples/declarations/refactor.c
[14:40:56 INF] --- AST MATCH ISSUES ---
[14:40:56 WRN] Parameter 1 mismatch for function f, at line 9: expected MyStruct x, got const MyStruct x
[14:40:56 WRN] Size mismatch for arr, declared at line 7: expected 8, got n
[14:40:56 INF] -----------------------
[14:40:56 INF] EQUALITY TEST RESULT: True
```

## LICC — Language Invariant Code Comparer

```
declare real x = 1.3e-2
declare real y = 1.2e-2
procedure update(dx : real, dy : real)
begin
    x = x + dx
    y = y + dy
end
```

```
x = 1.3e-2
y = 1.2e-2

function update(dx, dy)
    x, y = x + dx, y + dy
end
```

# LICC — Language Invariant Code Comparer



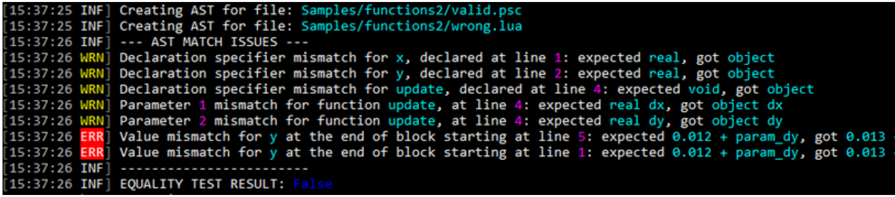
```
[15:32:17 INF] Creating AST for file: Samples/functions2/valid.psc
[15:32:17 INF] Creating AST for file: Samples/functions2/valid.lua
[15:32:17 INF] --- AST MATCH ISSUES ---
[15:32:17 WRN] Declaration specifier mismatch for x, declared at line 1: expected real, got object
[15:32:17 WRN] Declaration specifier mismatch for y, declared at line 2: expected real, got object
[15:32:17 WRN] Declaration specifier mismatch for update, declared at line 4: expected void, got object
[15:32:17 INF] ------------------------
[15:32:17 INF] EQUALITY TEST RESULT: True
```

## LICC — Language Invariant Code Comparer

```
declare real x = 1.3e-2
declare real y = 1.2e-2
procedure update(dx : real, dy : real)
begin
    x = x + dx
    y = y + dy
end
```

```
x = 1.3e-2
y = 1.2e-2

function update(dx, dy)
    x, y = x + dx, x + dy
end
```

# LICC — Language Invariant Code Comparer



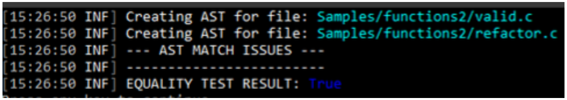
```
[15:37:25 INF] Creating AST for file: Samples/functions2/valid.psc
[15:37:25 INF] Creating AST for file: Samples/functions2/wrong.lua
[15:37:26 INF] --- AST MATCH ISSUES ---
[15:37:26 WRN] Declaration specifier mismatch for x, declared at line 1: expected real, got object
[15:37:26 WRN] Declaration specifier mismatch for y, declared at line 2: expected real, got object
[15:37:26 WRN] Declaration specifier mismatch for update, declared at line 4: expected void, got object
[15:37:26 WRN] Parameter 1 mismatch for function update, at line 4: expected real dx, got object dx
[15:37:26 WRN] Parameter 2 mismatch for function update, at line 4: expected real dy, got object dy
[15:37:26 ERR] Value mismatch for y at the end of block starting at line 5: expected 0.012 + param_dy, got 0.013
[15:37:26 ERR] Value mismatch for y at the end of block starting at line 1: expected 0.012 + param_dy, got 0.013
[15:37:26 INF] ------------------------
[15:37:26 INF] EQUALITY TEST RESULT: False
```

LICC — Language Invariant Code Comparer

```c
float x = 1.3e-2;
float y = 1.2e-2;

void update(float dx, float dy) {
    x = x + dx;
    y = y + dy;
}
```

```c
float x = 1.3e-2, y = 1.2e-2;

void update(float dx, float dy) {
    x += dx;
    y += dy;
}
```

## LICC — Language Invariant Code Comparer



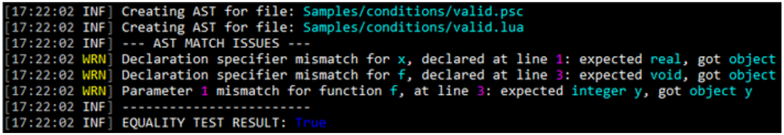
```
[15:26:50 INF] Creating AST for file: Samples/functions2/valid.c
[15:26:50 INF] Creating AST for file: Samples/functions2/refactor.c
[15:26:50 INF] --- AST MATCH ISSUES ---
[15:26:50 INF] ------------------------
[15:26:50 INF] EQUALITY TEST RESULT: True
```

## LICC — Language Invariant Code Comparer

```
declare real x = 3

procedure f(y : integer)
begin
    if y > 5 then x = 3 else x = 1
end
```

```
x = 3

function f(y)
    if y > 5 then x = 3 else x = 1 end
end
```

## LICC — Language Invariant Code Comparer



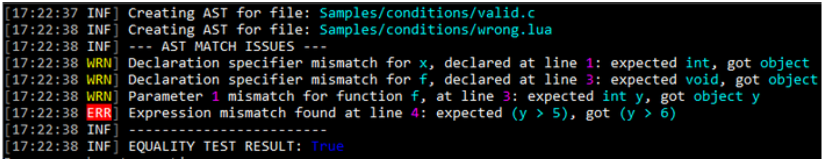
```
[17:22:02 INF] Creating AST for file: Samples/conditions/valid.psc
[17:22:02 INF] Creating AST for file: Samples/conditions/valid.lua
[17:22:02 INF] --- AST MATCH ISSUES ---
[17:22:02 WRN] Declaration specifier mismatch for x, declared at line 1: expected real, got object
[17:22:02 WRN] Declaration specifier mismatch for f, declared at line 3: expected void, got object
[17:22:02 WRN] Parameter 1 mismatch for function f, at line 3: expected integer y, got object y
[17:22:02 INF] ------------------------
[17:22:02 INF] EQUALITY TEST RESULT: True
```

# LICC — Language Invariant Code Comparer

```
int x = 3;
void f(int y) {
    if (y > 5)
        x = 3;
    else
        x = 1;
}
```

```
x = 3

function f(y)
    if y > 6 then x = 3 else x = 1 end
end
```

## LICC — Language Invariant Code Comparer



```
[17:22:37 INF] Creating AST for file: Samples/conditions/valid.c
[17:22:38 INF] Creating AST for file: Samples/conditions/wrong.lua
[17:22:38 INF] --- AST MATCH ISSUES ---
[17:22:38 WRN] Declaration specifier mismatch for x, declared at line 1: expected int, got object
[17:22:38 WRN] Declaration specifier mismatch for f, declared at line 3: expected void, got object
[17:22:38 WRN] Parameter 1 mismatch for function f, at line 3: expected int y, got object y
[17:22:38 ERR] Expression mismatch found at line 4: expected (y > 5), got (y > 6)
[17:22:38 INF] ------------------------
[17:22:38 INF] EQUALITY TEST RESULT: True
```

## Kraj!

Hvala na pažnji!

## Pitanja

???