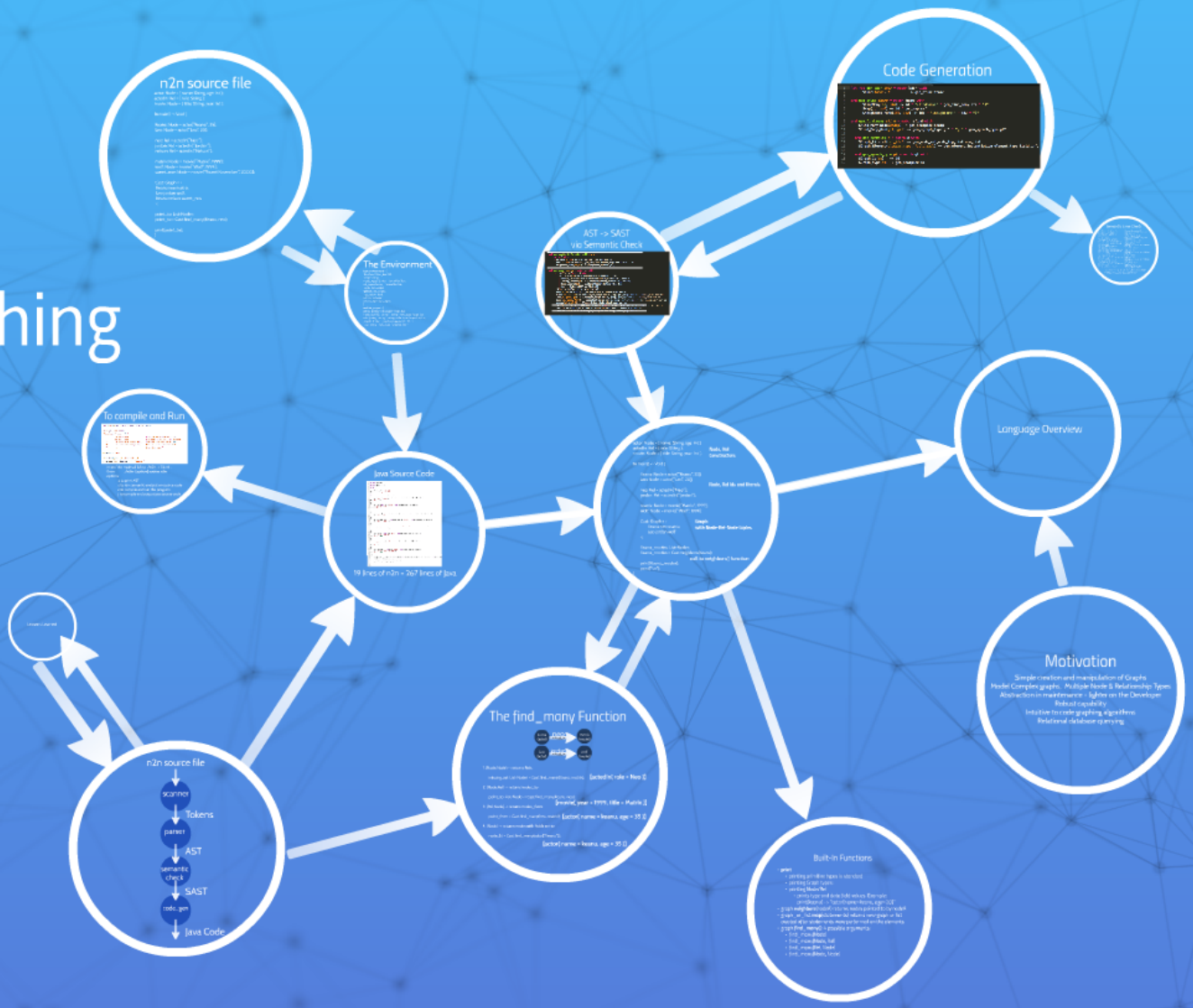
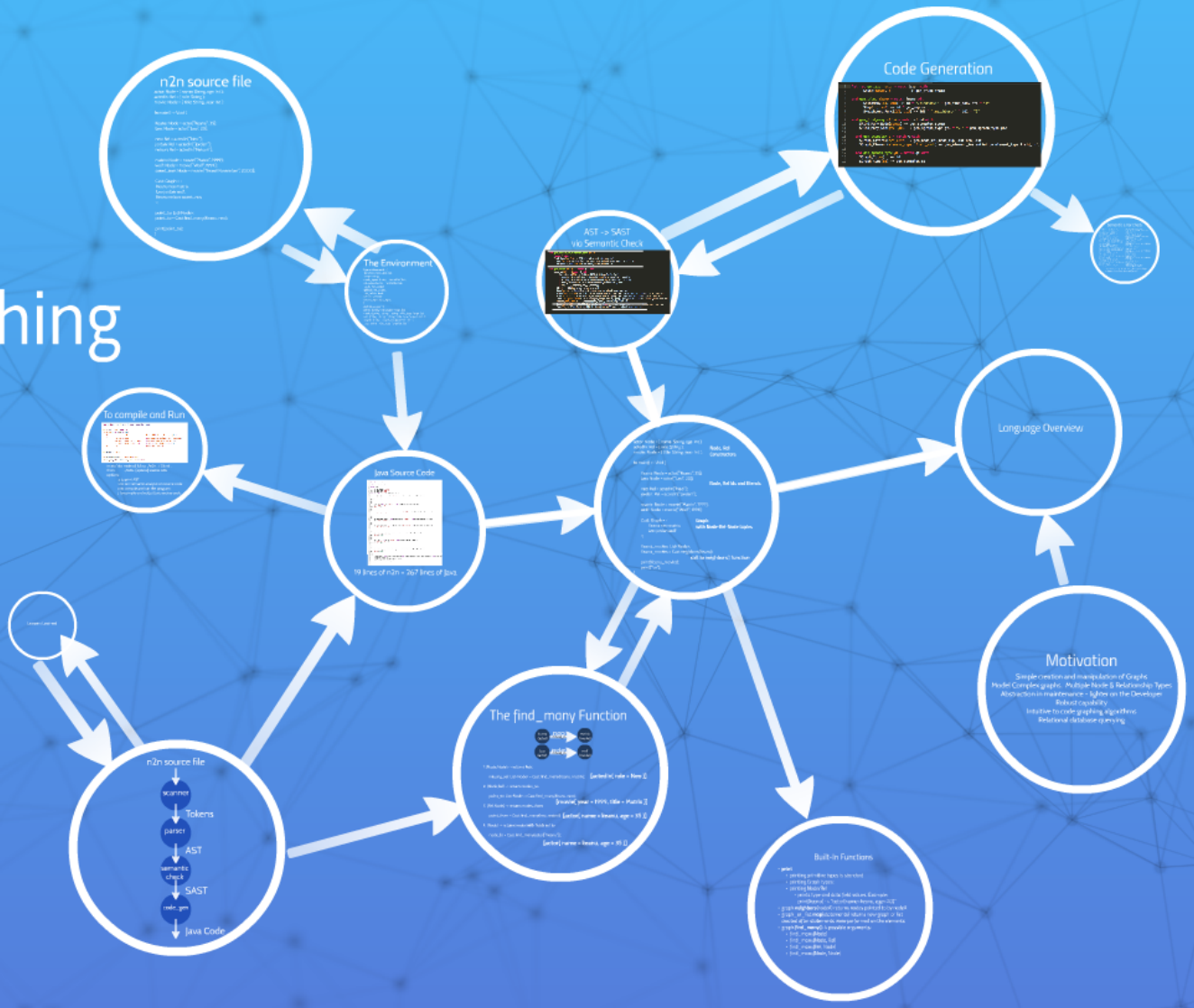
n2n:
A Relational Graphing Language

Team:
Nicholas Falba
Johan Mena
Jialun Liu
Elisheva Aeder



n2n:
A Relational Graphing Language

Team:
Nicholas Falba
Johan Mena
Jialun Liu
Elisheva Aeder



Motivation

Simple creation and manipulation of Graphs
Model Complex graphs, Multiple Node & Relationship Types
Abstraction in maintenance - lighter on the Developer
Robust capability
Intuitive to code graphing algorithms
Relational database querying

Language Overview

```
actor: Node = { name: String, age: Int };
actedIn: Rel = { role: String };
movie: Node = { title: String, year: Int };

fn main() -> Void {

    Keanu: Node = actor["Keanu", 35];
    Leo: Node = actor["Leo", 20];

    neo: Rel = actedIn["Neo"];
    jordan: Rel = actedIn["Jordan"];

    matrix: Node = movie["Matrix", 1999];
    wolf: Node = movie["Wolf", 1994];

    Cast: Graph = <
        Keanu neo matrix,
        Leo jordan wolf
    >;

    Keanu_movies: List<Node>;
    Keanu_movies = Cast.neighbors(Keanu);

    print(Keanu_movies);
    print("\n");
}
```

Node, Rel Constructors

Node, Rel ids and literals

Graph with Node-Rel-Node tuples

call to neighbors() function

Built-In Functions

- **print**
  - printing primitive types is standard
  - printing Graph types:
  - printing Node/Rel
    - prints type and data field values. Example: print(keanu) -> "actor{name=keanu, age=20}"
- graph.**neighbors**(nodeA) returns nodes pointed to by nodeA
- graph_or_list.**map**(statements) returns new graph or list created after statements were performed on the elements
- graph.**find_many**(): 4 possible arguments:
  - find_many(Node)
  - find_many(Node, Rel)
  - find_many(Rel, Node)
  - find_many(Node, Node)

The find_many Function

1. (Node,Node) -> returns Rels

```
missing_rel: List<Node> = Cast.find_many(Keanu, matrix);
```

[actedIn{ role = Neo }]

2. (Node,Rel) -> returns nodes_to

```
point_to: List<Node> = Cast.find_many(Keanu, neo);
```

[movie{ year = 1999, title = Matrix }]

3. (Rel,Node) -> returns nodes_from

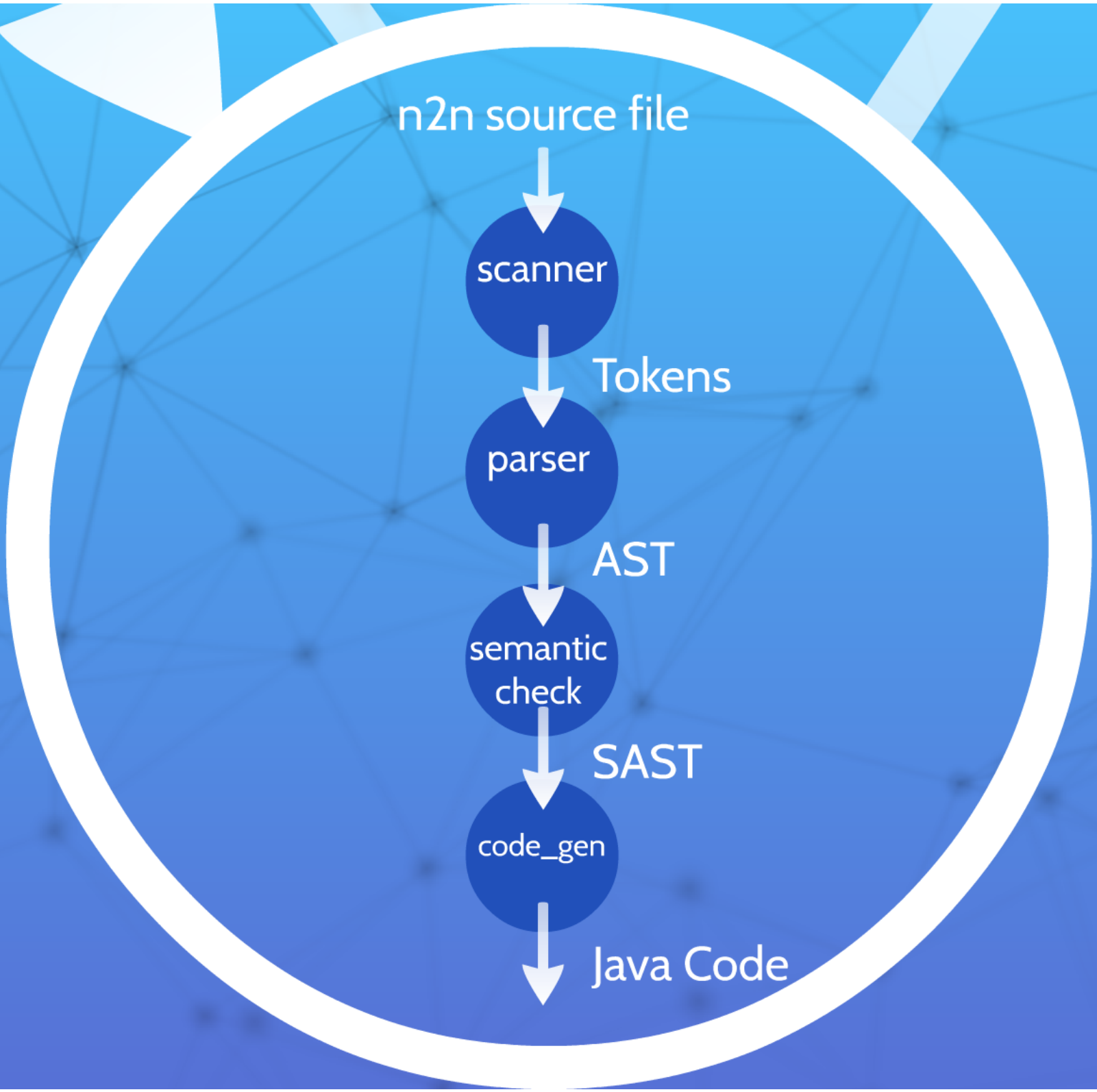
```
point_from = Cast.find_many(neo, matrix);
```

[actor{ name = keanu, age = 35 }]

4. (Node) -> returns nodes with fields set to

```
node_lit = Cast.find_many(actor["Keanu"]);
```

[actor{ name = keanu, age = 35 }]


n2n source file
scanner
Tokens
parser
AST
semantic check
SAST
code_gen
Java Code

n2n source file

```
actor: Node = { name: String, age: Int };
actedIn: Rel = { role: String };
movie: Node = { title: String, year: Int };

fn main() -> Void {

 Keanu: Node = actor["Keanu", 35];
 Leo: Node = actor["Leo", 20];

 neo: Rel = actedIn["Neo"];
 jordan: Rel = actedIn["Jordan"];
 nelson: Rel = actedIn["Nelson"];

 matrix: Node = movie["Matrix", 1999];
 wolf: Node = movie["Wolf", 1994];
 sweet_nov: Node = movie["Sweet November", 2000];

 Cast: Graph = <
  Keanu neo matrix,
  Leo jordan wolf,
  Keanu nelson sweet_nov
 >;

 point_to: List<Node>;
 point_to = Cast.find_many(Keanu, neo);

 print(point_to);
}
```

The Environment

```
type environment = {
 functions: func_decl list;
 scope: string;
 node_types: (string * formal list) list;
 rel_types: (string * formal list) list;
 locals: var_scope;
 globals: var_scope;
 has_return: bool;
 return_val: expr;
 return_type: n2n_type;
}
and var_scope = {
 prims: (string * n2n_type * expr) list;
 nodes: (string * string * (string * n2n_type * expr) list)
 rels: (string * string * (string * n2n_type * expr) list) list;
 graphs: (string * graph_component list) list;
 lists: (string * n2n_type * expr list) list }
```

AST -> SAST via Semantic Check

```
and get_sbuilt_in_function_call env f =
    match f with
    Find_Many(s, fm) -> SFindMany(s, get_sfm env fm)
    | Map (s, mf) -> SMap(s, get_smap env (check_expr env (Id(s))) mf)
    | Neighbors_Func(s1,s2) -> SNeighbors_Func(s1,s2)

and get_sexpr env ex = match ex with
    Literal(l) -> (match l with
        Int_Literal(i) -> SLiteral(SInt_Literal(i), Int)
        | Double_Literal(d)-> SLiteral(SDouble_Literal(d), Double)
        | String_Literal(s) -> SLiteral(SString_Literal(s), String)
        | Bool_Literal(b) -> SLiteral(SBool_Literal(b), Bool)
        | Any -> SLiteral(SAny, String))
    | Id(v) -> SId(v, check_expr env ex)
    | Unop(u, e) -> SUnop(u, get_sexpr env e, check_expr env ex)
    | Binop(e1, op, e2) -> SBinop(get_sexpr env e1, op, get_sexpr env e2, check_expr env ex)
    | Grop(e, grop, gc) -> SGrop(get_sexpr env e, grop, get_sgc env gc, check_expr env ex)
    | Geop (e, geop, form) -> SGeop(get_sexpr env e, geop, get_sformal form, check_expr env ex)
    | Access(str, str2) -> SAccess(str, str2, check_expr env ex)
    | Func(f) -> SFunc(get_sbuilt_in_function_call env f, check_expr env ex)
```

Semantic Error Check

Constructor actor being created
Constructor actedIn being created
Constructor movie being created
Starting to check function: main.
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking Keanu
Looking for actor constructor, finding: movie
Looking for actor constructor, finding: actor
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking Leo
Looking for actor constructor, finding: movie
Looking for actor constructor, finding: actor
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking neo
Looking for actedIn constructor, finding: actedIn
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking jordan
Looking for actedIn constructor, finding: actedIn
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking matrix
Looking for movie constructor, finding: movie
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking wolf
Looking for movie constructor, finding: movie
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking Cast
Calling check_stmt
Calling expression from check_stmtPrint function is being called
check_expr: Keanu id called
check_expr: Keanu id called
Calling check_stmt
Calling expression from check_stmtPrint function is being called
Keanu.name called
Keanu.name called
Calling check_stmt
Calling expression from check_stmtPrint function is being called
check_expr: Cast id called
check_expr: Cast id called
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking four

Fatal error: exception Semantic_check.Error("Type mismatch in local variable assignment")

Code Generation

```
let rec gen_expr expr = match expr with
    | SFunc(fname, t)          -> gen_sfunc fname

and gen_sfunc fname = match fname with
    | SFindMany(id, sfm) -> id ^ ".findMany(" ^ gen_find_many sfm ^ ")"
    | SMap(id, smf) -> id ^ gen_map smf
    | SNeighbors_Func(id1, id2) -> id1 ^ ".neighbors(" ^ id2 ^ ")"

and gen_find_many sfind = match sfind with
  | SFind_Many_Node(scomp) -> gen_scomplex scomp
  | SFind_Many_Gen(gt1, gt2) -> gen_sgraph_type gt1 ^ ", " ^ gen_sgraph_type gt2

  and gen_scomplex c = match c with
  | SGraph_Literal(nrn_list) -> gen_node_rel_node_tup_list nrn_list
  | SGraph_Element(element_type, field_info) -> gen_element_instantiation element_type field_info

  and gen_sgraph_type gt = match gt with
  | SGraph_Id(id)    -> id
  | SGraph_type(s1) -> gen_scomplex s1
```

Java Source Code

```java
package com.n2n;

import java.util.*;

class Main {

    String movie = "movie";
String actedIn = "actedIn";
String actor = "actor";
public static void main(String[] args) {

        Node Keanu = new Node("actor", new HashMap<String, Object>() {{
    put("name", "Keanu");
put("age", 35);

}}
);
    Node Leo = new Node("actor", new HashMap<String, Object>() {{
    put("name", "Leo");
put("age", 20);

}}
);
    Relationship neo = new Relationship("actedIn", new HashMap<String, Object>() {{
    put("role", "Neo");

}}
);
    Relationship jordan = new Relationship("actedIn", new HashMap<String, Object>() {{
    put("role", "Jordan");

}}
);
    Relationship nelson = new Relationship("actedIn", new HashMap<String, Object>() {{
    put("role", "Nelson");

}}
);
    Node matrix = new Node("movie", new HashMap<String, Object>() {{
    put("title", "Matrix");
put("year", 1999);

}}
);
    Node wolf = new Node("movie", new HashMap<String, Object>() {{
    put("title", "Wolf");
put("year", 1994);

}}
);
    Node sweet_nov = new Node("movie", new HashMap<String, Object>() {{
    put("title", "Sweet November");
put("year", 2000);

}}
);
    Graph Cast = new Graph(Arrays.asList(new Graph.Member<>(Keanu, neo, matrix), new Graph.Mem
    Set<Node> point_to;
    point_to = Cast.findMany(Keanu, neo);
    System.out.print(point_to);
    }

}
```

19 lines of n2n = 267 lines of Java

To compile and Run

```
type action = Ast | Sast | Java | Compile | Help

let usage (name:string) =
  "usage:\n" ^ name ^ "\n" ^
    "        -a source.n2n                (Print AST of an n2n source)\n"^
    "        -s source.n2n                (Run Semantic Analysis over source)\n"^
    "        -j source.n2n [target.java]  (Generate Java code for n2n)\n"^
    "        -c source.n2n [target.out]   (Compile n2n to executable)\n" ^
    "        -h                           (Shows this menu)"

let javac = "javac"

let backend_path = "../backend/src/"
let target_path = backend_path ^ "com/n2n/"
```

in src/ do: make all && cp ./n2n ../ && cd ..

then: **./n2n -[option] source.n2n**

options:

- a to print AST
- s to run semantic analysis on source code
- c to compile and run the program
- j to compile and output java source code

Lessons Learned

n2n:
A Relational Graphing Language

Team:
Nicholas Falba
Johan Mena
Jiglun Liu