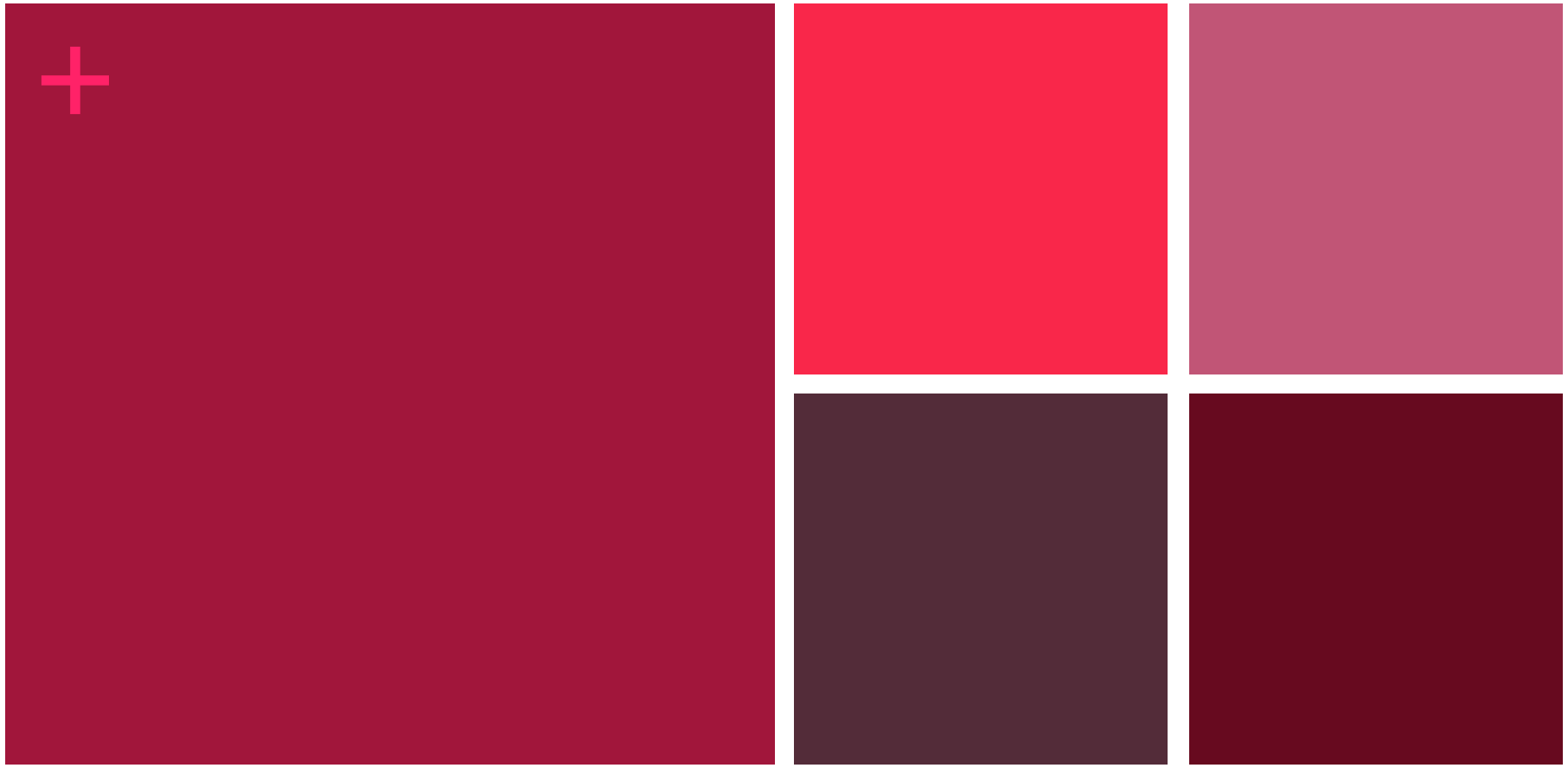


The HARPO Verifier

Status 2018 October 15

Verifying the Correctness of HARPO Programs

Presented at NECEC 2018, St. John's NL

Inaam Ahmed

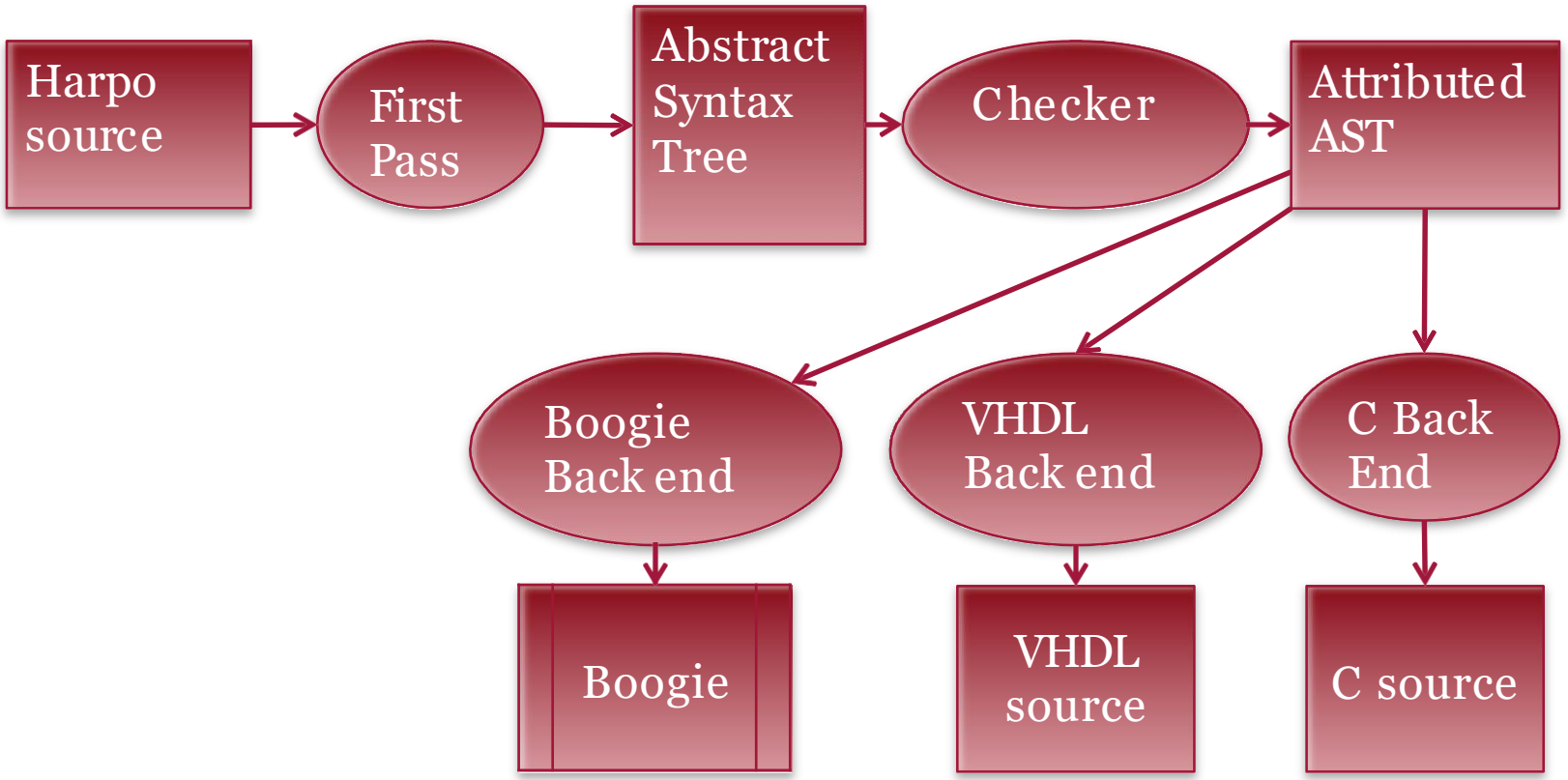
Computer Engineering Research Labs, Dept. ECE, MUN

2018 November 13

Review of Harpo

- HARdware Parallel Objects
- An executing Harpo program is a static network of objects.
  - Each contains 0 or more threads
  - Each contains 0 or more locations or arrays and connects to 0 or more other objects.
  - Objects communicate via client/server rendezvous.
- Objects are implemented as software (e.g. concurrent C) or hardware (e.g. FPGA).

Data flow architecture



HARPO Program

- HARPO Programs

  Program (ClassDecl | IntfDecl | ObjectDecl | ConstDecl |;)

  - Interfaces

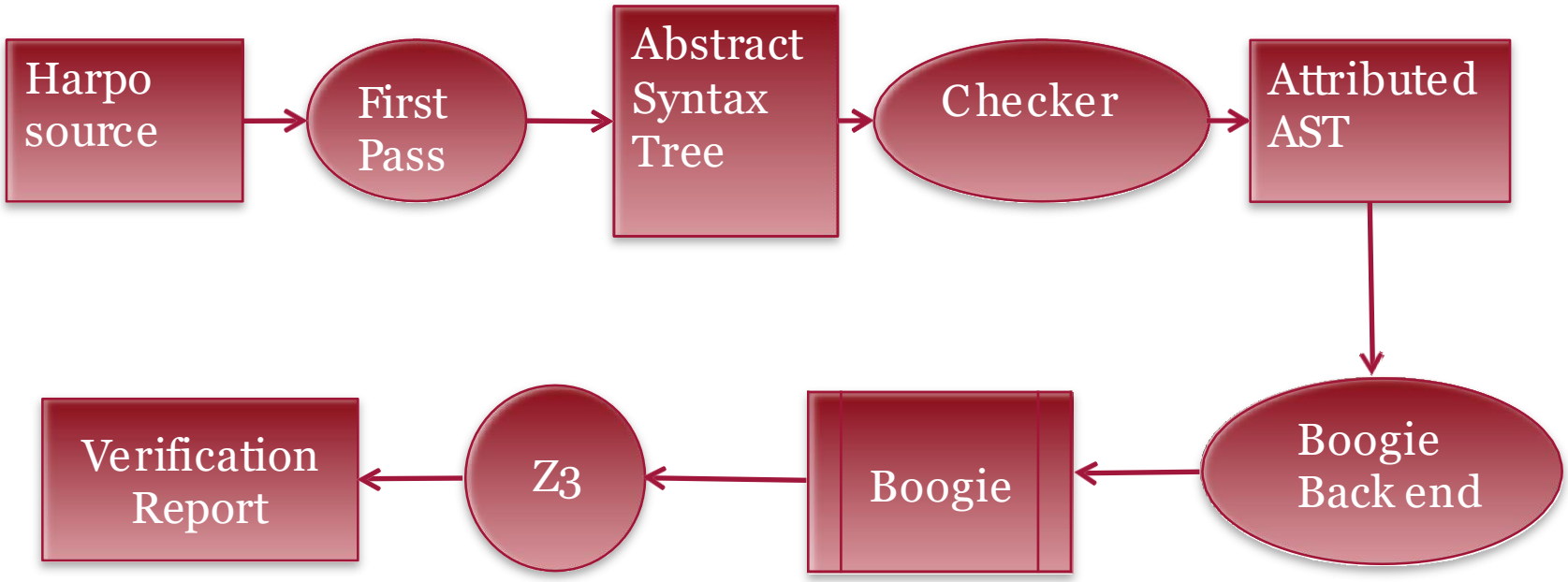
    Contains method declaration with *ghost* parameters
  - Classes

    Classes have the annotations *claim* and *invariant*
  - Objects

    Ghost objects are annotated with ghost keyword
  - Constants

    Constants having HARPO primitive type

Program Translation into Boogie



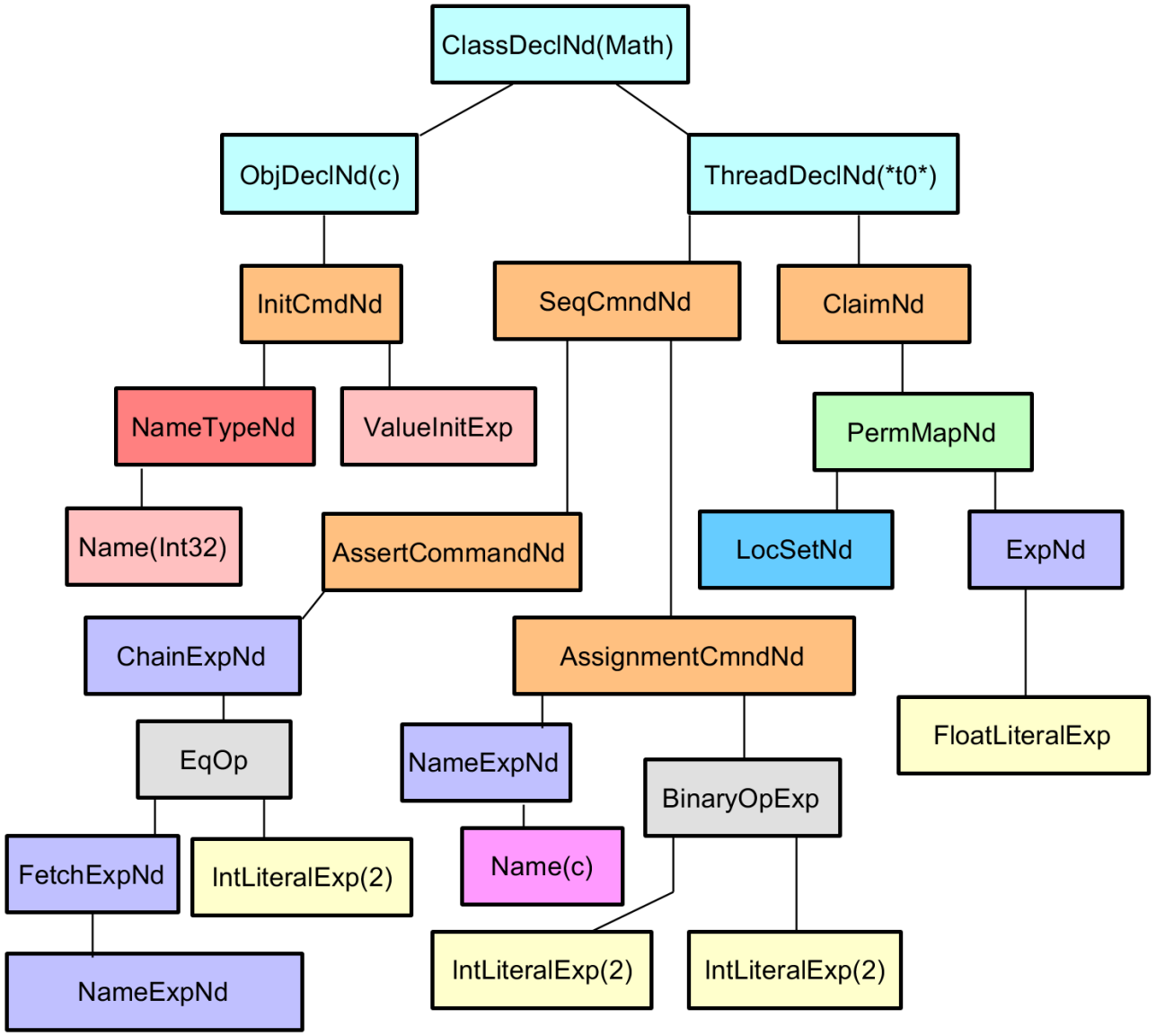
Running example: Input

```
(class Math
      obj c:int32 :=0;
      (thread (*t0*)claim c
            c:=2+2;
            assert c=4;
       thread)
class)
```

class AST {// slightly simplified

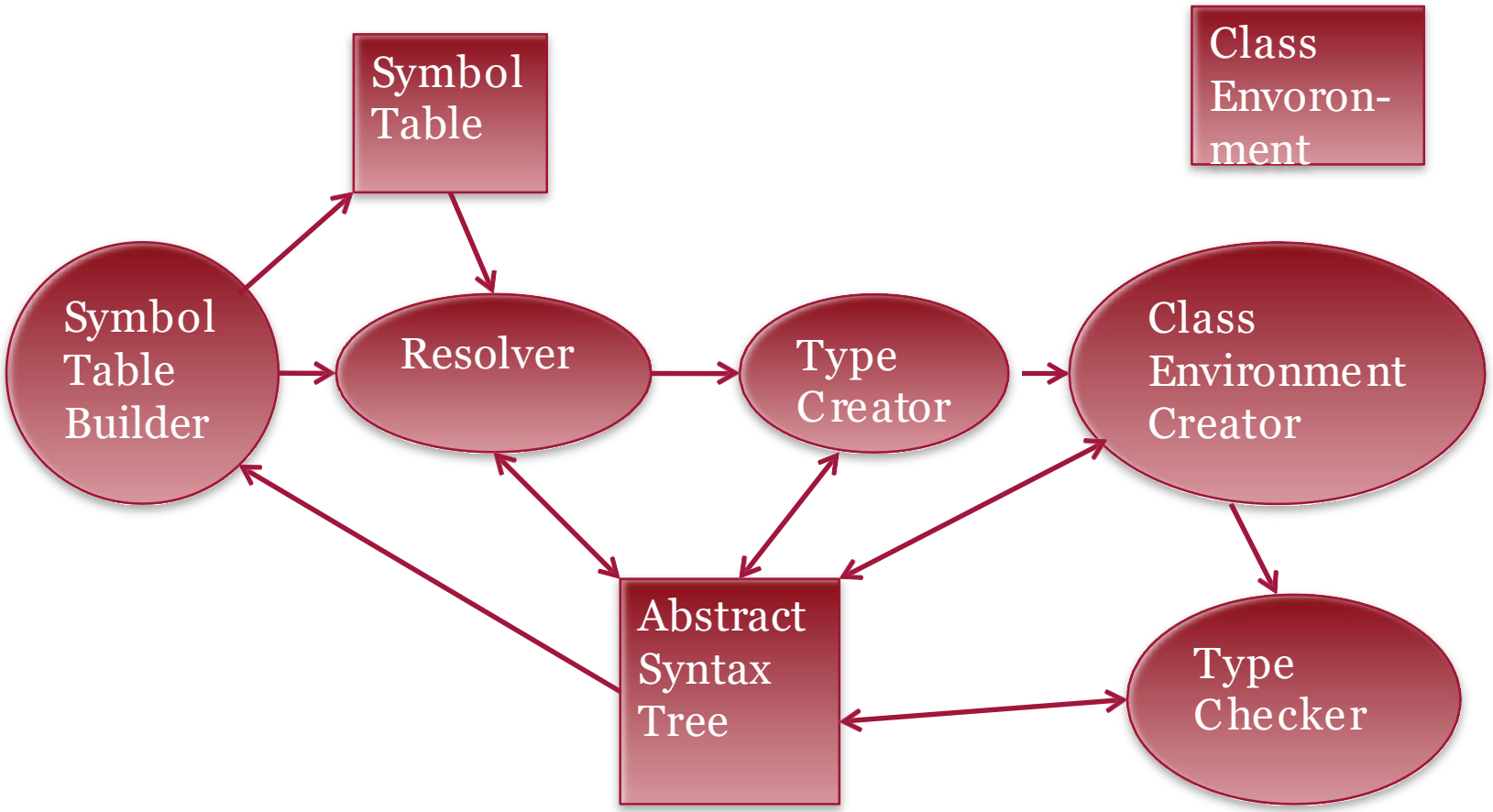
```
[ClassDeclNd(
   ObjDeclNd[c](
      NamedTypeNd( Int32 ) : loc{Int32},
      ValueInitExpNd(IntLiteralExpNd(0):Int32):Int32),
   ThreadDeclNd[t#0](
      ThrdClaimNd( [ NameExpNd( c ) : Int32),
       SeqCommandNd(
        AssertCmdNd(
         ChainExpNd(
         [ LessOp ],
               [ FetchExpNd( NameExpNd( c ) : loc{Int32} ) : Int32,
IntLiteralExpNd( 20 ) : Int32 ] ) : Bool ) ) ) ) ) )
,]
```

AST after first pass



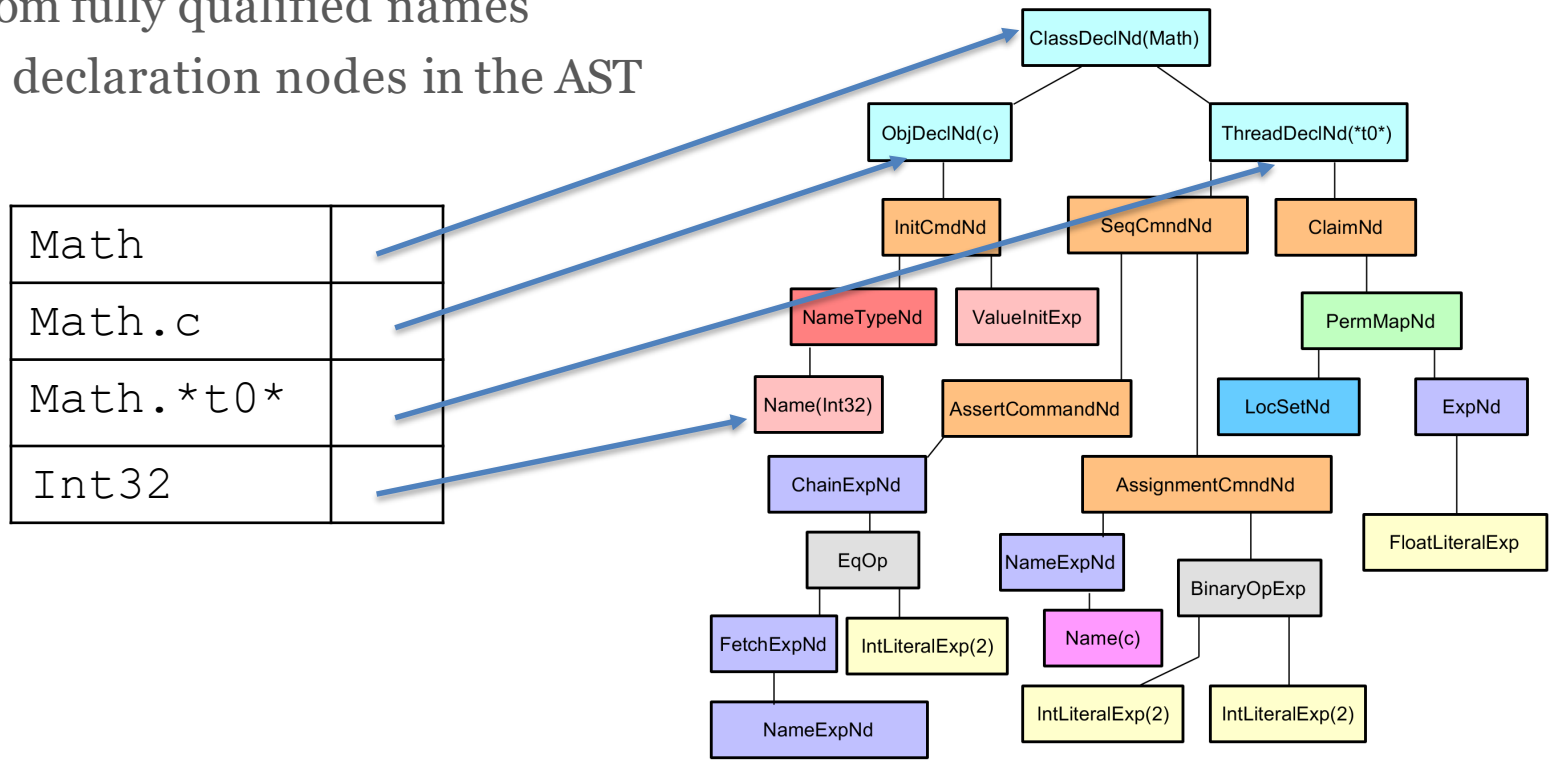
Checker Passes

modify and add attributes to the AST in-place



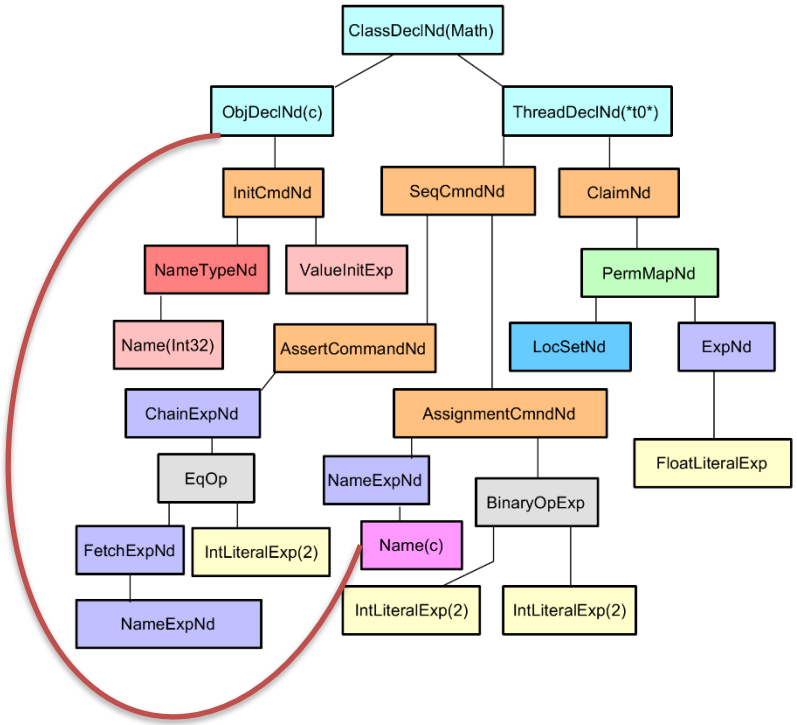
Running example after TableCreator

- Creates a map
  - from fully qualified names
  - to declaration nodes in the AST



Resolver pass

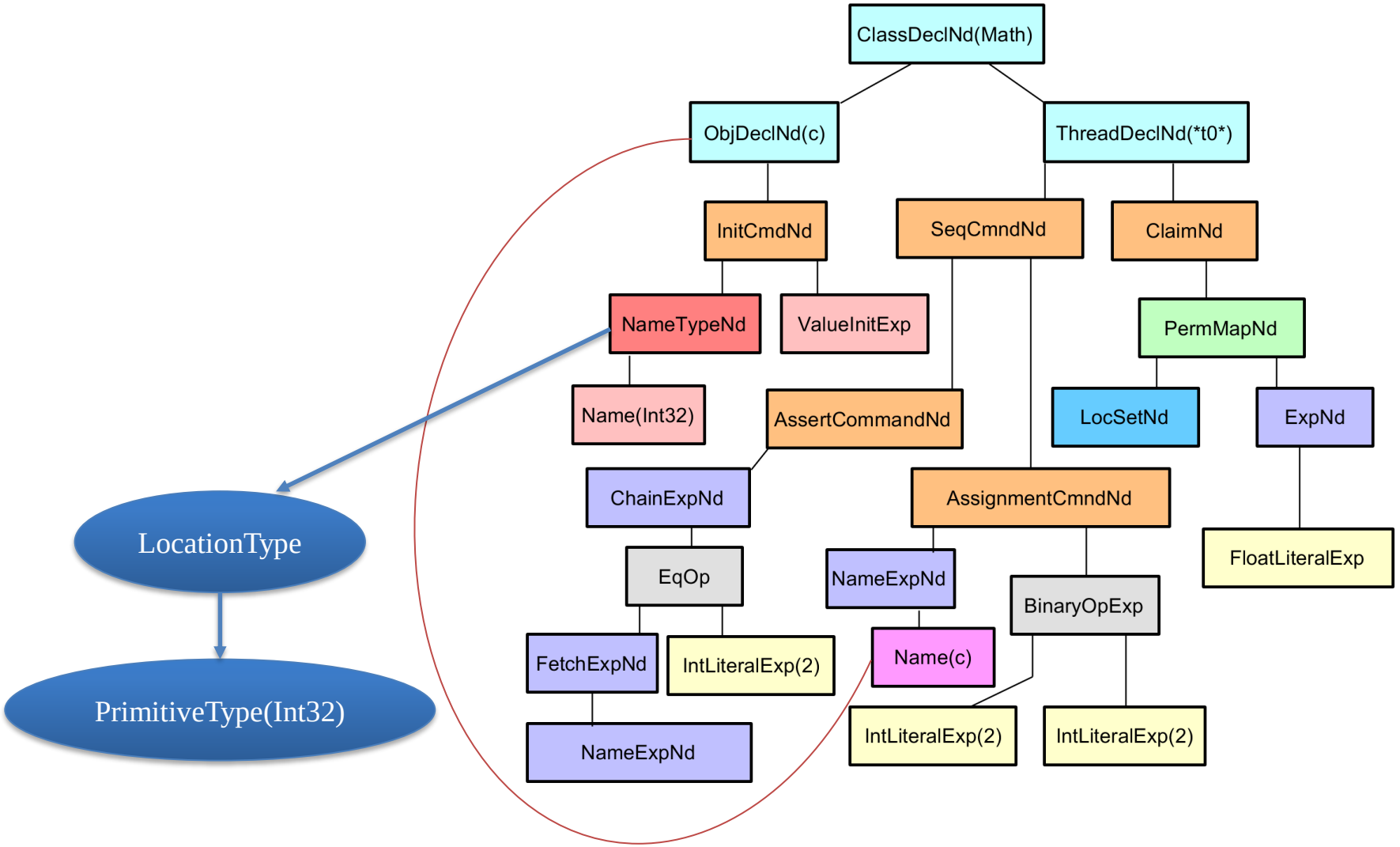
- Links each Name node to a declaration
- After this, the symbol table is no longer needed!



Type Creator Pass

- Creates types as needed.
- Associates all Type nodes (except NoType nodes) witha type.
- TypeNodes –syntactic representation of types
- Types –semantic representation of types

Running Example after Type Creation Pass



Boogie Back-end

- Goal: Translate Harpo to Boogie that can run on Boogie Verification Tool
  - A standard approach for program verification is to use the theorem proving
  - Source code with program specifications is converted into the verification conditions
  - Theorem prover use the verification to determine the correctness of the program
- Longer term goal: Implement the Boogie backend into an interactive tool

Complexity

- Generating the verification conditions is complex task
- Mitigate the complexity by dividing the task into two steps:
  - Convert source into IVL (Boogie)
  - Let Boogie talk with theorem prover (Z3)

Memory Model

- We are using heap memory model which maps the fields and object references to values.
  - Objects Heap
  - Array Heap

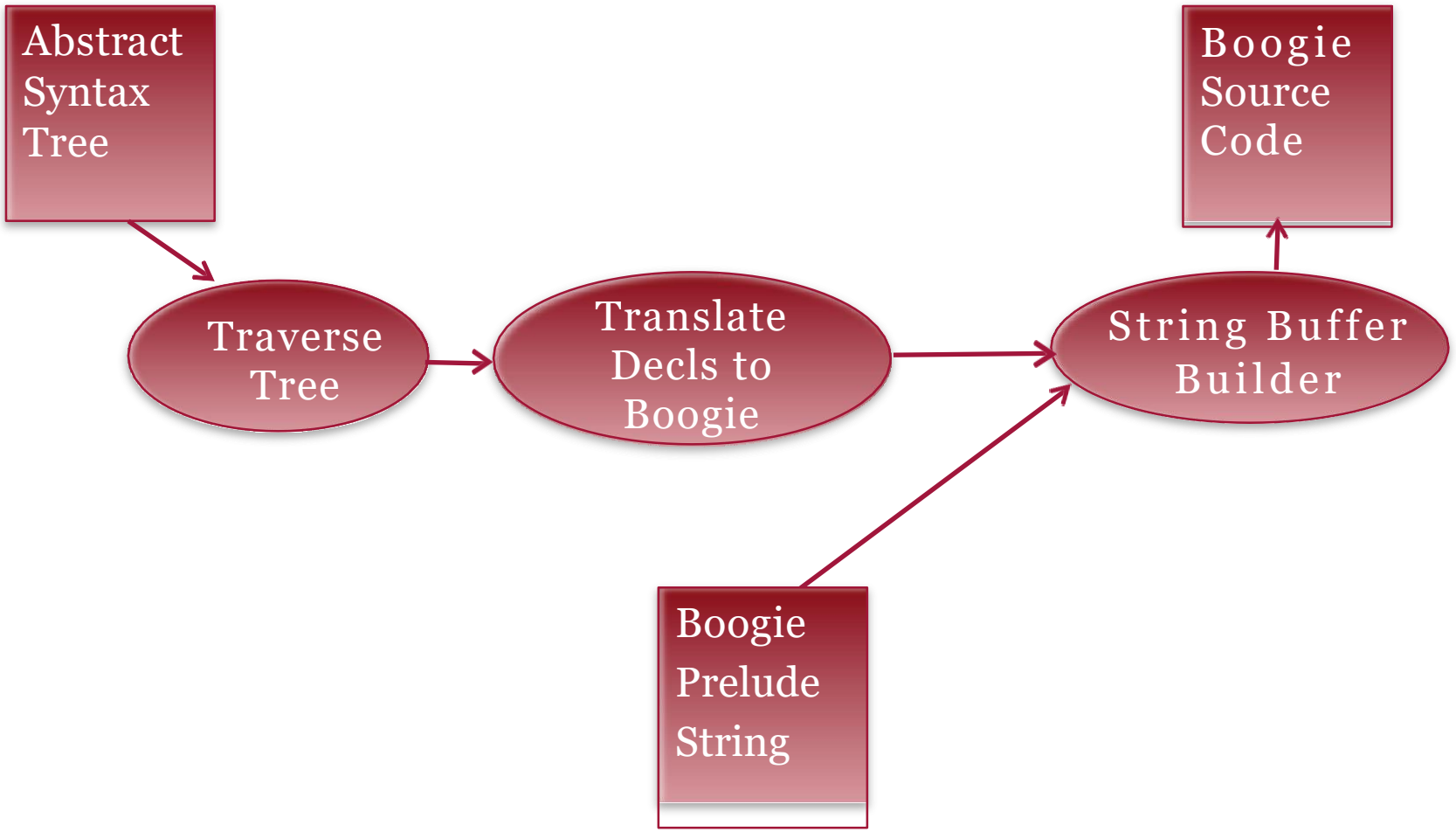
	x	**y**	**z**	**bool**	**m**
<*Ref*>	12	13		T	
<*Ref*>	15		16		4.0

Ref	*-2*	*-1*	*0*	*1*	*2*	*3*
Boolean	T	T	T	F	F	T
Integer	1	2	56	7	98	62
Real	2.6	76.0	8.6	233.6	8.8	98.0

Boogie Prelude

- Independent of the source program being translated.
- Contains some important properties such as,
  - modeling memory
  - reference types
  - type axioms
  - array length and permission type
- Required for translation of HARPO program.
- Final output program consists of boogie prelude and the translation of specific HARPO program.

Boogie Back End Pass



A Few Translations

Program components	***HARPO Code***	***Boogie Code***
Class	*(**class** A* *class members **class**)*	***const unique** A: ClassName;*
Interface	*(**interface** B* *Interface members **interface**)*	***const unique** B: ClassName;*
Field	***obj** h: Int8**: =** Exp*	***const unique** A.h: Field int;*
Constants	***const** c: real16: = Exp_h*	***const** c: real;* ***axiom** x == Exp_b*
While Statement	***(while** G_h **invariant** I statement(s) **while)***	***while** (G_b) **invariant** I* *Boogie statements*
Thread	***(thread** T **claim** init_Permission block* ***thread)***	***Procedure** A.T(this: Ref)* ***Modifies** H. ArrayH;* ***Requires** dtype(this) <: C;* *{…thread block* *…claim translation}*

Listing 0 Translated

- Boogie Source

```
                                        //prelude
1.  type Ref;
2.  type Field a;
3.  type HeapType = <a> [Ref,Field a]a;
4.  var Heap:HeapType;
5.  type Perm = real ;
6.  type PermissionType = <a>[Ref, Field a]Perm;
                        // Specific translated part of Listing 0
7.  type className;
8.  function dtype(Ref) returns (className);
9.  const unique Math:className;
10. const unique Math.c : Field int;
11. procedure Math.t0(this:Ref)
12. modifies Heap;
13. { var Permission : PermissionType where
      a. (forall <a> r:Ref, f : Field a :: Permission[r,f] == 0.0  ) ;
14. var oldHeap, tmpHeap : HeapType ;
    Permission[this, Math.c] := 1.0;
    assert Permission[ this, Math.c ] == 1.0 ;
17. Heap[this,Math.c]:= 2+2;
    assert Permission[ this, Math.c ] > 0.0 ;
19. assert Heap[this,Math.c]==4 ;}
```

Conclusion and Future Work

- Automated the process of translation
- Implementation will result an independent backend of verifier
- Support concurrent threads verification
- Some Language features, like functions and predicates, are needed to be added
- Develop a verification tool like Dafny

The End