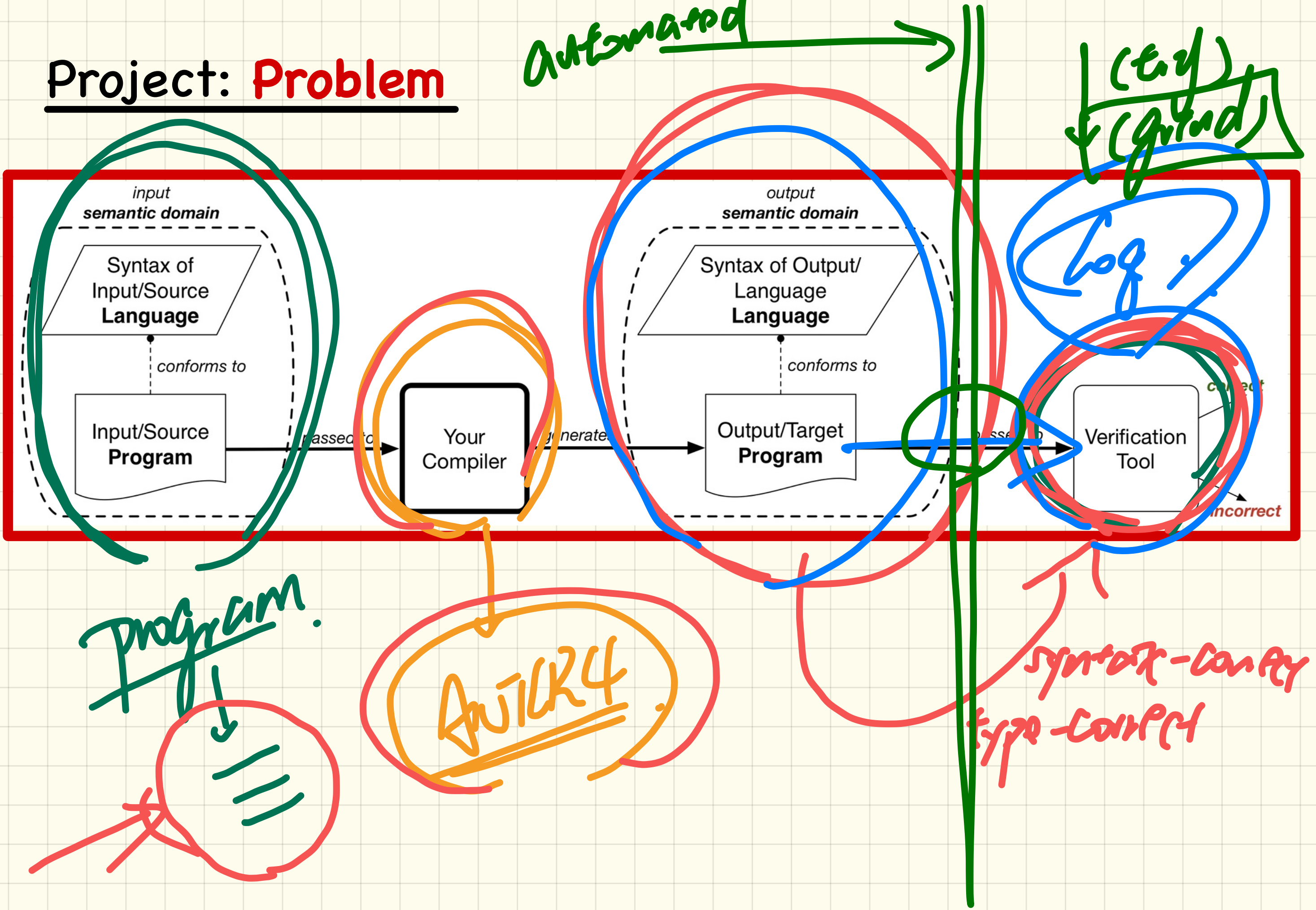
LECTURE 13

MONDAY, FEBRUARY 24

Project: Problem



Project: Milestones

Milestones of the Project

1. Confirm Team Member(s) and the Target Verification Tool

By the end of **Tuesday, March 3**, submit a plain text file `team.txt` for your team via the Prism account of a memer:

```
submit 4302 Project team.txt
```

2. Demonstrate Proficiency with the Chosen Target Verification Tool [**3%**]

- On **Thursday March 5 or Friday March 6** (about 2 weeks after the project is released), your team is required to meet with Jackie and demonstrate that you are familiar with the verification tool (and its specification language) you choose.
- During the meeting, you must demonstrate (using your computer) **5 non-trivial examples** (ones that show the various target language features that are relevant to your compilation) of verification. For each example, you will demonstrate how to verify it using the tool.
- **In this meeting, Jackie may suggest specific tasks that your team should complete and will be included in the evaluation of the second milstone.**

3. Demonstrate Satisfactory Progress on the Compiler [**5%**]

- On **Thursday March 19 or Friday March 20** (about 1 month after the project is released), your team is required to meet with Jackie and demonstrate a working version of your compiler on the following basic features of the source language features (of syntax of your own design), including:
  - variable declarations
  - variable assignments
  - variable references (i.e., referring to declared variables in expressions)
  - arithmetic, relational, and logical expressions
  - conditionals
  - specification (e.g., preconditions, postconditions, invariants, property assertions) in input programs that guide the target verification
- During the meeting, you must demonstrate (using your computer) **5 non-trivial examples** (ones that show the above source language features) of verification. For each example, you will demonstrate how to verify it using your compiler (e.g., given an input file, your tool will compile it into another file which can be taken as input by the target verification tool).
- **In this meeting, Jackie may suggest specific tasks that your team should complete and will be included in the evaluation of the final project in April.**

These two milestones are meant to make sure that you are on the right track. Based on your demo, Jackie will give you feedback.

April 6

Project: Verification Tool

You must use the ANTLR4 Parser Generator (for Java) to build your compiler.

For the target verification tool, you must choose from one of the following (and confirm by the due date; see Section 7), where a suggested starting point is provided for each tool:

- PVS https://pvs.csl.sri.com/
  - ◇ This tool is available in Prism and used by *EECS4312 Software Engineering Requirements.*
  - ◇ More info here: https://wiki.eecs.yorku.ca/project/sel-students/p:tutorials:pvs:start
- Coq https://coq.inria.fr/
- Isabelle https://isabelle.in.tum.de/
- Alloy https://alloytools.org/
- PAT https://pat.comp.nus.edu.sg/
- Spin http://spinroot.com/spin/whatispin.html
- Z3 https://rise4fun.com/z3/tutorial

Nonetheless, if there is a particular verification tool which you prefer to working with but it is not in the above list, speak to Jackie by the due date of submitting the `team.txt` file (See Section 7 for the due date).

NASA. most u.f.c. Cambridge Robin.

Paradigms of Verification (1)

Model Checking:

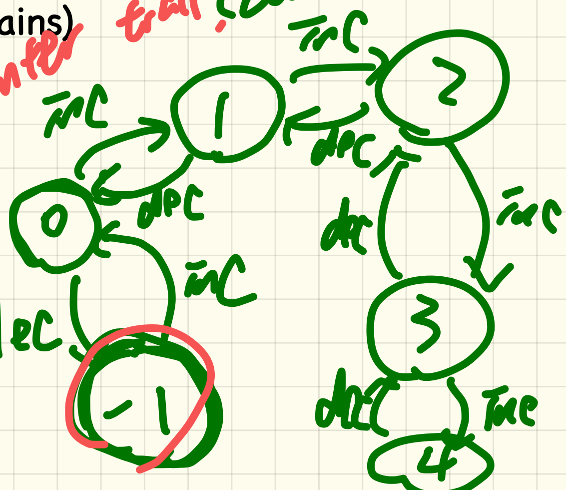
- A transition graph is built based on all possible states.
- An algorithm is run to ensure that certain properties are satisfied
- Automated
- State Explosion:

  not terminating if the state space is huge (combinatorial on sizes of variable domains)

Example:

```
inc
  require
    x < 5
  do
    x := x + 1
  end
```

```
dec
  require
    x ≥ 0
  do
    x := x - 1
  end
```



Paradigms of Verification (2)

Theorem Proving

- System is encoded as predicates (e.g., Hoare Triples in EECS3311)
- A proof system of deductions (axioms, lemmas) is used to prove that the system entails certain properties.
- Manual (EECS1090 proofs on a computer)
- Complete input domains of variables can be encoded.

Example:

```
swap
  require
    x > y
  do
    temp := x; x := y; y := temp
  ensure
    y > x
  end
```

Example:

Given $\neg(\neg p) \equiv p$

$p \Rightarrow q \equiv \neg p \vee q$

Prove: $\neg p \Rightarrow q \equiv p \vee q$

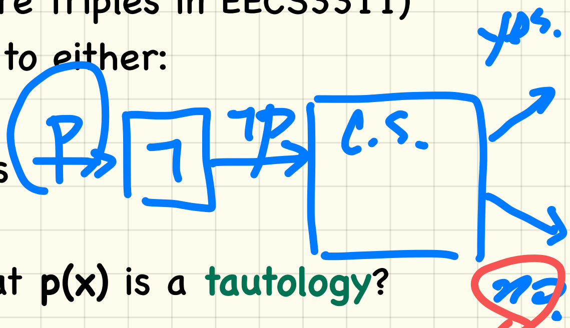
Paradigms of Verification (3)

Constraint Solving

- System is encoded as predicates (e.g., Hoare Triples in EECS3311)
- Given predicate p(x), an algorithm is used to either:

    (1) find a witness x s.t. p(x) is true.

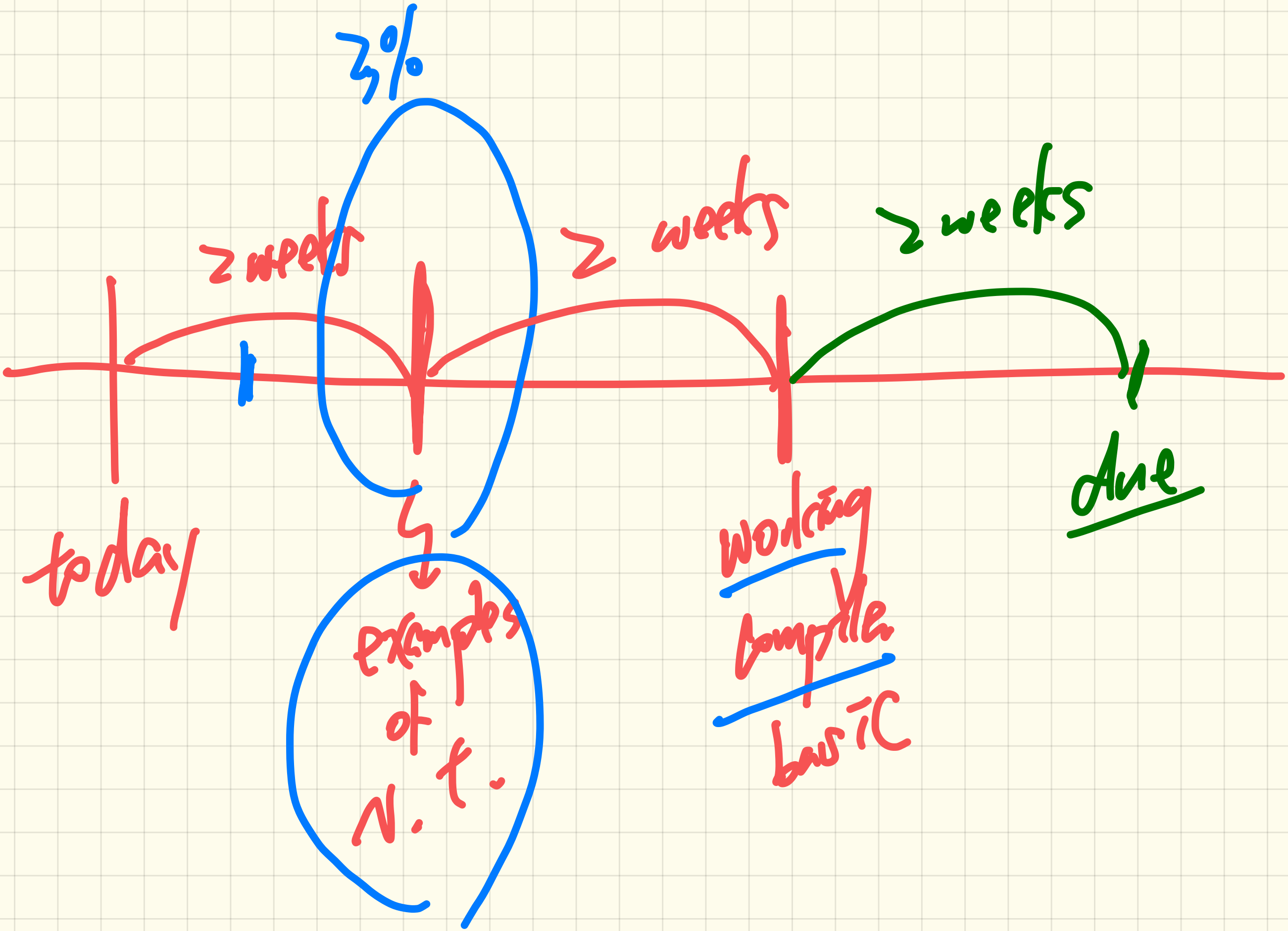
    (2) report that no such witness exists



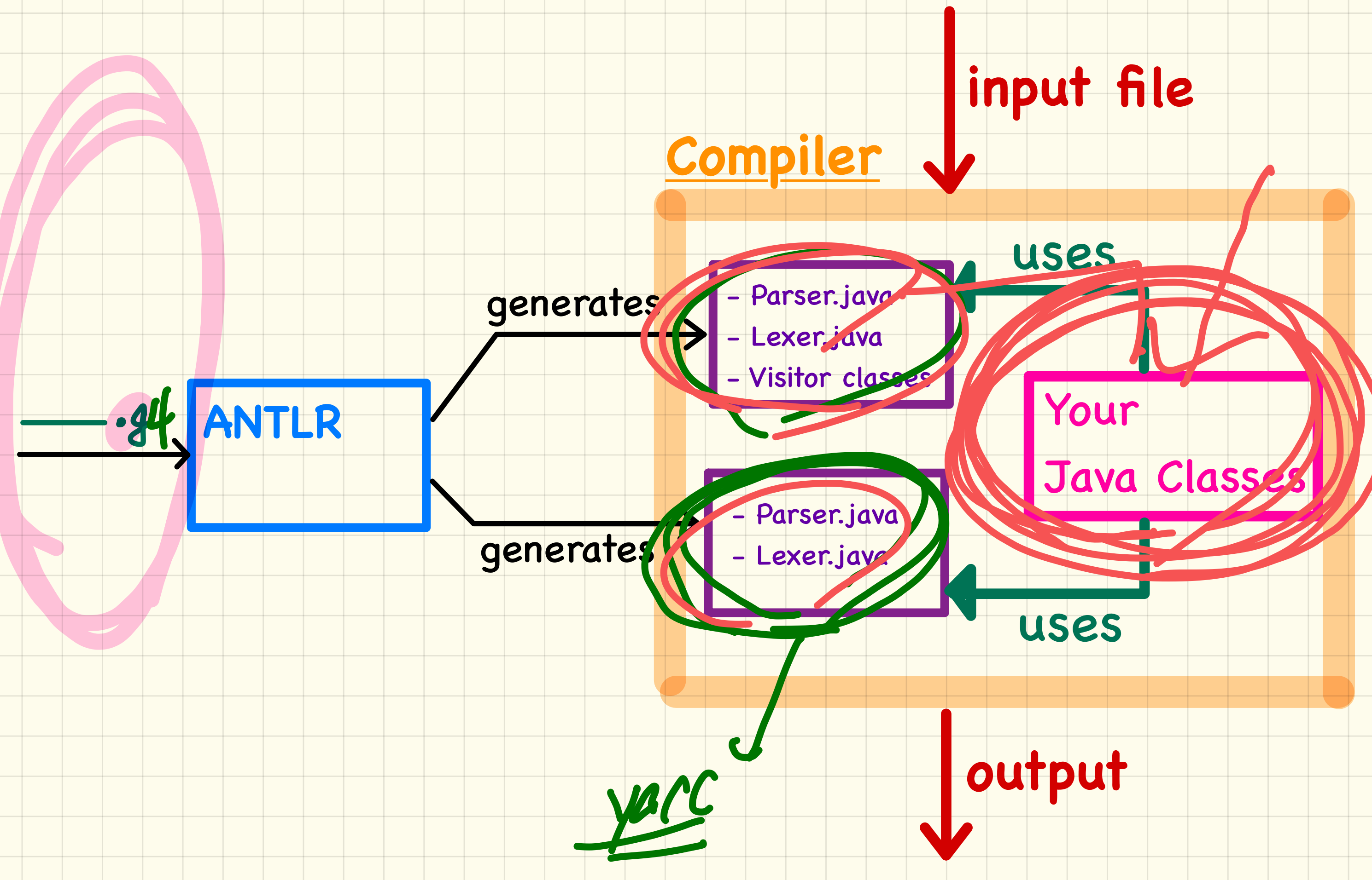
- Automated
- How do we then use a solver to prove that p(x) is a tautology?
- Combinatorial Explosion on Variable Domains

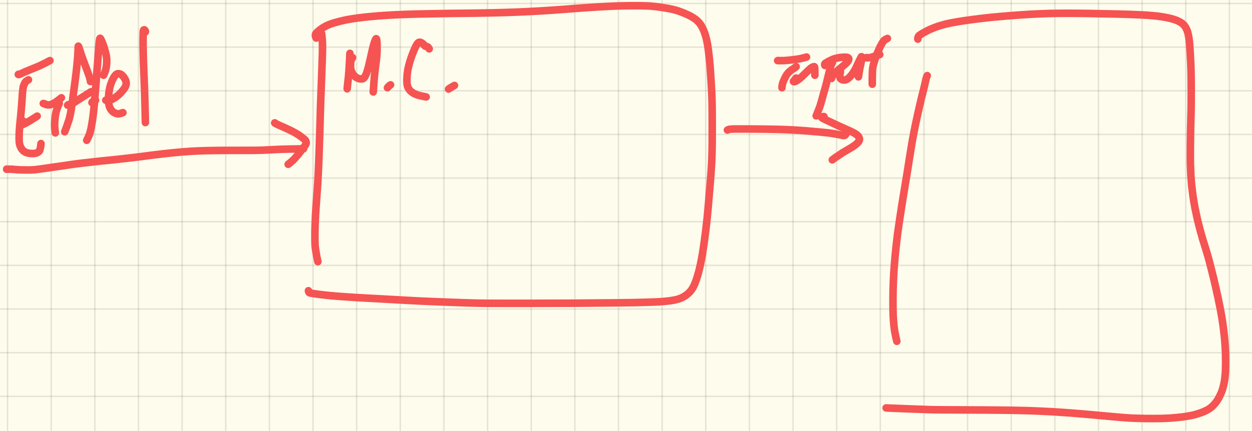
Example:

Prove that $p \wedge q \equiv q \wedge p$


3%
2 weeks
2 weeks
2 weeks
today
examples
of
N. f.
working
compiler
basic
due

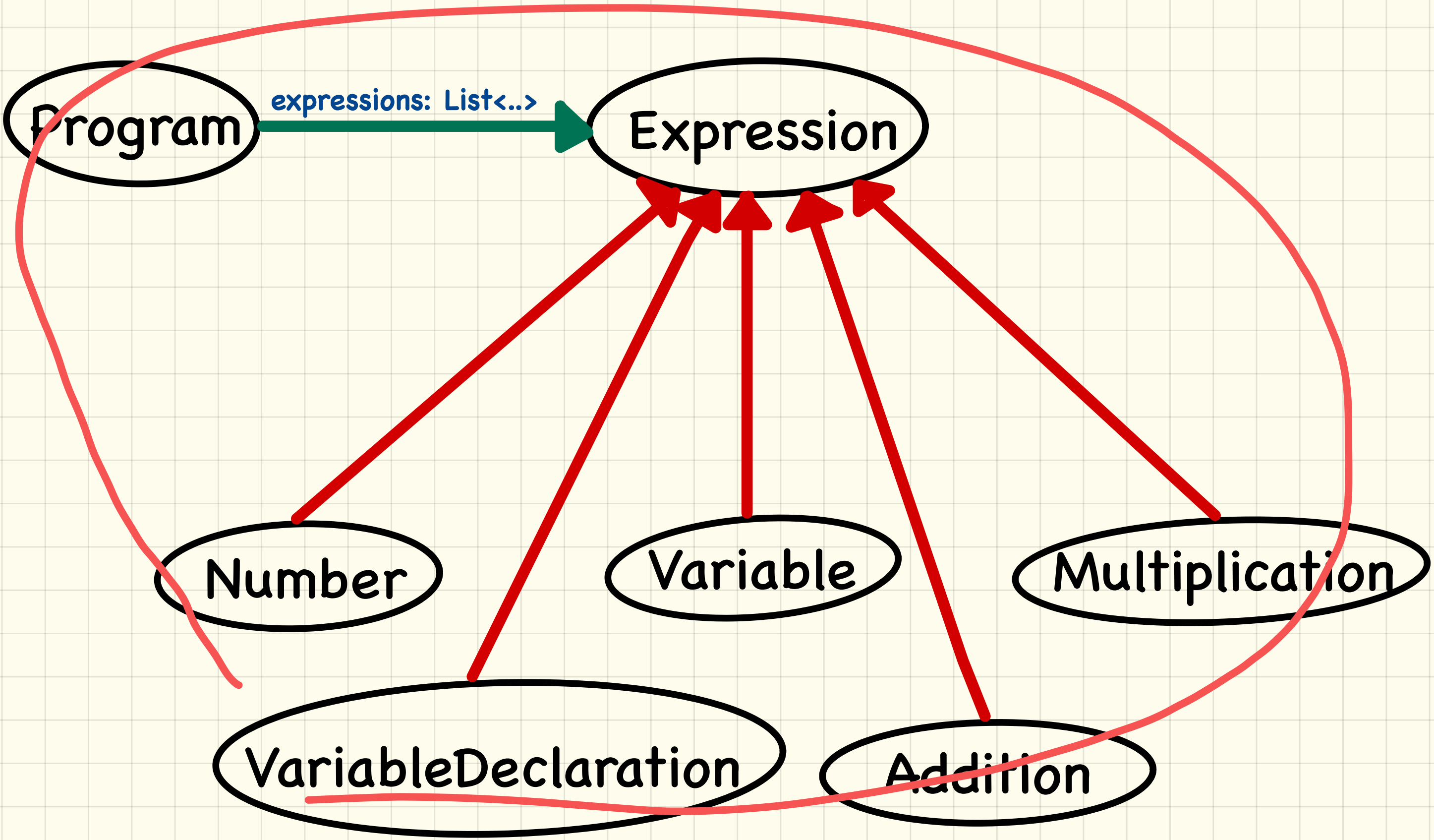
ANTLR (ANother Tool for Language Recognition)





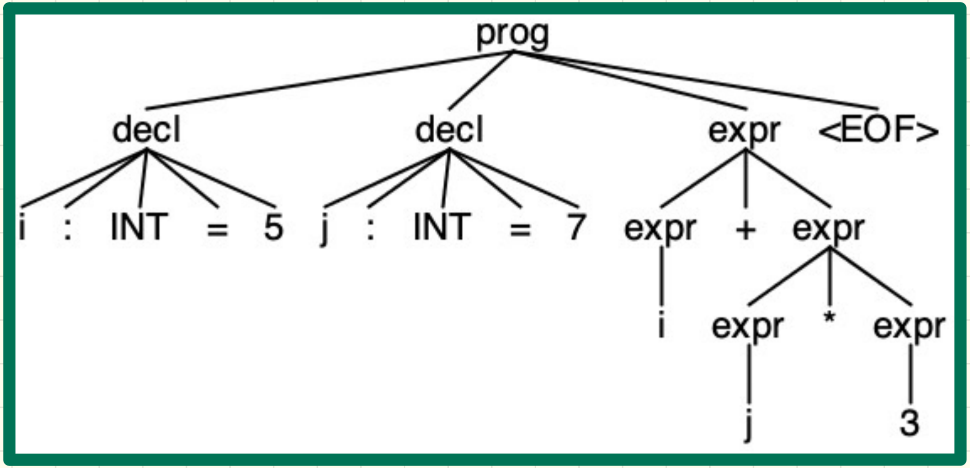
M.C.

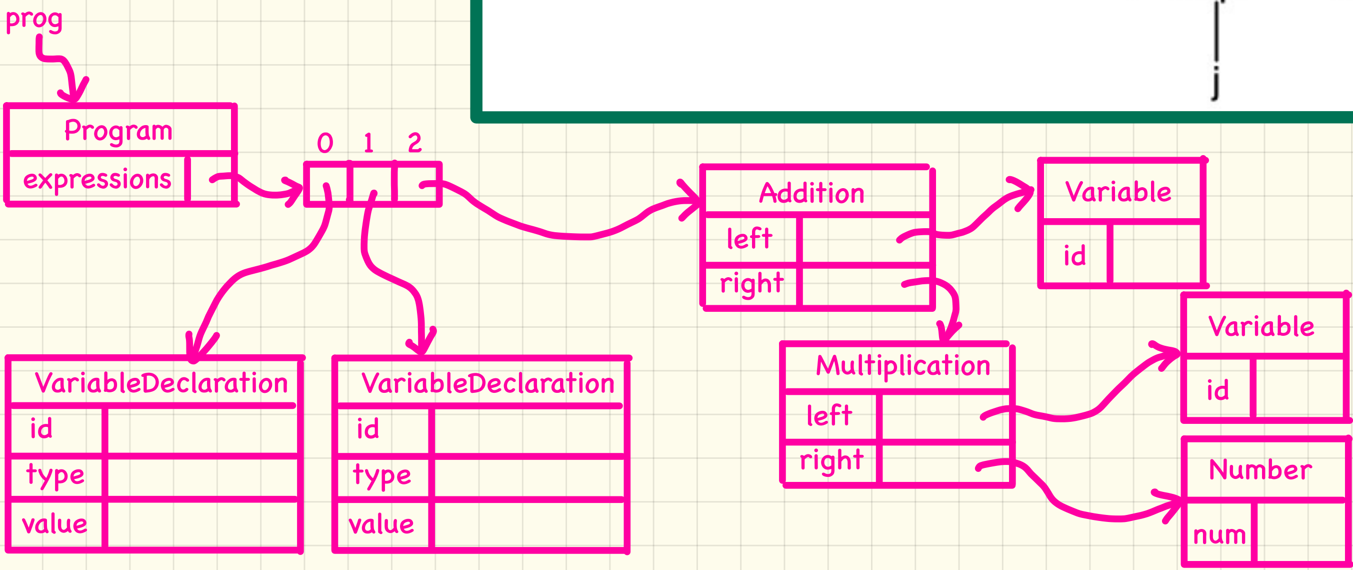
Composite Pattern of Model Classes



Building Model Objects from Parse Trees

```
i : INT = 5
j : INT = 7
i + j * 3
```





Backtrack-Free Grammar

$$\textsc{First}^+(A \rightarrow \beta) = \begin{cases} \textsc{First}(\beta) & \text{if } \epsilon \notin \textsc{First}(\beta) \\ \textsc{First}(\beta) \cup \textsc{Follow}(A) & \text{otherwise} \end{cases}$$

$\textsc{First}(\beta)$ is the extended version where β may be $\beta_1 \beta_2 \ldots \beta_n$

$A \rightarrow \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$ satisfying:

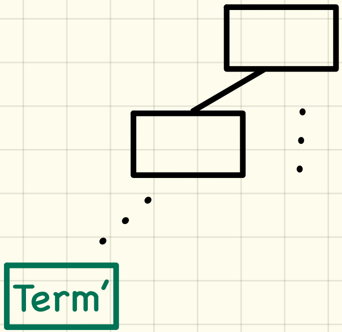
$$\forall i, j : 1 \leq i, j \leq n \wedge i \neq j \bullet \textsc{First}^+(\gamma_i) \cap \textsc{First}^+(\gamma_j) = \varnothing$$

Top-Down Parsing: Algorithm with lookahead

backtrack ≜ pop *focus*.siblings; *focus* := *focus*.parent; *focus*.resetChildren

```
ALGORITHM: TDParse
  INPUT: CFG G = (V, Σ, R, S)
  OUTPUT: Root of a Parse Tree or Syntax Error
PROCEDURE:
  root := a new node for the start symbol S
  focus := root
  initialize an empty stack trace
  trace.push(null)
  word := NextWord()
  while (true):
    if focus ∈ V then % use FOLLOW set as well?
      if ∃ unvisited rule focus → β1β2...βn ∈ R ∧ word ∈ FIRST+(β) then
        create β1, β2 ... βn as children of focus
        trace.push(βnβn−1 ... β2)
        focus := β1
      else
        if focus = S then report syntax error
        else backtrack
    elseif word matches focus then
      word := NextWord()
      focus := trace.pop()
    elseif word = EOF ∧ focus = null then return root
    else backtrack
```

0	*Goal*	→	*Expr*
1	*Expr*	→	*Term Expr′*
2	*Expr′*	→	+ *Term Expr′*
3		|	- *Term Expr′*
4		|	ϵ
5	*Term*	→	*Factor Term′*
6	*Term′*	→	x *Factor Term′*
7		|	÷ *Factor Term′*
8		|	ϵ
9	*Factor*	→	(*Expr*)
10		|	num
11		|	name



Backtrack-Free Grammar: Exercise

$$\text{FIRST}^+(A \to \beta) = \begin{cases} \text{FIRST}(\beta) & \text{if } \epsilon \notin \text{FIRST}(\beta) \\ \text{FIRST}(\beta) \cup \text{FOLLOW}(A) & \text{otherwise} \end{cases}$$

FIRST(β) is the extended version where β may be $\beta_1 \beta_2 \ldots \beta_n$

$A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$ satisfying:

$$\forall i, j : 1 \le i, j \le n \land i \neq j \bullet \text{FIRST}^+(\gamma_i) \cap \text{FIRST}^+(\gamma_j) = \varnothing$$

Is the following CFG ***backtrack free***?

11	*Factor*	→	name
12		|	name [*ArgList*]
13		|	name (*ArgList*)
15	*ArgList*	→	*Expr MoreArgs*
16	*MoreArgs*	→	, *Expr MoreArgs*
17		|	ϵ

Left-Factoring: Removing Common Prefixes

Identify a common prefix α:

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_j$$

[each of $\gamma_1, \gamma_2, \ldots, \gamma_j$ does not begin with α]

Rewrite that production rule as:

$$\begin{aligned} A &\rightarrow \alpha B \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_j \\ B &\rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \end{aligned}$$

11	*Factor*	$\rightarrow$	`name`
12		|	`name` [*ArgList*]
13		|	`name` (*ArgList*)
15	*ArgList*	$\rightarrow$	*Expr MoreArgs*
16	*MoreArgs*	$\rightarrow$	, *Expr MoreArgs*
17		|	ϵ

$F \rightarrow$ name F'

$F' \rightarrow \epsilon$

| [Arglist]

| (Arglist)

Implementing a Recursive-Descent Parser

generated by ANTLR4!

	Production				FIRST⁺
2	*Expr'*	$\rightarrow$	+	*Term Expr'*	{ + }
3		|	-	*Term Expr'*	{ - }
4		|	ϵ		ϵ, eof,)

```
ExprPrim()
  if word = + ∨ word = - then /* Rules 2, 3 */
    word := NextWord()
    if(Term())
      then return ExprPrim()
      else return false
  elseif word = ) ∨ word = eof then /* Rule 4 */
    return true
  else
    report a syntax error
    return false
  end

Term()
  ...
```

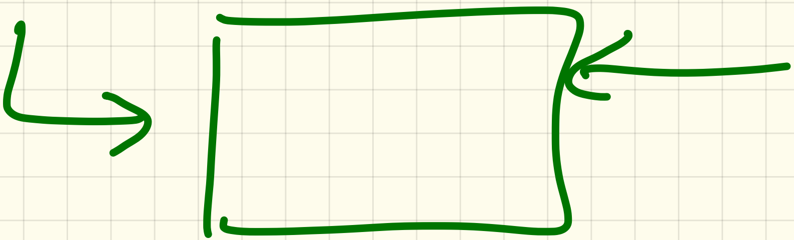
etf .

LL(*)

LL(1) .

↑ back-track free

[etf] -new —— goto.



$L \to Ra$

$\quad | \; Qba$

$R \to aba$

$\quad | \; caba$

$\quad | \; Rbc$

$Q \to bbc$

$\quad | \; bc$

	alternative	First	intersection
L	Ra Qba		∅
R			
R'			
Q			
Q'			

left recursion

$R \to abaR' \; | \; cabaR'$

$R' \to bcR'$

$\quad | \; \varepsilon$

common prefix

$Q \to bQ'$

$Q' \to bc$

$\quad | \; c$