Domain Specific Languages

Jason M. Carey

Who's This Guy?

NSA Cooperative Education Program out of high school

Graduated from University of Wisconsin-Madison

- B.S. in Computer Science
- B.S. in Mathematics

Several years applying machine learning techniques as a Department of Defense contractor

Graduated from Johns Hopkins University

- M.S. in Computer Science

Currently, active in addressing challenges in enterprise systems

What's a **D**omain **S**pecific **L**anguage (DSL)?

What's a **D**omain **S**pecific **L**anguage (DSL)?

A language focused on a particular domain, right?

What's a **D**omain **S**pecific **L**anguage (DSL)?

A language focused on a particular domain, right?

Not quite…

What's a **D**omain **S**pecific **L**anguage (DSL)?

A language focused on a particular domain.

What's a **D**omain **S**pecific **L**anguage (DSL)?

A language focused on a particular domain.

Is coffee ordering a DSL?

triple, venti, extra hot, no-whip, skinny white mocha

How about calling football plays?

ace right z-dig x-shallow

Or betting on horseracing?

1-all daily double wheel

What's a **D**omain **S**pecific **L**anguage (DSL)?

A computer programming language focused on a particular domain.

What's a **D**omain **S**pecific **L**anguage (DSL)?

A computer programming language focused on a particular domain.

Is Java a DSL?

Java is a programming language for writing object-oriented programs

What about PHP?

PHP is a programming language for developing web applications

What's a **D**omain **S**pecific **L**anguage (DSL)?

According to Martin Fowler…

"a computer programming language of limited expressiveness focused on a particular domain"

What's a **D**omain **S**pecific **L**anguage (DSL)?

According to Martin Fowler…

"a computer programming language of limited expressiveness focused on a particular domain"

Examples:

SQL

```
SELECT id, make, model FROM Car WHERE id > 100
```

Regular Expressions

```
<TAG[^>]*>(.*?)</TAG>
```

Why are DSLs useful?

Focused on a particular domain means context

order.withCrust(“thick”).withToppings(“onions”, “sausage”, “peppers”)

Why are DSLs useful?

Focused on a particular domain means context

order.withCrust("thick").withToppings("onions", "sausage", "peppers")

DSLs utilize the domain's vocabulary to create context

- Pizza, cheese, and tomato sauce were implied

With context, there is less clutter, terse, but readable

- Domain experts immediately understand instances of the DSL
- People not familiar with the domain may be confused, sorry!

Provided abstractions closely correspond to domain

- If pizza has crust and toppings, then the DSL must have them

Why are DSLs useful?

Limited expressiveness means accessibility

A HTML file is pretty simple to read/write/maintain. What if the same file was written as a JSP? How about as a servlet?

Why are DSLs useful?

Limited expressiveness means accessibility

A HTML file is pretty simple to read/write/maintain. What if the same file was written as a JSP? How about as a servlet?

DSLs only require knowledge of the domain

- HTML does not require knowledge of sessions, I/O, or exceptions

Abstractions already provided by the language

- No need to be creative (dangerous?) by adding new abstractions
- You cannot create a database connection with HTML

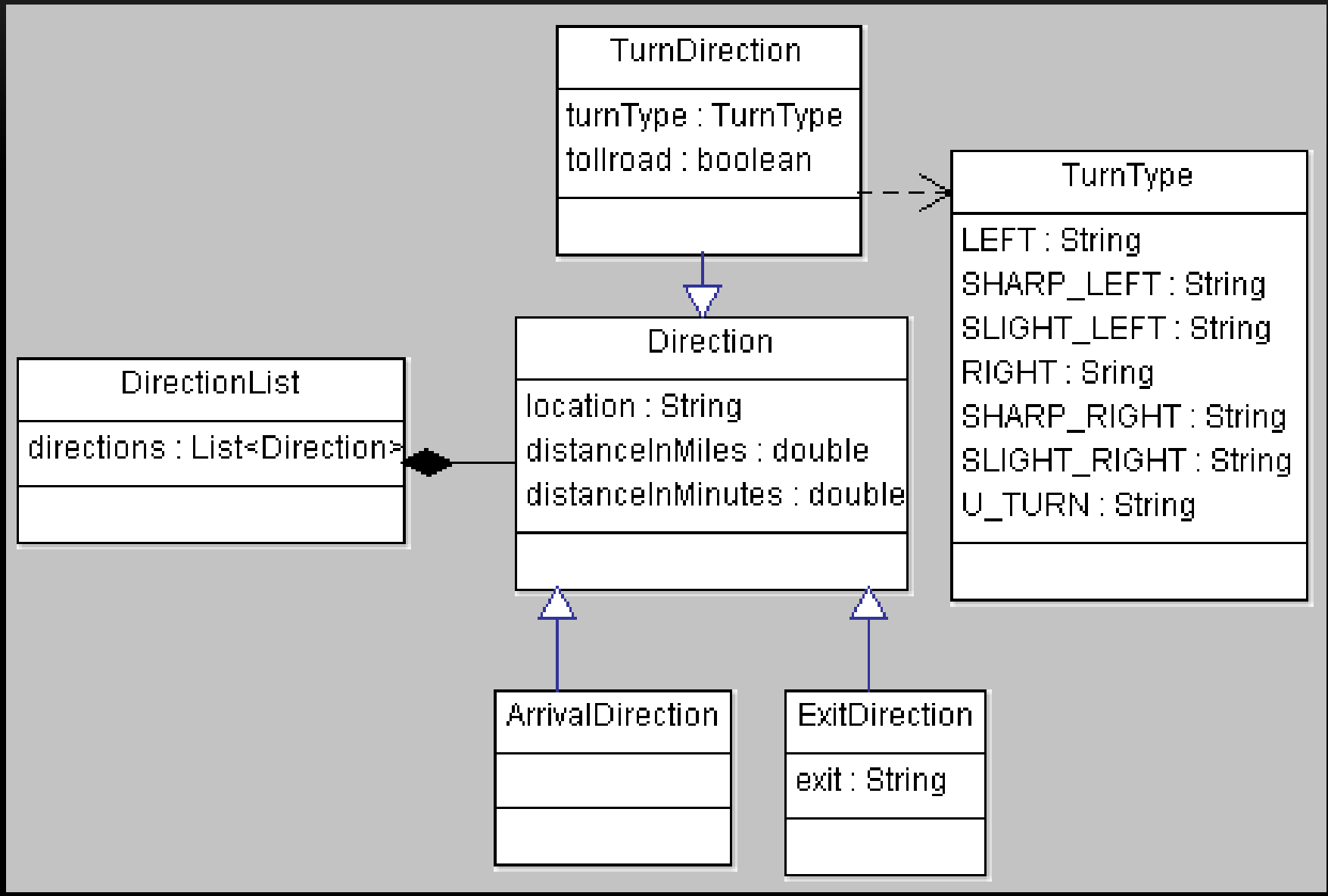
Why are DSLs useful?

When writing software, you are communicating with yourself, your team, and sometimes your users.

Exercise: Improving Google Driving Directions

Google has decided to improve the accuracy of Google Driving Directions. A small team will be responsible for developing software that allows people to submit suggested driving routes between two locations to Google.

Exercise: UML Class Diagram



Exercise: JavaBean Solution

```
DirectionList directions = new DirectionList();

TurnDirection td1 = new TurnDirection();
td1.setTurnType(TurnType.LEFT);
td1.setLocation("11th Ave");
td1.setDistanceInMiles(2.3);
td1.setDistanceInMinutes(4);
directions.add(td1);

TurnDirection td2 = new TurnDirection();
td2.setTurnType(TurnType.RIGHT);
td2.setLocation("Hwy 13");
td2.setDistanceInMiles(9.4);
td2.setDistanceInMinutes(11);
directions.add(td2);

TurnDirection td3 = new TurnDirection();
td3.setTurnType(TurnType.SLIGHT_RIGHT);
td3.setLocation("I-95");
td3.setDistanceInMiles(60);
td3.setDistanceInMinutes(55);
td3.setTollroad(true);
directions.add(td3);

ExitDirection ed = new ExitDirection();
ed.setExit("10A");
ed.setLocation("Frontage Road");
ed.setDistanceInMiles(4.4);
ed.setDistanceInMintues(4);
directions.add(ed);

TurnDirection td4 = new TurnDirection();
td4.setTurnType(TurnType.U_TURN);
td4.setTurnType(TurnType.U_TURN);
td4.setLocation("13th Street");
td4.setDistanceInMiles(0.1);
td4.setDistanceInMinutes(1);
directions.add(td4);

ArrivalDirection ad =
   new ArrivalDirection();
ad.setLocation("Burger King");
ad.setDistanceInMiles(0.2);
ad.setDistanceInMintues(4);
directions.add(ad);
```

Exercise: JavaBean Solution

```
DirectionList directions = new DirectionList();

TurnDirection td1 = new TurnDirection();
td1.setTurnType(TurnType.LEFT);
td1.setLocation("11th Ave");
td1.setDistanceInMiles(2.3);
td1.setDistanceInMinutes(4);
directions.add(td1);

TurnDirection td2 = new TurnDirection();
td2.setTurnType(TurnType.RIGHT);
td2.setLocation("Hwy 13");
td2.setDistanceInMiles(9.4);
td2.setDistanceInMinutes(11);
directions.add(td2);

TurnDirection td3 = new TurnDirection();
td3.setTurnType(TurnType.SLIGHT_RIGHT);
td3.setLocation("I-95");
td3.setDistanceInMiles(60);
td3.setDistanceInMinutes(55);
td3.setTollroad(true);
directions.add(td3);

ExitDirection ed = new ExitDirection();
ed.setExit("10A");
ed.setLocation("Frontage Road");
ed.setDistanceInMiles(4.4);
ed.setDistanceInMintues(4);
directions.add(ed);

TurnDirection td4 = new TurnDirection();
td4.setTurnType(TurnType.U_TURN);
td4.setLocation("13th Street");
td4.setDistanceInMiles(0.1);
td4.setDistanceInMinutes(1);
directions.add(td4);

ArrivalDirection ad =
   new ArrivalDirection();
ad.setLocation("Burger King");
ad.setDistanceInMiles(0.2);
ad.setDistanceInMintues(4);
directions.add(ad);
```

Exercise: Immutable POJO Solution

```
DirectionList directions = new DirectionList();

TurnDirection td;
ExitDirection ed;
ArrivalDirection ad;

td = new TurnDirection(TurnType.LEFT, “11th Ave", 2.3, 4);
directions.add(td);

td = new  TurnDirection(TurnType.RIGHT, "Hwy 13", 9.4, 11);
directions.add(td);

td = new TurnDirection(TurnType.SLIGHT_RIGHT, "I-95", true, 60, 55);
directions.add(td);

ed= new ExitDirection("10A", "Frontage Road", 4.4, 4);
directions.add(ed);

td = new TurnDirection(TurnType.U_TURN, "13th Street", 0.1, 1);
directions.add(td);

ad = new ArrivalDirection("Burger King", 0.2, 4);
directions.add(ad);
```

Exercise: XML Solution

```
<?xml version="1.0" encoding="UTF-8"?>

<directions>
  <direction type="turn">
    <location>11th Ave</location>
    <distanceInMiles>2.3</distanceInMiles>
    <distanceInMinutes>4</distanceInMinutes>
    <turnType>left</turnType>
  </direction>
  <direction type="turn">
    <location>Hwy 13</location>
    <distanceInMiles>9.4</distanceInMiles>
    <distanceInMinutes>11</distanceInMinutes>
    <turnType>right</turnType>
  </direction>
 <direction type="turn">
    <location>I-95</location>
    <distanceInMiles>60</distanceInMiles>
    <distanceInMinutes>55</distanceInMinutes>
    <turnType>slight_right</turnType>
    <tollroad>true</tollroad>
  </direction>

...

</directions>
```

Exercise: Can We Do Better?

Problems with JavaBean Solution

- Verbosity obscures intent
- Redundant (e.g., td.setXXX)
- Software changes require code compilation

Problems with Immutable POJO Solution

- Complex construction obscures intent
- Software changes require code compilation

Problems with XML Solution

- Overly verbose
- Unnatural syntax

Exercise: Revisiting Problem Statement

Google has decided to improve the accuracy of Google Driving Directions. A small team will be responsible for developing software that allows people to submit suggested driving routes between two locations to Google.

Exercise: Revisiting Problem Statement

Google has decided to improve the accuracy of Google Driving Directions. A small team will be responsible for developing software that allows people to submit suggested driving routes between two locations to Google.

Who will be submitting?

Exercise: Revisiting Problem Statement

Google has decided to improve the accuracy of Google Driving Directions. A small team will be responsible for developing software that allows people to submit suggested driving routes between two locations to Google.

Who will be submitting? Java Engineers?

Exercise: Revisiting Problem Statement

Google has decided to improve the accuracy of Google Driving Directions. A small team will be responsible for developing software that allows people to submit suggested driving routes between two locations to Google.

Who will be submitting? Java Engineers? Your Uncle?

Exercise: People == Java Engineers

Instead of

```
TurnDirection td = new TurnDirection();
td.setTurnType(TurnType.LEFT);
td.setLocation("11th Ave");
td.setDistanceInMiles(2.3);
td.setTimeInMinutes(4);
```

or

```
new TurnDirection(TurnType.LEFT, "11th Ave", 2.3, 4);
```

how about

```
turn(LEFT).onto("11th Ave").in_miles(2.3).in_minutes(4);
```

Fluent Interface

Coined by Mark Evans and Martin Fowler

Designed to be readable and flow naturally

- Methods return self or a promoter
- Methods make little sense out of context
- Contrasts the traditional Command-Query Interface
- Difficult to design, but easier to use

Fluent Interface Examples

Fluent Loops (Ruby)

- for i in 0..9 end
- (0..9).each do | i | end
- 0.upto(9) do | i | end

Fluent Testing (Mockito)

- when(mockedList.get(0)).thenReturn(“firstValue”)
- verify(mockedList, atLeastOnce()).add(“commonValue”)
- doThrow(new RuntimeException()).when(mockedList).clear()

Fluent ORM (Ruby Rails)

- class Manager < ActiveRecord::Base
  has_many :employees
  end

Exercise: Fluent Interface Solution

```
DirectionList directions = new DirectionList();

directions.add(

    turn(LEFT).onto(“11th Ave").in_miles(2.3).in_minutes(4),

    turn(SLIGHT_RIGHT).onto("Hwy 13").in_miles(9.4).in_minutes(11),

    turn(SLIGHT_RIGHT).onto("I-95").tollroad().in_miles(3.7).in_minutes(55),

    exit("10A").onto("Frontage Road").in_miles(60).in_minutes(4),

    turn(U_TURN).onto("13th Street").in_miles(.3).in_minutes(1),

    arrive("Burger King").in_miles(.2).in_minutes(4)

);
```

Exercise: Fluent Interface Implementation

Exercise: My Uncle Doesn't Speak Java

Fluent Interface is an API built on a host language

- Syntax confined to host language
- Requires knowledge of host language
- May require additional processing (e.g., compilation)

Exercise: My Uncle Doesn't Speak Java

Fluent Interface is an API built on a host language

- Syntax confined to host language
- Requires knowledge of host language
- May require additional processing (e.g., compilation)

What if we decoupled the DSL from a host language?

```
turn left onto "11th Ave" in 2.3 miles or 4 minutes
turn right onto "Hwy 13" in 9.4 miles or 11 minutes
turn slight right onto "I-94" tollroad in 60 miles or 55 minutes
exit at "10A" onto "Frontage Road" in 4.4 miles or 4 minutes
u-turn onto "13th Street" in 0.1 miles or 1 minutes
arrive at "Burger King" in 0.2 miles or 4 minutes
```

DSL Dichotomy

Internal ("Embedded") DSLs

- A convenient view of the host language
- Leverages the host language
  - Existing types, structures, and semantics
  - Lexical analysis, parsing, and interpretation
- Common Host Languages
  - Groovy, Java, Ruby, Scala

External DSLs

- Designed entirely from the ground-up
  - Responsible for defining types, syntax, semantics, and control structures
- Requires implementation of language
- Tooling and documentation for users
- Examples
  - SQL, Regular Expressions, CSS

Internal DSL or External DSL?

What is the technical background of the DSL users?

External DSLs provide complete flexibility in syntax and semantics allowing the language to be tailored to the domain. The syntax and semantics of the host language may limit or contradict the domain's vocabulary and rules.

Internal DSL or External DSL?

What is the technical background of the DSL users?

External DSLs provide complete flexibility in syntax and semantics allowing the language to be tailored to the domain. The syntax and semantics of the host language may limit or contradict the domain's vocabulary and rules.

Will the DSL be integrated into applications?

Whether Internal or External, the DSL implementation may need to be integrated with the application. Integration points should be understood in advance.

Internal DSL or External DSL?

What is the technical background of the DSL users?

External DSLs provide complete flexibility in syntax and semantics allowing the language to be tailored to the domain. The syntax and semantics of the host language may limit or contradict the domain's vocabulary and rules.

Will the DSL be integrated into applications?

Whether Internal or External, the DSL implementation may need to be integrated with the application. Integration points should be understood in advance.

How much time is permitted to create the DSL?

External DSLs require much more upfront cost because language types, control structures, exception flow, syntax, and semantics must be defined. The language must be implemented and may require tooling and training for users.

Internal DSL or External DSL?

What is the technical background of the DSL users?

External DSLs provide complete flexibility in syntax and semantics allowing the language to be tailored to the domain. The syntax and semantics of the host language may limit or contradict the domain's vocabulary and rules.

Will the DSL be integrated into applications?

Whether Internal or External, the DSL implementation may need to be integrated with the application. Integration points should be understood in advance.

How much time is permitted to create the DSL?

External DSLs require much more upfront cost because language types, control structures, exception flow, syntax, and semantics must be defined. The language must be implemented and may require tooling and training for users.

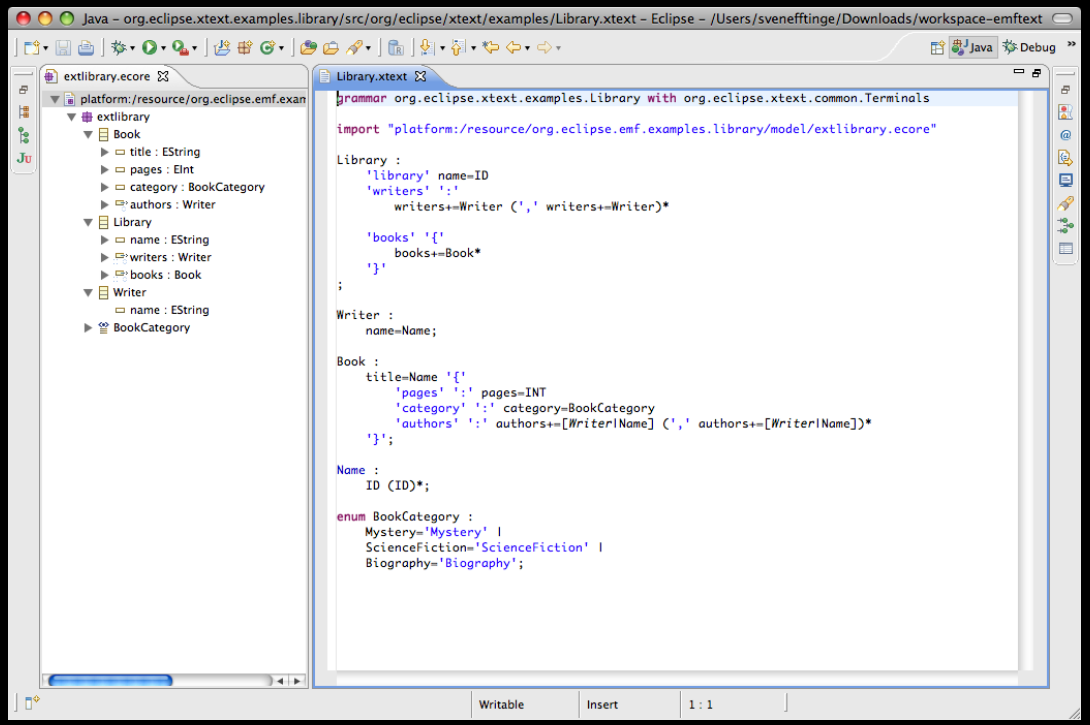
Are there performance requirements?

DSLs may improve or degrade performance. External DSLs are more prone to degrade application performance because the implementation may be less optimized.

Language Workbenches

IDEs for creating languages

- Language designers define DSL using various tooling (editors, views, wizards)
- Workbench generates model objects, parsers, and interpreters
- Information stored in a concrete syntax but viewed and edited using other representations such as outline views, call hierarchies, or refactoring actions
- Designers and users may leverage tooling



Xtext

http://www.eclipse.org/Xtext/

Summary

DSLs are not General Purpose Languages

- Limited expressiveness focused on a domain
- Think screwdriver not Swiss Army knife

DSLs help bridge the domain gap

- Accessibility of DSLs help engineers and users communicate about the domain and its rules

Understand External Vs. Internal Tradeoffs

- Internal can leverage host language
- External offers complete flexibility, but is much harder to develop

References

http://martinfowler.com/bliki/DomainSpecificLanguage.html

http://www.slideshare.net/adorepump/domain-specific-languages-presentation-816071

http://www.slideshare.net/ThoughtWorks0ffshore/construction-techniques-for-domain-specific-languages

http://nealford.com/downloads/conferences/4_Practical_Uses_for_DSLs(Neal_Ford).pdf

Ghosh, Debasish. DSLs in Action. Manning Publications, 2010

http://www.eclipse.org/Xtext

http://www.jetbrains.com/mps

Building External DSLs with ANTLR

Jason M. Carey

What's a **D**omain **S**pecific **L**anguage (DSL)?

According to Martin Fowler…

"a computer programming language of limited expressiveness focused on a particular domain"

Examples:

SQL

```
SELECT id, make, model FROM Car WHERE id > 100
```

Regular Expressions

```
<TAG[^>]*>(.*?)</TAG>
```

DSL Dichotomy

Internal ("Embedded") DSLs

- A convenient view of the host language
- Leverages the host language
  - Existing types, structures, and semantics
  - Lexical analysis, parsing, and interpretation
- Common Host Languages
  - Groovy, Java, Ruby, Scala

External DSLs

- Designed entirely from the ground-up
  - Responsible for defining types, syntax, semantics, and control structures
- Requires implementation of language
- Tooling and documentation for users
- Examples
  - SQL, Regular Expressions, CSS

External DSLs

Offer flexibility in the language design at the price of increased complexity in implementation.

External DSLs

Offer flexibility in the language design at the price of increased complexity in implementation.

In other words, External DSLs are a lot of work!

Are these Different?

```
// Example 1
int c = reader.read();
StringBuffer buf = new StringBuffer();
while(c != -1) {
  buf.append((char) c);
  c = reader.read();
}
```

```
// Example 2
int c = reader.read();
StringBuffer letters = new StringBuffer();
StringBuffer digits = new StringBuffer();
while(Character.isLetter((char) c)) {
  letters.append((char) c);
  c = reader.read();
}
while(Character.isDigit((char) c)) {
  digits.append((char) c);
  c = reader.read();
}
```

Are these Different?

```
// Example 1
int c = reader.read();
StringBuffer buf = new StringBuffer();
while(c != -1) {
  buf.append((char) c);
  c = reader.read();
}
```

```
// Example 2
int c = reader.read();
StringBuffer letters = new StringBuffer();
StringBuffer digits = new StringBuffer();
while(Character.isLetter((char) c)) {
  letters.append((char) c);
  c = reader.read();
}
while(Character.isDigit((char) c)) {
  digits.append((char) c);
  c = reader.read();
}
```

Are these Different?

```
// Example 1
int c = reader.read();
StringBuffer buf = new StringBuffer();
while(c != -1) {
  buf.append((char) c);
  c = reader.read();
}
```

```
// Example 2
int c = reader.read();
StringBuffer letters = new StringBuffer();
StringBuffer digits = new StringBuffer();
while(Character.isLetter((char) c)) {
  letters.append((char) c);
  c = reader.read();
}
while(Character.isDigit((char) c)) {
  digits.append((char) c);
  c = reader.read();
}
```

Are these Different?

```
// Example 1
int c = reader.read();
StringBuffer buf = new StringBuffer();
while(c != -1) {
  buf.append((char) c);
  c = reader.read();
}
```

Yes!

Example 2 *recognizes* letters followed by digits.

```
// Example 2
int c = reader.read();
StringBuffer letters = new StringBuffer();
StringBuffer digits = new StringBuffer();
while(Character.isLetter((char) c)) {
  letters.append((char) c);
  c = reader.read();
}
while(Character.isDigit((char) c)) {
  digits.append((char) c);
  c = reader.read();
}
```

Language Recognition

Recognizing input requires the ability to:

Identify the vocabulary

- The language defines the vocabulary
- The vocabulary recognizer is called the ***lexer***
  - Groups characters into vocabulary symbols called tokens

Compare the input structure against constraints

- The language defines the constraints
- The structure recognizer is called the ***parser***
  - Produces a data model from a stream of tokens

Input is recognizable if it satisfies the language constraints

Language Recognition

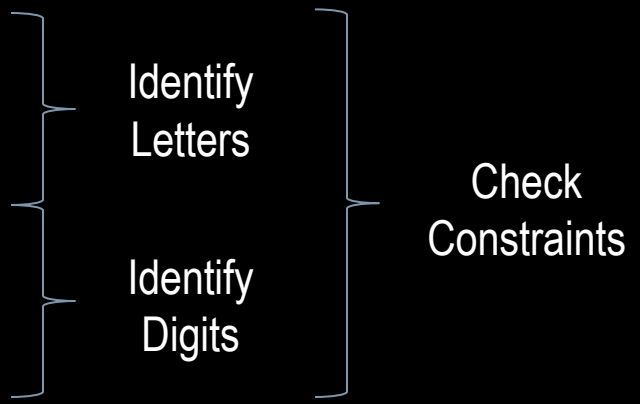
```
// Example 2
int c = reader.read();
StringBuffer letters = new StringBuffer();
StringBuffer digits = new StringBuffer();
while(Character.isLetter((char) c)) {
  letters.append((char) c);
  c = reader.read();
}
while(Character.isDigit((char) c)) {
  digits.append((char) c);
  c = reader.read();
}
```

Vocabulary ?

Constraints ?

Language Recognition

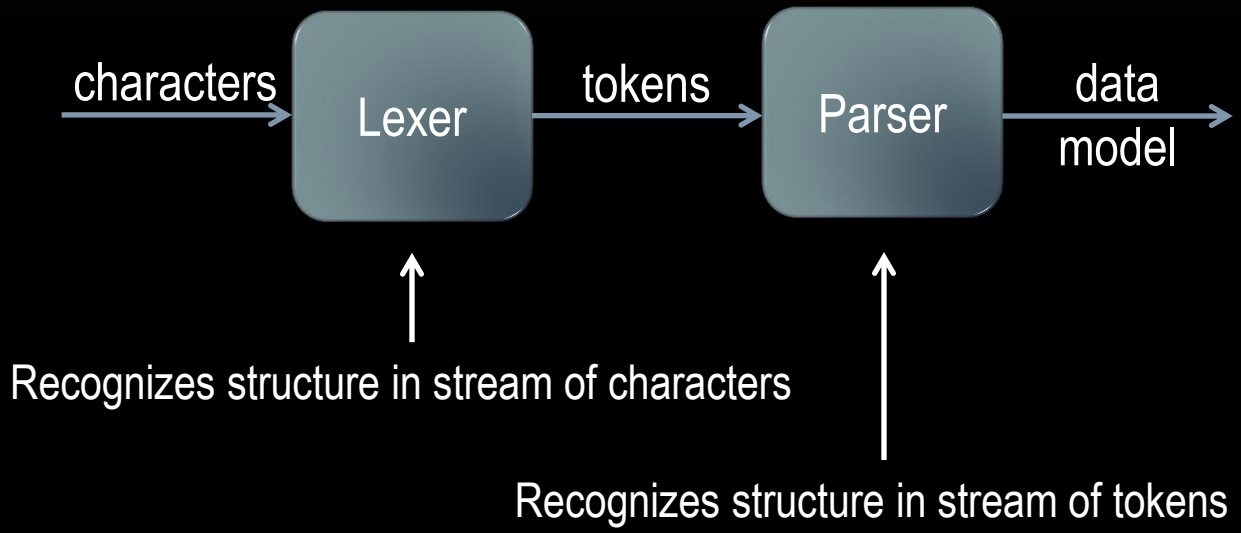
```
// Example 2
int c = reader.read();
StringBuffer letters = new StringBuffer();
StringBuffer digits = new StringBuffer();
while(Character.isLetter((char) c)) {
  letters.append((char) c);
  c = reader.read();
}
while(Character.isDigit((char) c)) {
  digits.append((char) c);
  c = reader.read();
}
```



Vocabulary :: letters, digits

Constraints :: zero or more letters followed by zero or more digits

The Big Picture (so far)



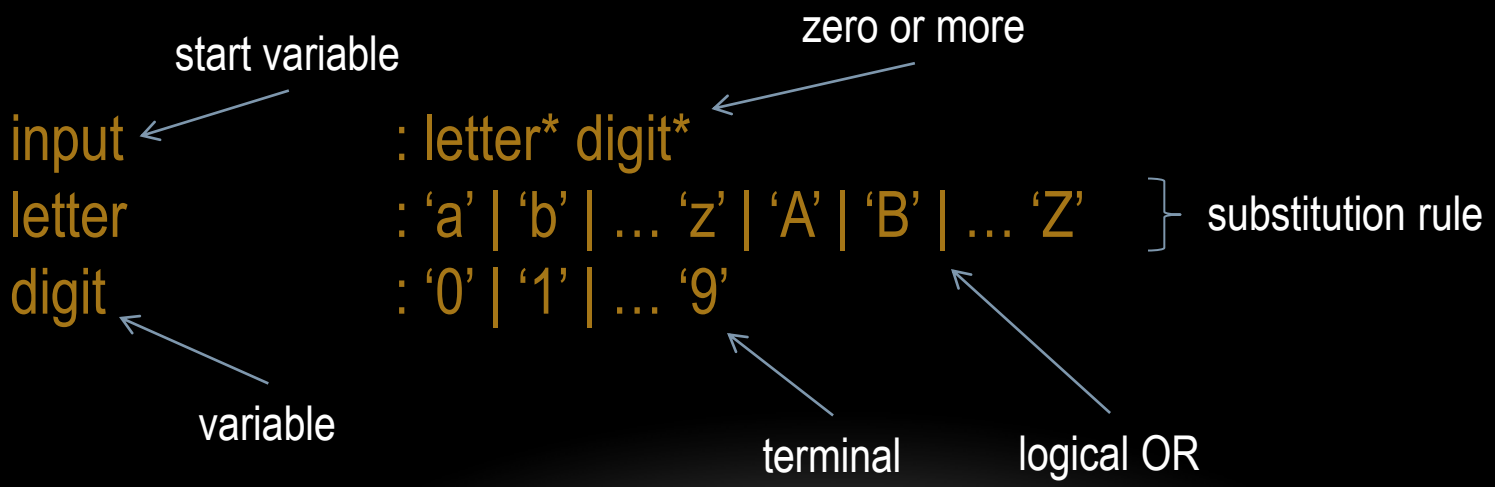
Language Grammars

A technique for describing language structure using variables and substitution rules

Language Grammars

A technique for describing language structure using variables and substitution rules

"Letters then Digits" Grammar



Grammar Derivations

A sequence of substitutions beginning with the start variable and ending with a string containing no variables.

A substitution replaces one variable at a time.

A language recognizes the input if a derivation exists.

Grammar Derivations

input : letter* digit*
letter : ‘a’ | ‘b’ | … ‘z’ | ‘A’ | ‘B’ | … ‘Z’
digit : ‘0’ | ‘1’ | … ‘9’

Does the language recognize ‘hello123’ ?

Grammar Derivations

input : letter* digit*
letter : ‘a’ | ‘b’ | … ‘z’ | ‘A’ | ‘B’ | … ‘Z’
digit : ‘0’ | ‘1’ | … ‘9’

Does the language recognize ‘hello123’ ?

input → letter* digit*
→ hello digit*
→ hello123

Yes, since a derivation exists!

Grammar Derivations

input : letter* digit*
letter : ‘a’ | ‘b’ | … ‘z’ | ‘A’ | ‘B’ | … ‘Z’
digit : ‘0’ | ‘1’ | … ‘9’

Does the language recognize ‘hello123world’ ?

Grammar Derivations

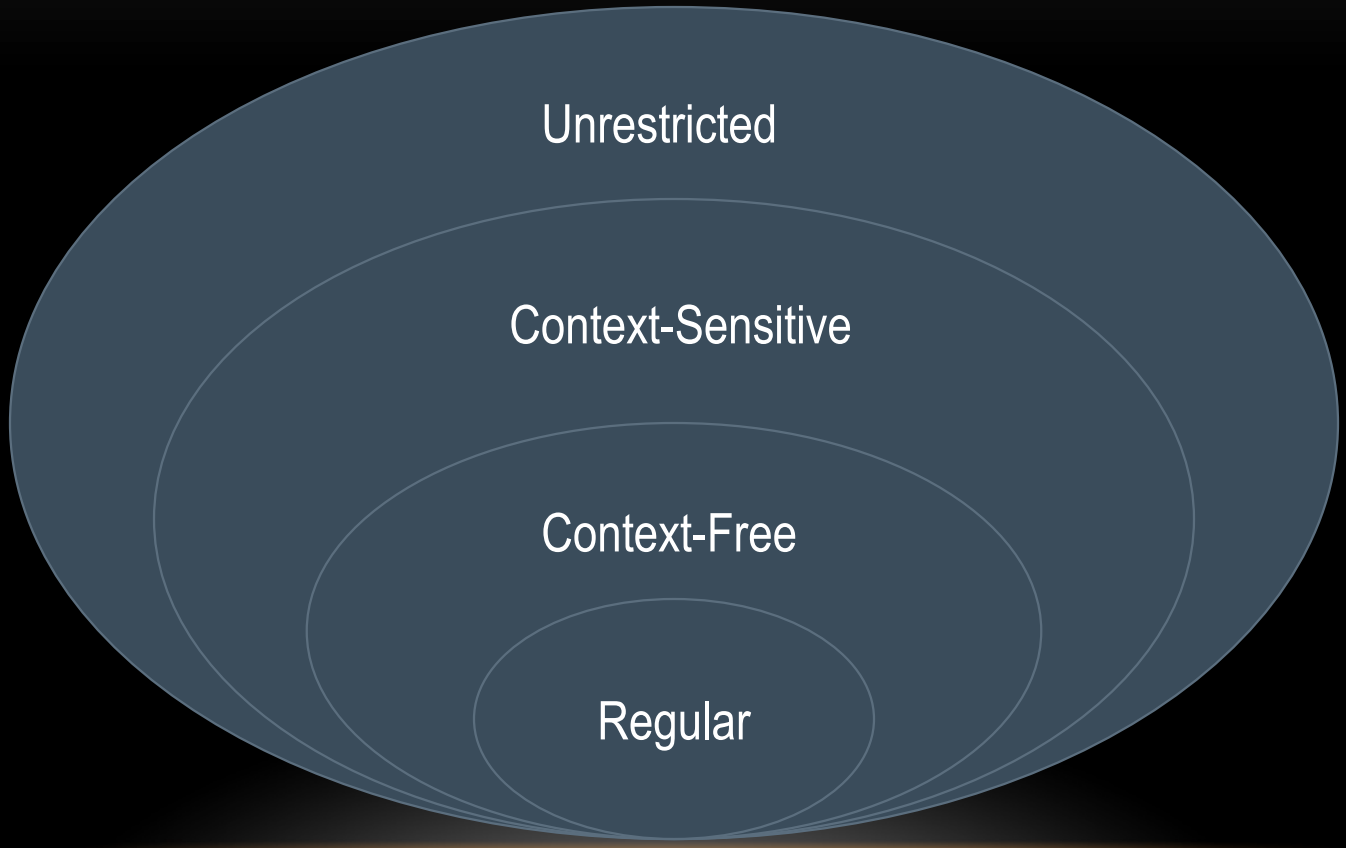
input : letter* digit*
letter : ‘a’ | ‘b’ | … ‘z’ | ‘A’ | ‘B’ | … ‘Z’
digit : ‘0’ | ‘1’ | … ‘9’

Does the language recognize ‘hello123world’ ?

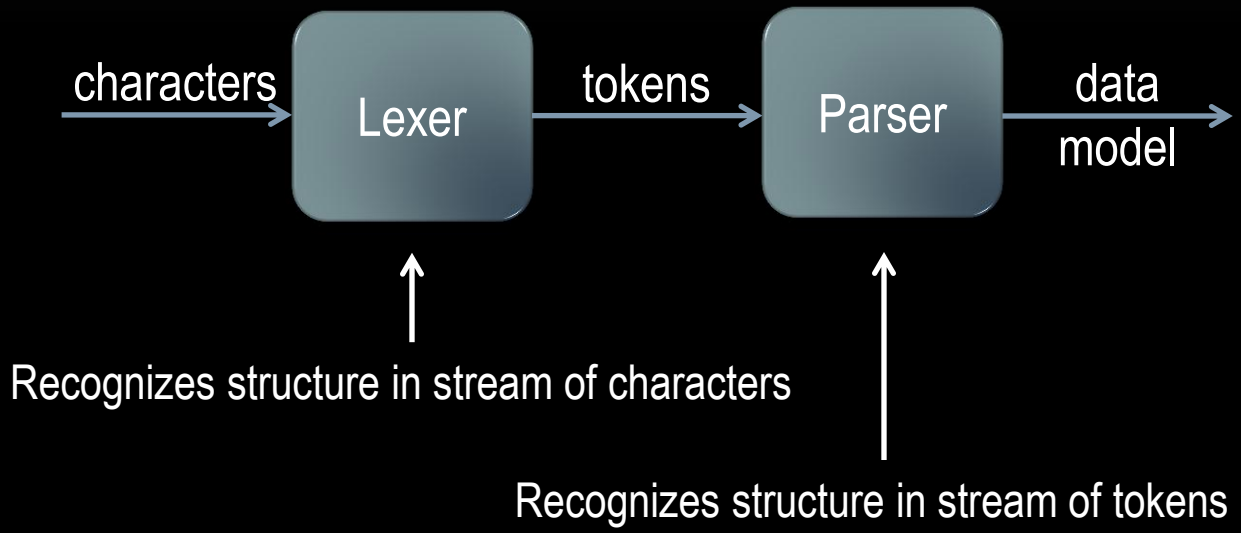
input → letter* digit*
→ hello digit*
→ hello123 // no variable left to match ‘world’

No, since a derivation does not exist!

Chomsky's Grammar Hierarchy



The Big Picture (so far)



The Data Model

Produced by the parser and is machine readable

The application dictates the nature of the data model

Task	Possible Data model
Word existence in a document	Unordered set of words
Word frequencies in a document	Map of words to frequencies
Ad hoc queries against an XML document	Document object model

Grammar-based applications usually require knowledge of tokens and their structural (nested) relationships

Trees as the Data Model

A simple grammar to recognize an integer assignment

```
statement     : assignment ';'
assignment    : id '=' integer
id            : 'x'
integer       : ('0' | '1' | … '9')+
```

Example Derivation: 'x = 123;'

```
statement → assignment ';'
          → id '=' integer ';'
          → 'x' '=' integer ';'
          → 'x' '=' 123 ';'
          → 'x' '=' 123 ';'
```

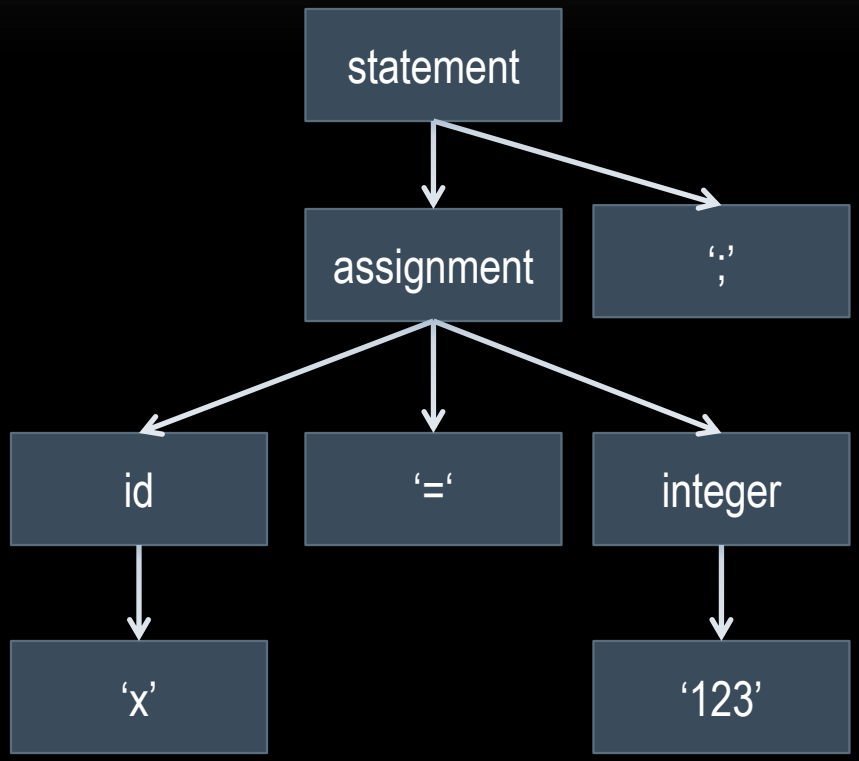
Trees as the Data Model

A simple grammar to recognize an integer assignment

```
statement     : assignment ';'
assignment    : id '=' integer
id            : 'x'
integer       : ('0' | '1' | … '9')+
```

Example Derivation: 'x = 123;'

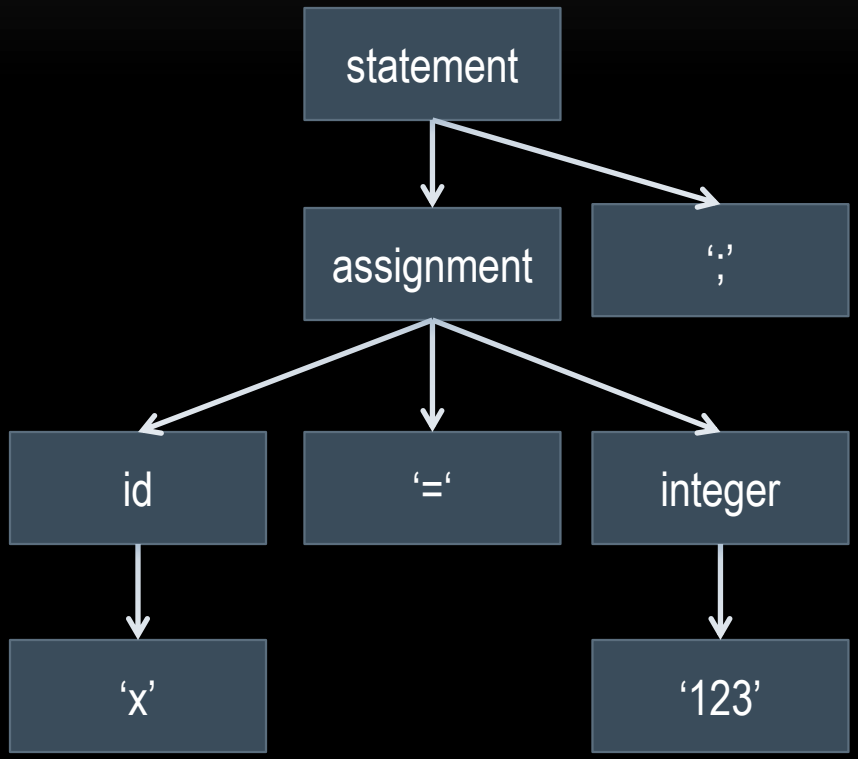
```
statement → assignment ';'
          → id '=' integer ';'
          → 'x' '=' integer ';'
          → 'x' '=' 123 ';'
          → 'x' '=' 123 ';'
```



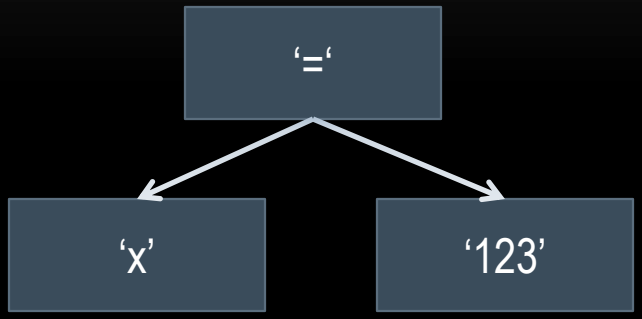
Called a *Parse Tree*

Not all Tress are Created Equal

Parse Tree



Abstract Syntax Tree (AST)



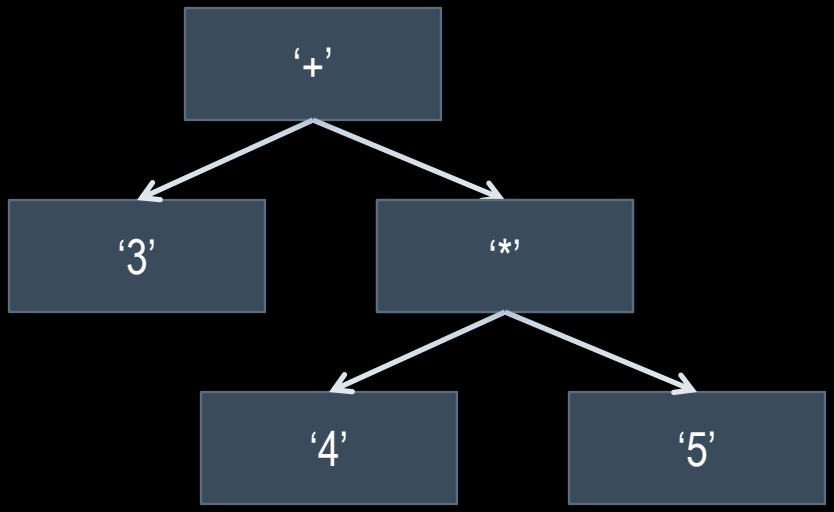
- **Dense**: no unnecessary nodes
- **Convenient**: easy to traverse
- **Meaningful**: emphasize operators, operands, and their relationship

AST Encoding Trickery

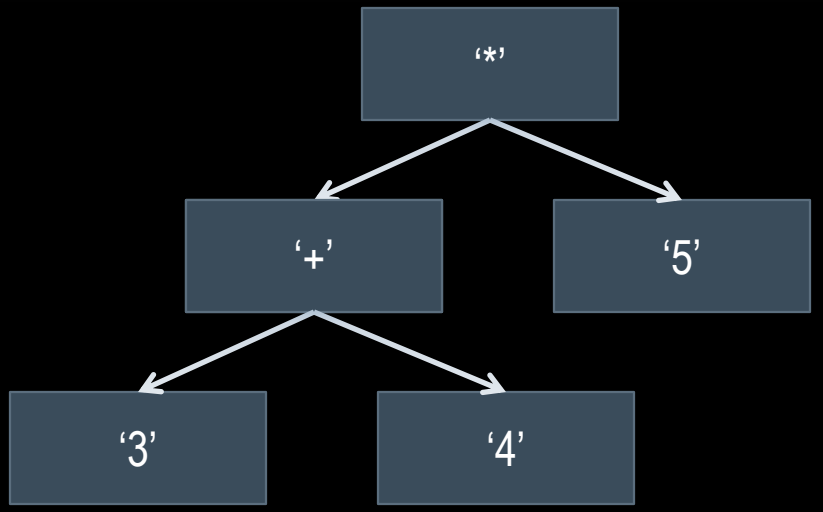
How can we represent ‘3+4∗5’ as an AST?

AST Encoding Trickery: Operator Precedence

AST for ‘3+4∗5’



AST for ‘(3+4)∗5’



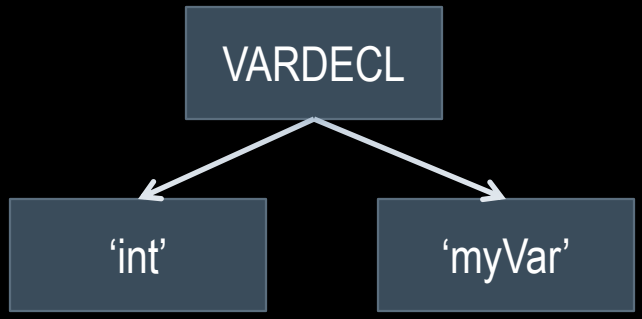
If “x happens before y”, encode “x” lower than “y” in the tree

AST Encoding Trickery

How can we represent ‘int myVar;’ as an AST?

AST Encoding Trickery: Imaginary Tokens

AST for ‘int myVar;’



Walking the AST

The AST encodes relevant tokens and their relationships

Tree walking may be used to:

Inspect the AST

- create a symbol table
- enforce static typing

Modify the AST

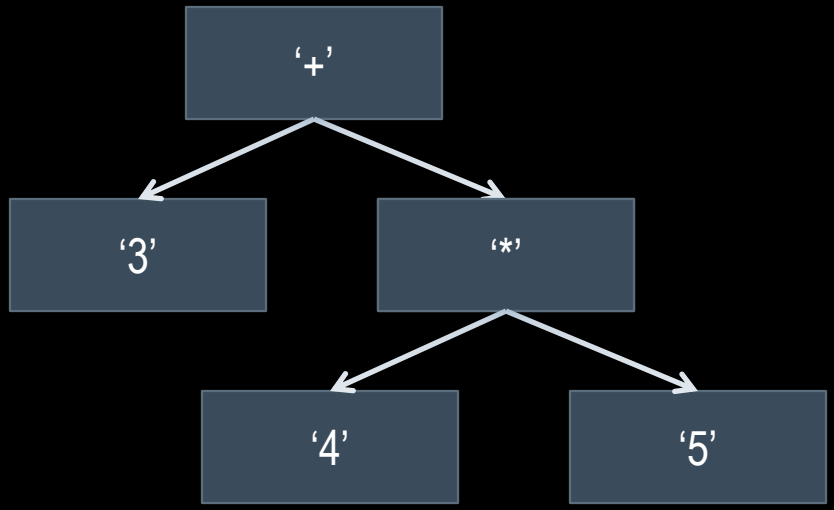
- simplify numerical or logical expressions

Execute actions

- compile source code to lower level representation
- analyze source code to produce a bug report

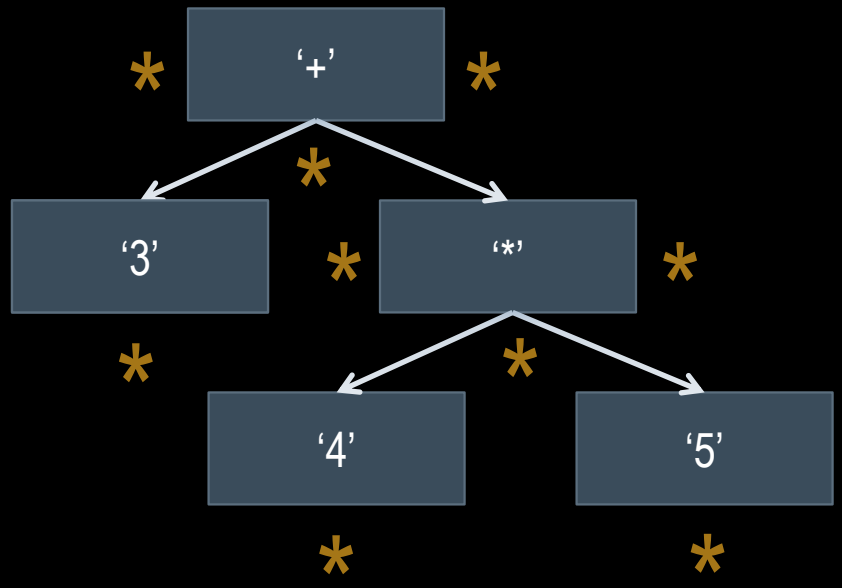
Walking Strategies

AST for ‘3+4*5’



Walking Strategies

Visitation Opportunities

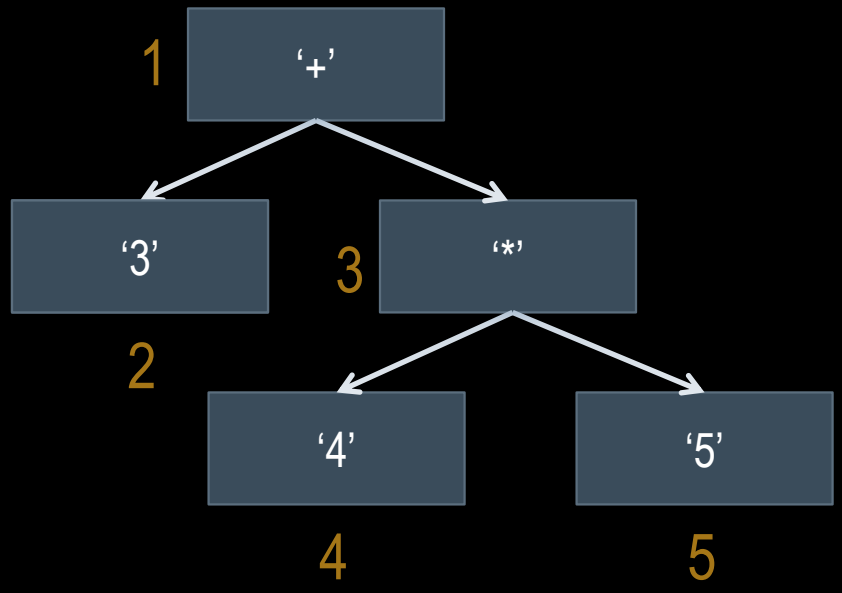


Visitation Pseudocode

```
public void walk() {
    <preorder-action>
    left.walk();
    <inorder-action>
    right.walk();
    <postorder-action>
}
```

Walking Strategies

Pre-Order Visitation



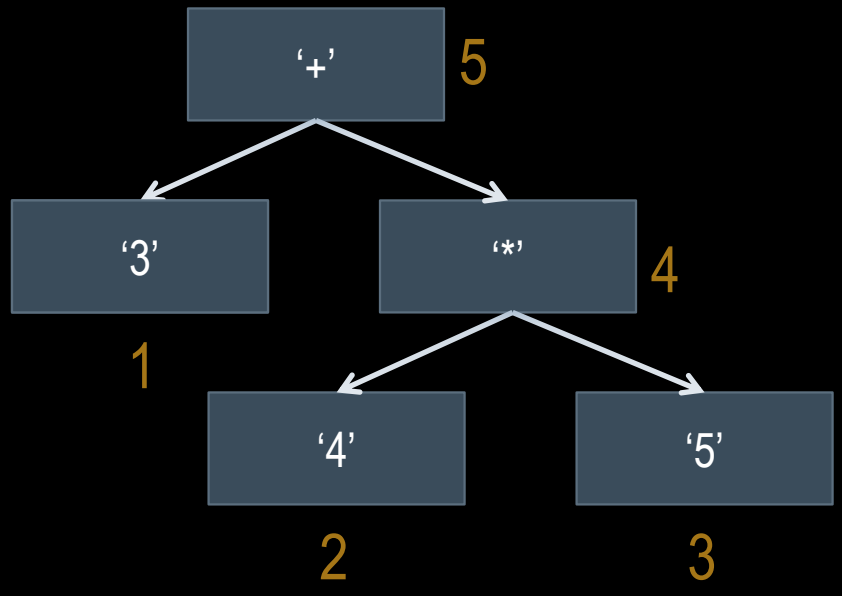
Visitation Pseudocode

```
public void walk() {
    print(token);
    left.walk();
    right.walk();
}
```

Output: +3*45

Walking Strategies

Post-Order Visitation



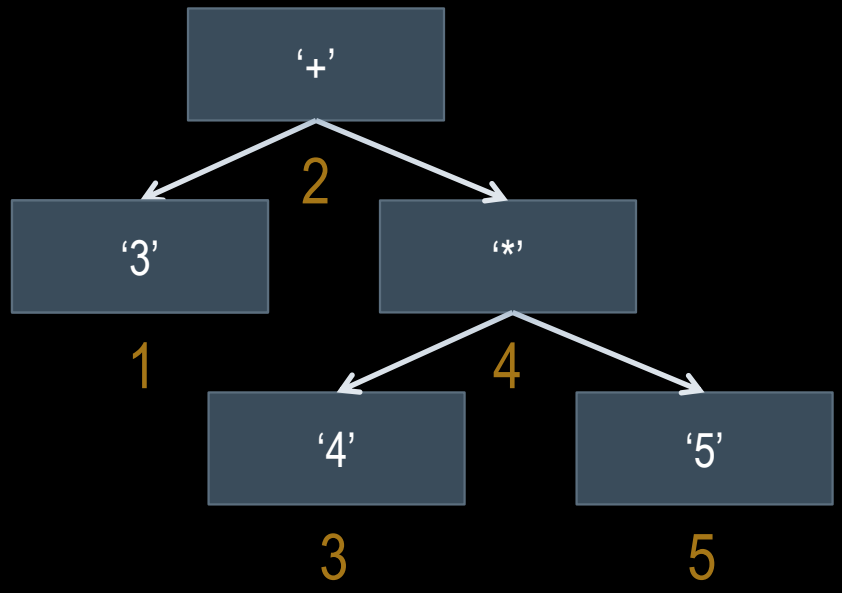
Visitation Pseudocode

```
public void walk() {
    left.walk();
    right.walk();
    print(token);
}
```

Output: 345*+

Walking Strategies

In-Order Visitation

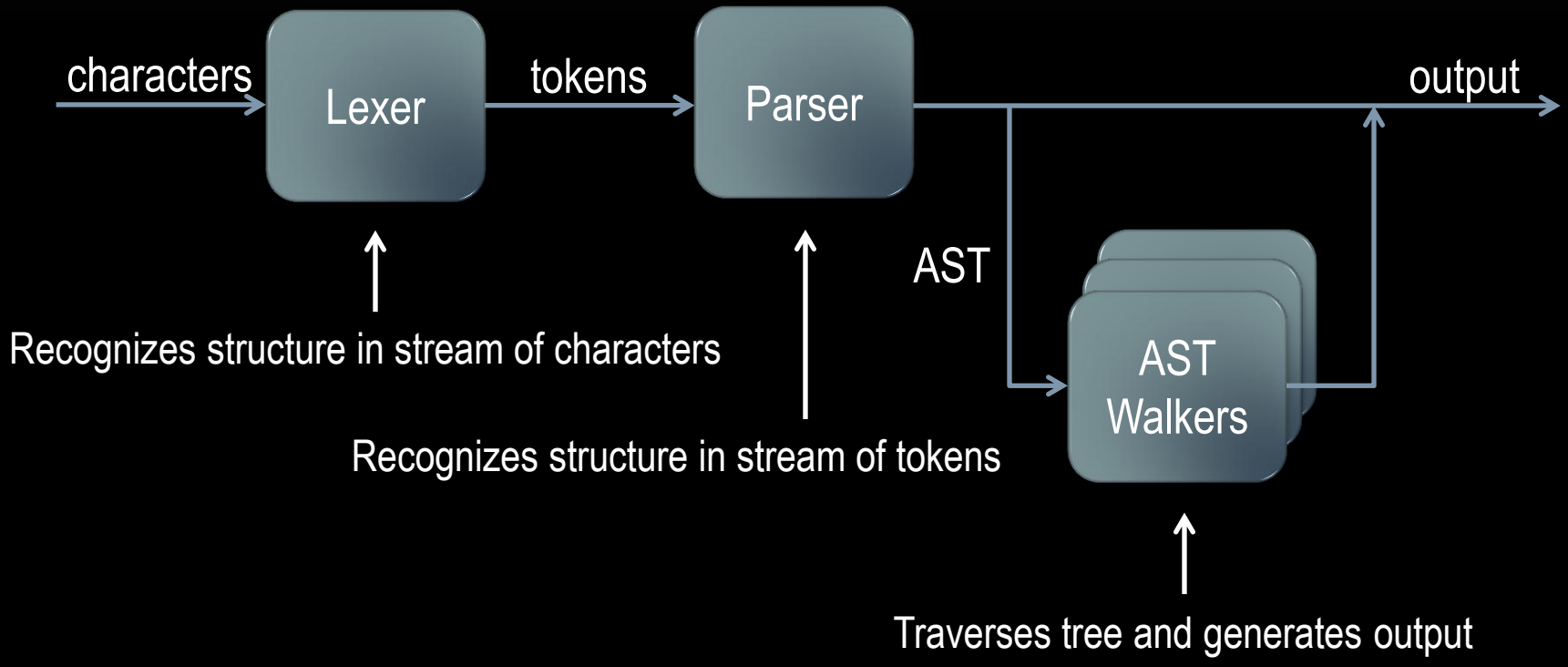


Visitation Pseudocode

```
public void walk() {
    left.walk();
    print(token);
    right.walk();
}
```

Output: 3+4*5

The Big Picture



External DSLs

I wasn't kidding…

External DSLs are a lot of work!

ANTLR to the Rescue

ANother Tool for Language Recognition

- Created and actively maintained by Terence Parr
- FOSS (3-clause BSD license)
- 100% Java
- 5,000 source downloads a month
- http://www.antlr.org

A framework for automating the construction of lexers, parsers, and tree walkers from grammatical descriptions

Code generation in many languages

- Java, JavaScript, Objective-C, Python, Ruby, C, C#, and more

ANTLR IDE

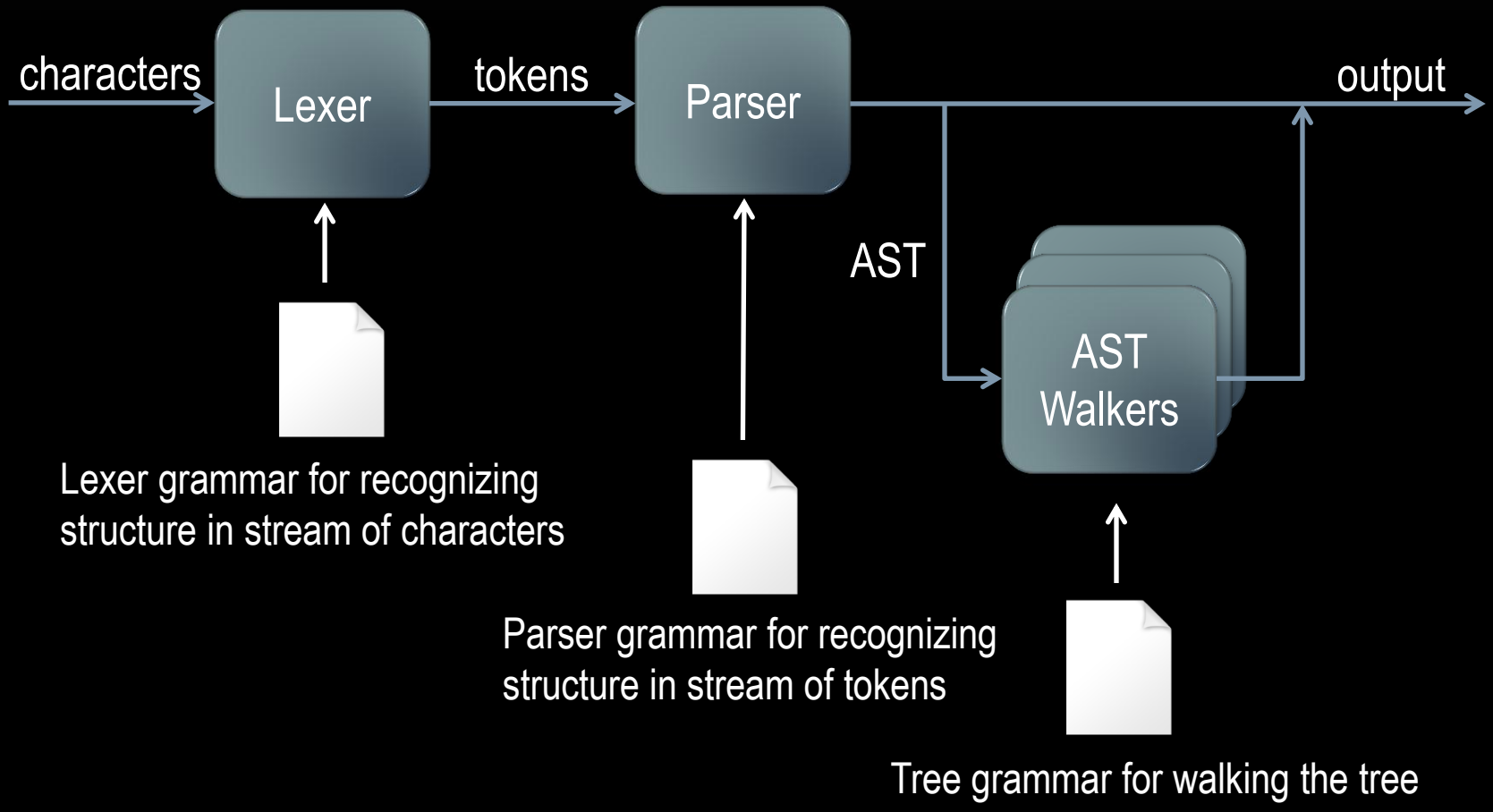
Eclipse plugin for developing ANTLR applications

- Syntax checking with auto-completion
- Source formatting
- Built-in debugger
- Visual interpreters for testing
- Automatic resource generation on save
- http://antlrv3ide.sourceforge.net/

A bit tricky to install

- Check out Scott Stanchfield's video tutorials
- http://javadude.com/articles/antlr3xtut/

The Big Picture with ANTLR

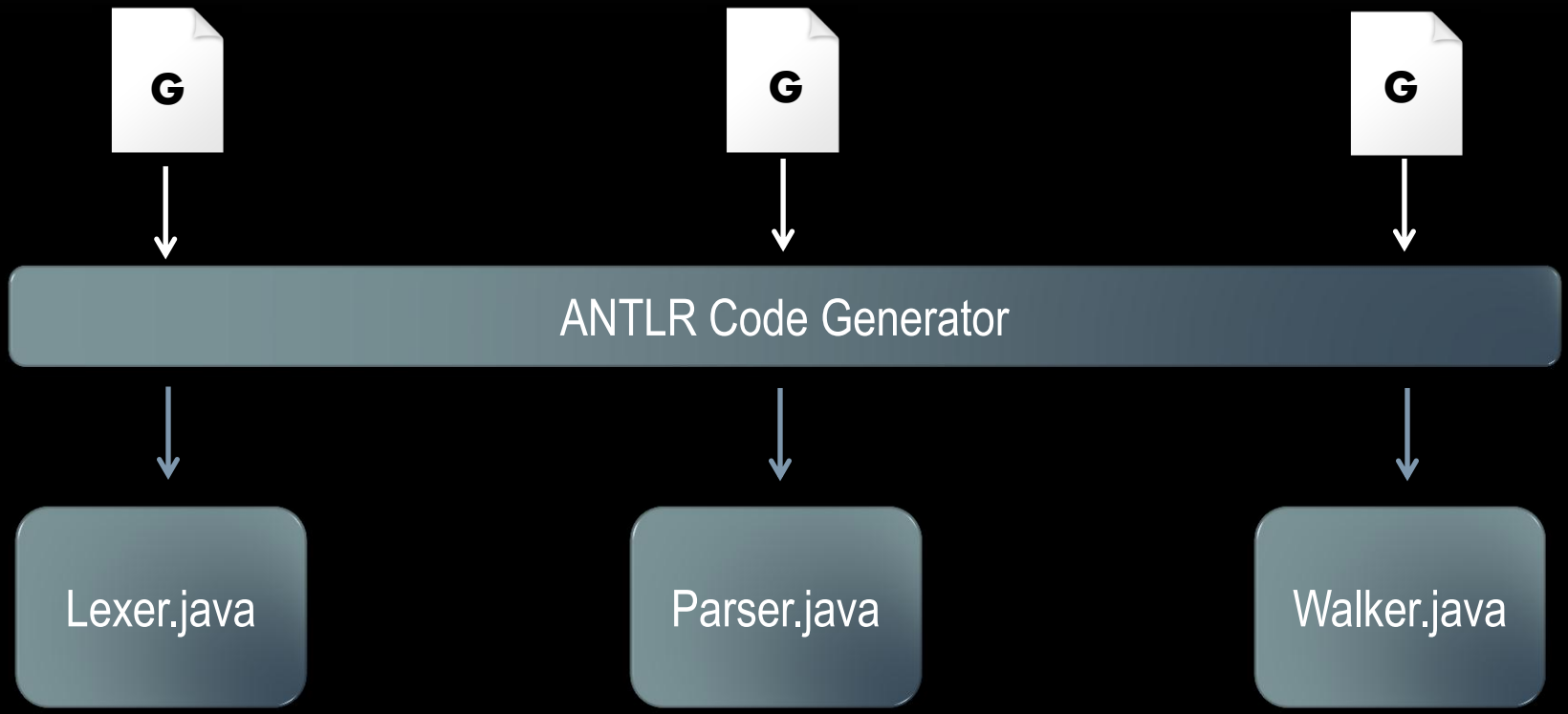


ANTLR Code Generation

Lexer grammar for recognizing structure in stream of characters

Parser grammar for recognizing structure in stream of tokens

Tree grammar for walking the tree



Lexer Grammar

```
lexer grammar LettersThenDigitsLexer;

@header {
  package lexer;
}

LETTER: LOWERCASE | UPPERCASE;

DIGIT:     '0'..'9';

fragment  LOWERCASE:      'a'..'z';

fragment  UPPERCASE:      'A'..'Z'

WS:
  (' ' | '\r' | '\t'  | '\f' | '\n') { $channel = HIDDEN; };
```

Grammar declaration

Java package of generated lexer class

Substitution rule
(rule must start with capital letter)

Fragment rule
(reusable "private" rule)

Filter whitespace tokens

Parser Grammar

```
parser grammar LettersThenDigitsParser;

options {
  tokenVocab = LettersThenDigitsLexer;
  language = Java;
}

@header {
  package parser;
}

start:      letter+ digit*;

letter:     LETTER;

digit:      DIGIT;
```

Grammar declaration

Lexer token vocabulary
Target language for code generation

Java package of generated parser class

Substitution rule
(rule must start with lowercase letter)

Embedding Custom Actions

```
parser grammar LettersThenDigitsParser;

options { … }

@header { … }

start:      letter+ digit*;

letter:     LETTER;

digit:      DIGIT;
```

Embedding Custom Actions

```
parser grammar LettersThenDigitsParser;

options { … }

@header { … }

start
  @init                                                    Rule initializer
  { StringBuilder buf = new StringBuilder(); }
  :
      ( t1 = letter { buf.append( $t1.text ); } )+         Aliasing with custom action
      ( t2 = digit { buf.append( $t2.text ); } )*
  { System.out.println(buf); };                            Custom action

letter:    LETTER
{ System.out.println( $LETTER.text ); };                   Custom action with token reference

digit:    DIGIT;
```

Overriding Code Generation Defaults

start: letter+ digit*

Overriding Code Generation Defaults

```
<access-modifier> start[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      letter+ digits* → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <alternative-1> → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <alternative-1> → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <alternative-1> → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <alternative-1> → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <alternative-1> → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <alternative-1> → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <alternative-1> → <rewrite-rule-1> |
      <alternative-2> → <rewrite-rule-2> |
      …
      <alternative-N> → <rewrite-rule-N>
      ;
    <exceptions-clause>
```

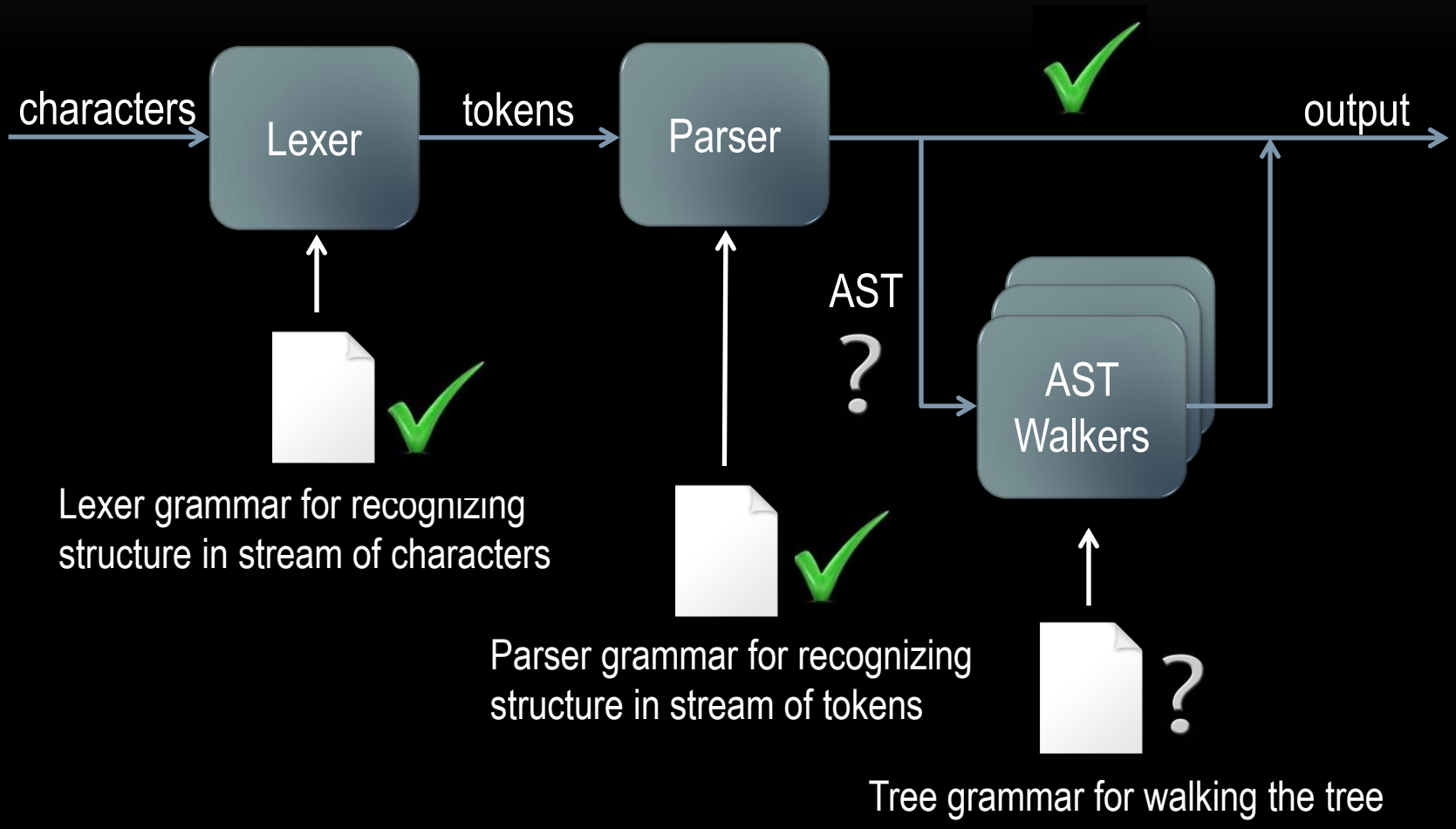
Letters Then Digits Example

Google Driving Directions Example

Example of Input:

```
turn left onto “11th Ave” in 2.3 miles or 4 minutes
turn right onto “Hwy 13” in 9.4 miles or 11 minutes
turn slight right onto “I-94” tollroad in 60 miles or 55 minutes
exit at “10A” onto “Frontage Road” in 4.4 miles or 4 minutes
u-turn onto “13th Street” in 0.1 miles or 1 minutes
arrive at “Burger King” in 0.2 miles or 4 minutes
```

The Big Picture with ANTLR



ANTLR, ASTs, and Tree Walkers

Instead of recognizing a token sequence and executing custom actions, configure ANTLR to generate an AST

Parser produces an AST encoded as a stream of tokens

A “tree” grammar is used to recognize tree structure

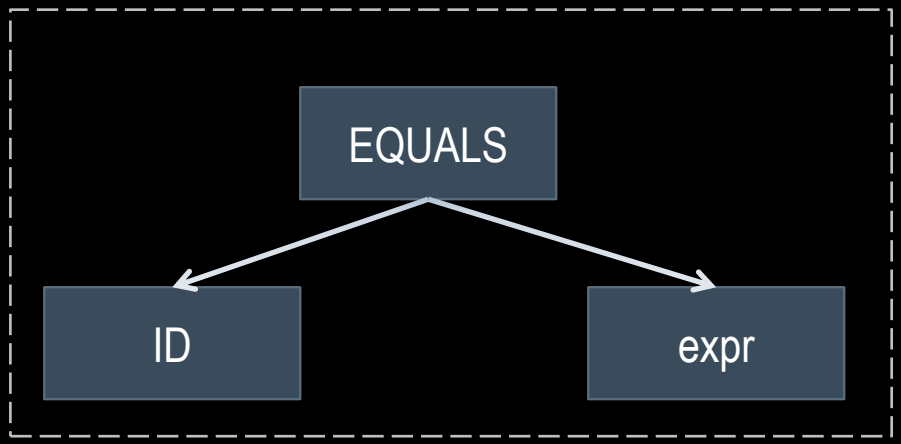
Custom actions embedded in “tree” grammar are executed during substitution rule evaluation

AST Construction in Parser Grammar

Given Substitution Rule:

assignment: ID EQUALS expr;

Desired AST:



Two Notations

Inline

assignment: ID EQUALS^ expr;

Rewrite Rule

assignment: ID EQUALS expr
→
^(EQUALS ID expr);

Recognizing AST in Tree Grammar

Parser AST Construction

assignment: ID EQUALS^ expr;

or

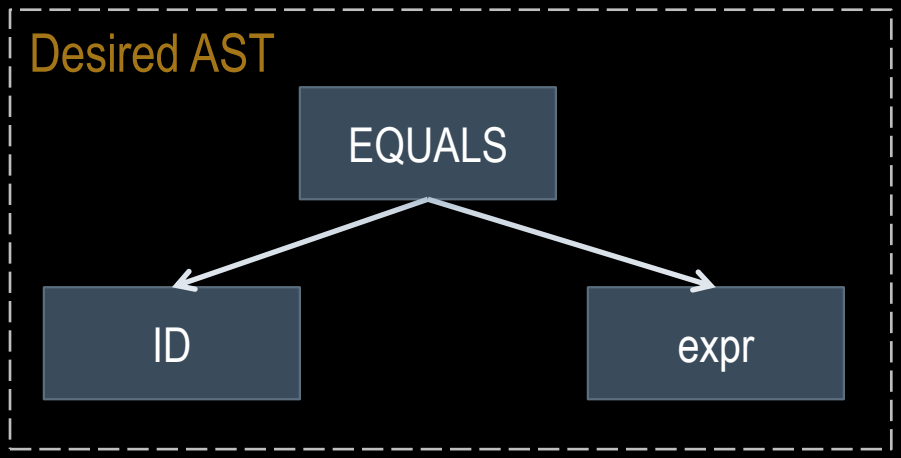
assignment: ID EQUALS expr

→

^(EQUALS ID expr);

Tree Grammar

assignment: ^(EQUALS ID expr);



A More Complicated Example

Given Substitution Rule:

declaration: DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;

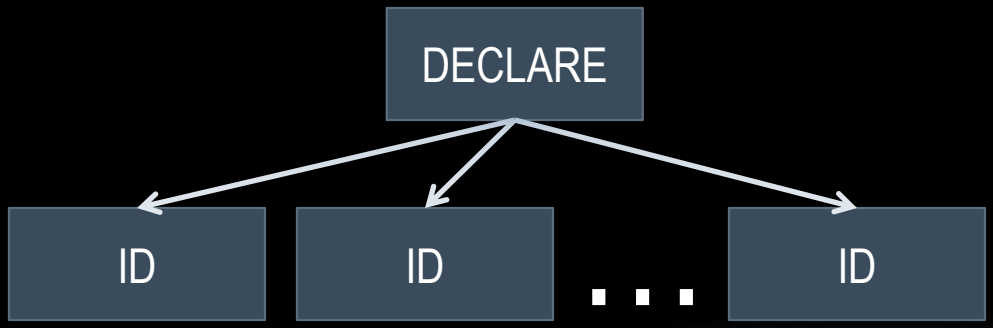
Desired AST: ?

A More Complicated Example

Given Substitution Rule:

declaration: DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;

AST with declaration only:

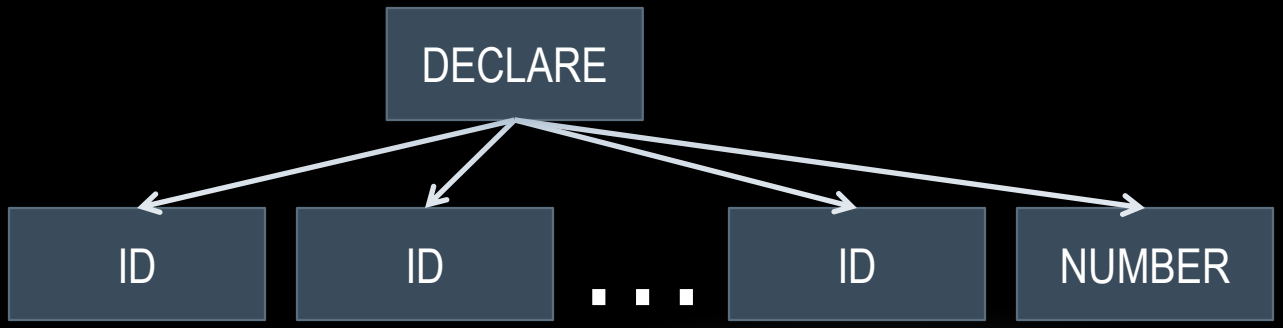


A More Complicated Example

Given Substitution Rule:

declaration: DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;

AST with declaration and initialization:



A More Complicated Example

Given Substitution Rule:

```
declaration:    DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;
```

Rewrite Rule for AST (in parser grammar):

```
declaration:    DECLARE ID (COMMA ID)* (EQUALS NUMBER)?
→
^(DECLARE ID+ (EQUALS NUMBER)?);
```

Tree Substitution Rule (in tree grammar):

```
declaration: ^(DECLARE ID+ (EQUALS NUMBER)?);
```

A More Complicated Example

Given Substitution Rule:

```
declaration:    DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;
```

Rewrite Rule for AST (in parser grammar):

```
declaration:    DECLARE ID (COMMA ID)* (EQUALS NUMBER)?
→
^(DECLARE ID+ (EQUALS NUMBER)?);
```

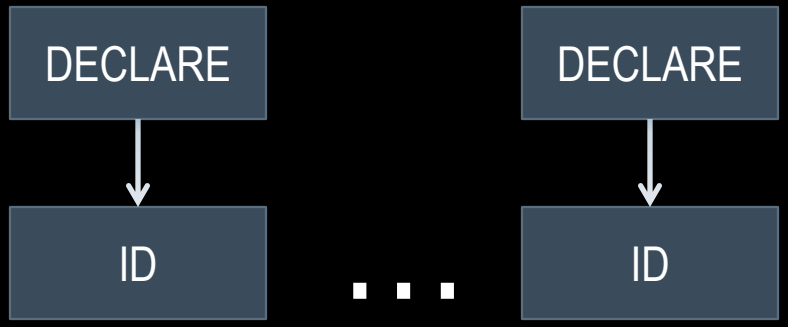
Tree Substitution Rule (in tree grammar):

```
declaration: ^(DECLARE idList += ID+ (EQUALS NUMBER)?)
 { for(Object id : idList) { … } };      // referencing IDs will require aliasing
```

A More Complicated Example

Alternative AST with declaration only:

declaration: DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;



A More Complicated Example

Alternative AST with declaration and initialization:

declaration: DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;

A More Complicated Example

Given Substitution Rule:

```
declaration:    DECLARE ID (COMMA ID)* (EQUALS NUMBER)?;
```

Rewrite Rule for AST (in parser grammar):

```
declaration:    DECLARE ID (COMMA ID)* (EQUALS NUMBER)?
→
^(DECLARE ID (EQUALS NUMBER)?)+;
```

Tree Substitution Rule (in tree grammar):

```
declaration:    ^(DECLARE ID (EQUALS NUMBER)?);    // no '+' on ID
```

Tree Grammar

```
tree grammar LettersThenDigitsWalker;

options {
  tokenVocab = LettersThenDigitsLexer;
  ASTLabelType = CommonTree;
}

@header {
  package walker;
}

declaration:
    ^(DECLARE ID (EQUALS NUMBER)?)
```

Grammar declaration

Lexer token vocabulary
Type of tree labels (Object by default)

Java package of generated parser class

Substitution rule
(rule must start with lowercase letter)

Embedding Actions

Same syntax as custom actions in parser grammar

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <tree-alternative-1> |
      <tree-alternative-2> |
      …
      <tree-alternative-N>
      ;
    <exceptions-clause>
```

Overriding Code Generation Defaults

```
<access-modifier> ruleName[<args>] returns [<ret-args>]
    <throws-clause>
    <options>
    <rule-attribute-scopes>
    <rule-actions>
      :
      <tree-alternative-1> |
      <tree-alternative-2> |
      …
      <tree-alternative-N>
      ;
    <exceptions-clause>
```

Simple Script Example

Example of Input:

```
/*
  Compute cylinder volume.
*/
declare circleArea
declare height = 10
declare radius = 1.2

// compute circleArea
circleArea = PI * radius**2
display circleArea

// compute cylinder volume and display
circleArea * height
```

Summary

External DSLs offer flexibility at the cost of complexity

- Designed entirely from the ground-up
- Requires implementation of language
- May require tooling and documentation for users
- Requires uncommon skills (e.g., language theory, design)

ANTLR Substantially Reduces Barrier of Entry

- Streamlines DSL development with code generation
- IDE support for development and testing
- Easily integrates with Java-based systems

References

http://www.antlr.org

http://www.en.wikipedia.org/wiki/Formal_grammar

http://javadude.com/articles/antlr3xtut/

Ghosh, Debasish. DSLs in Action. Manning Publications, 2010

Parr, Terence. Language Implementation Patterns. The Pragmatic Programmers, 2010

Parr, Terence. The Definitive ANTLR Reference. The Pragmatic Programmers, 2007

Sipser, Michael. Introduction to the Theory of Computation. PWS Publishing Company, 1997