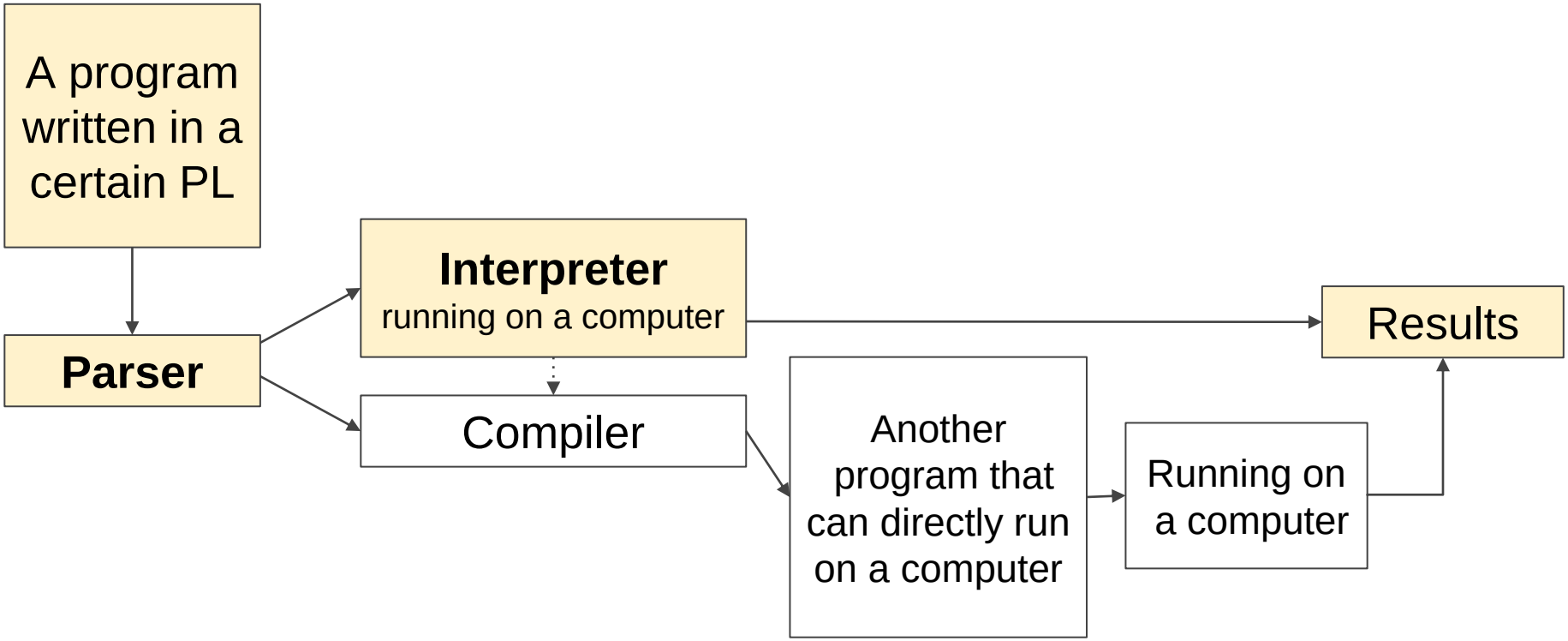
ITP20005
Modeling Languages (PLAI Chapter 1)

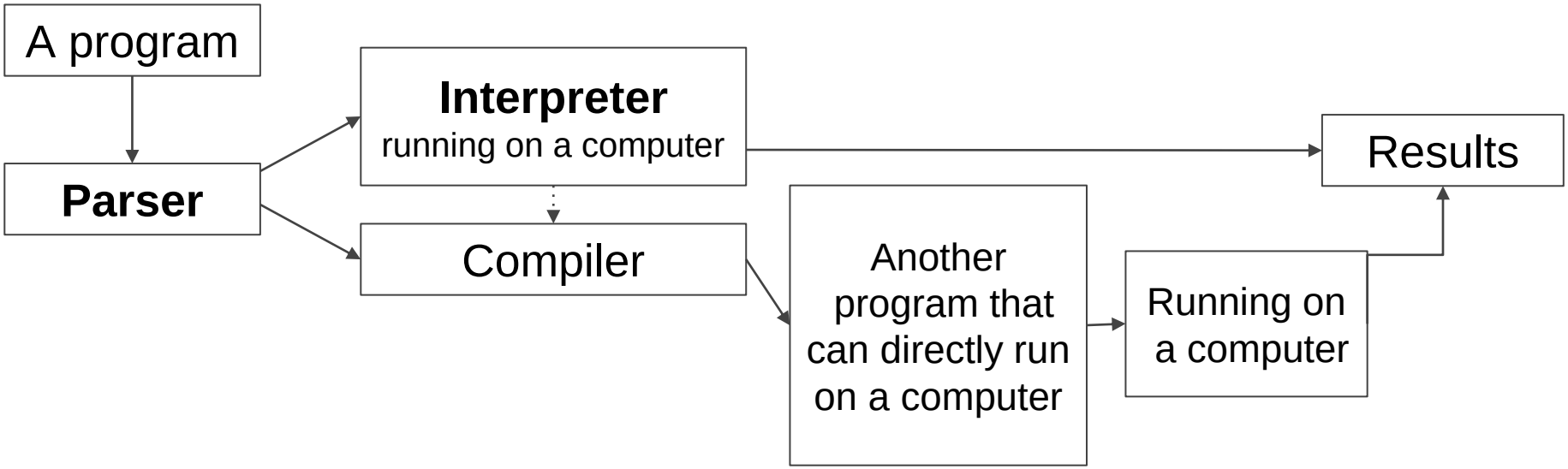
Lecture04
JC

Are we learning Raket or PLT???

Big Picture


A program written in a certain PL
Parser
Interpreter
running on a computer
Compiler
Results
Another program that can directly run on a computer
Running on a computer

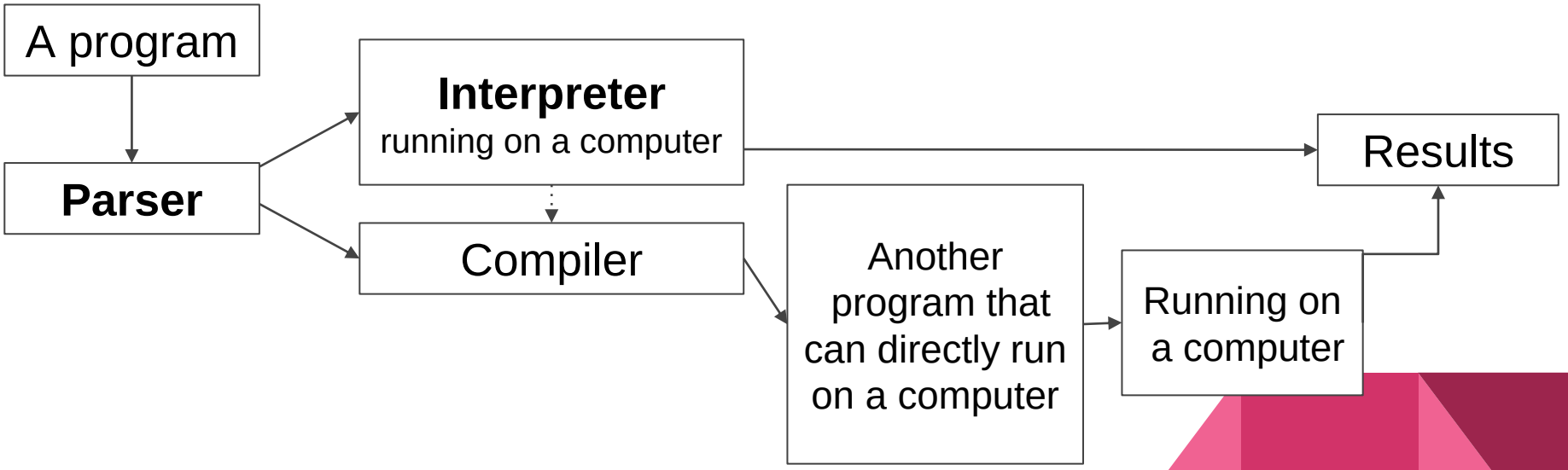
Big Picture (with a natural language example)



- An article (a set of sentences): I love you. [...]
- Parser to generate the abstract form of a sentence.
  - I (subject) love (verb) you (object).
- Interpreter (English → Korean)
  - 나는(subject) 여러분을 (object) 사랑합니다 (verb).
  - Love is (has been) growing in the writer's mind.
  - Feeling or actual behaviors from a writer.

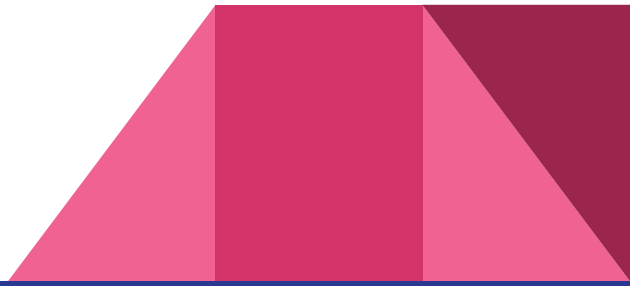
Big Picture (modeling languages)

- Just write an interpreter to explain a language.
- By writing an interpreter, we can understand the language!
- Interpreter can be converted into a compiler!!!

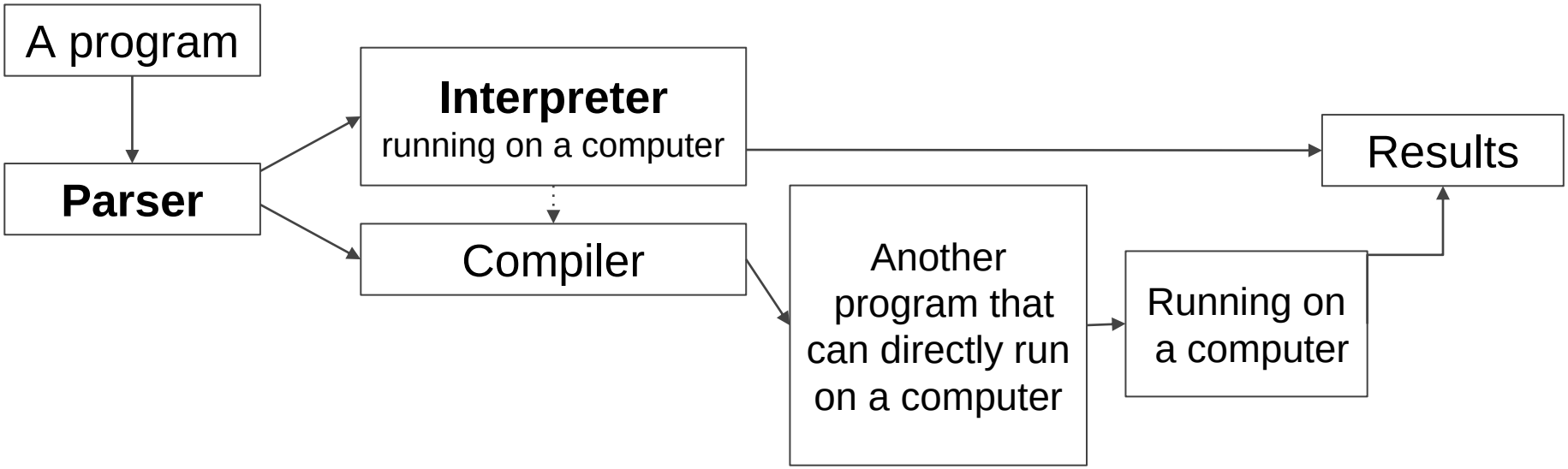


Agenda (Important PL knowledge)

- Syntax
- Semantics
- Parser
- Interpreter (we start to implement an interpreter from Lecture 5. Please, be ready!)



Big Picture (with a natural language example)



- An article (a set of sentences): I love you. [...]
- Parser to generate the abstract form of a sentence.
  - I (subject) love (verb) you (object).
- Interpreter (English → Korean)
  - 나는 (subject) 여러분을 (object) 사랑합니다 (verb).
  - Love is (has been) growing in the writer's mind.
  - Feeling or actual behaviors from a writer.

What does PL consist of?

Programing Languages (by Krishnamurthi)

- Peculiar syntax
- Some behaviors associated with each syntax
- Numerous useful libraries
- A collection of idioms that programmers of that language use

Programing Languages (by Krishnamurthi)

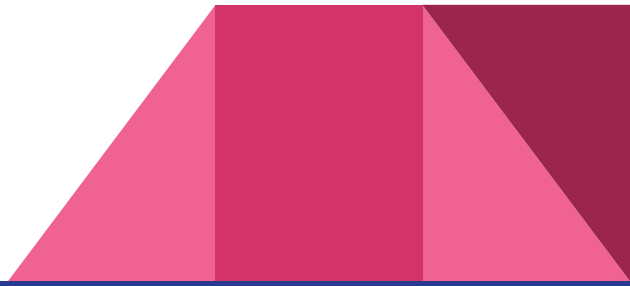
- Peculiar syntax
- Some behaviors associated with each syntax
- Numerous useful libraries
- A collection of idioms that programmers of that language use

Which one is most significant to learn PLT?

Syntax??

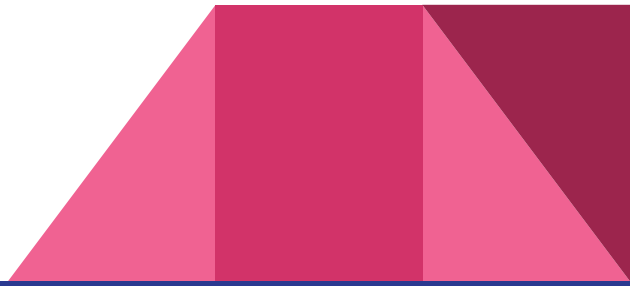
- Which two are most alike?
  - a[25] in Java
  - (vector-ref a 25) in Racket
  - a[25] in C
  - a[25] in ML or Haskell

- It's quite distracting to study a language.
  - So, we are going to use a uniform syntax for a new language we implement in our lectures.



The size of a language's library?

- Might be very important for developers to complete their tasks.
- Also distracting to study PL theories as the library is not the core of the language.



Idioms?

- Expressing a special feature of a recurring construct in one or more programming languages
  - See some examples: http://wiki.c2.com/?JavaIdioms
- Might be incomplete, dangerous,...
  - Should be careful to not read too much into it
- Developer-oriented but not language-oriented

Then, just semantics!!!

Behaviors associated with each Syntax

Behaviors associated with each Syntax

SEMANTICS!!

Modeling Semantics

The most precise languages in terms of semantics?

Mathematics?

- Denotational semantics
  (https://en.wikipedia.org/wiki/Denotational_semantics)
- Operational semantics
  (https://en.wikipedia.org/wiki/Operational_semantics)
- Axiomatic semantics
  (https://en.wikipedia.org/wiki/Axiomatic_semantics)

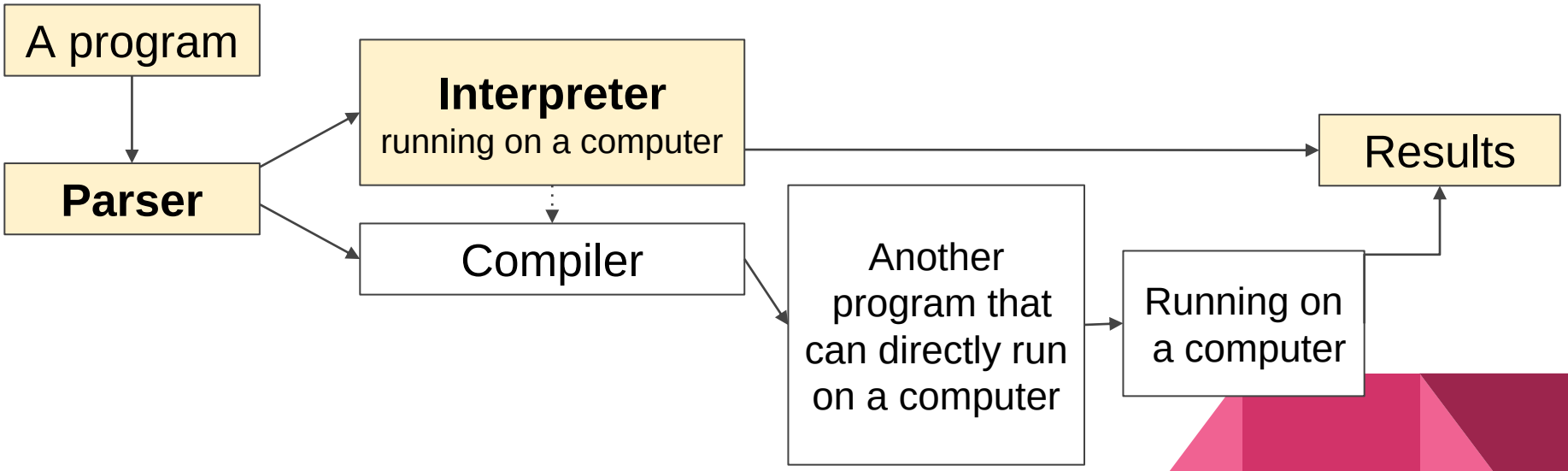
⇒ Too advanced

Interpreter semantics

for modeling languages
(A 'cousin' of the operational semantics)

It's simple~!

- Just write an interpreter to explain a language.
- By writing an interpreter, we can understand the language!
- Interpreter can be converted into a compiler!!!



An interpreter is a program.

An interpreter is a program.

Then, what language do we use to implement the interpreter??????

"Others have already worked out the mathematical semantics of the simple language."

So, we just start from here...

Modeling Syntax

(Semantics are most significant but we need syntax anyway.)

Modeling syntax

- Concrete Syntax ('expression')
  - 3 + 4 (infix)
  - 3 4 + (postfix)
  - (+ 3 4) (parenthesized prefix)

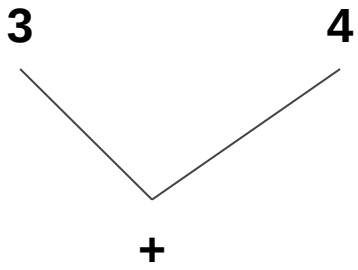
  ⇒ Each of these notations is in use by at least one programming language.

Can we have a general from of these syntax??

Modeling syntax

- Concrete Syntax ('expression')
  - 3 + 4 (infix)
  - 3 4 + (postfix)
  - (+ 3 4) (parenthesized prefix)

    ⇒ Each of these notations is in use by at least one programming language.
- Abstract syntax in a tree form (essence in a tree form)

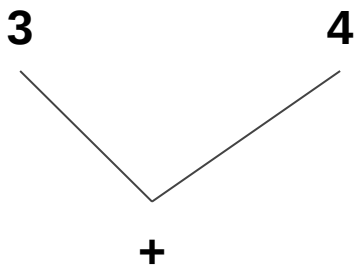


(abstract syntax tree)

Modeling syntax

- Concrete Syntax ('expression')
  - 3 + 4 (infix)
  - 3 4 + (postfix)
  - (+ 3 4) (parenthesized prefix)

    ⇒ Each of these notations is in use by at least one programming language.
- Abstract syntax in a tree form (essence in a tree form)



  - 'Representation' with the 'right data definition' in Racket
    - (add (num 3) (num 4))

Modeling syntax

- Another example
  - (3 - 4) + 7 (infix)
  - 3 4 - 7 + (postfix)
  - (+ (- 3 4) 7) (parenthesized prefix)
- Representation with the 'right data definition' (abstract syntax) in Racket
  - (add (sub (num 3) (num 4))
    (num 7))

Why do we need abstract syntax?

Modeling syntax

- Abstract syntax representation: 'one' data definition for the example above in Racket

```
(define-type AE
    [num (n number?)]
    [add (lhs AE?)
         (rhs AE?)]
    [sub (lhs AE?)
         (rhs AE?)])
```

⇒ AE stands for "Arithmetic Expression".

Modeling syntax

- Why do we use the *lhs* and *rhs* sub-expressions of type AE rather than those of type *num*?

```
(define-type AE
  [num (n number?)]
  [add (lhs AE?)
       (rhs AE?)]
  [sub (lhs AE?)
       (rhs AE?)])
```

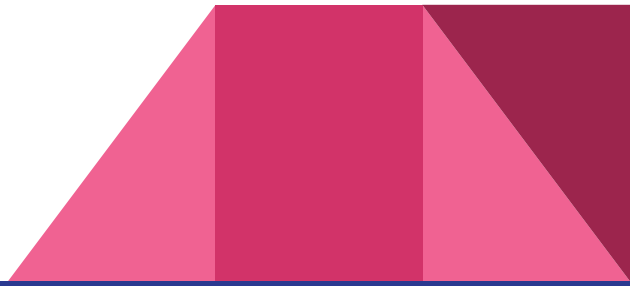
```
(define-type AE
  [num (n number?)]
  [add (lhs num?)
       (rhs num?)]
  [sub (lhs num?)
       (rhs num?)])
```

- Provide sample expressions permitted by both, and permitted by the former but rejected by the later, and argue that our choice is reasonable.

Parser

Revisit important concepts in PLs

- Syntax
- Semantics
- Grammar
  - Backus-Naur Form



Parser

- A parser is a component in an interpreter or compiler.
  - Identifies what kinds of program code it is examining, and
  - Converts concrete syntax (what we type) into abstract syntax.
- To do this, we need a clear specification of the concrete syntax of the language!!
  - How to specify???
    - We use Backus-Naur Form (BNF)

Specify the concrete syntax of the language

- An algebraic grammar in BNF (Backus-Naur Form):

〈expr〉 ::= (〈expr〉 + 〈expr〉)
| (〈expr〉 - 〈expr〉)
| 〈num〉
〈num〉 ::= 1, 42, 17, …

  - <expr>: Non-terminal (can rewrite it as one for the things on the right-hand side)
    Meta-variable
  - ::=: "Can be written as"
  - |: "one more choice" (a production)
  - <...>: literal syntax
  - Terminal

Specify the concrete syntax of the language (2)

- An algebraic grammar in BNF (Backus-Naur Form):

〈expr〉 ::= (〈expr〉 + 〈expr〉)
| (〈expr〉 - 〈expr〉)
| 〈num〉
〈num〉 ::= 1, 42, 17, …

Each meta variable, such as 〈expr〉, defines a set

Specify the concrete syntax of the language (3)

- Using a BNF Grammar: 〈num〉

〈num〉 ::= 1, 42, 17, … number

The set 〈num〉 is the set of all numbers.

To make an example 〈num〉, pick an element from it:

2 ∈ 〈num〉

298 ∈ 〈num〉

Specify the concrete syntax of the language (4)

- Using a BNF Grammar: 〈expr〉

〈expr〉 ::= (〈expr〉 + 〈expr〉)	addition
| (〈expr〉 - 〈expr〉)	subtraction
| 〈num〉	number

To make an example 〈expr〉:

- Choose one case in the grammar
- Pick an example for each meta variable
- Combine the examples with literal text

Specify the concrete syntax of the language (5)

- Using a BNF Grammar: 〈expr〉

〈expr〉 ::= (〈expr〉 + 〈expr〉)		addition
	| (〈expr〉 - 〈expr〉)	subtraction
	| 〈num〉	number

To make an example 〈expr〉:

- Choose one case in the grammar
- Pick an example for each meta variable
- Combine the examples with literal text

〈num〉
$7 \in$ 〈num〉
$7 \in$ 〈num〉

Specify the concrete syntax of the language (6)

- Using a BNF Grammar: 〈expr〉

〈expr〉 ::=	(〈expr〉 + 〈expr〉)	addition
|	(〈expr〉 - 〈expr〉)	subtraction
|	〈num〉	number

To make an example 〈expr〉:

- Choose one case in the grammar
- Pick an example for each meta variable
- Combine the examples with literal text

(〈expr〉 + 〈expr〉)
8 ∈ 〈num〉 ⊆ 〈expr〉
(8 + 8) ∈ 〈expr〉

Example: A Grammar for Arithmetic Expressions

(our preliminary language that can support simple arithmetic computation.)

- Example syntax of new arithmetic expressions (AE) we want to use.

  {+ {- 3 4 } 7}
- Specify in BNF

  <AE> ::= <num>

  | {+ <AE> <AE>}

  | {- <AE> <AE>}

Example: A Grammar for Arithmetic Expressions

- Example syntax of new arithmetic expressions (AE) we want to use.
  {+ {- 3 4 } 7}
- Specify in BNF
  <AE> ::= <num>
  | {+ <AE> <AE>}
  | {- <AE> <AE>}
- Abstract syntax representation In Racket

```
(define-type AE
  [num (n number?)]
  [add (lhs AE?)
       (rhs AE?)]
  [sub (lhs AE?)
       (rhs AE?)])
```

Example: A Grammar for Arithmetic Expressions

- Example syntax of new arithmetic expressions (AE) we want to use.
  {+ {- 3 4 } 7}
- Specify in BNF
  <AE> ::= <num>
  | {+ <AE> <AE>}
  | {- <AE> <AE>}
- Abstract syntax representation in Racket

```
(define-type AE
  [num (n number?)]
  [add (lhs AE?)
       (rhs AE?)]
  [sub (lhs AE?)
       (rhs AE?)])
```

*** Example usages based on AE.**

```
(define ae1 (add (sub (num 3) (num 4)) (num 7)))
(sub? ae1)                ; Checking type

; retrieving expressions
(add-rhs ae1)
(sub-rhs (add-lhs ae1))
```

BNF captures both the concrete syntax and a default abstract syntax!

(That is why BNF has been used in definitions of languages. Let's see some examples...)

https://users-cs.au.dk/amoeller/RegAut/JavaBNF.html

https://cs.wmich.edu/~gupta/teaching/cs4850/sumII06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm

https://docs.python.org/3/reference/grammar.html

JC
jcnam@handong.edu
https://lifove.github.io

TODO

Read Chapter 2. Interpreting arithmetic

* Slides are from Prof. Sukyoung Ryu's PL class in 2018 Spring or created by JC based on the main text book.