

An Example of a Simple Language, IMP

CS 336

Lecture 3

Tuesday, August 29, 2000

Reference

Textbook

Glynn Winskel, The Formal Semantics of Programming Languages: An Introduction, 1996, MIT Press, chapter 2, Introduction to operational semantics

Purpose

To define a simple (impractical) language IMP which is used throughout the text book

To use the definition of IMP as a basis for discussion of issues of programming language definition

Outline

1. Arithmetic Expressions
2. Relational Expressions
3. Logical Expressions
4. Statements
5. The IMP Language

1. Arithmetic Expressions

Addition

Subtraction

Multiplication

Division

Addition of Integers

1 is in the set S

closure: if s_1 and s_2 are in set S, then $s_1 + s_2$ is in S

This gives us all positive integers, $\{1, 2, \dots\}$

identify: there is a member of S, 0, such that $s + 0 = 0 + s$ for all s in S

This gives us all nonnegative integers, $\{0, 1, 2, \dots\}$

associativity: if s_1 , s_2 , and s_3 are in set S, then $(s_1 + s_2) + s_3 = s_1 + (s_2 + s_3)$

commutativity: if s_1 and s_2 are in set S, then $s_1 + s_2 = s_2 + s_1$

Subtraction of Integers

for each s in S does there exist an s' in S
such that $s + s' = 0$?

inverse

only if we include negative integers

Inverse Elements for Addition of Integers

if s is in S , then there is a unique s' in S such that $s + s' = 0$

We write s' as $-s$

We then define the subtraction operation by

$$s_1 - s_2 = s_1 + -s_2$$

Subtraction is Neither Associative Nor Commutative!

Associativity

example

$$(9 - 5) - 3 = 4 - 3 = 1$$

$$9 - (5 - 3) = 9 - 2 = 7$$

Commutativity

example

$$9 - 5 = 4$$

$$5 - 9 = -4$$

Multiplication of Integers

closure: if s_1 and s_2 are in set S , then $s_1 * s_2$ is in S

(Closure of multiplication gives us no more operations than does closure of addition with inverse.)

identify: there is a member of S , 1 , such that $s * 1 = 1 * s$ for all s in S

associativity: if s_1 , s_2 , and s_3 are in set S , then $(s_1 * s_2) * s_3 = s_1 * (s_2 * s_3)$

commutativity: if s_1 and s_2 are in set S , then $s_1 * s_2 = s_2 * s_1$

Inverse of Multiplication

Does each integer have an integer multiplicative inverse?

No! Only 1 has an integer inverse.

If we include the multiplicative inverses we expand the set to be the rational numbers.

Note: 0 has no multiplicative inverse in the rational numbers. $0 * a = a * 0 = 0$

Grammar 1 for Arithmetic Expressions

AEXP ::= AEXP + AEXP

A EXP ::= AEXP - AEXP

AEXP ::= AEXP * AEXP

AEXP ::= NUM

AEXP ::= LOC

NUM ::= string of one or more digits

LOC ::= string of one or more letters followed by zero or more digits

(Notice: this is *not* usual syntax for identifiers.)

Parsing using Grammar 1

AEXP \Rightarrow AEXP - AEXP

\Rightarrow LOC - AEXP

\Rightarrow a - AEXP

\Rightarrow a - AEXP * AEXP

\Rightarrow a - LOC * AEXP

\Rightarrow a - b * AEXP

\Rightarrow a - b * AEXP + AEXP

\Rightarrow a - b * LOC + AEXP

\Rightarrow a - b * c + AEXP

\Rightarrow a - b * c + LOC

\Rightarrow a - b * c + d

Parsing Same Expression using Grammar 1

AEXP \Rightarrow AEXP + AEXP
 \Rightarrow AEXP - AEXP + AEXP
 \Rightarrow LOC - AEXP + AEXP
 \Rightarrow a - AEXP + AEXP
 \Rightarrow a - AEXP * AEXP + AEXP
 \Rightarrow a - LOC * AEXP + AEXP
 \Rightarrow a - b * AEXP + AEXP
 \Rightarrow a - b * LOC + AEXP
 \Rightarrow a - b * c + AEXP
 \Rightarrow a - b * c + LOC
 \Rightarrow a - b * c + d

Ambiguous Grammars and Languages

A grammar G is ambiguous if some string x in $L(G)$ has two different parse trees.

A language L is ambiguous if every grammar for L is ambiguous.

Grammar 1 is ambiguous.

An Unambiguous Grammar 2 for Arithmetic Expressions

AEXP ::= AEXP + ATERM

A EXP ::= AEXP - ATERM

ATERM ::= ATERM * AFACTOR

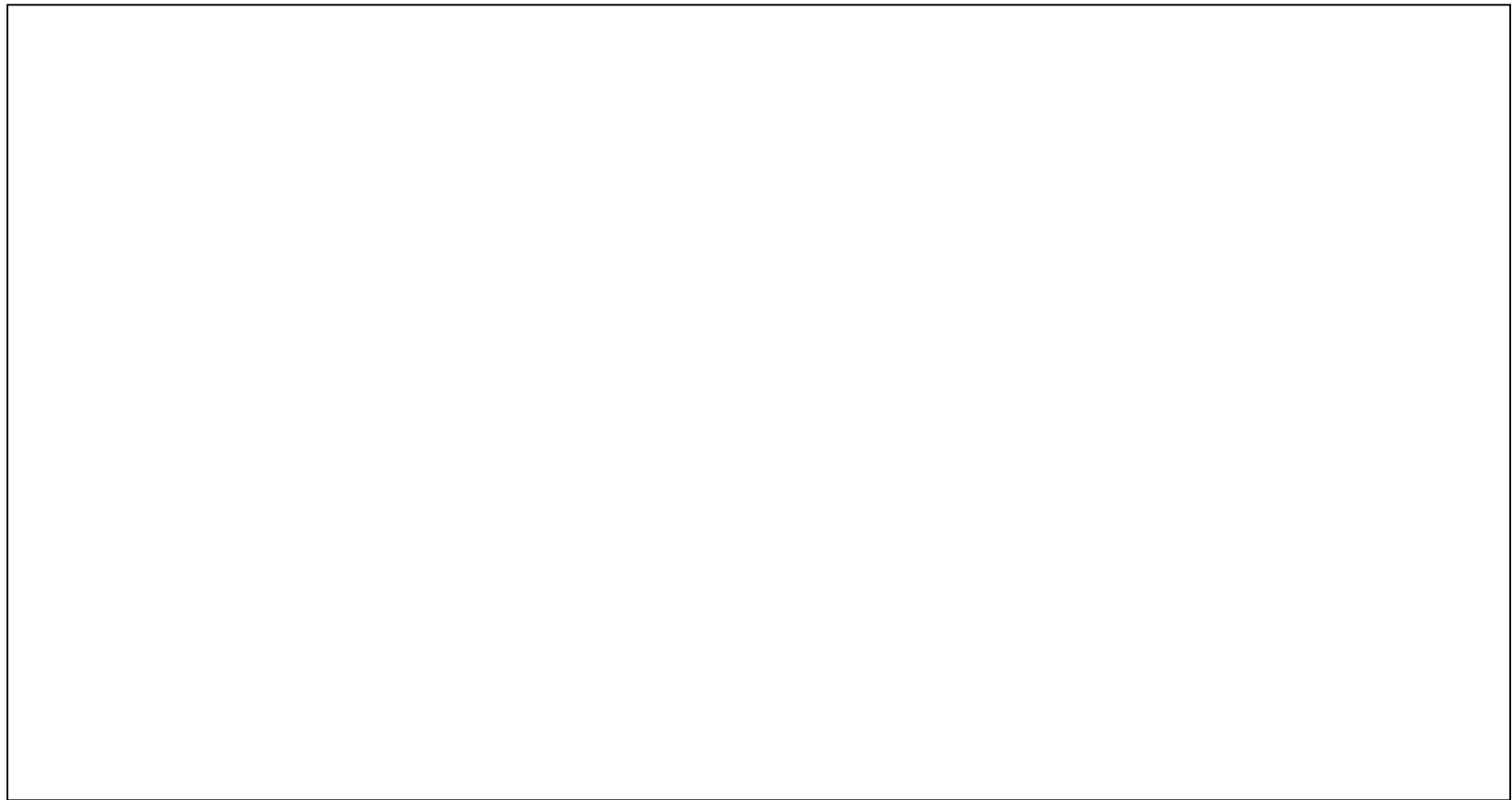
AFACTOR ::= NUM

AFACTOR ::= LOC

NUM ::= string of one or more digits

LOC ::= string of one or more letters followed by zero or more digits

Parsing using Grammar 2



Some Comments about Grammar 2

Two productions generally used to define arithmetic expressions are missing:

$ATERM ::= ATERM * AFACTOR$

$AFACTOR ::= (AEXP)$

Our interest in grammars in CS336 is different from our interest in grammars in CS236.

Abstract Syntax versus Concrete Syntax

Abstract Syntax of arithmetic expressions

how to build new arithmetic expressions

Concrete Syntax of arithmetic
expressions

how to add information for unique parsing
(through parentheses or precedence rules)

2. Relational Expressions

The Law of Trichotomy

for integers $n1$ and $n2$, exactly one of the following is true:

$$n1 = n2$$

$$n1 < n2$$

$$n1 > n2$$

The 6 Relational Operators (Fortran and C notation)

“is equal to” .EQ. ==

“is less than” .LT. <

“is less than or equal to” .LE. <=

“is greater than” .GT. >

“is greater than or equal to” .GE. >=

“is not equal to” .NE. !=

Constructing Relational Operations from Relational and Logical Operators

given =, <, >

$n1 \leq n2$ iff $n1 < n2$ or $n1 = n2$

$n1 \geq n2$ iff $n1 > n2$ or $n1 = n2$

$n1 \neq n2$ iff $n1 < n2$ or $n1 > n2$

given =, \leq

$n1 < n2$ iff $n1 \leq n2$ and not $n1 = n2$

$n1 > n2$ iff not $n1 \leq n2$

$n1 \neq n2$ iff not $n1 = n2$

Defining \geq in Terms of $=$, \leq , and Logical Operations

$n1 \geq n2$

iff $n1 = n2$ or $n1 > n2$

iff $n1 = n2$ or not $n1 \leq n2$

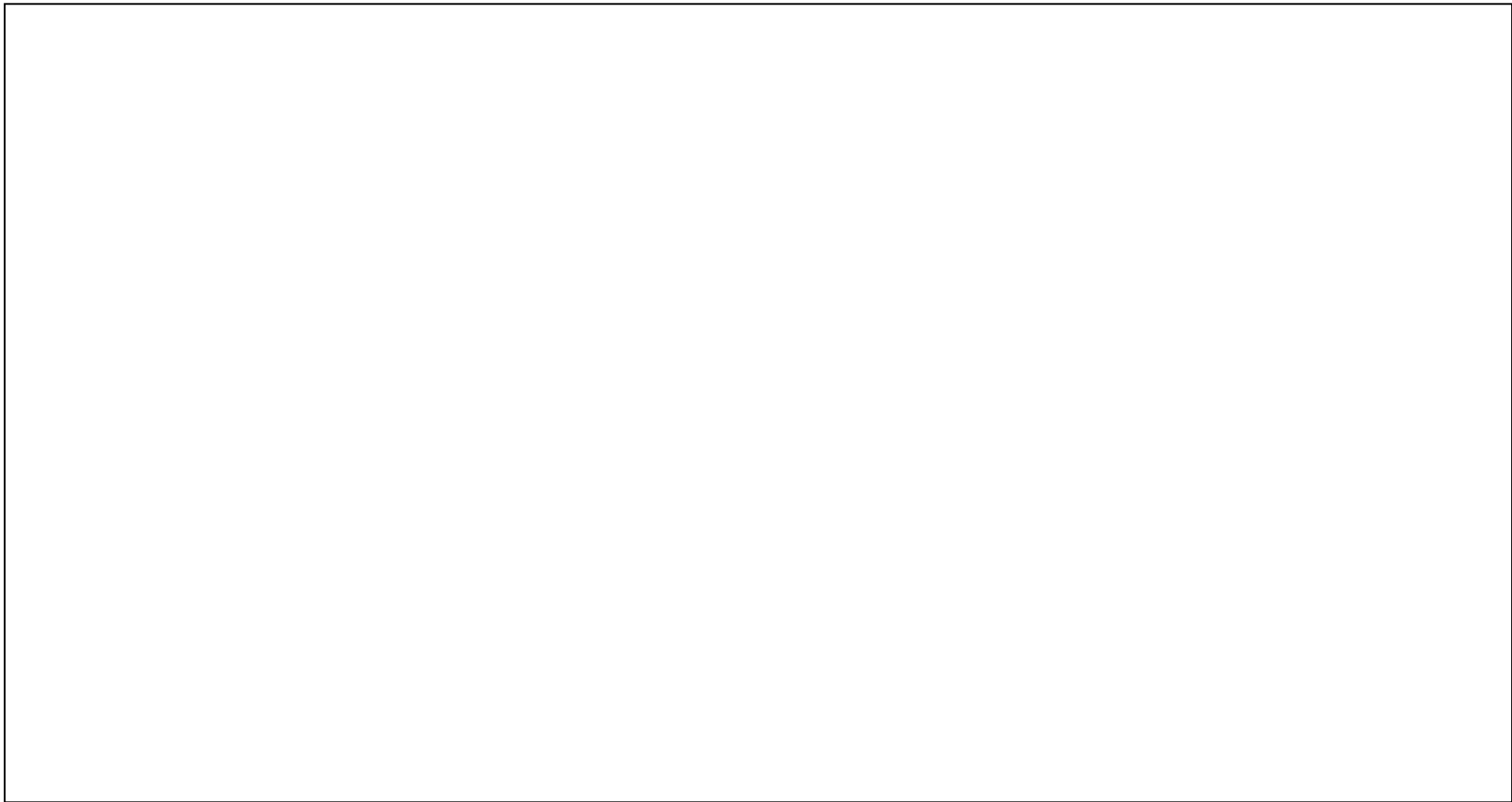
Functional Completeness

Let O be a set of operations and let P be a subset of O . P is functionally complete if every operation in O can be defined in terms of operations in P .

If we allow the logical operations,

$\{ =, <, > \}$ and $\{ =, \leq \}$ are functionally complete sets of relational operators.

3. Logical Expressions



Definitions of AND, OR, NOT

b1	b2		b1 AND b2	b1	b2		b1 OR b2
f	f		f	f	f		f
f	t		f	f	t		t
t	f		f	t	f		t
t	t		t	t	t		t
	b1		NOT b1				
	f		t				
	t		f				

All Binary Logical Operators

												1	1	1	1	1	1	
p	q		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5

f	f		f	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t
f	t		f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
t	f		f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
t	t		f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t

Recognizing Familiar Functions among These

0: false

1: $p \text{ AND } q$

3: p

7: $p \text{ OR } q$

12: q

15: true

Recognizing Some Familiar Logical Operators

0: false

1: p AND q

3: p

7: p OR q

12: q

15: true

6: p XOR q

exclusive or

8: p NOR q

14: p NAND q

13: p IMPLIES q

if p then q

9: p IFF q

p if and only if q

Some Functionally Complete Sets of Logical Operators

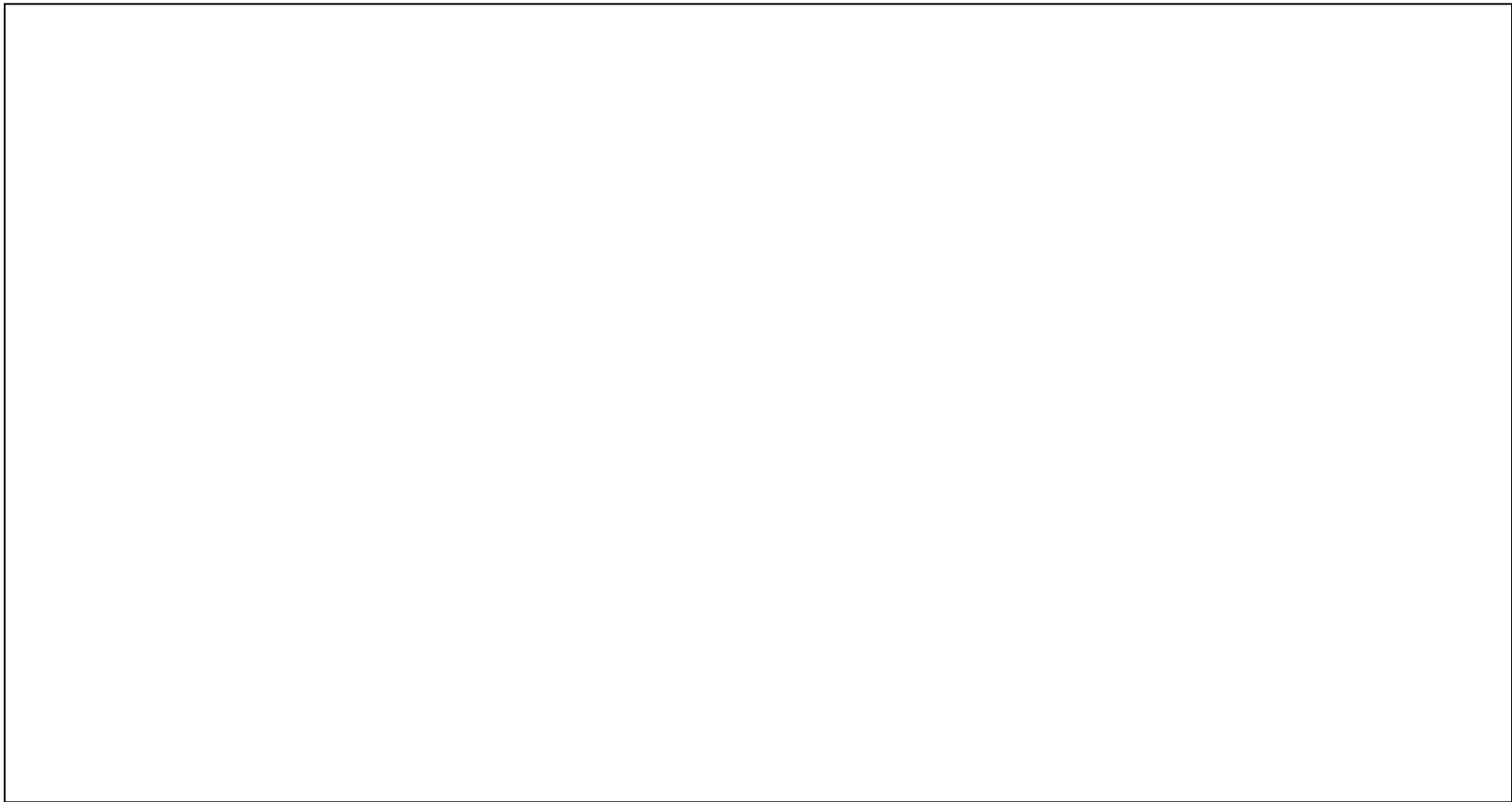
{ AND, OR, NOT }

{ AND, NOT }

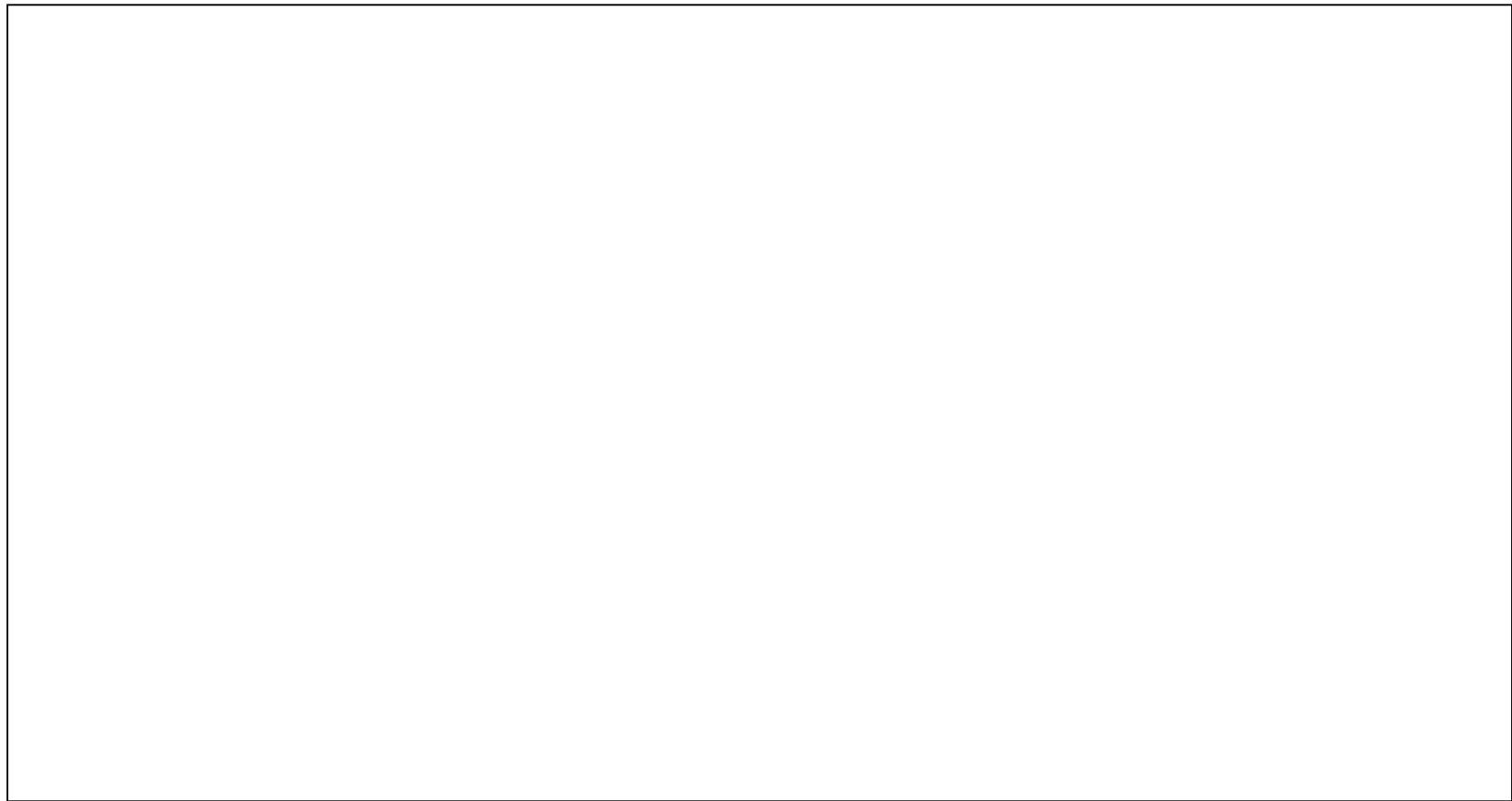
{ OR, NOT }

{ NAND }

4. Statements



5. Defintion of IMP



Quiz

