Language Specification

First Example: Propositional Logic

Professor William L. Harrison

September 28, 2016

An example

Q: Is the following a legal C program?

```
$ cat helloworld.c
#include <stdio.h>
int main() {
  printf("hello world\n")
}
```

An example

Q: Is the following a legal C program?

```
$ cat helloworld.c
#include <stdio.h>
int main() {
  printf("hello world\n")
}
```

Nope.

```
$ gcc helloworld.c
helloworld.c: In function ‘main’:
helloworld.c:3: error: parse error
                       before ’}’ token
```

Review

- C has some means of expressing and checking structure of an input file that a programmer claims is a program.

Review

- C has some means of expressing and checking structure of an input file that a programmer claims is a program.
- "Context-free Grammar" (CFG): structural rules that determine whether a sequence of symbols is, in fact, a sentence (program) in a language.

Review

- C has some means of expressing and checking structure of an input file that a programmer claims is a program.
- "Context-free Grammar" (CFG): structural rules that determine whether a sequence of symbols is, in fact, a sentence (program) in a language.
- CFGs are expressive enough to describe PL syntax and can be readily adapted to programming (parsing).

Review

- C has some means of expressing and checking structure of an input file that a programmer claims is a program.
- "Context-free Grammar" (CFG): structural rules that determine whether a sequence of symbols is, in fact, a sentence (program) in a language.
- CFGs are expressive enough to describe PL syntax and can be readily adapted to programming (parsing).
- Kernighan & Ritchie (2nd edition, App. 9.2, page 222):

$$\textit{expression-statement} : \ \textit{expression}_{opt} \ ;$$

Review

- C has some means of expressing and checking structure of an input file that a programmer claims is a program.
- "Context-free Grammar" (CFG): structural rules that determine whether a sequence of symbols is, in fact, a sentence (program) in a language.
- CFGs are expressive enough to describe PL syntax and can be readily adapted to programming (parsing).
- Kernighan & Ritchie (2nd edition, App. 9.2, page 222):

$$\textit{expression-statement} : \ \textit{expression}_{opt} \ ;$$

Says "an expression-statement is an expression (in this case the call to printf) followed by a semicolon."

The ideas and issues which we will consider are:

- What is a language?
- Syntax: How do we define *precisely* what are the well-formed sentences of a language?
- Semantics: Given a well-formed sentence, what does it mean?
- The separation between syntax and semantics.

Solving linear equation: $5x + 7 = 9$

Assume

$$5x + 4 \quad = \quad 9 \tag{i}$$

Solving linear equation: $5x + 7 = 9$

Assume

$$5x + 4 = 9 \quad \text{(i)}$$

Subtract 4 from each side of Equation (i):

$$5x = 5 \quad \text{(ii)}$$

Solving linear equation: $5x + 7 = 9$

Assume

$$5x + 4 = 9 \qquad \text{(i)}$$

Subtract 4 from each side of Equation (i):

$$5x = 5 \qquad \text{(ii)}$$

Divide both sides of Equation (ii) by 5:

$$x = 1 \qquad \text{(iii)}$$

Two rules

$$\begin{array}{lcll} kx + l = m & \Rightarrow & kx = m - l & \text{“subtract from both sides”} \\ kx = l & \Rightarrow & x = l/k & (k \neq 0)\,\text{“divide both sides”} \end{array}$$

Two rules

$$\begin{array}{lcll} kx + l = m & \Rightarrow & kx = m - l & \text{“subtract from both sides”} \\ kx = l & \Rightarrow & x = l/k & (k \neq 0)\,\text{“divide both sides”} \end{array}$$

Consider

$$5x + 4 = 9$$

Two rules

$$kx + l = m \quad \Rightarrow \quad kx = m - l \quad \text{“subtract from both sides”}$$
$$kx = l \quad \Rightarrow \quad x = l/k \quad (k \neq 0) \text{“divide both sides”}$$

Consider

$$5x + 4 = 9 \quad \Rightarrow \quad 5x = 9 - 4$$

Two rules

$$\begin{array}{lcll} kx + l = m & \Rightarrow & kx = m - l & \text{“subtract from both sides”} \\ kx = l & \Rightarrow & x = l/k & (k \neq 0)\,\text{“divide both sides”} \end{array}$$

Consider

$$5x + 4 = 9 \quad \Rightarrow \quad 5x = 9 - 4 \quad \Rightarrow \quad x = (9-4)/5$$

Two rules

$$\begin{array}{llll} kx + l = m & \Rightarrow & kx = m - l & \text{“subtract from both sides”} \\ kx = l & \Rightarrow & x = l/k & (k \neq 0)\ \text{“divide both sides”} \end{array}$$

Consider

$$5x + 4 = 9 \quad \Rightarrow \quad 5x = 9 - 4 \quad \Rightarrow \quad x = (9 - 4)/5$$

Question: Can we solve $3x + 5 + 6x = 0$ with these rules?

Two rules

$$\begin{array}{llll} kx + l = m & \Rightarrow & kx = m - l & \text{“subtract from both sides”} \\ kx = l & \Rightarrow & x = l/k & (k \neq 0)\ \text{“divide both sides”} \end{array}$$

Consider

$$5x + 4 = 9 \quad \Rightarrow \quad 5x = 9 - 4 \quad \Rightarrow \quad x = (9 - 4)/5$$

Question: Can we solve $3x + 5 + 6x = 0$ with these rules?
Answer: No.

Derivation System (High Level)

- Has some notion of a "sentence" or "formula"

$$-(E + -E) \qquad\qquad 2B \vee \neg 2B$$

Derivation System (High Level)

- Has some notion of a "sentence" or "formula"

$$-(E + -E) \qquad\qquad 2B \vee \neg 2B$$

- Rules for producing new formulae from existing ones

$$E \Rightarrow -E \qquad\qquad \frac{\varphi \quad \varphi \supset \gamma}{\gamma}$$

Derivation System (High Level)

- Has some notion of a "sentence" or "formula"

$$-(E + -E) \qquad\qquad 2B \vee \neg 2B$$

- Rules for producing new formulae from existing ones

$$E \Rightarrow -E \qquad\qquad \frac{\varphi \quad \varphi \supset \gamma}{\gamma}$$

- Notion of "proof" or "derivation". Sequence of sentences:

$$S_1, \ldots, S_n$$

where S_i result of applying a rule to (members of) $\{S_1, \ldots, S_{i-1}\}$

Derivation Systems Everywhere

- Defining syntax

$$E \Rightarrow i \qquad (\textit{where } i \in \{\ldots, -1, 0, 1, \ldots\})$$
$$E \Rightarrow -\ E$$

Derivation Systems Everywhere

- Defining syntax

$$E \Rightarrow i \qquad (\textit{where } i \in \{\ldots, -1, 0, 1, \ldots\})$$
$$E \Rightarrow - \; E$$

Derivation: E

Derivation Systems Everywhere

- Defining syntax

$$E \Rightarrow i \qquad (\textit{where } i \in \{\ldots, -1, 0, 1, \ldots\})$$
$$E \Rightarrow - \ E$$

Derivation: $E \Rightarrow -E$

Derivation Systems Everywhere

- Defining syntax

$$E \Rightarrow i \qquad (\text{where } i \in \{\ldots, -1, 0, 1, \ldots\})$$
$$E \Rightarrow - \ E$$

Derivation: $E \Rightarrow -E \Rightarrow - - E$

Derivation Systems Everywhere

- Defining syntax

$$E \Rightarrow i \qquad (\textit{where } i \in \{\ldots, -1, 0, 1, \ldots\})$$
$$E \Rightarrow -\ E$$

Derivation: $E \Rightarrow -E \Rightarrow - - E \Rightarrow - - 9$

Derivation Systems Everywhere

- Defining syntax

$E \Rightarrow i$ $\quad\quad$ $(\textit{where } i \in \{\ldots, -1, 0, 1, \ldots\})$

$E \Rightarrow - E$

Derivation: $E \Rightarrow -E \Rightarrow - - E \Rightarrow - - 9$

$\therefore - - 9$ is an E

Derivation Systems Everywhere

- Defining types

$$\frac{i \in \{\ldots, -1, 0, 1, \ldots\}}{i :: \mathit{Int}} \qquad \frac{e :: \mathit{Int}}{-e :: \mathit{Int}}$$

Derivation Systems Everywhere

- Defining types

$$\frac{i \in \{\ldots, -1, 0, 1, \ldots\}}{i :: \mathit{Int}} \qquad \frac{e :: \mathit{Int}}{-e :: \mathit{Int}}$$

Derivation

$$\cfrac{\cfrac{\cfrac{9 \in \{\ldots, -1, 0, 1, \ldots\}}{9 :: \mathit{Int}}}{-9 :: \mathit{Int}}}{- - 9 :: \mathit{Int}}$$

Derivation Systems Everywhere

- Defining types

$$\frac{i \in \{\ldots, -1, 0, 1, \ldots\}}{i :: \mathit{Int}} \qquad \frac{e :: \mathit{Int}}{-e :: \mathit{Int}}$$

Derivation

$$\cfrac{\cfrac{\cfrac{9 \in \{\ldots, -1, 0, 1, \ldots\}}{9 :: \mathit{Int}}}{-9 :: \mathit{Int}}}{- - 9 :: \mathit{Int}}$$

$$\therefore - - 9 :: \mathit{Int}$$

Derivation Systems Everywhere

- Defining meaning

  $length[x, y, z]$

Derivation Systems Everywhere

- Defining meaning

$$\begin{array}{l} \mathit{length}[x, y, z] \\ \qquad \Rightarrow 1 + \mathit{length}\,[y, z] \end{array}$$

Derivation Systems Everywhere

- Defining meaning

$$\begin{aligned} &\mathit{length}[x, y, z] \\ &\quad \Rightarrow 1 \;+\; \mathit{length}\,[y, z] \\ &\quad \Rightarrow 1 \;+\; 1 \;+\; \mathit{length}\,[z] \end{aligned}$$

Derivation Systems Everywhere

- Defining meaning

$$\begin{aligned} & \mathit{length}[x, y, z] \\ & \quad \Rightarrow 1 + \mathit{length}\,[y, z] \\ & \quad \Rightarrow 1 + 1 + \mathit{length}\,[z] \\ & \quad \Rightarrow 1 + 1 + 1 + \mathit{length}\,[] \end{aligned}$$

Derivation Systems Everywhere

- Defining meaning

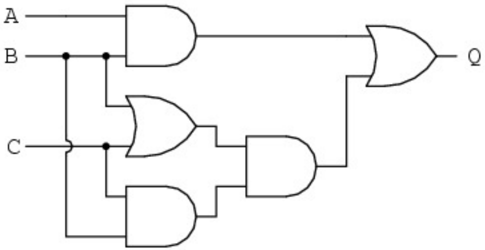
$$
\begin{aligned}
&\mathit{length}[x, y, z] \\
&\quad \Rightarrow 1 + \mathit{length}\,[y, z] \\
&\quad \Rightarrow 1 + 1 + \mathit{length}\,[z] \\
&\quad \Rightarrow 1 + 1 + 1 + \mathit{length}\,[] \\
&\quad \Rightarrow 1 + 1 + 1 + 0
\end{aligned}
$$

Derivation Systems Everywhere

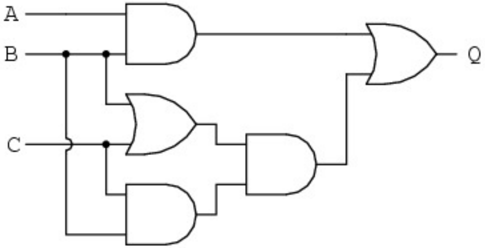
- Defining meaning

$$
\begin{aligned}
&\mathit{length}[x, y, z] \\
&\quad \Rightarrow 1 + \mathit{length}\,[y, z] \\
&\quad \Rightarrow 1 + 1 + \mathit{length}\,[z] \\
&\quad \Rightarrow 1 + 1 + 1 + \mathit{length}\,[] \\
&\quad \Rightarrow 1 + 1 + 1 + 0 \\
&\quad \Rightarrow 3
\end{aligned}
$$

Digital Logic



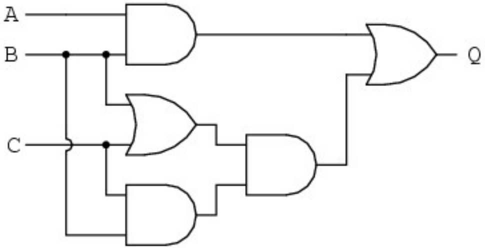
Digital Logic



An Equivalent Boolean Expression

```
(A and B) or ((B or C) and (C and B))
```

Truth Table



A	B	C	(A and B) or ((B or C) and (C and B))
T	T	T	?
T	T	F	?

Propositional Logic

- **Proposition**: a statement that is either true or false
  E.g., "*It is raining*", "*Socrates was Greek*", etc.

Propositional Logic

- **Proposition**: a statement that is either true or false
  E.g., "*It is raining*", "*Socrates was Greek*", etc.
- **Propositional Sentences**
  E.g., Let p and q stand for "it is raining" and "the street is wet", respectively, then $\mathrm{p} \supset \mathrm{q}$ is a propositional sentence. Connective $\supset$ stands for "implies".

Propositional Logic

- **Proposition**: a statement that is either true or false
  E.g., "*It is raining*", "*Socrates was Greek*", etc.
- **Propositional Sentences**
  E.g., Let p and q stand for "it is raining" and "the street is wet", respectively, then $\mathrm{p} \supset \mathrm{q}$ is a propositional sentence. Connective $\supset$ stands for "implies".
- **Propositional Logic**:
  A derivation system for logical consequence in Prop. Logic
  I.e., assuming $P_1, \ldots, P_n$, must Q hold?

Propositional Logic

- **Proposition**: a statement that is either true or false
  E.g., "*It is raining*", "*Socrates was Greek*", etc.
- **Propositional Sentences**
  E.g., Let p and q stand for "it is raining" and "the street is wet", respectively, then $\mathrm{p} \supset \mathrm{q}$ is a propositional sentence. Connective $\supset$ stands for "implies".
- **Propositional Logic**:
  A derivation system for logical consequence in Prop. Logic
  I.e., assuming $P_1, \ldots, P_n$, must Q hold?
- **Propositional Logic Semantics**: namely, truth tables.

Propositional Logic

- **Proposition**: a statement that is either true or false
  E.g., "*It is raining*", "*Socrates was Greek*", etc.
- **Propositional Sentences**
  E.g., Let p and q stand for "it is raining" and "the street is wet", respectively, then $\mathrm{p} \supset \mathrm{q}$ is a propositional sentence. Connective $\supset$ stands for "implies".
- **Propositional Logic**:
  A derivation system for logical consequence in Prop. Logic
  I.e., assuming $P_1, \ldots, P_n$, must Q hold?
- **Propositional Logic Semantics**: namely, truth tables.

Derivation systems will play a role in all of these.

The Language Syntax

The propositional calculus is the simplest form of mathematical logic.

Definition (Propositional Calculus)

A *propositional formula* has one of the following forms:

- a *propositional variable*; usually denoted by a roman letter, p, q, r, etc.
- a *negation*; e.g., $\neg\varphi$ where φ is a propositional formula.
- an *implication*; e.g., $(\varphi \supset \gamma)$ where φ and γ are propositional formulae.

The Language Syntax as Context Free Grammar

Before giving a precise definition, let's consider an example. Let *Var* be an infinite set of symbols. We will refer to typical elements of *Var* with lower case roman letters (e.g., p, q, r, etc.). Assume $\{(,),\neg,\wedge\} \cap Var = \emptyset$, then let alphabet A be the set $\{(,),\neg,\wedge\} \cup Var$. Here is a CFG:

$$Prop \rightarrow p \quad \text{for any } p \in Var \tag{1}$$

$$Prop \rightarrow (\neg Prop) \tag{2}$$

$$Prop \rightarrow (Prop \supset Prop) \tag{3}$$

This CFG defines a language, denoted $\mathcal{L}(Prop)$.

Deriving members of $\mathcal{L}(Prop)$

- How do we determine if a particular sequence[1] of symbols from A is in $\mathcal{L}(Prop)$?

[1] I use "string" and "sequence of symbols" interchangably.

Deriving members of $\mathcal{L}(Prop)$

- How do we determine if a particular sequence[1] of symbols from A is in $\mathcal{L}(Prop)$?
- We perform a *derivation* of the string.

[1] I use "string" and "sequence of symbols" interchangably.

Deriving members of $\mathcal{L}(Prop)$

- How do we determine if a particular sequence[1] of symbols from A is in $\mathcal{L}(Prop)$?
- We perform a *derivation* of the string.
- For instance, is the string $(\neg p) \in \mathcal{L}(Prop)$? Yes, and here's the derivation:

$$\begin{aligned} Prop &\rightarrow (\neg Prop) \\ &\rightarrow (\neg p) \end{aligned}$$

[1] I use "string" and "sequence of symbols" interchangably.

"Woofs"

Definition (Well-Formed Formulae of Propositional Logic)

The primitive symbols of L are:

$$\neg \quad \supset \quad (\;)$$

The propositional symbols of L are of the form A_i for any positive integer i. The symbols, $\neg$ and $\supset$, are called *connectives*. Any propositional symbol is a *well-formed formula* (wff) of L. Furthermore, if φ and γ are wffs, the so are:

$$(\neg\varphi)$$

and

$$(\varphi \supset \gamma)$$

Definitional Extensions

Definition (Disjunction, Conjunction and Equivalence)

Familiar connectives are defined by:

$$
\begin{array}{rcll}
(\varphi \vee \gamma) & \text{is} & \neg\varphi \supset \gamma & \text{(disjunction)} \\
(\varphi \wedge \gamma) & \text{is} & \neg(\neg\varphi \vee \neg\gamma) & \text{(conjunction)} \\
(\varphi \leftrightarrow \gamma) & \text{is} & (\varphi \supset \gamma) \wedge (\gamma \supset \varphi) & \text{(equivalence)}
\end{array}
$$

Definitional Extensions

Definition (Disjunction, Conjunction and Equivalence)

Familiar connectives are defined by:

$$
\begin{array}{rcll}
(\varphi \vee \gamma) & \text{is} & \neg\varphi \supset \gamma & \text{(disjunction)} \\
(\varphi \wedge \gamma) & \text{is} & \neg(\neg\varphi \vee \neg\gamma) & \text{(conjunction)} \\
(\varphi \leftrightarrow \gamma) & \text{is} & (\varphi \supset \gamma) \wedge (\gamma \supset \varphi) & \text{(equivalence)}
\end{array}
$$

I will typically drop the parentheses when possible.

Axiom System for Propositional Logic

$$\varphi \supset (\gamma \supset \varphi) \qquad \text{(Ax.1)}$$

$$(\varphi \supset (\gamma \supset \psi)) \supset ((\varphi \supset \gamma) \supset (\varphi \supset \psi)) \qquad \text{(Ax.2)}$$

$$((\neg\gamma \supset \neg\varphi) \supset ((\neg\gamma \supset \varphi) \supset \gamma)) \qquad \text{(Ax.3)}$$

There is only one inference rule in propositional logic, namely *Modus Ponens*.

$$\frac{\varphi \quad \varphi \supset \gamma}{\gamma} \ \text{(MP)}$$

Instances

An instance of an axiom is a substitution of a wff for φ, γ, ψ
Instances of Axiom 1 $(\varphi \supset (\gamma \supset \varphi))$ include

Instance	Substitution
$A \supset (B \supset A)$	$[\varphi \mapsto A, \gamma \mapsto B]$
$A \supset ((A \supset A) \supset A)$	$[\varphi \mapsto A, \gamma \mapsto (A \supset A)]$
$\vdots$	$\vdots$

Formal Proofs

Definition (Proof)

Let Φ be the sequence $\varphi_1, \ldots, \varphi_n$ of propositional wffs. Then, Φ is a *proof* of φ_n if, and only if, for each φ_i in Φ, φ_i is either:

- an instance of Ax.1, Ax.2, or Ax.3, or
- there are φ_j and φ_k such that $j < i$ and $k < i$ and φ_i follows from φ_j and φ_k by MP.

Example Proof

Say I want to prove that $A \supset A$.

Example Proof

Say I want to prove that $A \supset A$.

1. $(A \supset ((A \supset A) \supset A)) \supset ((A \supset (A \supset A)) \supset (A \supset A))$ Ax.2

Example Proof

Say I want to prove that $A \supset A$.

1. $(A \supset ((A \supset A) \supset A)) \supset ((A \supset (A \supset A)) \supset (A \supset A))$ Ax.2
2. $A \supset ((A \supset A) \supset A)$ Ax.1

Example Proof

Say I want to prove that $A \supset A$.

1. $(A \supset ((A \supset A) \supset A)) \supset ((A \supset (A \supset A)) \supset (A \supset A))$ Ax.2
2. $A \supset ((A \supset A) \supset A)$ Ax.1
3. $(A \supset (A \supset A)) \supset (A \supset A)$ MP2,1

Example Proof

Say I want to prove that $A \supset A$.

1. $(A \supset ((A \supset A) \supset A)) \supset ((A \supset (A \supset A)) \supset (A \supset A))$ Ax.2
2. $A \supset ((A \supset A) \supset A)$ Ax.1
3. $(A \supset (A \supset A)) \supset (A \supset A)$ MP2,1
4. $A \supset (A \supset A)$ Ax.1

Example Proof

Say I want to prove that $A \supset A$.

1. $(A \supset ((A \supset A) \supset A)) \supset ((A \supset (A \supset A)) \supset (A \supset A))$ Ax.2
2. $A \supset ((A \supset A) \supset A)$ Ax.1
3. $(A \supset (A \supset A)) \supset (A \supset A)$ MP2,1
4. $A \supset (A \supset A)$ Ax.1
5. $A \supset A$ MP3,4

Proof as Tree: $A \supset A$

$$\dfrac{\dfrac{}{A \supset (A \supset A)}\ (\text{Ax.1}) \qquad \dfrac{\dfrac{}{A \supset ((A \supset A) \supset A)}\ (\text{Ax.1}) \qquad \dfrac{}{\begin{array}{c}(A \supset ((A \supset A) \supset A)) \supset \\ ((A \supset (A \supset A)) \supset (A \supset A))\end{array}}\ (\text{Ax.2})}{(A \supset (A \supset A)) \supset (A \supset A)}\ (\text{MP})}{A \supset A}\ (\text{MP})$$

Semantics: What does it mean?

Semantics (a.k.a., model theory) is another way of establishing the validity of a wff. The semantics of propositional logic consists of the well-known "truth tables".

A	B	$\neg B$	(A $\supset$ B)	(A $\wedge$ B)	$\neg(A \supset \neg B)$
T	T	F	T	T	
T	F	T	F	F	
F	T	F	T	F	
F	F	T	T	F	

Another Truth Table

A	B	C	$(A \wedge B) \vee ((B \vee C) \wedge (C \wedge B))$
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	