1 Propositional Logic

1.1 Semantics

Definition 1.1 (Interpretation)**.** An *interpretation* I is a function that maps each propositional variable to either true or false.

$$I : \{p_1 \mapsto \text{true} \vee \text{false}, p_2 \mapsto \text{true} \vee \text{false}, \ldots, p_n \mapsto \text{true} \vee \text{false}\}$$

1.2 Syntax

Rule	$I \vDash \cdot$	$I \nvDash \cdot$
Neg	$\dfrac{I \vDash \neg F}{I \nvDash F}$	$\dfrac{I \nvDash \neg F}{I \vDash F}$
Conj	$\dfrac{I \vDash F_1 \wedge F_2}{I \vDash F_1 \qquad I \vDash F_2}$	$\dfrac{I \nvDash F_1 \wedge F_2}{I \nvDash F_1 \mid I \nvDash F_2}$
Disj	$\dfrac{I \vDash F_1 \vee F_2}{I \vDash F_1 \mid I \vDash F_2}$	$\dfrac{I \nvDash F_1 \vee F_2}{I \nvDash F_1 \qquad I \nvDash F_2}$
Impl	$\dfrac{I \vDash F_1 \to F_2}{I \nvDash F_1 \mid I \vDash F_2}$	$\dfrac{I \nvDash F_1 \to F_2}{I \vDash F_1 \qquad I \nvDash F_2}$
Contr	$\dfrac{I \vDash F \qquad I \nvDash F}{I \vDash \bot}$	never (by definition)

The notation $A \mid B$ means "either A or B (or both)".

Example 1.1. We want to show that the following formula is valid.

$$F \triangleq (P \wedge Q) \to (P \vee \neg Q)$$

We proceed via proof by contradiction, namely $I \nvDash F$

$$\dfrac{\dfrac{I \nvDash (P \wedge Q) \to (P \vee \neg Q)}{\dfrac{I \vDash P \wedge Q}{I \vDash P \qquad I \vDash Q}\ \textsc{Conj} \qquad \dfrac{I \nvDash P \vee \neg Q}{I \nvDash P \qquad I \nvDash \neg Q}\ \textsc{Disj}}\ \textsc{Impl}$$

Since our conclusion has both $I \vDash P$ and $I \nvDash P$ we can note this is a contradiction, and hence F is valid.

2 First-Order Logic

2.1 Semantics

Definition 2.1 (Scope)**.** The *scope* of a quantifier ($\forall$ or $\exists$) is the formula that immediately follows it. For example, in $\forall x.P(x) \to Q(x)$ the scope of $\forall x$ is $P(x) \to Q(x)$. Generally we say for a formula $\forall x.F(x)$ or $\exists x.F(x)$ that the scope of the quantifier is $F(x)$.

- The variable x is *bound* in $F(x)$.
- Any other occurence of x outside the scope of the quantifier is *free*.
  - A *closed* formula has no free variables.
  - An *open* formula has at least one free variable.

This is mirrors lambda calculus in which given a lambda abstraction $\lambda x.M$, the variable x is bound in M and any other occurence of x outside the scope of the lambda is free.

Definition 2.2 (Domain)**.** The *domain* (or universe) U_I of an interpretation I is a non-empty set of objects (e.g. people, numbers, etc.).

Definition 2.3 (Domain Cardinality)**.** The *cardinality* of a domain $\mathcal{U}_\mathcal{I}$ is the number of elements in $\mathcal{U}_\mathcal{I}$, denoted $|\mathcal{U}_\mathcal{I}|$. Domains can be fininte or infinite, and all are non-empty.

Definition 2.4 (Assignment)**.** An *assigment* $\alpha_\mathcal{I}$ or σ of an interpretation $\mathcal{I}$ can be classified into 3 types:

- (*variable symbols*) $x \mapsto x_\mathcal{I} \in \mathcal{U}_\mathcal{I}$
- (*n-ary function symbols*) $f(t_1, \ldots, t_n) \mapsto f_\mathcal{I} : \mathcal{U}_\mathcal{I}^n \to \mathcal{U}_\mathcal{I}$
- (*n-ary predicate symbols*) $p(t_1, \ldots, t_n) \mapsto p_\mathcal{I} : \mathcal{U}_\mathcal{I}^n \to \{\texttt{true}, \texttt{false}\}$

Constants are treated as 0-ary function symbols and propositional variables as 0-ary predicate symbols.
The lectures denote an assigment of a symbol (variable/function/predicate) x as

$$\sigma[x \mapsto v]$$

which expresses the assignment that agrees with σ to all variables except that x is mapped to v.

Example 2.1 (Assignment)**.** Let $\mathcal{U}_\mathcal{I} = \{a, b, c\}$ be a domain of people. An assignment σ could be:

- $x \mapsto a$
- $y \mapsto b$
- $f(x) \mapsto$ "the mother of x"
- $p(x, y) \mapsto$ "x is a parent of y"

Then $\sigma[x \mapsto b]$ would be the same as σ except that $x \mapsto b$.

Definition 2.5 (Interpretation)**.** An *interpretation* is a pair $\mathcal{I} : (\mathcal{U}_\mathcal{I}, \sigma)$ where $\mathcal{U}_\mathcal{I}$ is a domain and σ is an assignment.

Definition 2.6 (Structure)**.** A *structure* is a pair $\mathcal{S} : (\mathcal{U}_\mathcal{I}, \mathcal{I})$ where $\mathcal{U}_\mathcal{I}$ is a domain and $\mathcal{I}$ is an interpretation.

Definition 2.7 (Subformulae)**.** The *subformulae* of a formula F are the formulae that can be obtained by starting from F and repeatedly applying the following rules:

Algorithm 1 Subformulae extraction

```
Input: A formula F
Output: The set of subformulae of F
fn SUBFORMULAE(S)
    match S with
        case ¬G:
            S ← S ∪ Subformulae(G)
        case G₁ (∧ | ∨ |→) G₂:
            S ← S ∪ Subformulae(G₁) ∪ Subformulae(G₂)
        case ∀x.Gₓ | ∃x.Gₓ:
            S ← S ∪ Subformulae(Gₓ)
        case atomic formula:
            S ← S
    end
end fn
S ← {F} ∪ Subformulae(F)
```

Rule	$\mathcal{I} \models \cdot$	$\mathcal{I} \not\models \cdot$
All	$\dfrac{\mathcal{S}, \sigma \models \forall x.F(x)}{\forall c \in \mathcal{U} : \mathcal{S}, \sigma[x \mapsto c] \models F(c)}$	$\dfrac{\mathcal{S}, \sigma \not\models \forall x.F(x)}{\exists c \in \mathcal{U} : \mathcal{S}, \sigma[x \mapsto c] \not\models F(c)}$
Ex	$\dfrac{\mathcal{S}, \sigma \models \exists x.F(x)}{\exists c \in \mathcal{U} : \mathcal{S}, \sigma[x \mapsto c] \models F(c)}$	$\dfrac{\mathcal{S}, \sigma \not\models \exists x.F(x)}{\forall c \in \mathcal{U} : \mathcal{S}, \sigma[x \mapsto c] \not\models F(c)}$
Contr	$\dfrac{\mathcal{S}, \sigma \models p(s_1, \ldots, s_n) \qquad \mathcal{S}, \sigma \not\models p(s_1, \ldots, s_n)}{\mathcal{S}, \sigma \models \bot}$	never (by definition)

3 First-Order Theory

Definition 3.1 (Signature Σ)**.** A *signature* Σ is a set of constant, function and predicate symbols.

Definition 3.2 (Axioms A)**.** A set of *axioms* A is a set of closed formulae in which only symbols from the signature Σ are used. The axioms are what provide meaning to the symbols in the signature.

Definition 3.3 (First-Order Theory)**.** A *first-order theory* is a pair $T : (\Sigma, A)$ where Σ is a signature and A is a set of axioms.

Definition 3.4 (Σ-Formula)**.** A Σ-*formula* is a formula in which only symbols from the signature Σ are used as well as variables, logical connectives, and quantifiers.

Definition 3.5 (Validity in a Theory)**.** A Σ-*formula* F is *valid in a theory* $T : (\Sigma, A)$ if for every structure $S : (U_I, I)$ such that $S \vDash A$ it holds that $S \vDash F$. We denote this as $T \vDash F$. In other words F is valid in T if its true in all structures (i.e. interpretations) that satisfy the axioms of T. We can express this as a propositional check as follows for a given theory T_K

$$\textsc{ValidInTheory}(F, T_K) \triangleq \lambda S \mapsto (\lambda A_K \mapsto S \vDash A_K \to S \vDash F) \to T \vDash F : \texttt{Prop} \tag{1}$$

Note: This is not really formally rigorous but just pedagogically I think a nice way of expressing the idea of what constitutes validity in a theory.

4 Congruence

Definition 4.1 (Congruence Relation)**.** In the most general sense a *congruence relation* is a relation R on a given algebraic structure - such as in our case the domain of an interpretation U_I - if for a given n-ary operation μ (e.g predicate or function symbol) the following condition holds:

$$a_1 \mathrel{R} a_2 \wedge \ldots \wedge a_n \mathrel{R} b_n \implies \mu(a_1, \ldots, a_n) \mathrel{R} \mu(b_1, \ldots, b_n)$$

Within the context of first-order logic we equivalently say given a formula F_1 and F_2 then replacing a subformula F_1 with F_2 within a larger formula F preserves truth.

$$I \vDash F_1 \leftrightarrow F_2 \implies I \vDash F[F_1 \mapsto F_2] \Leftrightarrow \frac{I \vDash F_1 \leftrightarrow F_2}{I \vDash F[F_1] \leftrightarrow F[F_2]}$$

More generally we can say given n pairs of formulae $F_1, F_1', \ldots, F_n, F_n'$ then replacing each F_i with F_i' within a larger formula F preserves truth.

$$\frac{I \vDash F_1 \leftrightarrow F_1' \qquad \ldots \qquad I \vDash F_n \leftrightarrow F_n'}{I \vDash F[F_1 \mapsto F_1', \ldots, F_n \mapsto F_n']}$$

Definition 4.2 (Predicate Congruence)**.** Predicate congruence also known as *Leibniz's Law* or more straightforwardly *Substitution for formulas* states that if two terms t_1 and t_2 are equal then any predicate p that holds for t_1 also holds for t_2. Formally:

$$\textsc{PredCong}(p) \triangleq t = t' \to (p(\ldots, t, \ldots) \leftrightarrow p(\ldots, t', \ldots)) : \texttt{Prop} \tag{2}$$

Alternatively we can also use the same notation as for general congruence to express predicate congruence more generally

$$\frac{I \vDash t_1 = t_1' \wedge \ldots \wedge t_n = t_n'}{I \vDash p(t_1, \ldots, t_n) \leftrightarrow p(t_1', \ldots, t_n')}$$

Definition 4.3 (Function Congruence)**.** Function congruence is derived as a special case of predicate congruence. We start by defining a formula

$$\textsc{FunEquals}(t) \triangleq (f(\ldots, t, \ldots) = f(\ldots, t', \ldots)) : \texttt{Prop} \tag{3}$$

We substitute p in equation **??** with $\textsc{FunEquals}(t')$ to get

$$t = t' \to (f(\ldots, t, \ldots) = f(\ldots, t, \ldots) \to f(\ldots, t, \ldots) = f(\ldots, t', \ldots))$$

We simplify the first implication by noting that $f(\ldots, t, \ldots) = f(\ldots, t, \ldots)$ is always true and hence we can remove it to get the definition of function congruence.

$$\textsc{FunCong}(f) \triangleq t = t' \to (f(\ldots, t, \ldots) = f(\ldots, t', \ldots)) : \texttt{Prop} \tag{4}$$

Again we can also use the same notation as for general congruence to express function congruence as an inference rule

$$\frac{I \vDash t_1 = t_1' \wedge \ldots \wedge t_n = t_n'}{I \vDash f(t_1, \ldots, t_n) = f(t_1', \ldots, t_n')}$$

Definition 4.4 (Left / Right Congruence)**.** Left or Right congruence represents a special case of function congruence where the function is a binary operator (e.g. $+$, $\times$, etc.) and we only replace one of the two arguments. In the most general sense we define some arbitrary binary operator $\oplus$ and then define left and right congruence as follows:

$$\textsc{LeftCong}(\oplus) \triangleq t = t' \to (t \oplus s = t' \oplus s) : \texttt{Prop} \tag{5}$$

$$\textsc{RightCong}(\oplus) \triangleq s = s' \to (t \oplus s = t \oplus s') : \texttt{Prop} \tag{6}$$

Example 4.1 (Small-step operational semantics)**.** *Operational semantics* is just a fancy term for describing how a program executes, in other words the operational semantics of a language describe an idealized interpreter for that language. There are two categories of operational semantics - *big-step* and *small-step*. Big-step semantics are judgements in the form

$$(S, s) \to s' \equiv \text{Starting in state } s \text{ and executing } S \text{ may terminate in state } s'$$

Small-step semantics are judgements in the form

$$(S, s) \to (S', s')$$

the conceptual difference is that big-step semantics describe the overall effect of executing a program whereas small-step semantics describe the individual steps taken to execute a program thus giving us a more fine-grained view we can talk about the execution of intermediate states.
A place where Small-step semantics and congruence come together is particularly in the context of *Call by Value* (CBV) evaluation strategy. CBV semantics state that

1. *Right-congruence* : Reduce function expressions until they are values i.e. callable lambdas.

$$\frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2}\ \textsc{CBV-Fun} \tag{7}$$

2. *Left-congruence* : Function arguments must be fully evaluated before being passed to callable lambdas.

$$\frac{e_1 \to e_1'}{v\ e_1 \to v\ e_1'}\ \textsc{CBV-Arg} \tag{8}$$

3. Function application is only defined when the function is a callable lambda and all arguments are values.

$$\frac{v : \texttt{value}}{(\lambda x.e)\ v \to e[x \mapsto v]}\ \textsc{CBV-Subs} \tag{9}$$

For example consider the following expression

$$(\lambda x.x + 1)\ (1 + 2)$$

We can evaluate this expression step-by-step using the CBV small-step semantics defined above.

$$\frac{1 + 2 \to 3}{(\lambda x.x + 1)\ (1 + 2) \to (\lambda x.x + 1)\ 3}\ \textsc{CBV-Arg} \qquad \frac{3 : \texttt{value}}{(\lambda x.x + 1)\ 3 \to 3 + 1}\ \textsc{CBV-Subs} \qquad \frac{3 + 1 \to 4}{(\lambda x.x + 1)\ 3 \to 4}\ \textsc{CBV-Fun}$$

Note that we could not have applied the substitution rule (equation **??**) first because the argument $1 + 2$ is not a value. We had to first reduce it to a value 3 using the right-congruence rule (equation **??**) before we could apply the substitution rule.

5 Equality

5.1 Syntax

Definition 5.1 (Equality Signature)**.** The theory of equality T_{E} has the *signature* Σ_{E} defined as follows

$$\Sigma_{\mathsf{E}} : \{=, s\}$$

Where

- $=$ is a binary predicate symbol
- s represents any constant, function or predicate symbol

5.2 Axioms of Equality

The axioms of equality are as follows:

$$\begin{aligned}
&\textsc{Refl}(x) \triangleq (x = x) && (\textit{reflexivity})\\
&\textsc{Symm}(x, y) \triangleq (x = y) \to (y = x) && (\textit{symmetry})\\
&\textsc{Trans}(x, y, z) \triangleq (x = y \wedge y = z) \to (x = z) && (\textit{transitivity})\\
&\textsc{PredCong}(p) && (\textit{predicate congruence})\\
&\textsc{FunCong}(f) && (\textit{function congruence})
\end{aligned}$$

6 Peano Arithmetic

6.1 Syntax

Definition 6.1 (PA Signature)**.** The theory of PA has the *signature* Σ_{PA} defined as follows

$$\Sigma_{\text{PA}} : \{0, 1, +, \times, =\}$$

Where

- 0 and 1 are constant symbols (0-ary function symbols)
- $+$ and $\times$ are binary function symbols
- $=$ is a binary predicate symbol

6.2 Axioms of PA

The axioms of PA are as follows:

$$\begin{aligned}
&\forall x.\neg(0 = x + 1) : 0 = \textsc{min}(\mathbb{N}) && (\textit{zero})\\
&\forall x.x + 0 = x && (\textit{plus zero})\\
&\forall x.\forall y.x + (y + 1) = (x + y) + 1 && (\textit{plus successor})\\
&\forall x.x \times 0 = 0 && (\textit{times zero})\\
&\forall x.\forall y.x \times (y + 1) = (x \times y) + x && (\textit{times successor})\\
&\forall x.\forall y.(x = y) \to (x + 1 = y + 1) && (\textit{equality})\\
&(F(0) \wedge (\forall x.F(x) \to F(x + 1))) \to (\forall x.F(x)) && (\textit{induction})
\end{aligned}$$

7 Presburger Arithmetic

7.1 Syntax

Definition 7.1 (Presburger Signature)**.** The theory of Presburger arithmetic has the *signature* Σ_{Pres} defined as follows

$$\Sigma_{\text{Pres}} : \{0, 1, +, =\}$$

Where

- 0 and 1 are constant symbols (0-ary function symbols)
- $+$ is a binary function symbol
- $=$ is a binary predicate symbol

7.2 Axioms of Presburger Arithmetic

The axioms of Presburger arithmetic are as follows:

$$\forall x.\neg(0 = x + 1) : 0 = \text{MIN}(\mathbb{N}) \qquad (\textit{zero})$$
$$\forall x.x + 0 = x \qquad (\textit{plus zero})$$
$$\forall x.\forall y.x + (y + 1) = (x + y) + 1 \qquad (\textit{plus successor})$$
$$\forall x.\forall y.(x = y) \rightarrow (x + 1 = y + 1) \qquad (\textit{equality})$$
$$(F(0) \wedge (\forall x.F(x) \rightarrow F(x + 1))) \rightarrow (\forall x.F(x)) \qquad (\textit{induction})$$

8 Theory of Arrays

8.1 Syntax

Definition 8.1 (Array Signature)**.** The theory of arrays has the *signature* Σ_{Arrays} defined as follows

$$\Sigma_A : \{\_\,[\,\_\,], \langle\_ \triangleleft \_\rangle, =\}$$

Where

- $\_\,[\,\_\,]$ is a binary function symbol equivalent to the function $\texttt{read}(a, i)$ which returns the element at index i of array a.
- $\langle\_ \triangleleft \_\rangle$ is a ternary function symbol equivalent to the function $\texttt{write}(a, i, v)$ which returns a new array that is the same as array a except that index i now contains value v.
- $=$ is a binary predicate symbol.

8.2 Axioms of Arrays

The axioms of the theory of arrays are as follows:

$$\forall a.\forall i.\forall v.(a\langle i \triangleleft v\rangle)[i] = v \qquad (\textit{read-over-write})$$
$$\forall a.\forall i.\forall j.\forall v.(i \neq j) \rightarrow (a\langle i \triangleleft v\rangle)[j] = a[j] \qquad (\textit{read-over-write (different index)})$$
$$\forall a.\forall b.(\forall i.a[i] = b[i]) \rightarrow (a = b) \qquad (\textit{extensionality})$$