Formalisms
Every Computer Scientist Should Know

1 Lecture 1: How to write an informal proof

1.1 Example 1: Sum of two bounded functions

Definition 1.1. A real number $b \in \mathbb{R}$ is a *bound* of a function $f \colon \mathbb{R} \to \mathbb{R}$ from $\mathbb{R}$ to $\mathbb{R}$, if for all x in $\mathbb{R}$, we have $f(x) \leq b$.

Definition 1.2. Given two functions $f \colon \mathbb{R} \to \mathbb{R}$ and $g \colon \mathbb{R} \to \mathbb{R}$ from $\mathbb{R}$ to $\mathbb{R}$, their *sum* is the function $f + g$ such that for all x in $\mathbb{R}$, we have $(f + g)(x) = f(x) + g(x)$.

Theorem 1.1. *For all functions $f \colon \mathbb{R} \to \mathbb{R}$ and $g \colon \mathbb{R} \to \mathbb{R}$, if f and g are bounded, then $f + g$ is bounded.*

Proof. Consider arbitrary functions $\hat{f}$ and $\hat{g}$ from $\mathbb{R}$ to $\mathbb{R}$. We assume that $\hat{f}$ and $\hat{g}$ are bounded, and want to show that $\hat{f} + \hat{g}$ is bounded.

By the definition of boundedness, we show that there exists a $b \in \mathbb{R}$ such that for all $x \in \mathbb{R}$, $(\hat{f} + \hat{g})(x) \leq b$.

For this, let $\hat{b_1}$ be such that for all $x \in \mathbb{R}$, $\hat{f}(x) \leq \hat{b_1}$. Similarly, let $\hat{b_2}$ be a bound for $\hat{g}$. We show that $\hat{b_1} + \hat{b_2}$ is a bound for $\hat{f} + \hat{g}$, i.e. for all $x \in \mathbb{R}$, we have $(\hat{f} + \hat{g})(x) \leq \hat{b_1} + \hat{b_2}$.

Consider an arbitrary $\hat{x} \in \mathbb{R}$. We show that $(\hat{f} + \hat{g})(\hat{x}) \leq \hat{b_1} + \hat{b_2}$. By the definition of $+$ for functions, we have to show $\hat{f}(\hat{x}) + \hat{g}(\hat{x}) \leq \hat{b_1} + \hat{b_2}$.

For all $x_1, x_2, y_1, y_2 \in \mathbb{R}$, if $x_1 \leq y_1$ and $x_2 \leq y_2$, then $x_1 + x_2 \leq y_1 + y_2$. This can be broken down even further:

(i) for all $x, y, z \in \mathbb{R}$, if $x \leq y$ and $y \leq z$, then $x \leq z$

(ii) for all $x, y \in \mathbb{R}$, $x + y = y + x$

(iii) for all $x, y, z \in \mathbb{R}$, if $x \leq y$, then $x + z \leq y + z$

□

1.2 Example 2: Schröder-Bernstein Theorem

Definition 1.3. Two sets A and B are *equipollent* ("have the same size"), if there is a bijection from A to B.

Definition 1.4. A function f from A to B is: (i) *one-to-one* if for all x and y in A, if $x \neq y$, then $f(x) \neq f(y)$; (ii) *onto* if for all z in B, there exists x in A such that $f(x) = z$; (iii) *bijective* if f is one-to-one and onto.

Theorem 1.2 (Schröder-Bernstein theorem)**.** *If there is a one-to-one function on a set A to a subset of a set B and there is also a one-to-one function on B to a subset of A, then A and B are equipollent.*

Exercise 1.1. Correct the following incorrect proof of Theorem 1.2 *in the presented style*:

> Suppose that f is a one-to-one map of A into B and g is one-to-one on B to A. It may be supposed that A and B are disjoint. The proof of the theorem is accomplished by decomposing A and B into classes which are most easily described in terms of parthenogenesis. A point x (of either A or B) is an ancestor of a point y iff y can be obtained from x by successive application of f and g (or g and f). Now decompose A into three sets: let A_e consist of all points of A which have an even number of ancestors, let A_o consist of points which have an odd number of ancestors, and let A_∞ consist of points with infinitely many ancestors. Decompose B similarly and observe: f maps A_e onto B_o and A_∞ onto B_∞, and g^{-1} maps A_o onto B_e. Hence the function which agrees with f on $A_e \cup A_\infty$, and agrees with g^{-1} on A_o is a one-to-one map of A onto B.

2 Lecture 2

2.1 Style Guide

We provide an informal style guide for writing mathematical proofs.

Goal	Knowledge	Outermost symbol
Show for all x, $G(x)$. Consider arbitrary $\hat{x}$[1]. Show $G(\hat{x})$	We know for all x, $K(x)$ In particular we know $K(\hat{t})$[2]	$\forall$
Show: exists x s.t. $G(x)$. We show $G(\hat{t})$[2]	We know exists x s.t. $K(x)$ Let $\hat{x}$[1] be s.t. $K(\hat{x})$	$\exists$
Show if G_1 then G_2 Assume G_1 Show G_2	We know if K_1 then K_2 To show K_2 it suffices to show K_1 Know K_1, Also know K_2	$\Rightarrow$
Show G_1 iff G_2 1. Show if G_1 then G_2 2. Show if G_2 then G_1	We know K_1 iff K_2 In particular we know if K_1 then K_2 and if K_2 then K_1	$\Longleftrightarrow$
Show G_1 and G_2 1. Show G_1, **and** 2. Show G_2	Know K_1 and K_2 Also Know K_1 Also Know K_2	$\wedge$

Show G_1 or G_2
Either, assume $\neg G_1$, show G_2, **or**
Assume $\neg G_2$, show G_1
	We know K_1 or K_2. Show G.
	case 1: Assume K_1, Show G
	case 2: Assume K_2, Show G
	Case split ↑
		$\vee$

Move Negation Inside, as far as possible		$\neg$

[1] where $\hat{x}$ is a new constant. [2] where $\hat{t}$ is any constant expression

2.2 Lattices and Fixpoints

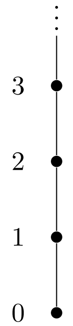
We define relations and their properties.

Definition 2.1 (Relation)**.** A binary *relation* R on a set A is a subset $R \subseteq A \times A$. The relations R is:

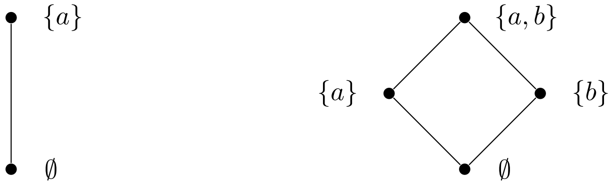
- *reflexive* if for all x in A, we have $R(x, x)$.
- *anti-symmetric* if for all x and y in A, if $R(x, y)$ and $R(y, x)$ then $x = y$.
- *transitive* if for all x, y and z in A, if $R(x, y)$ and $R(y, z)$ then $R(x, z)$.
- a *partial order* if R is reflexive, anti-symmetric and transitive.

Definition 2.2 (poset)**.** A *poset* $(A, \sqsubseteq)$ is a set A and a partial order $\sqsubseteq$ on A.

Example 2.1. Let $\mathbb{N}$ be the natural numbers. The pair $(\mathbb{N}, \leq)$ is a poset.



Example 2.2. For any set B let $\mathcal{P}(B)$ be the powerset. The pair $(\mathcal{P}(B), \subseteq)$ is a poset. E.g., for $B \coloneqq \{a\}$ (left) and for $B \coloneqq \{a, b\}$ (right).



Definition 2.3 (Upper Bound)**.** Let $(A, \sqsubseteq)$ be a poset. Then (i) x in A is an *upper bound* of a subset B of A if for all y in B, it holds that $y \sqsubseteq x$; (ii) x is the *least upper bound* (lub) of B if x is an upper bound of B and for all upper bounds y of B, we have $x \sqsubseteq y$.

Definition 2.4 (Lower Bound)**.** Let $(A, \sqsubseteq)$ be a poset. Then (i) x in A is a *lower bound* of a subset B of A if for all y in B, it holds that $x \sqsubseteq y$; (ii) x is the *greatest lower bound* (glb) of B if x is a lower bound of B and for all lower bounds y of B, we have $y \sqsubseteq x$.

Example 2.3. Consider the poset $(\mathbb{N}, \leq)$. Then for any $B \subseteq \mathbb{N}$, if B is finite, $\text{lub}(B)$ is well-defined and equal to $\max(B)$. If B is infinite, then $\text{lub}(B)$ does not exist.

Example 2.4. Consider the poset $(\mathbb{N} \cup \{\infty\}, \leq)$ where for all x in $\mathbb{N}$, it holds that $x \leq \infty$. Then for all $B \subseteq \mathbb{N}$, $\text{lub}(B)$ is well-defined.

Example 2.5. Let A be any set and consider the poset $(\mathcal{P}(A), \subseteq)$. For any subset B of $\mathcal{P}(A)$, it holds that $\text{lub}(B) = \bigcup B$ and $\text{glb}(B) = \bigcap B$.

Lemma 2.1. *Let $(A, \sqsubseteq)$ be a poset. For all x, y in A, and any subset B of A, if x and y are lubs (glbs) of B, then $x = y$.*

Proof. Consider arbitrary $\hat{x}, \hat{y} \in A$ and $\hat{B} \subseteq A$. Assume $\hat{x}$ is a lub of $\hat{B}$ and $\hat{y}$ is a lub of $\hat{B}$. We know:

(i) for all $z \in B$, $z \sqsubseteq \hat{x}$,

(ii) for all $z \in B$, $z \sqsubseteq \hat{y}$,

(iii) for all $z \in A$, if z is an upper bound of B then $\hat{x} \sqsubseteq z$ and

(iv) for all $z \in A$, if z is an upper bound of B then $\hat{y} \sqsubseteq z$.

From (ii) we know that $\hat{y}$ is an upper bound, and therefore, by (iii), $\hat{x} \sqsubseteq \hat{y}$. Symmetrically, by (i) and (iv) we get $\hat{y} \sqsubseteq \hat{x}$. By antisymmetry, $x = y$. □

Given that the *least upper bound* and the *greatest lower bound* are unique (if they exist), for any set A and $B \subseteq A$, we write $\bigsqcup B$ to denote $\mathrm{lub}(B)$ and $\bigsqcap B$ to denote $\mathrm{glb}(B)$.

Definition 2.5 (Lattice)**.** A poset $(A, \sqsubseteq)$ is a *lattice* if for all x, y in A, the lub $\bigsqcup\{x, y\}$ ($x \sqcup y$, join) and the glb $\bigsqcap\{x, y\}$ ($x \sqcap y$, meet) exist.

Definition 2.6 (Complete Lattice)**.** A poset $(A, \sqsubseteq)$ is a *complete lattice* if for all $B \subseteq A$, the lub $\bigsqcup B$ and glb $\bigsqcap B$ exist.

Example 2.6. Let $(A, \sqsubseteq)$ be a complete-lattice. Then: $\bigsqcup A = \top$, $\bigsqcap A = \bot$, $\bigsqcup \varnothing = \bot$, and $\bigsqcap \varnothing = \top$. $\top$ and $\bot$ are called top and bottom, respectively.

Definition 2.7 (Monotone Function)**.** Let $(A, \sqsubseteq)$ be a poset. A function $F : A \to A$ is *monotone* if for all x, y in A, if $x \sqsubseteq y$ then $F(x) \sqsubseteq F(y)$.

Definition 2.8. Let $(A, \sqsubseteq)$ be a poset and $F : A \to A$. An element x in A is,

1. a *prefixpoint* of F if $x \sqsubseteq F(x)$,

2. a *postfixpoint* of F if $F(x) \sqsubseteq x$.

Definition 2.9 (Fixpoint)**.** Let $(A, \sqsubseteq)$ be a poset and $F : A \to A$. An element x in A is a *fixpoint* of F if it is both a *pre-* and *postfixpoint* of F.

Theorem 2.2 (Knaster-Tarski)**.** *Let $(A, \sqsubseteq)$ be a complete lattice and let F be a monotone function on A. Further, let $\hat{y} = \bigsqcup\{x \mid x \sqsubseteq F(x)\}$ and $\hat{z} = \bigsqcap\{x \mid F(x) \sqsubseteq x\}$. It holds, that:*

1. $\hat{y}$ and $\hat{z}$ are fixpoints of F,

2. for all fixpoints x of F, $\hat{z} \sqsubseteq x \sqsubseteq \hat{y}$

We write $\mathrm{lfp}(F) = \bigsqcap\{x \mid F(x) \sqsubseteq x\}$ and $\mathrm{gfp}(F) = \bigsqcup\{x \mid x \sqsubseteq F(x)\}$, to denote the least fixpoint and the greatest fixpoint, respectively.

Exercise 2.1. Prove the Knaster Tarski Theorem (Theorem 2.2).

Definition 2.10 ($\bigsqcup$-continuous)**.** Consider a complete lattice $(A, \sqsubseteq)$. A function $f \colon A \to A$ is $\bigsqcup$-continuous if, for all increasing sequences $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq x_2 \sqsubseteq \ldots$, we have

$$f\left(\bigsqcup\{x_n : n \in \mathbb{N}\}\right) = \bigsqcup\{f(x_n) : n \in \mathbb{N}\}\,.$$

Definition 2.11 ($\bigsqcap$-continuous)**.** Consider a complete lattice $(A, \sqsubseteq)$. A function $f\colon A \to A$ is $\bigsqcap$-continuous if, for all increasing sequences $x_0 \sqsupseteq x_1 \sqsupseteq x_2 \sqsupseteq x_2 \sqsupseteq \ldots$, we have

$$f\left(\bigsqcap\{x_n : n \in \mathbb{N}\}\right) = \bigsqcap\{f(x_n) : n \in \mathbb{N}\}\,.$$

Lemma 2.3. *Both $\bigsqcup$-continuous and $\bigsqcap$-continuous imply monotonicity respectively.*

Theorem 2.4 (Constructive Fixpoints)**.** *Consider a complete lattice $(A, \sqsubseteq)$ and a monotonic function $f\colon A \to A$. Then,*

$$\mathrm{lfp} f = \bigsqcup\{f^n(\bot) : n \in \mathbb{N}\} \quad \textit{and} \quad \mathrm{gfp} f = \bigsqcap\{f^n(\top) : n \in \mathbb{N}\}\,.$$

Exercise 2.2. Prove the Constructive Fixedpoints Theorem (Theorem 2.4)

2.3 Induction and Co-Induction

Definition 2.12. Define $\mathbb{N}$ as the smallest set X s.t. (i) $0 \in X$ and (ii) if $n \in X$, then $Sn \in X$

Remark 2.1. In Definition 2.12, we consider a universal set U sufficiently big, the complete lattice $(2^U, \subseteq)$ and the function on sets given by $f(Y) \coloneqq \{0\} \cup \{Sn : n \in Y\}$. Then, $\mathrm{lfp} f = \mathbb{N}$.

Definition 2.13. Consider a finite alphabet Σ. Define Σ^* as the smallest set X such that (i) $\epsilon \in X$ and (ii) for all $a \in \Sigma$, we have $aX \subseteq X$.

Remark 2.2. Inductively defined sets are countable and consist of finite elements. They can be written as rules of the form

$$\frac{x}{f(x)}$$

expressing that if $x \in X$, then $f(x) \in X$. Moreover, they allow proof by induction.

> Consider proving that for all $x \in X$ we have $G(x)$. This can be proven by showing (i) $G(\bot)$ and (ii) for all $x \in X$, if $G(x)$, then $G(f(x))$.

Definition 2.14 (Balanced binary sequences)**.** Define the set S as the largest set X such that $X \subseteq 01X \cup 10X$.

Remark 2.3. In the definition of balanced binary sequences, we consider the complete lattice $(\Sigma^\omega, \subseteq)$ and the function on sets given by $f(X) \coloneqq 01X \cup 10X$. Then, balanced binary sequences correspond to $\mathrm{gfp} f$.

Definition 2.15 (Interval $[0, 1]$)**.** Define the set S as the largest set X such that $X \subseteq 0X \cup 1X \cup \ldots \cup 9X$.

Remark 2.4. Co-inductively defined sets are uncountable and consist of infinite elements. They can be written as rules of the form

$$\frac{x}{f(x)}$$

expressing that for all $y \in X$, there exists x such that $y = f(x)$ and $x \in X$. Moreover, they allow proof by co-induction.

> Consider proving that for all x, if $G(x)$, then $x \in X$. This can be proven by showing (i) for all x and i, if $G(f_i(x))$, then $G(x)$, where $\{f_1, \dots, f_n\}$ is the set of rules that define the set X.

Exercise 2.3 (Prove balanced binary sequences)**.** Consider S generated by the rules $X \Leftarrow 01X$ and $X \Leftarrow 10X$. Prove that, for all binary words x, we have that $x\ inS$ if and only if every finite prefix of even length of x has the same number of 0s and 1s. (Hints: The direction $\Leftarrow$ can be proven by co-induction. The direction $\Rightarrow$ can be proven by induction on the length of the prefix.)

2.4 Correctness of programs on (co-)inductive sets

Definition 2.16 (`merge`)**.** Let Σ^* be an alphabet with linear operator $\leq$. Then for all $x, y \in \Sigma^*$ and $a, b \in \Sigma$ we have that: (i) $\texttt{merge}(\epsilon, x) = x$, (ii) $\texttt{merge}(y, \epsilon) = y$, and (iii) $\texttt{merge}(a{\cdot}x, b{\cdot}y) = a{\cdot}\texttt{merge}(x, b{\cdot}y)$ if $a \leq b$, otherwise $b{\cdot}\texttt{merge}(a{\cdot}x, y)$.

Definition 2.17 (Concatenation)**.** For all $y \in \Sigma^*$, we have that $\epsilon, y = y$. For all $a \in \Sigma$, and $x, y \in \Sigma^*$, we have that $(a \cdot x), y = a \cdot (x, y)$.

Definition 2.18.

Theorem 2.5 (Associativity)**.** $\forall x, y, z \in \Sigma^*$ *we have that* $(x, y), z = x, (y, z)$*.*

Proof. Consider arbitrary $\hat{y}, \hat{z} \in \Sigma^*$. The goal is to show that for all $x \in \Sigma^*$, holds that $(x, \hat{y}), \hat{z} = x, (\hat{y}, \hat{z})$. We use induction on x and we have the following cases:

- $x = \epsilon$. We get that $(\epsilon, \hat{y}), \hat{z} = \epsilon, (\hat{y}, \hat{z})$. Using Definition 2.17 we get $\hat{y}, \hat{z} = \hat{y}, \hat{z}$.
- $x = \hat{a}{\cdot}\hat{u}$ for some $a \in \Sigma$ and $\hat{u} \in \Sigma^*$, we have that the induction hypothesis is $(\hat{u}, \hat{y}), \hat{z} = \hat{u}, (\hat{y}, \hat{z})$ and the goal is to show that

  $$(\hat{a} \cdot \hat{u}, \hat{y}), \hat{z} = (\hat{a} \cdot \hat{u}), (\hat{y}, \hat{z}). \tag{1}$$

  If we apply Definition 2.17 we get

  $$(\hat{a} \cdot (\hat{u}, \hat{y})), z = \hat{a} \cdot (\hat{u}, (\hat{y}, \hat{z})). \tag{2}$$

  Applying Definition 2.17 to the left side and the induction hypothesis to the right side of equation 2 we get

  $$\hat{a} \cdot ((\hat{u}, \hat{y}), \hat{z}) = \hat{a} \cdot ((\hat{u}, \hat{y}), \hat{z}). \tag{3}$$

□

Definition 2.19 (`reverse`)**.** $\texttt{reverse}(\epsilon) = \epsilon$. For all $a \in \Sigma$ and $x \in \Sigma^*$, we have that $\texttt{reverse}(a \cdot x) = \texttt{reverse}(x) \cdot a$.

Definition 2.20 (`r`)**.** For all $y \in \Sigma^*$, we have that $\texttt{r}(\epsilon, y) = y$. For all $a \in \Sigma$ and $x, y \in \Sigma^*$, we have that $\texttt{r}(a \cdot x, y) = \texttt{r}(x, a \cdot y)$.

Theorem 2.6. *For all $x \in \Sigma^*$, $\texttt{reverse}(x) = \texttt{r}(x, \epsilon)$.*

Proof. We prove it by induction on x.

- For the base case we have to prove that for $x = \epsilon$, $\texttt{reverse}(\epsilon) = \texttt{r}(\epsilon, \epsilon)$. Using Definition 2.19 we get that $\epsilon = \epsilon$.

- As induction hypothesis we assume that for all $y \in \Sigma^*$ and $x = \hat{a} \cdot \hat{u}$, where $\hat{a} \in \Sigma$ and $\hat{u} \in \Sigma^*$, it holds that $\texttt{reverse}(\hat{u}), y = \texttt{r}(\hat{u}, y)$. Then we have to show that

$$\texttt{reverse}(\hat{a} \cdot \hat{u}), y = \texttt{r}(\hat{a} \cdot \hat{u}, y). \tag{4}$$

  - Consider an arbitrary $\hat{y}$. If we apply Definition 2.20 to the left and the right side respectively we get

$$(\texttt{reverse}(\hat{u}), \hat{a}), \hat{y} = \texttt{r}(\hat{u}, \hat{a} \cdot \hat{y}). \tag{5}$$

  - Then we apply Theorem 2.5 to the left side and get

$$\texttt{reverse}(\hat{u}), (\hat{a}, \hat{y}) = \texttt{r}(\hat{u}, \hat{a} \cdot \hat{y}). \tag{6}$$

    Finally, we apply the induction hypothesis to the left side of our previous equation and get

$$\texttt{r}(\hat{u}, \hat{a} \cdot \hat{y}) = \texttt{r}(\hat{u}, \hat{a} \cdot \hat{y}). \tag{7}$$

□

Definition 2.21 (`odd` and `even`)**.** For all $x \in \Sigma^*$ and $a \in \Sigma$, we have that $\texttt{odd}(\epsilon) = \epsilon$ otherwise $\texttt{odd}(a \cdot x) = a \cdot \texttt{even}(x)$, where $\texttt{even}(\epsilon) = \epsilon$ and $\texttt{even}(a \cdot x) = \texttt{odd}(x)$.

Definition 2.22 (Well-founded)**.** A binary relation $\prec$ on a set A is well-founded if there is no infinite descending sequence $x_0 \succ x_1 \succ x_2 \succ \ldots$ on A.

Example 2.7. Two examples are $<$ over the set of natural numbers and ”shorter than” on Σ^*.

Remark 2.5 (Well-founded induction)*.* In order to show $\forall x \in A$, $G(x)$ for well-founded $\prec$. Consider an arbitrary $\hat{x} \in A$. Assume $\forall y \prec \hat{x}$, $G(y)$. Show $G(\hat{x})$.

Definition 2.23 (`mergesort`)**.** For all $x \in \Sigma^*$, we have that $\texttt{mergesort} = \texttt{merge}(\texttt{mergesort}(\texttt{odd}(x))$ and $\texttt{mergesort}(\texttt{even}(x)))$

Definition 2.24 ($\texttt{sorted}$)**.** For all $a \in \Sigma$, we have that $\texttt{sorted}(\epsilon)$, $\texttt{sorted}(a \cdot \epsilon)$ and for all $x \in \Sigma^*$, we have that $\texttt{sorted}(a \cdot b \cdot x)$ iff $a \leq b$ and $\texttt{sorted}(b \cdot x)$.

Lemma 2.7. *For all $x, y \in \Sigma^*$, if $\texttt{sorted}(x)$ and $\texttt{sorted}(y)$, then $\texttt{sorted}(\texttt{merge}(x, y))$.*

Theorem 2.8 (Merge-Sort)**.** *For all $x \in \Sigma^*$, we have that $\texttt{sorted}(\texttt{mergesort}(x))$.*

Theorem 2.9 (Permutation)**.** *For all $x \in \Sigma^*$, we have that permutation$(x, \texttt{mergesort}(x))$.*

Exercise 2.4. (i) Proof Lemma 2.7 and Theorem 2.8. (ii) Write a definition of permutation. (iii) Proof Theorem 2.9

2.4.1 Humans and Monkeys

Definition 2.25. For all x and y, we have $x > y$ iff $\texttt{parent}(x, y)$ or there exists z such that $\texttt{parent}(x, z)$ and $z > y$.

Definition 2.26. For all x and y, we have $x > y$ iff $\texttt{parent}(x, y)$ or there exists z such that $x > z$ and $\texttt{parent}(z, y)$.

Axiom 2.1. For all x, we have that $h(x)$ implies $\neg m(x)$ and $m(x)$ implies $\neg h(x)$.

Axiom 2.2. $<$ is well-founded.

Theorem 2.10. *Provide a proof of the following using Def. 2.25 and then using Def. 2.26. If there exist x and y such that $x > y$ and $m(x)$ and $h(y)$, then there exist x and y such that $\texttt{parent}(x, y)$ and $m(x)$ and $h(y)$.*

Exercise 2.5. Proof Theorem 2.10.

3 Propositional (Boolean) Logic

In propositional (Boolean) Logic we differentiate between:

- Syntax (Proof theory) where we look at strings of letters, e.g. $a, b, A, B, \#$, in a concrete syntax.
- Semantics (Model Theory) where we look at boolean ($\mathbb{B} := \{\textbf{true}, \textbf{false}\}$) functions (gates) $f \colon \mathbb{B}^n \to \mathbb{B}$.

A	B	$A \wedge B$	$A \vee B$	$\neg A$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Definition 3.1 (Syntax)**.** Given a finite set $P = \{p_0, p_1, ..., p_n\}$ of propositions. We define the set Φ of (well-formed) formulas (countable).

$$\varphi ::= P \mid \top \mid \bot \mid \neg\varphi \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \Rightarrow \psi)$$

Remark 3.1. The definition is inductive. That is, Φ is the smallest set X generated by the following rules creating an abstract syntax tree:

$$\frac{}{p}\;{\scriptstyle p\,\in\,P} \qquad \frac{}{\bot} \qquad \frac{}{\top} \qquad \frac{\varphi}{\neg\varphi} \qquad \frac{\varphi_1 \qquad \varphi_2}{\varphi_1 \land \varphi_2} \qquad \frac{\varphi_1 \qquad \varphi_2}{\varphi_1 \lor \varphi_2}$$
$$\frac{\varphi_1 \qquad \varphi_2}{\varphi_1 \Rightarrow \varphi_2}$$

Remark 3.2. The presence order is implied by the order of the rules, e.g. $\neg a \lor b \land c \equiv \neg a \lor (b \land c)$ Binary connectives are right associative, e.g., $a \land b \land c \equiv a \land (b \land c)$ or $a \Rightarrow b \Rightarrow c \equiv a \Rightarrow (b \Rightarrow c)$ The connective $\Leftrightarrow$ is syntactic sugar for $(\varphi \Rightarrow \psi) \lor (\psi \Rightarrow \varphi)$.

Definition 3.2 (Semantics)**.** Let $\nu : P \to \mathbb{B}$ be a truth assignment and let $V = \mathbb{B}^P$. Then we define an interpretation $\llbracket\cdot\rrbracket : \Phi \times V \to \mathbb{B}$ as follows:

- $\llbracket p \rrbracket_\nu = \nu(p)$, $\llbracket \bot \rrbracket_\nu =$ **false**, $\llbracket \bot \rrbracket_\nu =$ **true**;
- $\llbracket \neg\varphi \rrbracket_\nu =$ if $\llbracket \neg\varphi \rrbracket_\nu =$ **true** then **false** else **true**;
- $\llbracket \varphi_1 \land \varphi_2 \rrbracket_\nu =$ if $\llbracket \varphi_1 \rrbracket_\nu =$ **true** and $\llbracket \varphi_2 \rrbracket_\nu =$ **true** then **true** else **false**;
- $\llbracket \varphi_1 \lor \varphi_2 \rrbracket_\nu =$ if $\llbracket \varphi_1 \rrbracket_\nu =$ **true** or $\llbracket \varphi_2 \rrbracket_\nu =$ **true** then **true** else **false**;
- $\llbracket \varphi_1 \Rightarrow \varphi_2 \rrbracket_\nu =$ if $\llbracket \varphi_1 \rrbracket_\nu =$ **true** or $\llbracket \varphi_2 \rrbracket_\nu =$ **false** then **false** else **true**;

Definition 3.3 (Model, Satisfiable, Validity)**.** The truth assignment ν is called a *model* of φ if $\llbracket \varphi \rrbracket_\nu =$ **true**, written using entailment as $\nu \models \varphi$. The formula φ is *satisfiable*, if $\exists \nu \in V : \nu \models \varphi$. The formula φ is *valid*, if $\forall \nu \in V : \nu \models \varphi$, written as $\models \varphi$.

Definition 3.4 (Problems in Logic)**.** The common problems in logic are:

- the *evaluation problem*: given φ and ν, find $\llbracket \varphi \rrbracket_\nu$. It can be solved in linear time.
- the *validity problem*: given φ, is φ valid? It is co-NP-complete.
- the *satisfiability problem (SAT)*: given φ, is φ satisfiable? It is NP-complete.

For propositional logic, these problems are dual: φ is valid iff $\neg\varphi$ is unsatisfiable.

Definition 3.5 (Soundness)**.** An algorithm (not necessarily halting) to solve satisfiability with input φ and output $\mathbb{B}$ is sound iff: (i) If output is **true**, then φ is satisfiable. (ii) If the output is **false**, then φ is unsatisfiable.

Definition 3.6 (Complete)**.** An algorithm to solve satisfiability with input φ and output $\mathbb{B}$ is *semi-complete* iff (i) if φ is satisfiable, then the algorithm halts with output true. The algorithm is *complete* if (i) and (ii) if φ is unsatisfiable, then the algorithm halts with false. It is called a *decision procedure.*

Remark 3.3 (SAT Solvers). Decision procedures for propositional logic are called SAT Solvers. SAT solvers typically first transform φ into CNF (Conjunctive Normal Form): (i) conjunction of clauses,(ii) each clause is a disjunction of literals, and (iii) each literal is a proposition or its negation.

Definition 3.7 (SAT Solver: Exhaustive Search)**.** On possibility is to enumerate all possible 2^n interpretations of the n propositions. Let Γ be a set of formulas and let $\Gamma[p]$ denote all occurrences of p in Γ then we define the rule

$$\frac{\Gamma[\bot] \;\; \textbf{unsat} \qquad \Gamma[\top] \;\; \textbf{unsat}}{\Gamma[p] \;\; \textbf{unsat}} \qquad \frac{\Gamma[\top] \;\; \textbf{unsat}}{\Gamma, \bot \;\; \textbf{unsat}}$$

Definition 3.8 (SAT Solver: Resolution)**.** The resolution rule is

$$\frac{\Gamma, C_1[\bot] \vee C_2[\top] \;\; \textbf{unsat}}{\Gamma, C_1[p], C_2[\neg p] \;\; \textbf{unsat}}$$

Exercise 3.1. Show that resolution is sound and complete.

3.1 Formal Systems

Definition 3.9 (Formal System)**.** A *formal system* F is a set of rules. Rule is a finite set of (formulas) premises $p_0, \ldots, p_k$ and (a formula called) conclusion c. An axiom is a rule without premises.

Remark 3.4. We usually have infinitely many rules but only finitely many different rule schemata. For example, schema

$$\frac{}{\varphi \Rightarrow \varphi}$$

gives infinitely many rules, e.g.,

$$\frac{}{p_3 \Rightarrow p_3}$$

Definition 3.10 (Proof: Linear View)**.** A *proof* (derivation) is a finite sequence of formulas $\varphi_0, \ldots, \varphi_n$ such that every formula in the sequence is (i) either an axiom (which can be viewed as a special case of the following), or (ii) the conclusion of a rule whose premises occur earlier in the sequence.

Remark 3.5. Linear view is usually easier for proving meta theorems. Tree view (inductive definition) is usually better in practice.

Definition 3.11 (Theorem)**.** A theorem can be defined (i) syntactically or (ii) semantically. The formula φ is a *theorem*, (i) (in the formal system F) if it has a proof in a formal system F, denoted as $\vdash_F \varphi$; (ii) if φ is valid, denoted as $\models \varphi$.

Remark 3.6 (Logic). Formal system equipped with semantics is called a logic. The core problem in logic is establishing $\vdash \varphi$ iff $\models \varphi$. We speak of (i) *soundness* if $\vdash \varphi \implies \models \varphi$ and of (ii) *completeness* if $\models \varphi \implies \vdash \varphi$.

Definition 3.12 (Soundness and Completeness)**.** Let F be a formal system, let R be a rule of this formal system, let L be a logic, and let ψ be a formula then:

$$\begin{aligned}
R \text{ is } \textit{sound} &\iff \text{if all premises of } R \text{ are valid, then its conclusion is valid;}\\
F \text{ is } \textit{sound} &\iff \text{all rules are sound (or every theorem is valid);}\\
F \text{ is } \textit{complete} &\iff \text{every valid formula is a theorem;}\\
F \text{ is } \textit{consistent} &\iff \nvdash_F \bot \text{ (or there exists a formula that is not a theorem);}\\
R \text{ is } \textit{derivable} \text{ in } F &\iff \text{for all formulas } \varphi, \vdash_{F\cup\{R\}} \varphi \text{ iff } \vdash_F \varphi;\\
\psi \text{ is } \textit{expressible} \text{ in } L &\iff \text{there exists a formula } \chi \text{ of } L \text{ s.t., for all } \nu,\ [[\psi]]_\nu = [[\chi]]_\nu;
\end{aligned}$$

Example 3.1. For example $\varphi_1 \wedge \varphi_2$ is expressible using only $\neg$ and $\vee$ (de Morgan) as $\psi = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$.

Remark 3.7. We can enumerate all theorems by systematically enumerating all possible proofs. The proof is a witness for validity.

- Sound formal system $\Longrightarrow$ sound procedure for validity (but not necessarily complete).
- Sound and complete formal system $\Longrightarrow$ sound semi-complete procedure for validity (may not terminate on inputs that represent a formula that is not valid).

To get a decision procedure (sound and complete procedure for validity), we need both: (i) sound complete formal system for validity, and (ii) sound complete formal system for satisfiability (to define a formal system for satisfiability. Those are formal systems systems where valid formulas are replaced by satisfiable judgments. Then all axioms are satisfiable and all rules go from satisfiables to satisfiable. If we have both (i) and (ii) then for every input φ, one of them will eventually terminate and we can conclude either φ is valid, or $\neg\varphi$ is satisfiable (which means that φ is not valid). This is only possible the set is decidable, i.e., both the set and its complement are recursively-enumerable implying that the set is recursive (decidable).

3.2 Hilbert formal system $\mathcal{H}$ for propositional logic

Definition 3.13. The Hilbert system $\mathcal{H}$ uses connectives $\Rightarrow$ and $\neg$ only, has three axioms

$$\begin{aligned}
(K) \quad & \varphi \Rightarrow \psi \Rightarrow \varphi\\
(S) \quad & (\varphi \Rightarrow \psi \Rightarrow \chi) \Rightarrow ((\varphi \Rightarrow \psi) \Rightarrow (\varphi \Rightarrow \chi))\\
(\textit{ex middle}) \quad & (\neg\varphi \Rightarrow \neg\psi) \Rightarrow (\psi \Rightarrow \varphi)
\end{aligned}$$

and one rule, called *modus ponens* (MP)

$$\frac{\varphi \qquad \varphi \Rightarrow \psi}{\psi}\,(\text{MP})$$

Example 3.2. The proof of $\varphi \Rightarrow \varphi$ in Hilbert system:

$$\begin{aligned}
(K) &\quad \varphi \Rightarrow (\psi \Rightarrow \varphi) \Rightarrow \varphi \\
(S) &\quad (\varphi \Rightarrow (\psi \Rightarrow \varphi) \Rightarrow \varphi) \Rightarrow ((\varphi \Rightarrow \psi \Rightarrow \varphi) \Rightarrow (\varphi \Rightarrow \varphi)) \\
(MP) &\quad (\varphi \Rightarrow \psi \Rightarrow \varphi) \Rightarrow (\varphi \Rightarrow \varphi) \\
(K) &\quad \varphi \Rightarrow \psi \Rightarrow \varphi \\
(MP) &\quad \varphi \Rightarrow \varphi
\end{aligned}$$

Theorem 3.1 (Soundness and Completeness)**.** *The formal system $\mathcal{H}$ is sound and complete for propositional logic.*

Corollary 3.2. *The formal system $\mathcal{H}$ is consistent for propositional logic.*

Theorem 3.3 (Deduction Theorem)**.** $\Gamma \vdash \varphi \Rightarrow \psi$ *iff* $\Gamma, \varphi \vdash \psi$ *where* $\Gamma \vdash \varphi$ *denotes* $\underset{F \cup \Gamma}{\vdash} \varphi$*, i.e., the set of formulas* Γ *is used as added axioms.*

Proof. “ $\Longrightarrow$ ”: One application of modus ponens.
“ $\Longleftarrow$ ”: Assume ψ has a proof π using axioms Γ, φ, (K), (S), (ex). Show that $\varphi \Rightarrow \psi$ has a proof π' using Γ, (K), (S), (ex). Proof by induction on length n of π.

- Case $n = 1$: ψ must be an axiom. Either $\psi \in \Gamma \cup \{\mathrm{K}, \mathrm{S}, \mathrm{em}\}$ so we prove it by (K), or $\psi = \varphi$ so we use $\vdash \varphi \Rightarrow \varphi$ as derived above.

- Case $n > 1$: ψ is the result of an application of modus ponens. We have χ and $\chi \Rightarrow \psi$, both of which were derived from Γ, φ in fewer steps. Induction hypothesis gives us $\Gamma \vdash \varphi \Rightarrow \chi$ and $\Gamma \vdash \varphi \Rightarrow \chi \Rightarrow \psi$. We use (S) in the form $(\varphi \Rightarrow \chi \Rightarrow \psi) \Rightarrow (\varphi \Rightarrow \chi) \Rightarrow (\varphi \Rightarrow \psi)$ and apply modus ponens twice, resulting in $\varphi \Rightarrow \psi$ derived from Γ only.

□

3.3 Natural Deduction for Propositional Logic

Remark 3.8. For this chapter, we replace the word *formula* in rules and proofs of a formal system by the word *judgement*.

Remark 3.9. Judgements in *Natural Deduction* have the form $\Gamma \vdash \varphi$, where Γ is a set of formulas and φ is a formula. Semantically, $\models \Gamma \vdash \varphi$, i.e., Γ derives φ, meaning that for all interpretations ν, if all formulas in Γ are true in ν, then φ is true in ν. Formal system for natural deduction defines a meta-symbol $\vdash_{\text{meta}}$ such that $\vdash_{\text{meta}} (\Gamma \vdash \varphi)$.

Definition 3.14 ($\mathcal{NJ}$)**.** We use the notation Γ, φ to show $\Gamma \cup \{\varphi\}$. The system $\mathcal{NJ}$ consists of the following rules: The axiom introduction

$$\frac{}{\Gamma, \varphi \vdash \varphi}\ (\mathrm{ax})$$

The true-introduction, expressing that $\top$ can always be derived, and the false-elimination, expressing if $\bot$ can be derived, then anything can be derived.

$$\dfrac{}{\Gamma \vdash \top}\;(\top\text{-intro}) \qquad \dfrac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\;(\bot\text{-elim})\;.$$

Conjunction elimination and introduction

$$\dfrac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\;(\wedge\text{-elim}) \qquad \dfrac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\;(\wedge\text{-elim}) \qquad \dfrac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\;(\wedge\text{-intro})\;.$$

Disjunction elimination and introduction

$$\dfrac{\Gamma \vdash \varphi \vee \psi \qquad \Gamma, \varphi \vdash \chi \qquad \Gamma, \psi \vdash \chi}{\Gamma \vdash \chi}\;(\vee\text{-elim})$$

$$\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\;(\wedge\text{-intro}) \qquad \dfrac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\;(\wedge\text{-intro})\;.$$

Negation elimination and introduction

$$\dfrac{\Gamma \vdash \varphi \qquad \Gamma \vdash \neg\varphi}{\Gamma \vdash \bot}\;(\neg\text{-elim}) \qquad \dfrac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}\;(\neg\text{-intro})\;.$$

Implication elimination and introduction (observe how $\rightarrow$-elim is similar to modus ponens)

$$\dfrac{\Gamma \vdash \varphi \qquad \Gamma \vdash \varphi \Rightarrow \psi}{\Gamma \vdash \psi}\;(\Rightarrow\text{-elim}) \qquad \dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \Rightarrow \psi}\;(\Rightarrow\text{-intro})\;.$$

Exercise 3.2. Prove implication transitivity using Natural Deduction.

Example 3.3 (Contrapositive)**.** Show $(\varphi \rightarrow \psi) \rightarrow (\neg\psi \rightarrow \neg\varphi)$.

Proof. We proof this using $\mathcal{NJ}$. The proof is represented as a tree and constructed from the root (bottom).

$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{ax})\,\dfrac{}{\varphi \Rightarrow \psi, \neg\psi, \varphi \vdash \neg\psi} \qquad \dfrac{(\text{ax})\,\dfrac{}{\varphi \Rightarrow \psi, \neg\psi, \varphi \vdash \varphi} \qquad (\text{ax})\,\dfrac{}{\varphi \Rightarrow \psi, \neg\psi, \varphi \vdash \varphi \Rightarrow \psi}}{\varphi \Rightarrow \psi, \neg\psi, \varphi \vdash \psi}\;(\Rightarrow\text{-elim})}{\varphi \Rightarrow \psi, \neg\psi, \varphi \vdash \bot}\;(\neg\text{-elim})}{\varphi \Rightarrow \psi, \neg\psi \vdash \neg\varphi}\;(\neg\text{-intro})}{\dfrac{\varphi \Rightarrow \psi \vdash \neg\psi \Rightarrow \neg\varphi}{\vdash (\varphi \Rightarrow \psi) \Rightarrow (\neg\psi \Rightarrow \neg\varphi)}\;(\Rightarrow\text{-intro})}\;(\Rightarrow\text{-intro})$$

□

Remark 3.10. Observe how this method of writing proofs in Natural Deduction requires us to rewrite the context for every step. We can use a slightly different notation to avoid this repetition. We can draw boxes to introduce *contexts* in a proof. Every formula written inside a box is assumed to hold only within that box. The following "proofs" are examples of using this notation.

$$\frac{\boxed{\begin{matrix}\varphi\\ \vdots\\ \psi\end{matrix}}}{\varphi\to\psi}\qquad\frac{\boxed{\begin{matrix}\varphi\\ \vdots\\ \bot\end{matrix}}}{\neg\varphi}\qquad\frac{\varphi\lor\psi\qquad\boxed{\begin{matrix}\varphi\\ \vdots\\ \chi\end{matrix}}\qquad\boxed{\begin{matrix}\varphi\\ \vdots\\ \chi\end{matrix}}}{\chi}$$

Example 3.4. Using the notation from Remark 3.10, we can rewrite the proof for Example 3.3:

$$\dfrac{\boxed{\begin{array}{c}1.\ \varphi\to\psi\\ \dfrac{\boxed{\begin{array}{c}2.\ \neg\psi\\ \dfrac{\boxed{\begin{array}{c}3.\ \varphi\\ 4.\ \psi\ (\to\text{-elim, 1, 3})\\ 5.\ \bot\ (\neg\text{-elim, 2, 4})\end{array}}}{\neg\varphi}\end{array}}}{\neg\psi\to\neg\varphi}\end{array}}}{(\varphi\to\psi)\to(\neg\psi\to\neg\varphi)}$$

Remark 3.11. The system $\mathcal{NJ}$ defined is in fact the Intuitionistic Natural Deduction system. The following rule, namely the law of excluded middle, cannot be derived in $\mathcal{NJ}$:

$$\frac{}{\Gamma\vdash\varphi\lor\neg\psi}\ (\text{ex})\ .$$

Assumption of the law of excluded middle is in fact an important distinction between intuitionistic and classical logic.

Definition 3.15 ($\mathcal{NK}$)**.** The system $\mathcal{NK}$ consists of the same rules as $\mathcal{NJ}$ in addition to the law of excluded middle.

$$\frac{}{\Gamma\vdash\varphi\lor\neg\psi}\ (\text{ex})\ .$$

Theorem 3.4. *The $\mathcal{NK}$ system is sound and complete for propositional logic.*

Example 3.5. Show that there exist $a, b \notin \mathbb{Q}$ such that $a^b \in \mathbb{Q}$.

Proof. Let $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$. We know that $\sqrt{2} \notin \mathbb{Q}$. We do a "classical" case-splitting on $a \in \mathbb{Q}$:

- Case $a \notin \mathbb{Q}$: We have

$$a^b = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^2 = 2 \in \mathbb{Q}$$

- Case $a \in \mathbb{Q}$: We are already done with the proof; let $a_1 = b_1 = \sqrt{2}$. We know $a_1, b_1 \notin \mathbb{Q}$ and, by assumption, ${a_1}^{b_1} \in \mathbb{Q}$.

Observe how this classical-style proof utilizes the law of excluded middle in the case-splitting: $\sqrt{2}^{\sqrt{2}}$ is either in $\mathbb{Q}$ or not in $\mathbb{Q}$; there is no *middle*. □

3.4 Kripke Semantics

Remark 3.12. Classically, an interpretation $\nu\colon P \to \mathbb{B}$ is defined as a mapping from a set of propositions to boolean values $\top$ and $\bot$. For intuitionistic reasoning, we define a new semantics.

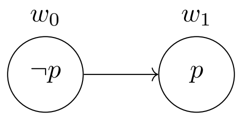
Definition 3.16. A Kripke model m is defined as a tuple $(W, \preceq, w_0, \nu\colon W \times P \to \mathbb{B})$, where W is a set of classical worlds, $\preceq$ is a pre-order relation on W, w_0 is the initial world, and ν is a function from pairs of world and proposition to boolean values such that for any $w, w' \in W$ and any $p \in P$, if $w \preceq w'$, then $v(w,p) \preceq v(w',p)$.

Remark 3.13. Informally, a Kripke model is an interpretation model for intuitionistic proof systems.

Theorem 3.5. *The following facts hold for any Kripke model m: (i) $m \models \varphi$ iff $m \vdash_{w_0} \varphi$; (ii) $m \not\models_w \bot$ for any world w; (iii) $m \models_w p$ iff $\nu(w,p) = \top$; (iv) $m \models_w \varphi \to \psi$ iff for any w', if $w \preceq w'$ and $m \models_{w'} \varphi$, then $m \models_{w'} \psi$.*

Remark 3.14. In $\mathcal{NJ}$, whenever you show $\varphi \vee \psi$, you need to show either φ, or ψ. As previously stated, the law of excluded middle cannot be derived in $\mathcal{NJ}$. To show this, we need to show that there exists a Kripke model m such that excluded middle is false in a world w of m.

Example 3.6. Let us define a Kripke model with only one proposition p and only two worlds w_0 and w_1, where $w_0 \preceq w_1$, and p is false in w_0 and true in w_1.



Let us examine what formulas are true (or false) in each world. By definition, p is false in w_0. Let us examine the value of $\neg p$ in w_0. We can safely substitute $\neg p$ with $p \Rightarrow \bot$. By definition, $p \Rightarrow \bot$ holds in w_0 iff for any world w, if $w_0 \preceq w$ and p is true in w, then $\bot$ is true in w. We know p is true in w_1. We also know that $\bot$ is not true in w_1, as it is not true in any world. So, by definition, $p \Rightarrow \bot$ is false in w_0. So, both p and $\neg p$ are false in w_0, from which we obtain that $p \vee \neg p$ is also false in w_0.

3.5 Sequent (Gentzen) Calculus and $\mathcal{LK}$

Remark 3.15. Judgements in the $\mathcal{LK}$ proof system have the form $\Gamma \vdash \Delta$, where both Γ and Δ are sets of formulas. Judgement $\Gamma \vdash \Delta$ should be read as "the *conjunction* of the formulas in Γ implies the *disjunction* of the formulas in Δ". Semantically, for a classical interpretation ν, $\nu \models (\Gamma \vdash \Delta)$ if and only if, if all formulas in Γ are true under ν, then some formula in Δ is true under ν.

Definition 3.17. The Sequent (Gentzen) Calculus is composed of the following rules. (Observe how every non-axiom judgement increases the number of logical connectives in the set of formulas.) Axioms

$$\frac{}{\Gamma, \varphi \vdash \varphi, \Delta}\,(\text{ax}) \qquad \frac{}{\Gamma, \bot \vdash \Delta}\,\bot\text{-elim} \qquad \frac{}{\Gamma \vdash \top, \Delta}\,\top\text{-intro}$$

Conjunction

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta}\,(\wedge\text{-left}) \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}\,(\wedge\text{-right})$$

Disjunction

$$\frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}\,(\vee\text{-left}) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}\,(\vee\text{-right})$$

Negation

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}\,\neg\text{-right} \qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta}\,\neg\text{-left}$$

Implication

$$\frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \to \psi, \Delta}\,\to\text{-right} \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \to \psi \vdash \Delta}\,\to\text{-left}$$

Theorem 3.6. *The $\mathcal{LK}$ proof system is sound and complete for propositional logic. This actually means that excluded middle can be derived in $\mathcal{LK}$.*

Exercise 3.3. Prove the following in $\mathcal{LK}$: (i) The law of excluded middle: $\vdash p \vee \neg p$ and (ii) implication transitivity.

3.6 Decision Procedures

Remark 3.16. We now talk about three proof systems for validity:

Hilbert	Natural Deduction	Grentzen
$\vdash \varphi$	$\Gamma \vdash \varphi$	$\Gamma \vdash \Delta$

Remark 3.17. We now talk abou three decision procedures (either for validity or for satisfiability)

Definition 3.18. We define three decision procedures (either for validity or for satisfiability): (i) Branching (if we don't derive $\bot$ then it is satisfiable)

$$\frac{\Gamma[\top]\ \textbf{unsat} \qquad \Gamma[\bot]\ \textbf{unsat}}{\Gamma[p]\ \textbf{unsat}}$$

(ii) Resolution

$$\frac{\Gamma, C_1, C_2, C_1[\bot] \vee C_2[\top]\ \ \textbf{unsat}}{\Gamma, C_1[p], C_2[\neg p]\ \ \textbf{unsat}}$$

(iii) DPLL using unit resolution combined with branching of lower priority (ℓ is a literal)

$$\frac{\Gamma, C[\bot]\ \ \textbf{unsat}}{\Gamma, \ell, C[\neg \ell]\ \ \textbf{unsat}}$$

Example 3.7. Consider $p \vee q \vee r,\ \neg p \vee \neg q \vee \neg r,\ \neg p \vee q \vee r,\ \neg q \vee r,\ q \vee \neg r$ First we branch on r.

$$\begin{array}{llclcl}
\text{Case } r = \bot : & p \vee q,\ \neg p \vee q,\ \neg q & \xrightarrow{Res. \neg q} & p,\ \neg p & \xrightarrow{Res. \neg p} \bot \\
\text{Case } r = \top : & \neg p \vee \neg q & \xrightarrow{q = \top} & \neg p & \xrightarrow{p = \bot} \bot
\end{array}$$

Definition 3.19 (Horn Clauses)**.** Horn clause is a clause with at most one positive literal.

Remark 3.18. We can view Horn clauses as implications where LHS is a conjunction of positive propositions:

$$\begin{aligned}
\neg p \vee \neg q &\iff p \wedge q \Rightarrow \bot \\
\neg p \vee \neg q \vee r &\iff p \wedge q \Rightarrow r \\
r &\iff \top \Rightarrow r
\end{aligned}$$

3.7 Metatheorems

Theorem 3.7. *A countable set Γ of formulas is satisfiable iff every finite subset of Γ is satisfiable.*

Theorem 3.8 (Craig's interpolation)**.** *We have $\vdash \varphi \Rightarrow \psi$ iff there exists a third formula χ (the "interpolant") which only uses nonlogical symbols (in propositional logic, it is propositions only) that occur in both φ and ψ such that $\vdash \varphi \Rightarrow \chi$ and $\vdash \chi \Rightarrow \psi$.*

Exercise 3.4. How hard is it to compute interpolates in propositional logic?

Definition 3.20. The cut rule in $\mathcal{NK}$, $\mathcal{NJ}$ (left) and $\mathcal{LK}$ (right) is respectively

$$\frac{\Gamma \vdash \varphi \qquad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi} \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \varphi \vdash \Delta}{\Gamma \vdash \Delta}$$

Theorem 3.9 (Cut elimination)**.** *If a judgement can be proved with the cut rule, it can also be proved without the cut rule.*

Remark 3.19 (Theorem 3.9). In fact, it is "iff"; the other direction is trivial. Of course, the proof may become longer (introducing a lemma φ often helps in practice). It is a purely syntactic metatheorem. We cannot use deduction to prove it.

4 First-order logic

4.1 Definition

Remark 4.1. First-order logic, also called "predicate logic", is propositional logic with quantifiers $\forall$ (for all) and $\exists$ (exists).

Definition 4.1 (Signature)**.** A Signature defines the non-logical symbols

(i) finite set of variables $X = \{x, y, z, \dots\}$

(ii) finite set of function symbols $F = \{f, g, h, \dots\}$

(iii) finite set of predicate symbols $P = \{p, q, r, \dots\}$

Each function symbol has a fixed "arity" (number of arguments), which can be zero (arity 0 gives a constant). Each predicate symbol has also a fixed "arity" (number of arguments), which can be zero (arity 0 gives a proposition).

Definition 4.2 (Logical symbols)**.** The logical symbols consist of:

(i) connectives $(\bot, \top, \neg, \wedge, \vee, \Rightarrow)$

(ii) quantifiers $(\forall, \exists)$

(iii) finite set of predicate symbols $P = \{p, q, r, \dots\}$

Remark 4.2. We don't add parentheses to the symbols; we will talk about syntax trees; only if we want to write them down as strings, we add parentheses (as few as possible).

Definition 4.3 (Syntax)**.** The syntax of first-order logic is given by the following grammar for terms and formulas repsectively

$$\begin{aligned} t &::= f_0 \mid f_n(t_1, \dots, t_n) \\ \varphi &::= p_0 \mid p_n(t_1, \dots, t_n) \mid \bot \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \forall x.\varphi \mid \exists x.\varphi \end{aligned}$$

When we write $\forall x.\varphi$, every occurence of x is bound in φ. A variable that is not bound is called free. A change of bound variables does not change the abstract syntax tree (same in abstract syntax).

Remark 4.3 (Safe substitution). If we want to substitute y^2 for x in $\forall x\,.\ \exists y\,.\ y \geq x + 1$, we must first rename y to z, i.e., $\forall x\,.\ \exists z\,.\ z \geq x + 1$, and then substitute, i.e., $\exists z\,.\ z \geq y^2 + 1$.

Definition 4.4 (Interpretation)**.** Interpretation $\mathcal{I}$ in first-order structure:

(i) Domain $D_{\mathcal{I}}$

(ii) for each n-ary function symbol $f \in F$, $[\![f]\!]_{\mathcal{I}} : D_{\mathcal{I}}^n \to D_{\mathcal{I}}$

(iii) for each n-ary predicate symbol $p \in P$, $[\![p]\!]_{\mathcal{I}} : D_{\mathcal{I}}^n \to B = \{true, false\}$

(iv) ”context” (environment) for each (free) variable $x \in X$, $[\![x]\!]_{\mathcal{I}} \in D_{\mathcal{I}}$

Formula φ is closed if it has no free variables.

Definition 4.5 (Semantics)**.** Given an interpretation $\mathcal{I}$, a term t and a formula φ have the following meaning:

$$[\![t]\!]_{\mathcal{I}} \coloneqq \begin{cases} [\![x]\!]_{\mathcal{I}} & t = x \\ [\![f]\!]_{\mathcal{I}}([[\![t_1]\!]_{\mathcal{I}}, [\![t_2]\!]_{\mathcal{I}}, \ldots, [\![t_n]\!]_{\mathcal{I}}) & t = f(t_1, \ldots, t_n) \end{cases}$$

$$[\varphi]_{\mathcal{I}} \coloneqq \begin{cases} [\![x]\!]_{\mathcal{I}} & t = x \\ [\![p]\!]_{\mathcal{I}}([\![t_1]\!]_{\mathcal{I}}, [\![t_2]\!]_{\mathcal{I}}, \ldots, [\![t_n]\!]_{\mathcal{I}}) & \varphi = p(t_1, \ldots, t_n) \end{cases}$$

$$\begin{aligned} [\![\varphi_1 \Rightarrow \varphi_2]\!]_{\mathcal{I}} &\coloneqq \textbf{true} \ \ \textit{iff} \ \ [\![\varphi_1]\!]_{\mathcal{I}} = \textbf{false} \ \textit{or} \ [\![\varphi_2]\!]_{\mathcal{I}} = \textbf{true} \\ [\![\forall x.\varphi]\!]_{\mathcal{I}} &\coloneqq \textbf{true} \ \ \textit{iff for all} \ \ d \in D_{\mathcal{I}}, [\![\varphi]\!]_{I[x \to d]} = \textbf{true} \\ [\![\exists x.\varphi]\!]_{\mathcal{I}} &\coloneqq \textbf{true} \ \ \textit{iff for some} \ \ d \in D_{\mathcal{I}}, [\![\varphi]\!]_{I[x \to d]} = \textbf{true} \end{aligned}$$

Definition 4.6 (Satisfiable and Valid)**.** Let φ be a first-order formula. The formula φ is: (i)*satisfiable* iff it is true for *some* interpretation $\mathcal{I}$, i.e., $\exists \mathcal{I}. \models \varphi$. (ii) *valid* iff it is true for *all* interpretations $\mathcal{I}$, i.e., $\forall \mathcal{I}. \models \varphi$.

Definition 4.7 (Post Correspondence Problem)**.** PCP (Post Correspondence Problem): Given a finite set S of dominoes $\frac{s}{t}$ where $s, t \in \{0, 1\}^*$. Is there a finite sequence $\frac{s_1}{t_1}, \ldots \frac{s_n}{t_n}$ of (possibly repeating) dominoes from S such that the

$$s_1 \cdot s_2 \cdot \cdots \cdot s_n = t_1 \cdot t_2 \cdot \cdots t_n$$

Theorem 4.1. *The PCP problem is undecidable.*

Theorem 4.2 (Compactness)**.** *The set of formulas* Γ *is satisfiable iff every finite subset of* Γ *is satisfiable.*

Theorem 4.3 (Lowenheim-Skolem)**.** *If a set* Γ *of formulas is satisfiable then* Γ *is satisfiable by an interpretation with countable domain.*

Theorem 4.4 (Undecidability)**.** *Both the validity and satisfiability problems for FOL are undecidable.*

Proof. Let $F = \{e, f_0, f_1\}$ where e is 0-ary and f_0, f_1 are unary. With those functions we can encode the string 011 as $f_1(f_1(f_0(e)))$. Let $P = \{p\}$ where p is a binary predicate. Intuitively, the domino $\frac{s}{t}$ is represented by $p(s, t)$. Given an instance $R \coloneqq \{r_1, \ldots, r_k\}$ of PCP we define the formulas

$$\begin{aligned}
\varphi_1 &\coloneqq \bigwedge_{1 \leq i \leq k} P(s_{\mathcal{I}}(e), t_{\mathcal{I}}(e)) \\
\varphi_2 &\coloneqq \forall v, w.(p(v, w) \Rightarrow \bigwedge_{1 \leq i \leq k} p(s_{\mathcal{I}}(v), t_{\mathcal{I}}(w))) \\
\varphi_3 &\coloneqq \exists z.p(z, z)
\end{aligned}$$

It remains to be shown that the formula $\varphi_R = (\varphi_1 \wedge \varphi_2) \Rightarrow \varphi_3$ is valid iff the answer to R is yes. □

Exercise 4.1. Use PCP to show satisfiability of FOL is undecidable.

4.2 Three Sound and Complete Proof Systems

Definition 4.8 (Hilbert)**.** We extend the Hilbert system $\mathcal{H}$ by adding the following axioms

$$\begin{array}{ll}
(1) & (\forall x.\varphi) \Rightarrow \varphi[x \coloneqq t] \\
(2) & \forall x.(\varphi \Rightarrow \psi) \Rightarrow (\forall x.\varphi) \Rightarrow (\forall x.\psi) \\
(3) & \varphi \Rightarrow \forall x.\varphi \quad \text{provided } x \text{ is not free in } \varphi
\end{array}$$

We denote the safe replacement of every free variable x in φ by t as $\varphi[x \coloneqq t]$

Definition 4.9 (Genzen)**.** We extend the Genzen system $\mathcal{LK}$ by adding the following rules for the $\forall$-quantifier

$$\frac{\Gamma, \varphi[x \coloneqq t] \vdash \Delta}{\Gamma, \forall x.\varphi \vdash \Gamma}\ (\forall\text{-elim}) \qquad \frac{\Gamma \vdash \Delta, \varphi[x \coloneqq y]}{\Gamma \vdash \Delta, \forall x.\varphi}\ (\forall\text{-intro})$$

and for the $\exists$-quantifier

$$\frac{\Gamma, \varphi[x \coloneqq y]}{\Gamma, \exists x.\varphi \vdash \Delta}\ (\exists\text{-elim}) \qquad \frac{\Gamma \vdash \Delta, \varphi[x \coloneqq t]}{\Gamma \vdash \Delta, \exists x.\varphi}\ (\exists\text{-intro})$$

where y is a new (fresh) variable.

Definition 4.10 (Natural Deduction)**.** We extend the Natural Deduction system $\mathcal{NK}$ (and $\mathcal{NJ}$) by adding the following rules for the $\forall$-quantifier

$$\frac{\forall x.\varphi}{\varphi[x \coloneqq t]}\ (\forall\text{-elim}) \qquad \frac{\varphi[x \coloneqq y]}{\forall x.\varphi}\ (\forall\text{-intro})$$

and for the $\exists$-quantifier

$$\frac{\exists x.\varphi \qquad \begin{array}{c} \varphi[x \coloneqq y] \\ \vdots \\ \psi \end{array}}{\psi}\ (\exists\text{-elim}) \qquad \frac{\varphi[x \coloneqq t]}{\exists x.\varphi}\ (\exists\text{-intro})$$

where y is a new (fresh) variable.

Exercise 4.2. Prove de Morgan, i.e., $\forall x.\varphi \Leftrightarrow \neg\exists x.\neg\varphi$, for quantifiers in all three systems.

4.3 First Order Resolution: Classical Automated Theorem Proving

Definition 4.11 (First Order Resolution)**.** To proof $\models \varphi$ for a closed φ:

1. Negate. Show $\neg\varphi$ is unsat.
2. Bring $\neg\varphi$ into "prenex" form (see Remark 4.4).
3. Bring γ into CNF.
4. "Skolemization" gets rid of $\exists$ (see Remark 4.5).
5. Drop $\forall$.
6. See Theorem 4.5

Remark 4.4 (Prenex). A formula is in prenex normal all quantifiers are in the front, e.g., $\exists x.\varphi(x) \wedge \gamma \Leftrightarrow \exists x(\varphi(x) \wedge \gamma)$.

Remark 4.5 (Skolemization). Consider that while for $\exists x.\varphi(x)$ is sat iff $\varphi(\hat{x})$ is sat, we have that $\forall y \exists x \varphi(x)$ is sat iff $\varphi(f(y))$ is sat where $f(y)$ is a new function symbol. Thus,for each $\exists x$ within the scope of a universally quantified variables $y_1, \cdots, y_n$, drop $\exists x$ and replace each bound occurrence of x by $f(y_1, \cdots, y_n)$ where f is a new n-ary function symbol. E.g.

$$\forall x, \exists y \forall z_1, z_2 \exists y.\varphi(x, y, z_1, z_2, u) \to \forall x \forall z_1, z_2.\varphi(x, f(x), z_1, z_2, g(x, z_1, z_2)).$$

Theorem 4.5 (Unifier)**.** *For an interpretation $\mathcal{I}$, clauses C_1 and C_2, and literals ℓ and ℓ'. If $\mathcal{I} \models C_1[\ell]$ and $\mathcal{I} \models C_2[\neg\ell']$ and ℓ, ℓ' are "unifiable" (i.e, there is a substitution, which is a function from variables to terms, that when applied to ℓ and ℓ', makes them equal), then $\mathcal{I} \models X_1\theta[\bot] \vee C_2\theta[\bot]$ where θ is the most general unifier of ℓ and ℓ'.*

Example 4.1. For example for a variable x and a constant S, we have that $m(x)$ and $\neg m(S)$ are unifiable by replacing x by s. The purpose is to make literals the same.

4.4 First order theories

Remark 4.6. Theories defined by (i) signature (set of functions + predicate symbols) and (ii) either by a r.e. set of closed formulas called axioms A (set of all A-valid closed formulas) or by a specific "intended" interpretation $\mathcal{I}$ (true formulas in $\mathcal{I}$).

Definition 4.12. A theory T is a r.e. set of closed formulas (there are no free variables) that are closed under $\models$, i.e $T \models \varphi$ iff $\forall \mathcal{I}((\forall \psi \in T.\mathcal{I} \models \psi) \Rightarrow \mathcal{I} \models \varphi)$.

Formally,

Definition 4.13. We say that

- φ is T-valid iff $T \models \varphi$
- φ is T-satisfiable iff $\exists \mathcal{I}.\mathcal{I}$ T-model φ.
- $\mathcal{I}$ is a T-model iff $\forall \psi \in T.\mathcal{I} \models \psi$.
- φ, ψ are T-equivalent iff φ, ψ have truth value in all T-models.

Remark 4.7. Therefore, T is either (i) the set of all A-valid closed formulas or (ii) T is the set of all formulas true in $\mathcal{I}$.

Definition 4.14. The theory T is:

- *consistent* iff $\exists \mathcal{I}.\forall \psi \in T$, $\mathcal{I} \models \psi$ (always the case for (ii) in Remark 4.7);
- *complete* iff for all closed formulas ψ of the signature, either $T \models \psi$ or $T \models \neg\psi$ (always the case for (ii) in Remark 4.7);
- T is *decidable* iff the problem "given ψ, is $T \models \psi$" is decidable.

Remark 4.8. Common theories are the theory of: (i) propositional logic with quantifiers (QBF); (ii) equality with uninterpreted functions and predicates; (iii) the integers, either Peano- or Pressburger arithmetic; (iv) the reals over real closed fields; (v) the rational numbers with linear arithmatic; (vi) lists and arrays.

Definition 4.15 (T_{EQ})**.** The theory of equality and uninterpreted function + predicate symbols T_{EQ} has as signature $(=)$ and is composed of the following axioms

(i) equivalence axiom: $=$ is equivalence

(ii) congruence axiom: for every function symbol f, $\forall x_1, \cdots, x_n, y_1, \cdots, y_n$,

$$x_1 = y_1 \wedge \cdots \wedge x_n = y_n \Rightarrow f(x_1, \cdots, x_n) = f(y_1, \cdots, y_n)$$

and for every predicate p, $\forall x_1, \cdots, x_n, y_1, \cdots, y_n$,

$$x_1 = y_1 \wedge \cdots \wedge x_n = y_n \Rightarrow p(x_1, \cdots, x_n) = p(y_1, \cdots, y_n).$$

Theorem 4.6. *Satisfiability of quantifier free T_{EQ} is NP-complete. Satisfiability of quantifier free conjunction of T_{EQ} with congruence closure is $O(n \log n)$.*

Remark 4.9 (Congruence Closure). First replace predicate by function symbols, i.e., replace $p(t_1, \cdots, t_n)$ by $f_p(t_1, \cdots, t_n) = T$. Second the conjunction $\phi := (\wedge_i s_i = t_i) \wedge (\wedge_j s_j \neq t_j)$ is satisfiable iff there exists a congruence relation on terms (i.e., a relation satisfying (i) and (ii) in Definition 4.15, and for all i, $s_i = t_i$, and for all j, $s_j \neq t_j$)

Example 4.2 (Congruence Closure)**.** To show that $\phi := f^3(a) = a \wedge f^5(a) = a \wedge f(a) \neq a$ is unsatisfiable with congruence closure we follow the steps

$$\begin{aligned}
&\{a\}, \{f(a)\}, \cdots, \{f^5(a)\}\\
&\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\\
&\{\mathbf{a}, f^2(a), f^3(a), f^5(a), \mathbf{f}(\mathbf{a}), f^4(a)\}
\end{aligned}$$

Exercise 4.3. Given union find, write congruence closure (subquadratic)?

Definition 4.16 (T_{GR})**.** The theory of groups (T_{GR}) has the signature $(=, \cdot, i, -)$ and consists of the following axioms.

$$\begin{aligned}
\text{(i)} &\quad \forall x, y, z, \quad (x \cdot y) \cdot z = x \cdot (y \cdot z)\\
\text{(ii)} &\quad \forall x, \quad x \cdot i = x\\
\text{(iii)} &\quad \forall x, \quad x \cdot (-x) = i\\
\text{(iv)} &\quad \forall x, y, \quad x \cdot y = y \cdot x \quad \text{(Abelian)}.
\end{aligned}$$

Theorem 4.7. *The theory T_{GR} is incomplete and undecidable.*

Exercise 4.4. Analyze the quantifier-free T_{GP}. Is it undecidable and/or incomplete?

Definition 4.17 ($T_{\mathbb{N}}$)**.** Theory of the natural numbers ($T_{\mathbb{N}}$) has the signature $(=, 0, 1, +, \cdot)$. There are two different set of axioms, i.e., the Peano arithmetic (Definition 4.18) and Presburger arithmetic (Remark 4.10).

Definition 4.18 (T_{PE})**.** Given the signature in Definition 4.17, the theory of Peano arithmetic (T_{PE}) is defined w.r.t. the following axioms.

$$\begin{aligned}
\text{(i)} &\quad \forall x, \quad Sx \neq 0\\
\text{(ii)} &\quad \forall x, y, \quad Sx = Sy \Rightarrow x = y\\
\text{(iii)} &\quad \phi(0) \wedge \forall x(\phi(x) \Rightarrow \phi(x+1)) \Rightarrow \forall x, \phi(x)\\
\text{(iv)} &\quad \forall x, \quad x + 0 = x\\
\text{(v)} &\quad \forall x, y, \quad x + Sy = S(x + y)\\
\text{(vi)} &\quad \forall x, \quad x \cdot 0 = 0\\
\text{(vii)} &\quad \forall x, y, \quad x \cdot Sy = xy + x.
\end{aligned}$$

Theorem 4.8. *The theory T_{PE} is incomplete and undecidable, including its quantifier-free fragment (see Goedel).*

Remark 4.10 (T_{PR}). The theory of Presburger arithmetic is defined over the same signature as in Definition 4.17 is an alternative axiomatisation of the natural numbers. The axioms are omitted. No quantifier elimination (QE) is possible in this theory. Where QE means that for every formula $\forall x$, $\phi(x, y_1, \cdots, y_n)$, there exists a T-equivalent $\psi(y_1, \cdots, y_n)$.

Theorem 4.9. *The theory T_{PR} is complete and decidable in* 3EXP*. The quantifier free fragment is in* NP*.*

Exercise 4.5. Give a T_{PR} formula that has no quantifier-free equivalent.

Definition 4.19 ($T_{\mathbb{R}}$)**.** The theory of the reals ($T_{\mathbb{R}}$) has the signature $(=, 0, 1, +, -, \cdot, \leq)$ and consists of the following axioms.

$$\begin{array}{rll}
\text{G-(i)} & (+, 0); & \text{Abelian group} \\
\text{R-(i)} & (xy)z = x(yz), & \text{Ring} \\
\text{R-(ii)} & x \cdot 1 = 1 \cdot x = x, & \\
\text{R-(iii)} & (x+y)z = xz + yz, & \\
\text{R-(iv)} & x(y+z) = xy + xz; & \\
\text{F-(i)} & xy = yx, & \text{Field} \\
\text{F-(ii)} & 0 \neq 1, & \\
\text{F-(iii)} & x \neq 0 \Rightarrow \exists y. xy = 1; & \\
\text{O-(i)} & x \leq y \wedge y \leq x \Rightarrow x = y & \text{Order} \\
\text{O-(ii)} & x \leq y \wedge y \leq z \Rightarrow x \leq z, & \\
\text{O-(iii)} & x \leq y \vee y \leq x; & \\
\text{(i)} & \forall x \exists y.\ x = yy \vee -x = yy & \\
\text{(ii)} & \forall x_1, \cdots, x_n \exists y.\ y^n + x_1 y^{n-1} + \cdots + x_{n-1} y + x_n = 0 & (\text{odd } n)
\end{array}$$

Theorem 4.10. *The theory $T_{\mathbb{R}}$ is complete and decidable in* 2EXP *using cylindrical algebraic decomposition for QE.*

Definition 4.20 (T_{LI})**.** The theory of linear arithmetic (T_{LI}) has the signature $(=, 0, 1, +, -, \leq)$ and consists of the following axioms.

$$\begin{array}{rll}
\text{(i)} & \leq & \text{anti-symmetric, transitive, total} \\
\text{(ii)} & + & \text{associative, commutable, identity } 0\text{, inverse } - \\
\text{(iii)} & \multicolumn{2}{l}{\forall x\, y\, z.\ x \geq y \Rightarrow x + z \geq y + z} \\
\text{(iv)} & \multicolumn{2}{l}{\forall x.\ nx = 0 \Rightarrow x = 0 \quad \text{for all } n} \\
\text{(v)} & \multicolumn{2}{l}{\forall x\, \exists y.\ x = ny \quad \text{for all } n > 0}
\end{array}$$

Theorem 4.11. *The theory T_{LI} is complete and decidable in* 2EXP *using Fourier-Motzkin for QE.*

Example 4.3. Starting with the set of equations

$$\{x_1 - x_2 \leq 0,\ x_1 - x_3 \leq 0,\ -x_1 + x_2 + 2x_3 \leq 0,\ -x_3 \leq 1\}\,.$$

we first eliminate x_1 obtaining

$$\{x_1 \leq x_2,\ x_1 \leq x_2,\ x_1 \geq x_2 + 2x_3\}\,.$$

Next we get

$$\{2x_3 \leq 0,\ x_2 + x_3 \leq 0,\ -x_3 \leq 1\}\,.$$

Eliminating x_2 we obtain

$$\{x2 \leq -x_3,\ x_3 \leq 0,\ -x_3 \leq 1\}\,.$$

Finally eliminating x_3 to obtain $\top$.

Remark 4.11. Linear inequalities over $\mathbb{Q}$ are NP-complete, e.g., integer linear programming.

Definition 4.21 (T_{List})**.** The theory of list (T_{List}), or recursive data structures, has the signature $(=, \text{atom}, \text{car}, \text{cdr}, \text{cons})$ where: atom is a unary predicate, car and cdr are unary functions, and cons is a binary function. The theory consists of the following axioms

$$\begin{aligned}
(i) &\quad \forall x, y.\, \text{car}(\text{cons}(x, y,)) = x \\
(ii) &\quad \forall x, y.\, \text{cdr}(\text{cons}(x, y,)) = y \\
(iii) &\quad \forall x, y.\, \neg\text{atom}(\text{cons}(x, y,)) \\
(iv) &\quad \forall x.\, \neg\text{atom}(x) \Rightarrow \text{cons}(\text{car}(x), \text{cdr}(x)) = x \\
(v) &\quad \forall x.\, \text{atom}(x) \Rightarrow \varphi[x] \wedge \forall x, y.(\text{atom}(x) \wedge \underbrace{\varphi[y]}_{\text{IH}} \Rightarrow \varphi[\text{cons}(x, y)]) \Rightarrow \forall x.\varphi[x]
\end{aligned}$$

Where (i),(ii) are destructor axioms, (v) is the constructor axiom, and (vi) is the induction scheme.

Theorem 4.12. *The theory T_{List} is undecidable (decidable if quantifier-free), consistent and incomplete.*

Example 4.4. The list (1 2 3) can be represented as $(\text{cons}(1, \text{cons}(2, 3)))$

Definition 4.22 (T_{Array})**.** The theory of arrays (T_{Array}) has the signature $(=, \text{read}, \text{write}, \text{cdr}, \text{cons})$ where: read is a binary function, e.g., $\text{read}(a, i)$ reads array a at index i and returns a value; $\text{write}(a, i, x)$ writes the value x at position i in array a. The theory consists of the following axioms

$$\begin{aligned}
(i) &\quad \forall a \forall i, j \forall x.\ i = j \Rightarrow \text{read}(\text{write}(a, i, x), j) = x \\
(ii) &\quad \forall a \forall i, j \forall x.\ i \neq j \Rightarrow \text{read}(\text{write}(a, i, x), j) = \text{read}(a, j)
\end{aligned}$$

Where (i),(ii) are destructor axioms, (vi) is the constructor axiom, and (v) is the induction scheme.

Theorem 4.13. *The theory T_{List} is undecidable (decidable if quantifier-free), consistent and incomplete.*

Remark 4.12 (QBF). Quantified boolean formulas have an empty signature and no axioms. They are in PSPACE, i.e., decidable, (quantifier-free NP), and complete.

Remark 4.13 (Combination of theories). The Nelson-Oppen Algorithm can be used to combine theories. If the satisfiability of quantifier-free formulas of T_1 and T_2 are decidable, then the Nelson-Oppen Algorithm decides the satisfiability of quantifier-free formulas in the combined theory $T_1 \oplus T_2$ (the axioms of $T_1 \oplus T_2$ is the union of the respective axioms). The condition is that T_1 and T_2 are stably infinite and contain $=$.

Definition 4.23. The theory T is stable infinite if for all quantifier-free φ, if $\varphi$is satisfiable then there exists a T-interpretation $\mathcal{I}$ with inf. domain s.t. $\mathcal{I} \models \varphi$.

Remark 4.14 (Nelson-Oppen Algorithm). The Nelson-Oppen Algorithm for the combined theory $T_1 \oplus T_2$ of T_1 and T_2 has two steps:

- *Step 1: Purification* Given a formula φ in $T_1 \oplus T_2$ construct φ_1 in T_1 and φ_2 in T_2 s.t. φ is satisfiable iff $\varphi_1 \wedge \varphi_2$ is satisfiable.
- *Step 2: Equality propagation:* $\varphi_1 \wedge \varphi_2$ is satisfiable iff there exists an equivalence relation $\sim$ on the variables with characteristic formula $\psi_\sim$ s.t. $\varphi_1 \wedge \psi_\sim$ and $\varphi_2 \wedge \psi_\sim$ satisfiable

$$\psi_\sim \coloneqq \bigwedge_{x \sim y} x = y \wedge \bigwedge_{x \not\sim y} x \neq y$$

Example 4.5. Consider the combined theory of Presburger arithmetic and uninterpreted functions. Then the purification step of φ into φ_1 and φ_2 results in the following

$$\begin{aligned}
\varphi &\coloneqq (1 \leq x \leq 2 \wedge f(1) \neq f(x) \wedge f(2) \neq f(x)) \\
\varphi_1 &\coloneqq (1 \leq x \leq 2 \wedge y_1 = 1 \wedge y_2 = 2) \\
\varphi_2 &\coloneqq (f(1) \neq f(x) \wedge f(2) \neq f(x))
\end{aligned}$$

We then have the equivalence relation $\sim \coloneqq \{\{x, y_1\}, \{y_2\}\}$ and

$$\psi_\sim \coloneqq (x = y_1 \wedge x \neq y_2 \wedge y_1 \neq y_2)$$

for the equality propagation step.

5 Lambda Calculus

Remark 5.1. The term $\lambda x.t$ denotes the function that maps x to $t(x)$, e.g., $(\lambda y.\lambda x.x^2 \cdot y)(2) = \lambda x.2x^2$.

Definition 5.1 (Lambda Calculus: Syntax)**.** The syntax of Lambda calculus is given be the following grammar

$$t ::= x \mid \lambda x.t \mid t_1\, t_2$$

where x is a variable, $\lambda x.t$ is an abstraction, and $t_1\, t_2$ is an application. Application is left-associated, i.e., $(t_1\, t_2)\, t_3 = t_1\, t_2\, t_3$, and binds stronger than abstraction, i.e., $\lambda x.(t_1\, t_2) = \lambda x.t_1\, t_2$ (not to be confused with $(\lambda x.t_1)\, t_2$).

Definition 5.2 (Lambda Calculus: Operational semantics)**.** We have the axiom β-reduction, i.e.,

$$\frac{}{(\lambda x.t)s \to_\beta t[x \mapsto s]}$$

where $t[x \mapsto s]$ indicates safe substitution, and the proof rules congruence

$$\frac{t \to_\beta t'}{\lambda x.t \to_\beta \lambda x.t'} \qquad \frac{t \to_\beta t'}{t\, s \to_\beta t'\, s} \qquad \frac{t \to_\beta t'}{s\, t \to_\beta s\, t'}$$

Remark 5.2 (Nondeterministic)*.* Lambda calculus is nondeterministic, e.g.,

$$\begin{tikzcd}
& (\lambda x.y)((\lambda z.z\, z)(\lambda u, u)) \arrow[dl, "\text{call-by-name}"'] \arrow[dr, "\text{call-by-value}"] & & \\
y & & (\lambda x.y)((\lambda z.z\, z)(\lambda u, u)) \arrow[dl] \arrow[dr] & \\
& y & & (\lambda x.y)((\lambda z.z\, z)(\lambda u, u)) \\
& & & \vdots
\end{tikzcd}$$

Theorem 5.1 (Confluence)**.** *For all t, t_1, t_2 if $t \to_\beta^* t_1$ and $t \to_\beta^* t_2$ then there exists an s such that $t_1 \to_\beta^* s$ and $t_2 \to_\beta^* s$, where $\to_\beta^*$ is the reflexive, transitive closure of $\to_\beta$.*

Remark 5.3*.* The confluence theorem is also known as the Church Rosser theorem or the Diamond theorem.

$$\begin{tikzcd}
& t \arrow[dl] \arrow[dr] & \\
t_1 \arrow[dr] & & t_2 \arrow[dl] \\
& s &
\end{tikzcd}$$

Definition 5.3. The term t is irreducible (in "normal form"), if for all t', $t \not\to_\beta t'$.

Corollary 5.2. *For all t, t_1, t_2 if $t \to_\beta^* t_1$ and $t \to_\beta^* t_2$ and if t_1 and t_2 are in normal form, then $t_1 = t_2$.*

Remark 5.4. The term t can only be irreducible, if it has the form $\lambda x.t'$ with t' being irreducible, or if it has the form $t = t_1 \dots t_n$ and $t_1, \dots t_n$ are irreducible. If you start from a closed term then the only normal forms are λxt terms.

Remark 5.5. There are infinite computations of the term $(\lambda x.x\,x)(\lambda x.x\,x)$.

Definition 5.4. A term t is weakly normalizing, if some computation from t terminates. It is strongly normalizing if all computations from t terminate

Theorem 5.3 (Church)**.** *The problem whether t is (weakly) normalizing is undecidable, i.e., the halting problem for λ-calculus.*

5.1 Encoding $\mathbb{N}$ in λ (Church numerals)

Definition 5.5 ($\mathbb{B}$)**.** The booleans $\mathbb{B}$ are defined in λ-calculus as

$$\top \coloneqq \lambda x \lambda y.x = \lambda xy.x \quad \text{and} \quad \bot \coloneqq \lambda x \lambda y.y = \lambda xy.y.$$

Moreover, if is defined as if $\coloneqq \lambda bxy.bxy$.

Example 5.1. The expression if$\top st$ reduces as follows.

$$\begin{aligned} \text{if}\top st = \quad & (\lambda bxy.bxy)(\lambda xy.x)st \to (\lambda xy.(\lambda xy.x)xy)st \to (\lambda y.(\lambda xy.x)sy)t \to \\ & (\lambda xy.x)st \to (\lambda y.s)t \to s \end{aligned}$$

We can derive t from $if\ \bot st$ similarly.

Definition 5.6 ($\mathbb{N}$)**.** The natural numbers $\mathbb{N}$ are defined in λ-calculus as

$$\underline{0} \coloneqq \lambda fx.x,\ \underline{1} \coloneqq \lambda fx.fx\ ,\underline{2} \coloneqq \lambda fx.f(fx)\ ,\underline{n} \coloneqq \lambda fx.\underbrace{f(\dots f}_{n\text{times}} x)$$

The successor function is defined as $S \coloneqq \lambda nfx.f(nfx)$.

Example 5.2. The expression if$\top st$ reduces as follows.

$$\begin{aligned} S\underline{n} = \quad & (\lambda nfx.f(nfx))(\lambda fx.f^n x) \to \lambda fx.f((\lambda fx.f^n x)fx) \to \lambda fx.f((\lambda x.f^n x)x) \to \\ & \lambda fx.f(f^n x) \to \lambda fx.f^{n+1}x \to n+1 \end{aligned}$$

Definition 5.7 (Fixed point operator)**.** The expression $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ is a fixed point operator.

Theorem 5.4. *$Yt =_\beta t(Yt)$ are β equivalence. That is, there exists t' that both Yt and $t(Yt)$ can be β-reduced to t'.*

Proof. First the left side

$$Yt = \quad (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))t \to (\lambda x.t(xx))(\lambda x.t(xx)) \to t((\lambda x.t(xx))(\lambda x.t(xx))).$$

Second the right side

$$t(Yt) = \quad t((\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))t) \to t((\lambda x.t(xx))(\lambda x.t(xx))).$$

□

Example 5.3. fact $= \lambda fn.$if (iszero n)1else (mult $n(f($pred $n)))$

5.2 Combinatory logic

Definition 5.8 (Combinatory logic)**.** In combinatory logic is defined with the following combinators

$$I \coloneqq \lambda x.x \quad S \coloneqq \lambda xyz.(xz)(yz) \quad K = \lambda xy.x$$

Theorem 5.5. *Every λ term can be built from I, S, and K.*

Example 5.4.

$$\begin{aligned} K((SI)I) = \quad & (\lambda xy.x)(((\lambda xyz.(xz)(yz))(\lambda x.x))(\lambda x.x)) \to \\ & \lambda y.(((\lambda xyz.(xz)(yz))(\lambda x.x))(\lambda x.x)) \to \\ & \lambda y.((\lambda yz.((\lambda x.x)z)(yz))(\lambda x.x)) \to \\ & \lambda y \lambda z((\lambda x.x)z)((\lambda x.x)z) \to \\ & \lambda yz.zz \end{aligned}$$

Exercise 5.1. This exercise is threefold. (i) Define the λ-terms of functions iszero, mult, and pred. (ii) Evaluate fact(3) ($\to_\beta^\star$ 6). (iii) Write fact in terms of I, S, and K.

5.3 Simply typed λ

Definition 5.9 (Simply typed λ-calculus)**.** To define simply typed λ-calculus we define: types, context, terms, typing judgments, and typing rules.

- The *Types* are defined by the grammar

  $$A ::= X \mid A \to A$$

  with $A \to B$ being a right associates function from A to B (,e.g., $A \to B \to C = A \to (B \to C)$).

- The *context* $\Gamma \coloneqq \{x_1 \colon A_1, \dots x_n \colon A_n\}$ provides a type A_i to each λ-variable x_i e.g. $\Gamma(x_i) = A_i$. Note that domain$(\Gamma) = \{x_1, ..., x_n\}$.

- The terms are defined by the following grammar

  $$t ::= x \mid t_1 t_2 \mid \lambda x \colon A.t$$

- The *typing judgments* are of the form $\Gamma \vdash t \colon A$.

- There are *typing rules* three typing rules, axiom (ax), application (ap), and abstraction (ab)

$$\frac{}{\Gamma \vdash x \colon \Gamma(x)}\ \text{(ax)} \quad \frac{\Gamma \vdash s \colon A \qquad \Gamma \vdash t \colon A \to B}{\Gamma \vdash ts \colon B}\ \text{(ap)}$$

$$\frac{\Gamma[x \mapsto A] \vdash t \colon B}{\Gamma \vdash (\lambda x \colon A.t) \colon A \to B}\ \text{(ab)}$$

Remark 5.6.

Example 5.5. $(\lambda f\colon A \to A.\lambda x\colon A.f(fx))\colon (A \to A) \to A \to A$

Proof. The proof tree has the following shape

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{true}}{\{f\colon A \to A, x\colon A\} \vdash f\colon A \to A}\,(\text{ax}) \qquad \dfrac{\dfrac{\mathbf{true}}{\{f\colon A \to A, x\colon A\} \vdash f\colon A \to A} \qquad \dfrac{\mathbf{true}}{\{f\colon A \to A, x\colon A\} \vdash x\colon A}}{\{f\colon A \to A, x\colon A\} \vdash fx\colon A}}{\{f\colon A \to A, x\colon A\} \vdash (f(fx))\colon A}\,(\text{ap})}{\{f\colon A \to A\} \vdash (\lambda x\colon A.f(fx))\colon A \to A}\,(\text{ab})}{\vdash (\lambda f\colon A \to A.\lambda x\colon A.f(fx))\colon (A \to A) \to A \to A}\,(\text{ab})$$

□

Definition 5.10 (Problems)**.** The main problems are: (i) *type checking*, i.e., given Γ, t, A, is $\Gamma \vdash t\colon A$? (ii) *type inference*, i.e., given Γ, t, is t "typable" and if so, find A such that $\Gamma \vdash t\colon A$?

Theorem 5.6. *In simply typed λ, type inference is $O(|t|)$ i.e. linear time.*

Theorem 5.7 (Curry-Howard isomorphism)**.** *The Curry-Howard isomorphism (also known as correspondence, propositions as types, and proofs as programs) expresses that there is a one to one correspondence between type derivations of $\Gamma \vdash t\colon A$ and proof derivations of $\Gamma \vdash_{\mathcal{NJ}} A$.*

Definition 5.11 (β-reduction)**.** The β-reduction in simply typed λ-calculus is the same as for the untyped λ-calculus, i.e.,

$$(\lambda x\colon A.t)s \to_\beta t[x \mapsto s]$$

$(\lambda x\colon A.t)s$ is typable if $\exists B.(t\colon G \wedge s\colon A)$ (also $(\lambda x\colon A.t\colon A \to B)$)

Theorem 5.8 (Subject reduction theorem)**.** *If* $\Gamma \vdash t\colon A$ *and* $t \to_\beta t', then \Gamma \vdash t'\colon A$.

Remark 5.7. Same as cut-elimination in $\mathcal{NJ}$, i.e., cut-free proofs is equivalent to typable terms in normal form.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \qquad \dfrac{\vdots}{\Gamma \vdash A}}{\Gamma \vdash B} \quad \rightsquigarrow \quad \dfrac{\vdots}{\Gamma \vdash B}$$

Remark 5.8. Untyped λ-calculus is confluent and undecidable, i.e., all partial recursive functions are expressible. Simply typed λ-calculus is confluent and satisfies subject reduction. Regarding the expressiveness of simply typed λ over church numerals, every typable term corresponds to an extended polynomial function.

Theorem 5.9. *Every typable term is strongly normalizable.*

Corollary 5.10. *(i) Y is not typable and (ii) $=_\beta$ is decidable.*

5.4 New types

Definition 5.12. To define simply typed λ-calculus we define: types, terms, typing rules, reduction rules, and congruence rules

- The *Types* are defined by the grammar

  $$A ::= X \mid A_1 \to A_2 \mid A_1 \times A_2 \mid 1$$

  where the type system of the simply typed lambda calculus is extended by the product type and unit type (empty product) respectively.

- The terms are defined by the following grammar

  $$t ::= x \mid \lambda x \colon t \mid t_1, t_2 \mid \langle t_1, t_2 \rangle \mid \Pi_1(t) \mid \Pi_2(t) \mid \langle\rangle$$

- The *typing rules* are,

  $$\frac{}{\Gamma \vdash \langle\rangle \colon 1} \qquad \frac{\Gamma \vdash t_1 \colon A \qquad \Gamma \vdash t_2 \colon B}{\Gamma \vdash \langle t_1, t_2 \rangle \colon A \times B} \qquad \frac{\Gamma \vdash t \colon A \times B}{\begin{array}{c}\Gamma \vdash \pi_1(t) \colon A \\ \Gamma \vdash \pi_2(t) \colon B\end{array}}$$

- The *reduction rules* are,

  $$\frac{}{\pi_1(\langle t_1, t_2 \rangle) \to t_1} \qquad \frac{}{\pi_1(\langle t_1, t_2 \rangle) \to t_2}$$

- The*congruence rules* are,

  $$\frac{s \to t}{\begin{array}{c}\pi_1(s) \to \pi_1(t) \\ \pi_2(s) \to \pi_2(t) \\ \langle s, s' \rangle \to \langle t, s' \rangle \\ \langle s', s \rangle \to \langle s', t \rangle\end{array}}$$

Remark 5.9. We will have: (i) subject reduction, (ii) confluence, (iii) strong normalization, and (iv) curry-howard, i.e., → corresponds to ⇒, × corresponds to $\wedge$.

Remark 5.10 (Expressiveness over $\mathbb{N}$). Regarding the expressiveness over $\mathbb{N}$ Simply typed λ-calculus (extended polynomial functions) can be enriched by (i) still staying total, e.g., type for primitive recursion (Godel "system T") and (ii) adding fixedpoints, e.g., Y.

Definition 5.13 (Plotkin's PCF)**.** Plotkin's Programming Computable Functions (PCF) expresses all computable functions (total and partial) while losing strong normalization. They are defined w.r.t. to the grammar

$$\begin{aligned} A &::= \mathbb{N} \mid A_1 \to A_2 \\ t &::= 0 \mid Sx \mid \lambda x \colon A.t \mid t_1, t_2 \mid \text{if } t_1 \text{ then } t_2 \text{ else } t3 \mid \text{fix}_A \end{aligned}$$

and has the following typing rules

$$\frac{}{\Gamma \vdash O\colon \mathbb{N}} \qquad \frac{\Gamma \vdash x\colon \mathbb{N}}{\Gamma \vdash Sx\colon \mathbb{N}} \qquad \frac{\Gamma \vdash t\colon A \to A}{\Gamma \vdash \mathrm{fix}_A t\colon A}$$

$$\frac{\Gamma \vdash t\colon \mathbb{N} \qquad \Gamma \vdash s\colon A \qquad \Gamma \vdash s'\colon A}{\Gamma \vdash \text{if } t \text{ then } s \text{ else } s'\colon A}$$

and has the following reduction rules

$$\frac{}{\text{if } 0 \text{ then } s \text{ else } s' \to s} \qquad \frac{}{\text{if } Sx \text{ then } s \text{ else } s' \to s'} \qquad \frac{}{\mathrm{fix}_A t \to t(\mathrm{fix}_A t)}$$

Example 5.6. The factorial function can be defined as follows:

$$\mathrm{fact} \coloneqq \mathrm{fix}_{\mathbb{N}\to\mathbb{N}}(\lambda f.\lambda x.\text{ if } x \text{ then } S0 \text{ else } x \times f(x-1))$$

Exercise 5.2. Define multiplication and subtraction in Example 5.6.

5.5 Typed λ calculus ”without types”

Remark 5.11. Type inference is concerned with checking typability, i.e., i.e., given closed t, is there a type A such that $\vdash t\colon A$?, if so, find A . For typed λ-calculus ”without types” this can be checked in linear time.

Definition 5.14. Typed λ calculus ”without types” follows the grammar

$$A ::= X \mid A_1 \to A_2 \mid 1 t ::= \mid \lambda x.t \mid t_1 t_2$$

and uses the following typing rules

$$\frac{}{\Gamma \vdash x\colon \Gamma(x)} \qquad \frac{\Gamma, x\colon A \vdash t\colon B}{\Gamma \vdash \lambda x.t\colon A \to B} \qquad \frac{\Gamma \vdash t\colon A \to B \qquad \Gamma \vdash s\colon A}{\Gamma \vdash ts\colon B}$$

Remark 5.12. In this type system terms can have multiple types, e.g.,

$$\lambda x.x\colon 1 \to 1 \quad \text{and} \quad \lambda x.x\colon (1 \to 1) \to 1 \to 1$$

However, there is always a *most general* type called the principal type, e.g., $\lambda x.x\colon X \to X$.

Remark 5.13 (Substitution on types). We denote $\sigma \coloneqq A[X \mapsto B]$ substitution on types. If $\vdash t\colon A$ then for all substitutions σ, $\vdash t\colon A\sigma$.

Theorem 5.11 (Principal type)**.** *If $\vdash t\colon A$, then there exists B such that for all B' is $\vdash t\colon B'$ then exists σ such that $B' = B\sigma$.*

Remark 5.14 (cont. Thrm 5.11). To find B, collect “=”-constraints on types. In particular, type variables are equal, i.e. $A = A$”, if they can be unified, i.e., $\exists \sigma, \sigma'\colon A\sigma = A'\sigma'$. Judgments can be used to add “=”-constraints on types to typing rules, i.e, $\Gamma \vdash t\colon A \mid E$ where E is a set of equations on types and for Y being free

$$\frac{\Gamma \vdash t\colon A|E \qquad \Gamma \vdash s\colon B|F}{\Gamma \vdash ts|E \cup F \cup \{A = B \to Y\}}$$

This gives you an algorithm for computing principal types.

Definition 5.15 (Polymorphic types)**.** Polymorphic types, i.e., $\lambda x.x\colon \forall X.X \to X$, are the simplest form is Hinolley-Milner types. They follow the grammar

$$A ::= X \mid A \to A \mid \forall X.A$$

6 IMP

Remark 6.1. IMP is a simple imperative language corresponding to PCF.

Definition 6.1 (IMP Syntax)**.** There are three types of *expressions* in IMP, i.e., arithmetic, boolean, and command expressions. Programs in IMP belong to the following grammar:

$$\begin{aligned} a &::= n \mid x \mid a + a \mid a - a \mid a * a \mid Sa \mid 0 \\ b &::= \mathbf{true} \mid \mathbf{false} \mid a \leq a \mid \neg b \mid b \wedge b \\ c &::= \mathtt{skip} \mid x = a \mid c; c \mid \text{if } b \text{ then } c \text{ else } c \mid \text{while } b \text{ do } c \end{aligned}$$

Definition 6.2 (IMP Semantics)**.** There are three different types of semantics for the IMP programming language. All consider the set of variables X as locations, the define a state as the function $\sigma \colon X \to \mathbb{Z}$. We use $\perp$ to denote the *undefined* state (e.g., for a non-terminating program). Moreover, we denote the set of all states (including $\perp$) as Σ ($\Sigma_\perp$).

6.1 Operational Semantics

Definition 6.3 (Operational Semantics)**.** The operational semantics for IMP was originally introduced as structured operational semantics (SOS) by Plotkin. It is defined by a set of rules. We use $\langle c, \sigma \rangle \downarrow \sigma' \in \Sigma_\perp$ to denote that, starting with state σ, a command expression c takes us to state σ'. $\langle a, \sigma \rangle \downarrow n \in \mathbb{Z}$ and $\langle b, \sigma \rangle \downarrow v \in \mathbb{B}$ are defined similarly for arithmetic and Boolean expressions. The judgements of the operational semantics are as follows:

(i) for *arithmetic expressions*, note that the same structure can be used for judgements on subtraction and multiplication.

$$\frac{}{\langle \mathtt{n},\ \sigma \rangle \downarrow n} \qquad \frac{}{\langle \mathtt{x},\ \sigma \rangle \downarrow \sigma(x)} \qquad \frac{\langle \mathtt{a}_1,\ \sigma \rangle \downarrow n_1 \qquad \langle \mathtt{a}_2,\ \sigma \rangle \downarrow n_2}{\langle \mathtt{a}_1 + \mathtt{a}_2,\ \sigma \rangle \downarrow n_1 + n_2}$$

(ii) for *boolean expressions*, note that the same structure can be used for completing the set of judgements on Boolean operators.

$$\frac{}{\langle \mathtt{true},\ \sigma \rangle \downarrow \top} \qquad \frac{\langle \mathtt{b}_1,\ \sigma \rangle \downarrow \top \qquad \langle \mathtt{b}_2,\ \sigma \rangle \downarrow \top}{\langle \mathtt{b}_1 \wedge \mathtt{b}_2,\ \sigma \rangle \downarrow \top}$$

$$\frac{\langle \mathtt{a}_1,\ \sigma \rangle \downarrow n_1 \qquad \langle \mathtt{a}_2,\ \sigma \rangle \downarrow n_2 \qquad n_1 \leq n_2}{\langle \mathtt{a}_1 \leq \mathtt{a}_2,\ \sigma \rangle \downarrow \top}$$

(iii) for *command expressions.* In the expression below, note that $\perp$ in $\langle \mathtt{b},\ \sigma \rangle \downarrow \perp$ is the Boolean false. The other case for if b then c_1 else c_2 is relatively easy to work out.

$$\frac{}{\langle \mathtt{skip},\ \sigma \rangle \downarrow \sigma} \qquad \frac{\langle \mathtt{a},\ \sigma \rangle \downarrow n}{\langle \mathtt{x} = \mathtt{a},\ \sigma \rangle \downarrow \sigma[x \mapsto n]} \qquad \frac{\langle \mathtt{b},\ \sigma \rangle \downarrow \perp}{\langle \text{while } \mathtt{b} \text{ do } \mathtt{c},\ \sigma \rangle \downarrow \sigma}$$

$$\frac{\langle \mathtt{b},\ \sigma\rangle \Downarrow \top \qquad \langle \mathtt{c},\ \sigma\rangle \Downarrow \sigma' \qquad \langle \text{while } \mathtt{b} \text{ do } \mathtt{c},\ \sigma'\rangle \Downarrow \sigma''}{\langle \text{while } \mathtt{b} \text{ do } \mathtt{c},\ \sigma\rangle \Downarrow \sigma''}$$

$$\frac{\langle \mathtt{c}_1,\ \sigma\rangle \Downarrow \sigma' \qquad \langle \mathtt{c}_2,\ \sigma'\rangle \Downarrow \sigma''}{\langle \mathtt{c}_1; \mathtt{c}_2,\ \sigma\rangle \Downarrow \sigma''} \qquad \frac{\langle \mathtt{b},\ \sigma\rangle \Downarrow \top \qquad \langle \mathtt{c}_1,\ \sigma\rangle \Downarrow \sigma'}{\langle \text{if } \mathtt{b} \text{ then } \mathtt{c}_1 \text{ else } \mathtt{c}_2,\ \sigma\rangle\rangle \Downarrow \sigma'}$$

Remark 6.2. What we have defined above is *multi-step* operational semantics. We can also define it as rules over *single* steps; however, there are many versions of single-step semantics, depending on the *step size* we pick. For single-step semantics, we use $\langle \mathtt{c},\ \sigma\rangle \rightarrow \langle \mathtt{c'},\ \sigma'\rangle$ to denote that, starting with state σ, a command expression $\mathtt{c}$ takes us to state σ' *after one step*, and program $\mathtt{c'}$ is yet to be executed.

Example 6.1. An example of rules in are single-step semantics are as follows.

$$\frac{\langle \mathtt{a},\ \sigma\rangle \Downarrow n}{\langle \mathtt{x} \coloneqq \mathtt{a},\ \sigma\rangle \rightarrow \langle \mathtt{skip},\ \sigma[x \mapsto a]\rangle} \qquad \frac{\langle \mathtt{c}_1,\ \sigma\rangle \rightarrow \langle \mathtt{skip},\ \sigma'\rangle}{\langle \mathtt{c}_1; \mathtt{c}_2,\ \sigma\rangle \rightarrow \langle \mathtt{c}_2,\ \sigma'\rangle}$$

$$\frac{\langle \mathtt{c}_1,\ \sigma\rangle \rightarrow \langle \mathtt{c}_1',\ \sigma'\rangle \qquad \mathtt{c}_1' \neq \mathtt{skip}}{\langle \mathtt{c}_1; \mathtt{c}_2,\ \sigma\rangle \rightarrow \langle \mathtt{c}_1'; \mathtt{c}_2,\ \sigma'\rangle}$$

Example 6.2. For the program $\mathtt{c} = \mathtt{x} \coloneqq \mathtt{0}; \mathtt{x} \coloneqq \mathtt{1}$, we have:

$$\begin{aligned}
\langle \mathtt{x} \coloneqq \mathtt{0},\ \sigma\rangle &\rightarrow \langle \mathtt{skip},\ \sigma[x \mapsto 0]\rangle \\
\langle \mathtt{x} \coloneqq \mathtt{0}; \mathtt{x} \coloneqq \mathtt{1},\ \sigma\rangle &\rightarrow \langle \mathtt{x} \coloneqq \mathtt{1},\ \sigma[x \mapsto 0]\rangle \\
&\rightarrow \langle \mathtt{skip},\ \sigma[x \mapsto 1]\rangle
\end{aligned}$$

6.2 Denotational Semantics

Definition 6.4 (Denotational Semantics)**.** We define functions $\llbracket \mathtt{a} \rrbracket : \Sigma \rightarrow \mathbb{Z}$ and $\llbracket \mathtt{b} \rrbracket : \Sigma \rightarrow \mathbb{B}$ as semantic functions for arithmetic and boolean expressions. For command expressions, we define $\llbracket \mathtt{c} \rrbracket : \Sigma \rightarrow \Sigma_\perp$ (alternatively define $\llbracket \mathtt{c} \rrbracket$ as the *partial* function $\Sigma \rightharpoonup \Sigma$). Similar to operational semantics, we define denotational semantic functions for different types of expressions in IMP:

1. Arithmetic expressions:

   $$\begin{aligned}
   \llbracket \mathtt{n} \rrbracket &\coloneqq \lambda\sigma.\ n \\
   \llbracket \mathtt{x} \rrbracket &\coloneqq \lambda\sigma.\ \sigma(x) \\
   \llbracket \mathtt{a}_1 + \mathtt{a}_2 \rrbracket &\coloneqq \lambda\sigma.\ \llbracket \mathtt{a}_1 \rrbracket + \llbracket \mathtt{a}_2 \rrbracket
   \end{aligned}$$

   Notice the difference between the syntactic + in $\llbracket \mathtt{a}_1 + \mathtt{a}_2 \rrbracket$ and the semantic one $\lambda\sigma.\ \llbracket \mathtt{a}_1 \rrbracket + \llbracket \mathtt{a}_2 \rrbracket$ the syntactic + and the semantic '+' operators.

2. Boolean expressions:

   $$\begin{aligned}
   \llbracket \mathtt{true} \rrbracket &\coloneqq \lambda\sigma.\ \top \\
   \llbracket \mathtt{a}_1 \leq \mathtt{a}_2 \rrbracket &\coloneqq \lambda\sigma.\ \llbracket \mathtt{a}_1 \rrbracket\sigma \leq \llbracket \mathtt{a}_2 \rrbracket\sigma \\
   \llbracket \mathtt{b}_1 \land \mathtt{b}_2 \rrbracket &\coloneqq \lambda\sigma.\ \llbracket \mathtt{b}_1 \rrbracket\sigma \land \llbracket \mathtt{b}_2 \rrbracket\sigma
   \end{aligned}$$

   Again, notice the difference between the syntactic and semantic $\land$.

3. Command expressions:

$$\begin{aligned}
[\![\mathtt{skip}]\!] &\coloneqq \lambda\sigma.\ \sigma \\
[\![\mathtt{x} \coloneqq \mathtt{a}]\!] &\coloneqq \lambda\sigma.\ \sigma[x \mapsto [\![\mathtt{a}]\!]\sigma] \\
[\![\mathtt{c}_1; \mathtt{c}_2]\!] &\coloneqq \lambda\sigma.\ \text{if } [\![\mathtt{c}_1]\!]\sigma = \bot \text{ then } \bot \text{ else } [\![\mathtt{c}_2]\!]([\![\mathtt{c}_1]\!]\sigma) \\
[\![\text{if } \mathtt{b} \text{ then } \mathtt{c}_1 \text{ else } \mathtt{c}_2]\!] &\coloneqq \lambda\sigma.\ \text{if } [\![\mathtt{b}]\!]\sigma = \top \text{ then } [\![\mathtt{c}_1]\!]\sigma \text{ else } [\![\mathtt{c}_2]\!]\sigma
\end{aligned}$$

The case for a while a do b command is more elaborate: its semantic function is defined as the *least fixed point* of a higher order function F, which takes a program $f : \Sigma \to \Sigma_\bot$ and a state σ, and returns a state $\sigma' \in \Sigma_\bot$. We formally define function F as follows:

$$F(f) = \lambda\sigma.\ \text{if } [\![\mathtt{b}]\!]\sigma = \top \text{ then (if } [\![\mathtt{c}]\!]\sigma = \bot \text{ then } \bot \text{ else } f([\![\mathtt{c}]\!]\sigma)) \text{ else } \sigma$$

Exercise 6.1. We can approximate the lfp of a function F by constructing the sequence $F(\bot), F^2(\bot), \ldots$. For the factorial program, first *define* a program $\hat{\bot}$, then calculate $\hat{F}(\hat{\bot}), \hat{F}^2(\hat{\bot}), \ldots$, where $\hat{F}$ is the semantic function of the while a do b command in factorial.

The following theorem connects operational and denotational semantics:

Theorem 6.1 (Connect Operational and Denotational Semantic)**.** *The following equivalences hold over expressions in IMP:*

$$\begin{aligned}
&\forall \mathtt{a}, n, \sigma\ .\ \langle \mathtt{a},\ \sigma\rangle \downarrow n && \iff [\![\mathtt{a}]\!]\sigma = n \\
&\forall \mathtt{b}, n, \sigma\ .\ \langle \mathtt{a},\ \sigma\rangle \downarrow v && \iff [\![\mathtt{b}]\!]\sigma = v \\
&\forall \mathtt{c}, \sigma, \sigma'.\ \langle \mathtt{c},\ \sigma\rangle \downarrow \sigma' && \iff [\![\mathtt{c}]\!]\sigma = \sigma' \qquad (*) \\
&\forall \mathtt{c}, \sigma\quad .\ (\neg\exists\sigma'.\ \langle \mathtt{c},\ \sigma\rangle \downarrow \sigma') && \iff [\![\mathtt{c}]\!]\sigma = \bot
\end{aligned}$$

Arithmetic, Boolean, and command expressions are denoted with $\mathtt{a}$, $\mathtt{b}$, *and* $\mathtt{c}$.

Exercise 6.2 (Optional)**.** Prove (*) in the theorem above. You can assume the equivalences for arithmetic and Boolean expressions.

6.3 Axiomatic Semantics (Hoare Logic)

Definition 6.5 (Notation)**.** Let $\mathtt{c}$ be an IMP program, let $\sigma \in \Sigma$ is a state, and $\mathcal{I} : Y \mapsto \mathbb{Z}$ is a mapping of free variables to integers, then

$$\sigma \models^{\mathcal{I}} \{\varphi\}\ \mathtt{c}\ \{\psi\}$$

denotes that the program $\mathtt{c}$ satisfies the assertions in $\{\psi\}$, if the assertions in $\{\varphi\}$ are satisfied before execution. Each of $\{\varphi\}$ and $\{\psi\}$ are a set of *assertions* ($\sim$ first-order logic formulas) over the set of program locations X, the set of free variables Y, and at least all functions and predicate symbols that occur in arithmetic and Boolean expressions of IMP.

Definition 6.6 (Partial Correctness)**.** The expression $\models^{\mathcal{I}}$ is defined as follows. $\sigma \models^{\mathcal{I}} \{\varphi\}\ \mathtt{c}\ \{\psi\}$ if and only if for any σ', if $\langle \mathtt{c},\ \sigma\rangle \downarrow \sigma'$ and $\sigma \models^{\mathcal{I}} \varphi$, then $\sigma' \models^{\mathcal{I}} \psi$. Moreover, we write $\models \{\varphi\}\ \mathtt{c}\ \{\psi\}$ iff $\forall \sigma, I.\ \sigma \models^{I} \{\varphi\}\ \mathtt{c}\ \{\psi\}$

Definition 6.7 (Axiomatic Semantics)**.** Similar to other proof systems, axiomatic semantics for IMP is a set of inference rules.

1. Skip and assignment command

$$\frac{}{\{\varphi\}\ \mathtt{skip}\ \{\varphi\}} \qquad \frac{\varphi[x \mapsto a]}{\mathtt{x} := \mathtt{a}\ \{\varphi\}}$$

2. Command expressions

$$\frac{\{\varphi\}\ \mathtt{c}_1\ \{\psi\} \qquad \{\psi\}\ \mathtt{c}_2\ \{\chi\}}{\{\varphi\}\ \mathtt{c}_1;\mathtt{c}_2\ \{\chi\}} \qquad \frac{\{\varphi \wedge b\}\ \mathtt{c}_1\ \{\psi\} \qquad \{\varphi \wedge \neg b\}\ \mathtt{c}_2\ \{\psi\}}{\{\varphi\}\ \text{if } \mathtt{b} \text{ then } \mathtt{c}_1 \text{ else } \mathtt{c}_2\ \{\psi\}}$$

3. Loop invariant

$$\frac{\{\varphi \wedge b\}\ \mathtt{c}\ \{\varphi\}}{\{\varphi\}\ \text{while } \mathtt{b} \text{ do } \mathtt{c}\ \{\varphi \wedge \neg b\}}$$

4. The consequence rule, it shows how we can *strengthen* our pre-conditions and *weaken* our post-conditions:

$$\frac{\varphi \Rightarrow \varphi' \qquad \{\varphi'\}\ \mathtt{c}\ \{\psi'\} \qquad \psi' \Rightarrow \psi}{\{\varphi\}\ \mathtt{c}\ \{\psi\}}$$

Remark 6.3. Be careful with the pre- and post-conditions of the assignment command.

Exercise 6.3. For the factorial program `fact`, prove the following using Hoare logic:$\{x = i\}$ `fact` $\{y = i!\}$

Theorem 6.2 (Soundness)**.** *If* $\vdash \{\varphi\}\ \mathtt{c}\ \{\psi\}$ *then* $\models \{\varphi\}\ \mathtt{c}\ \{\psi\}$ *s.t.*

$$\forall \sigma\sigma'\forall \mathcal{I},\ \sigma \models^{\mathcal{I}} \varphi \wedge \langle \mathtt{c},\ \sigma\rangle \downarrow \sigma' \Rightarrow \sigma' \models^{\mathcal{I}} \psi \qquad \textit{partial correctness,}$$
$$\forall \sigma\forall \mathcal{I},\ \sigma \models^{\mathcal{I}} \varphi \Rightarrow \exists \sigma', \langle \mathtt{c},\ \sigma\rangle \downarrow \sigma' \wedge \sigma' \models^{\mathcal{I}} \psi \qquad \textit{total correctness.}$$

Theorem 6.3 (Relative Completeness)**.** *Assume we have an oracle for deciding the validity of assertions* $\models \chi$*. If* $\models \{\varphi\}\ \mathtt{c}\ \{\psi\}$ *then* $\vdash \{\varphi\}\ \mathtt{c}\ \{\psi\}$*.*

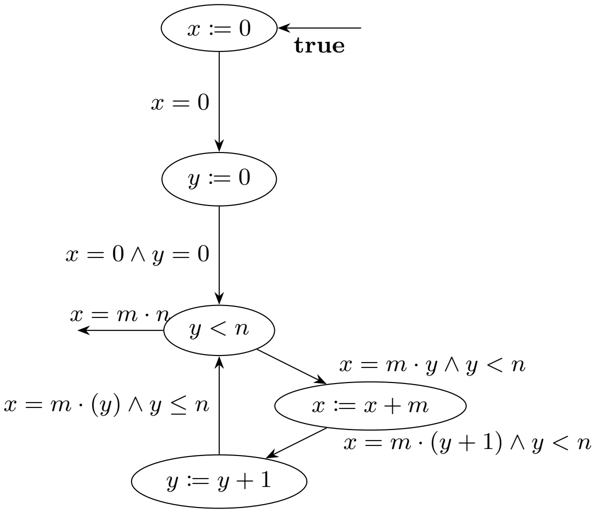
Remark 6.4. The following issues arise: (i) $\models \{\textbf{true}\}\ \mathtt{c}\ \{\textbf{false}\}$ iff `c` does not terminate; (ii) $\models \{\textbf{true}\}\ \mathtt{skip}\ \{\psi\}$ iff $\models \psi$, but according to Goedel theorem there is no complete proof system for $(\mathbb{N}, +, *)$.

6.4 Verification Conditions

Example 6.3 (Annotated Program)**.** From the annotations shown in Algorithm 1 we can derive the verification conditions which are the assertions that need to be valid for the Hoare proof:

- $\textbf{true} \Rightarrow 0 = 0.$
- $x = 0 \Rightarrow x = 0 \land 0 = 0.$
- $x = 0 \land y = 0 \Rightarrow x = m * y \land y \leq n.$
- $x = m * y \land y \leq n \land y \geq n \Rightarrow x = m * n.$
- $x = m * y \land y \leq n \land y < n \Rightarrow x = m * y \land y < n.$
- $x = m * y \land y < n \Rightarrow x + m = m * (y + 1) \land y < n.$
- $x = m * (y + 1) \land y < n \Rightarrow x = m * (y + 1) \land (y + 1) \leq n.$

This can be represented using a *Flow Chart* (Control Flow Graph - "Floyd Style"). Below is such a chart for this example



Definition 6.8 (General Verification Conditions)**.** In general, we can derive verification conditions using the following rules:

- $\mathrm{vc}(\{\varphi\}\ \texttt{skip}\ \{\psi\}) = \{\varphi \Rightarrow \psi\}.$
- $\mathrm{vc}(\{\varphi\}\ \texttt{x} = \texttt{a}\ \{\psi\}) = \{\varphi \Rightarrow \psi[x \mapsto a]\}.$
- $\mathrm{vc}(\{\varphi\}\ \texttt{c}_1\ \{\chi\}\ \texttt{c}_2\ \{\psi\}) = \mathrm{vc}(\{\varphi\}\ \texttt{c}_1\ \{\chi\}) \cup \mathrm{vc}(\{\chi\}\ \texttt{c}_2\ \{\psi\}).$
- $\mathrm{vc}(\{\varphi\}\ \text{if } \texttt{b} \text{ then } \texttt{c}_1 \text{ else } \texttt{c}_2\ \{\psi\}) = \mathrm{vc}(\{\varphi \land b\}\ \texttt{c}_1\ \{\psi\}) \cup \mathrm{vc}(\{\varphi \land \neg b\}\ \texttt{c}_2\ \{\psi\}).$
- $\mathrm{vc}(\{\varphi\}\text{while } \texttt{b} \text{ do } \texttt{c}\{\psi\}) = \{\varphi \Rightarrow \chi, \chi \land \neg b \Rightarrow \psi\} \cup \mathrm{vc}(\{\chi \land b\}\ \texttt{c}\ \{\chi\}).$

Algorithm 1 A program that computes $m * n$

```
$\{\textbf{true}\}$
$x \coloneqq 0$
$\{x = 0\}$
$y \coloneqq 0$
$\{x = 0 \land y = 0\}$
while $y \leq n$ do                          ▷ loop invariant: $\{x = m * y \land y \leq n\}$
    $\{x = m * y \land y < n\}$
    $x \coloneqq x + m$
    $\{x = m * (y + 1) \land y < n\}$
    $y \coloneqq y + 1$
end while
$\{x = m * n\}$
```

6.5 Programs as predicate transforming (Dijkstra): Weakest (liberal) preconditions

Definition 6.9 (Weakest Liberal Preconditions)**.** $\mathrm{wlp}(\mathtt{c}, \psi)$ is the *weakest* φ s.t $\{\varphi\}\ \mathtt{c}\ \{\psi\}$, i.e., for all φ with $\{\varphi\}\ \mathtt{c}\ \{\psi\}$ we have $\varphi \Rightarrow \mathrm{wlp}(\mathtt{c}, \psi)$.

Definition 6.10 (Weakest Preconditions)**.** $\mathrm{wp}(\mathtt{c}, \psi)$ is the *weakest* φ s.t $\{\varphi\}\ \mathtt{c}\ \{\psi\} \land \forall \sigma$, if $\sigma \models \varphi$ then $\mathtt{c}$ terminates from σ.

Definition 6.11. An assertion language is *expressive* if $\mathrm{wlp}(\mathtt{c}, \psi) = \{\sigma \in \Sigma \mid \langle \mathtt{c},\ \sigma\rangle = \bot \lor \forall \mathcal{I}. [\![\mathtt{c}]\!]\sigma \models_{\mathcal{I}} \psi\}$.

Remark 6.5. For IMP, arithmetic $(\mathbb{N}, +, *)$ is expressive (proof by Goedelization of program executions).

Example 6.4. A few examples:

$$\begin{aligned}
\mathrm{wlp}(\mathtt{skip}, \psi) &= \psi \\
\mathrm{wlp}(\mathtt{x} = \mathtt{a}, \psi) &= \psi[x \mapsto a] \\
\mathrm{wlp}(\mathtt{c}_1, \mathtt{c}_2, \psi) &= \mathrm{wlp}(\mathtt{c}_1, \mathrm{wlp}(\mathtt{c}_1, \psi)) \\
\mathrm{wlp}(\text{if } \mathtt{b} \text{ then } \mathtt{c}_1 \text{ else } \mathtt{c}_2, \psi) &= (b \Rightarrow \mathrm{wlp}(\mathtt{c}_1, \psi)) \land (\neg b \Rightarrow \mathrm{wlp}(\mathtt{c}_2, \psi)) \\
&= (b \land \mathrm{wlp}(\mathtt{c}_1, \psi)) \lor (\neg b \land \mathrm{wlp}(\mathtt{c}_2, \psi)) \\
\mathrm{wlp}(\text{while } \mathtt{b} \text{ do } \mathtt{c}, \psi) &= \text{weakest } \chi : \chi \land b \Rightarrow \mathrm{wlp}(\mathtt{c}, \chi) \land \chi \land \neg b \Rightarrow \psi \\
\mathrm{wp}(\text{while } \mathtt{b} \text{ do } \mathtt{c}, \psi) &= \text{weakest } \chi : \chi \land b \land (e = n) \Rightarrow \mathrm{wlp}(\mathtt{c}, \chi \land e > n) \\
&\qquad \land \chi \land \neg b \Rightarrow \psi \\
&\qquad \land \chi \Rightarrow e \in \text{well-founded set}
\end{aligned}$$

where n is a fresh variable.

Definition 6.12 (`assert` and `skip`)**.** We define `assume` and `skip` using various semantics.

$$\begin{aligned}[\![\texttt{skip}]\!] &:- \lambda\sigma.\sigma \\ [\![\texttt{assert b}]\!] &:- \lambda\sigma.\text{if } \texttt{b} \text{ then } \sigma \text{ else } \texttt{fail}\end{aligned}$$

and

$$\frac{\langle \texttt{b}, \sigma\rangle \downarrow \top}{\langle \texttt{assert b}, \sigma\rangle \downarrow \sigma} \qquad \frac{\langle \texttt{b}, \sigma\rangle \downarrow \bot}{\langle \texttt{assert b}, \sigma\rangle \downarrow \texttt{fail}}$$

and

$$\begin{aligned}\text{wlp}(\texttt{skip}, \psi) &\coloneqq \psi \\ \text{wlp}(\texttt{assert b}, \psi) &\coloneqq \texttt{b} \wedge \psi\end{aligned}$$

and

$$\frac{}{\{\texttt{b} \wedge \psi\}\texttt{assert b }\{\psi\}}$$

Moreover, we can define

$$[\![\texttt{c}]\!] \colon \Sigma \to \Sigma_\bot, \quad [\![\texttt{c}]\!] \colon \Sigma \to \Sigma_\bot^{\texttt{fail}} \quad \text{or} \quad [\![\texttt{c}]\!] \colon \Sigma \times \Sigma_\bot^{\texttt{fail}}$$

for $\Sigma_\bot^{\texttt{fail}} :- \Sigma_{\{\texttt{fail},\bot\}}$.

Remark 6.6. In nondeterministic settings, the symbol $\downarrow$ means “can reach”.

Remark 6.7. The relational semantics generalizes to non-determinism

Definition 6.13 (`assume`)**.** We define `assume` using various semantics.

$$[\![\texttt{assume b}]\!] :- \{(\sigma, \sigma') \mid \sigma \models \texttt{b} \wedge \sigma' = \sigma\}$$

and

$$\text{wlp}(\texttt{assume b}, \psi) \coloneqq (b \Rightarrow \psi)$$

and

$$\frac{}{\{\texttt{b} \Rightarrow \psi\}\texttt{assume b }\{\psi\}}$$

Definition 6.14.

$$\text{wlp}(\texttt{c}, \psi) :- \{\sigma \mid \forall \sigma'.(\sigma, \sigma') \in [\![\texttt{c}]\!] \Rightarrow \sigma' \models \psi\}$$

Definition 6.15. If the assert is false we define

$$[\![\texttt{abort}]\!] :- \lambda\sigma.\texttt{fail} \quad \text{and} \quad [\![\texttt{havoc b}]\!] :- \{(\sigma, \sigma') \mid \sigma' \models \texttt{b}\}$$

6.6 Disjkstra's guarded commands

Definition 6.16 (Syntax)**.** Disjkstra's guarded commands are defined by the following grammar

$$\begin{aligned} \texttt{gc} &::= \overbrace{\texttt{b} \to \texttt{c}}^{\texttt{b}\text{ guard}} \mid \overbrace{\texttt{gc} \,\square\, \texttt{gc}}^{\square\ \text{alternatively}} \\ \texttt{c} &::= \texttt{skip} \mid \texttt{abort} \mid \texttt{x} := \texttt{a} \mid \texttt{c};\texttt{c} \mid \text{if } \texttt{gc} \text{ fi} \mid \text{do } \texttt{gc} \text{ od} \end{aligned}$$

Example 6.5. Intuitively: We say $\texttt{b}$ is a "guard" and we say □ means "alternatively". For example: $x \leq 5 \to x :- x+1 \,\square\, x \geq 5 \to x :- x-1$

Definition 6.17 (Rules)**.** The behavior of Disjkstra's guarded commands is given by the following inference rules.

$$\frac{\langle \texttt{b}, \sigma\rangle \downarrow \top \qquad \langle \texttt{c}, \sigma\rangle \downarrow \sigma'}{\langle \texttt{b} \to \texttt{c}, \sigma\rangle \downarrow \sigma'} \qquad \frac{\langle \texttt{b}, \sigma\rangle \downarrow \bot}{\langle \texttt{b} \to \texttt{c}, \sigma\rangle \downarrow \texttt{fail}}$$

$$\frac{\langle \texttt{gc}_1, \sigma\rangle \downarrow \sigma'}{\langle \texttt{gc}_1 \vee \texttt{gc}_2, \sigma\rangle \downarrow \sigma'}$$

$$\frac{\langle \texttt{gc}_1, \sigma\rangle \downarrow \texttt{fail} \qquad \langle \texttt{gc}_2, \sigma\rangle \downarrow \texttt{fail}}{\langle \texttt{gc}_1 \vee \texttt{gc}_2, \sigma\rangle \downarrow \texttt{fail}}$$

$$\frac{\langle \texttt{gc}, \sigma\rangle \downarrow \sigma'}{\langle \text{if } \texttt{gc} \text{ fi}, \sigma\rangle \downarrow \sigma'} \qquad \frac{\langle \texttt{gc}, \sigma\rangle \downarrow \texttt{fail}}{\langle \text{if } \texttt{gc} \text{ fi}, \sigma\rangle \downarrow \texttt{fail}}$$

$$\frac{\langle \texttt{gc}, \sigma\rangle \downarrow \texttt{fail}}{\langle \text{do } \texttt{gc} \text{ od}, \sigma\rangle \downarrow \sigma} \qquad \frac{\langle \texttt{gc}, \sigma\rangle \downarrow \sigma' \qquad \langle \text{do } \texttt{gc} \text{ od}, \sigma'\rangle \downarrow \sigma''}{\langle \text{do } \texttt{gc} \text{ od}, \sigma\rangle \downarrow \sigma''}$$

Example 6.6. Euclid(gcd): do $x > y \to x := x - y \,\square\, y > x \to y := y - x$ od

Definition 6.18. We define the grammar

$$\texttt{c} ::= \texttt{skip} \mid \texttt{x} := \texttt{a} \mid \texttt{c};\texttt{c} \mid \underbrace{\texttt{b}?}_{\text{assume } \texttt{b}} \mid \underbrace{\texttt{c}_1 + \texttt{c}_2}_{\text{alternative}} \mid \underbrace{\texttt{c}^*}_{\text{iterate 0 or more}}$$

Example 6.7. We can define while $\texttt{b}$ do $\texttt{c}$ by $(\texttt{b}?;\texttt{c})^* + (\neg\texttt{b}?)$.

Exercise 6.4. Give the operational rules, the wlp definition (or the Hoare rules), and the denotational definition of $[\![\texttt{c}]\!]$. Moreover, what is F in $[\![\texttt{c}^*]\!] = \text{lfp}F$.

7 Parallelism

Example 7.1. The expression $x := 0 \parallel x := 1$ is either $(x := 0;\ x := 1)$ or $(x := 1;\ x := 0)$

Remark 7.1. Parallelism as nondeterministic interleaving of atomic actions (programs): (i) nondeterministic; (ii) noncompositional; (iii) nonterminating "reactive programs". The study of parallelism falls under concurrency theory and process algebra. Here the operator $\parallel$ is added to languages. There were dozens of process algebras in the 1990s.

7.1 CCS (Calculus of Communicating Systems) by Milner

Definition 7.1 (Syntax)**.** We talk about: processes P, Q; nil 0 (like skip); atomic actions $a, b, \ldots, \in A$; prefix $a.P$ (like sequencing); alternative $P + Q$; process definitions $C \hat{=} P$; parallelism $P \parallel Q$; restriction: $(\nu a)P$;

Example 7.2. We can formulate constructs such as Iterate $= a.$Iterate $+ b0$

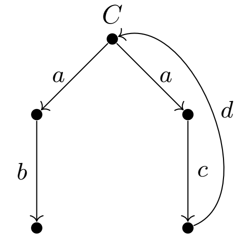
Definition 7.2 (Structured Operational Semantics)**.** The (single-step) structured operational semantics (SOS) of CCS is given by the rules

$$\frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'} \qquad \frac{P \xrightarrow{a} P'}{C \xrightarrow{a} P'}\, C \hat{=} P$$

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q} \qquad \frac{Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P \parallel Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \frac{P \xrightarrow{b} P'}{(\nu a)P \xrightarrow{b} (\nu a)P'}\, a \neq b,\ \bar{a} \neq b$$

Example 7.3. Sequential Processes are action labeled directed graphs, i.e., a labeled transition system. For example $c \hat{=} ab0 + acdC$ corresponds to the graph



Definition 7.3. Let A be a set of atomic actions. For each $a \in A$ we denote $a!$ as send a and $a?$ as receive a. Then a sequential process and a concurrent process is given by the respective grammars

$$\begin{aligned} P &::= 0 \mid a.P \mid P + P && \text{(sequential process)} \\ P &::= P \mid Q \parallel Q \mid (\nu a)Q \mid C && \text{(concurrent process)} \end{aligned}$$

where C is a process constant. If ν is omitted then we obtain a BPP ("basic parallel processes"). Note: the binding strength of the operators are $\nu > . > \parallel > +$; . and 0 are often omitted.

Definition 7.4 (CCS network)**.** A CCS network $\mathcal{P}$ is a (possibly infinite) set of equations that define process constants such that every constant that occurs in $\mathcal{P}$: (i) is defined by an equation in $\mathcal{P}$; (ii) every r.h.s. occurrence is "guarded" i.e. occurs within an action prefix.

Example 7.4.

$$\{C_0 = a.C_1 + b.O, C_1 = a.C_0 \parallel bbaO\}$$

Definition 7.5 (Semantics)**.** The semantics of CCS is given by the following rules. We define τ as a fixed operator symbol and μ as a meta-variable representing any action (including τ).

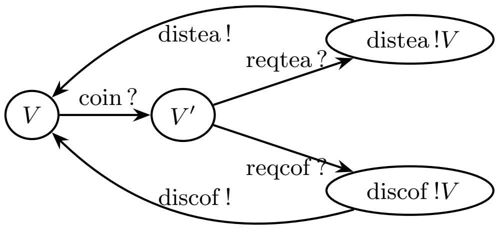
$$\frac{}{aP \xrightarrow{a} P} \qquad \frac{P \xrightarrow{\mu} P'}{(\nu a)P \xrightarrow{\mu} (\nu a)P'}\ \mu \neq a!, a?$$

$$\frac{P \xrightarrow{a} P'}{\begin{array}{c} P + Q \xrightarrow{a} P' + Q \\ Q + P \xrightarrow{a} Q + P' \\ P \parallel Q \xrightarrow{a} P' \parallel Q \\ Q \parallel P \xrightarrow{a} Q \parallel P' \end{array}} \qquad \frac{P \xrightarrow{a?} P' \qquad Q \xrightarrow{a!} Q'}{\begin{array}{c} P \parallel Q \xrightarrow{\tau} P' \parallel Q' \\ Q \parallel P \xrightarrow{\tau} Q' \parallel P' \end{array}}$$

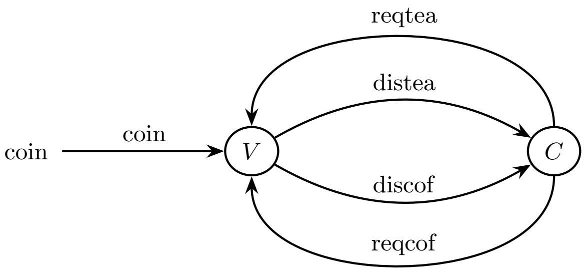
Example 7.5 (Vending Machine)**.** The $\mathcal{P}$ for a coffee and tea vending machine consists of the following equations

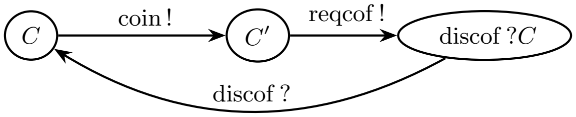
$$\begin{aligned} V &= \text{ coin ?(reqcof ? discof !}V + \text{ reqtea ? distea !}V), \\ C &= \text{ coin ! reqcof ! discof ?}C, \\ S &= (\nu \text{ coin })(\nu \text{ reqcof })(\nu \text{ reqtea , distea , discof })V \parallel C \end{aligned}$$

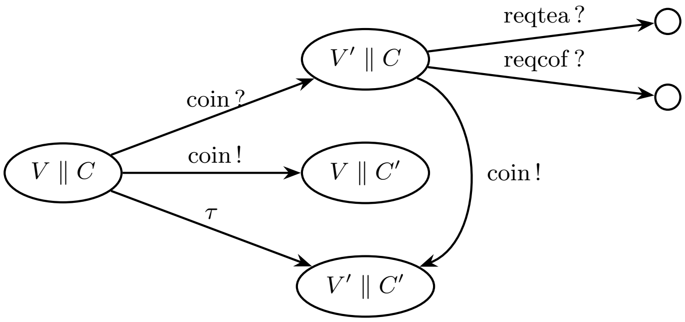
constructed over the set $A = \{$ coin , reqcof , reqtea , discof , distea$\}$ of atomic actions.

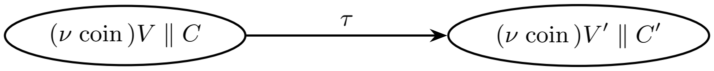


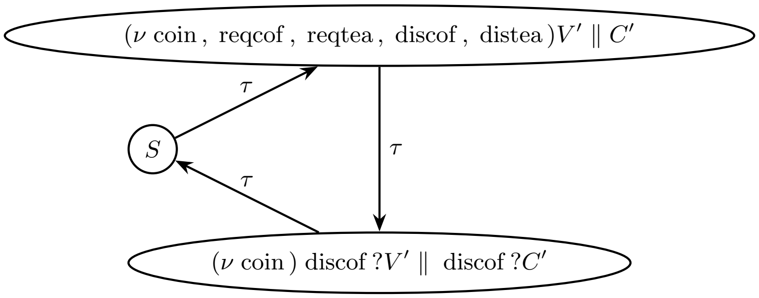
The interface S of V is given as the graph










(ν coin , reqcof , reqtea , discof , distea)$V' \parallel C'$
τ
S
τ
τ
(ν coin) discof ?$V' \parallel$ discof ?C'

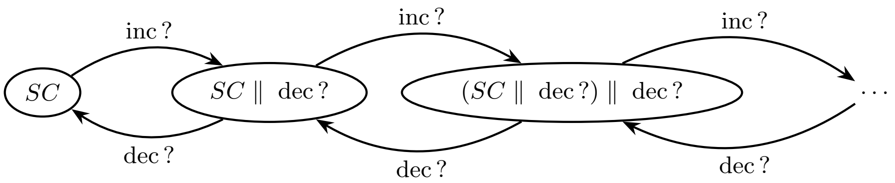
Remark 7.2. CCS networks are Turing-complete but finite CCS networks are still infinite state but not Turing-complete.

Example 7.6 (Counter)**.** Let $A :\!- \{\,\text{inc}\,,\ \text{dec}\,,\ \text{zero}\,\}$, then we define

$$\text{Counter} \coloneqq \left\{ \begin{array}{l} C_0 = \ \text{inc}\,?C_1 + \ \text{zero}\,?C_0, \\ C_1 = \ \text{inc}\,?C_2 + \ \text{dec}\,?C_0, \\ C_2 = \ \text{inc}\,?C_3 + \ \text{dec}\,?C_1, \\ \vdots \end{array} \right\}$$

and

$$\text{Semicounter} :\!- \{SC = \ \text{inc}\,?(SC \parallel \ \text{dec}\,?0)\}$$



Definition 7.6 (Regular CCS Network)**.** Regular (finite) CCS networks are defined by the following grammar. Regular networks are finite state.

$$\begin{aligned} R &::= O \mid aP \mid R + R \\ P &::= R \mid C \\ Q &::= P \mid Q \parallel Q \mid (\nu a)Q \end{aligned}$$

Remark 7.3. Regular CCS networks represent a finite set of CCS processes, each parallel composition of (possibly nested) sequential processes.

Theorem 7.1 (Laws)**.** *For an appropriately chosen notion of equivalence, the following equation hold*

$$\begin{aligned} P + Q &\approx Q + P, \\ (P + Q) + R &\approx P + (Q + R), \\ P + P &\approx P, \\ P + O &\approx P \end{aligned}$$

In addition we have the expansion law

$$a \parallel b \approx ab + ba$$

8 Labeled Transition Systems

Definition 8.1 (Labeled Transition Systems)**.** A labeled transition systems (LTS) for a set of actions A is a tuple $T := (S, \to,)$ where S is a set of states, and $\to \subseteq S \times A \times S$ is a labeled transition relation. It defines a finite or infinite directed graph whose edges are labeled by actions (including τ which is an invisible action).

Example 8.1. A process network $\mathcal{P}$ can be encoded as a LTS $T_{\mathcal{P}}$. We define S to be the set of sub-processions that occur in $\mathcal{P}$ closed under $\to$ and $\to$ is given by the SOS rules.

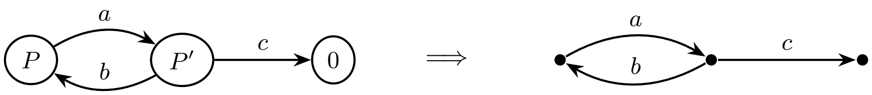
Remark 8.1. A LTS $T = (S, \to)$ is finitely branching (for guarded process terms) if A is finite and finite state if S is finite.

Exercise 8.1. Design a coffee machine that needs refilling after every second drink, together with a coffee preferring customer, and a second tea preferring customer that never refills. Give CCS network and the corresponding LTS (for all sub-processions).

Example 8.2. When are two processes equal? For example consider the following syntactic CCS expressions

$$P = a(bP + c), \quad P' = bP + c, \text{and} \quad P'' = c + bP$$

To answer this we look at their semantics, which is given by LTS, e.g.,



Hence, the question can be reduced to defining equivalences between LTS.

Definition 8.2 (Equivalences)**.** the four common equivalence relations on LTS are: (i) $\approx_{\text{iso}}$ isomorphism between LTS; (ii) $\approx_{\text{bis}}$ bisimilarity; (iii) $\approx_{\text{sim}}$ similarity; (iv) $\approx_{\text{lin}}$ language equivalence.

Theorem 8.1. *We know that:*

$$\approx_{\text{iso}} \implies \approx_{\text{bis}} \implies \approx_{\text{sim}} \implies \approx_{\text{lin}}$$
$$\approx_{\text{lin}} \not\Longrightarrow \approx_{\text{sim}} \not\Longrightarrow \approx_{\text{bis}} \not\Longrightarrow \approx_{\text{iso}}$$

Definition 8.3 (Isomorphism)**.** Let $T_1 = (S_1, \mapsto)$ and $T_2 = (S_2, \mapsto)$. Let $s_1 \in S_1$ and $s_2 \in S_2$. Then $s_1 \approx_{\text{iso}} s_2$ iff there exists $h : S_1 \mapsto S_2$ such that:

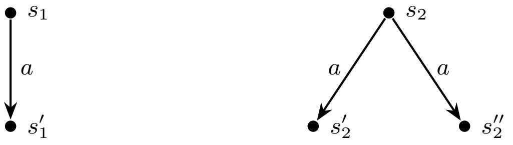
(i) h is a bijection (onto and 1-1);

(ii) for all $s, s' \in S_1$ and all actions $a \in A$, we have $s \xrightarrow{a} s'$ iff $h(s) \xrightarrow{a} h(s')$;

(iii) $h(s_1) = s_2$.

Remark 8.2. Isomorphism can be seen as a structural equivalence.

Example 8.3 (Ex. 8.2 cont.)**.** It is easy to see that $P' \approx_{\text{iso}} P''$ due to the commutativity of $+$, i.e., $P + Q = Q + P$. However, this notion of equivalence is too strong to capture $P = P + P$, i.e., semantically P and $P + P$ can be represented as



As there is no bijection to be found the two LTS are not isomorphic. Hence, we are interested in weaker notions of similarity.

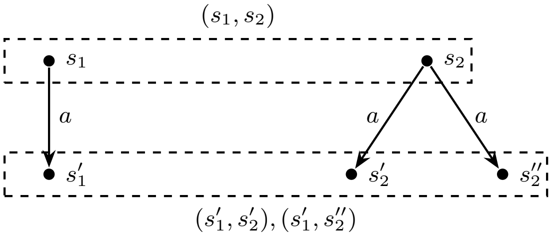
Definition 8.4 (Bisimilarity)**.** Let $T_1 = (S, \mapsto)$. Let $s_1 \in S$ and $s_2 \in S$. Then $s_1 \approx_{\text{bis}} s_2$ iff there exists $r \subseteq S \times S$ such that:

(i) $\forall s_1, s_2 \in S$ if $(s_1, s_2) \in r$, then $\forall s_1' \in S, \forall a \in A$ if $s_1 \xrightarrow{a} s_1'$ then $\exists s_2'$ such that $s_2 \xrightarrow{a} s_2'$ and $(s_1', s_2') \in r$;

(ii) $\forall s_1, s_2$ if $(s_1, s_2) \in r$, then $\forall s_2' \in S, \forall a \in A$ if $s_2 \xrightarrow{a} s_2'$, then $\exists\ s_1'$ such that $s_1 \xrightarrow{a} s_1'$ and $(s_1', s_2') \in r$;

(iii) $(s_1, s_2) \in r$.

Example 8.4 (Ex. 8.2 cont.)**.** Continuing. While P and $P+P$ are not isomorphic, they are bisimilar, i.e., we define $r \coloneqq \{(s_1, s_2), (s_1', s_2'), (s_1', s_2'')\}$



Remark 8.3. Both the identity and all graph isomorphism are bisimulations. The converse does not hold. Computing isomorphisms is hard, i.e., it is somewhere between P and NP. Whereas computing bisimulations is relatively easy, i.e., $\mathcal{O}(m \log n)$ where n is the number of states and m the number of edges.

Theorem 8.2. *Let $r \subseteq S \times S$ and let*

$$C_{(i)}(r, s_1, s_2) \coloneqq \forall s_1' \forall a.\ s_1 \xrightarrow{a} s_1' \Rightarrow \exists s_2'.\ s_2 \xrightarrow{a} s_2' \wedge r(s_1', s_2')$$
$$C_{(ii)}(r, s_1, s_2) \coloneqq \forall s_2' \forall a.\ s_2 \xrightarrow{a} s_2' \Rightarrow \exists s_1'.\ s_1 \xrightarrow{a} s_1' \wedge r(s_1', s_2')\}$$

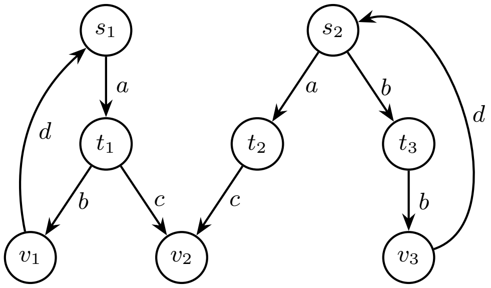
We define $F(r) = \{(s_1, s_2) | C_{(i)}(r, s_1, s_2) \wedge C_{(ii)}(r, s_1, s_2)\}$. Then $\approx_{\text{bis}} = \text{gfp} F$.

Remark 8.4. From Theorem 8.2 we can derive a simple algorithm for computing $\approx_{\text{bis}}$ by starting from $S \times S$ and iteratively refining the relation.

$$\begin{aligned}
& \approx_{\text{bis}}^0 = S \times S \\
\supseteq \quad & \approx_{\text{bis}}^1 = F(\approx_{\text{bis}}^0) = \{(s_1, s_2) | C_{(i)}(\approx_{\text{bis}}^0, s_1, s_2) \wedge C_{(ii)}(\approx_{\text{bis}}^0, s_1, s_2)\} \\
\supseteq \quad & \approx_{\text{bis}}^2 = F(\approx_{\text{bis}}^1) = \{(s_1, s_2) | C_{(i)}(\approx_{\text{bis}}^1, s_1, s_2) \wedge C_{(ii)}(\approx_{\text{bis}}^1, s_1, s_2)\} \\
\supseteq \quad & \quad \vdots \\
\supseteq \quad & \approx_{\text{bis}} = \text{gfp} F(S \times S)
\end{aligned}$$

This is called the simple fixed-point approximation algorithm (aka. partition refinement): $O(m \cdot n)$

Example 8.5. Consider the following LTS



Then applying the fixed-point approximation algorithm we get

$$\begin{aligned}
&\approx^0_{\text{bis}} = S \times S\\
&\approx^1_{\text{bis}} = \text{id} \cup \{(v_1,v_2),(v_1,v_3),(v_2,v_1),(v_2,v_3),(v_3,v_1),(v_3,v_2)\}\\
&\quad\vdots\\
&\approx_{\text{bis}} = \text{id} \cup \{(v_1,v_2),(v_2,v_1)\}
\end{aligned}$$

where $\text{id} \coloneqq \{(s_1,s_1),(s_2,s_2),(t_1,t_1),(t_2,t_2),(t_3,t_3),(v_1,v_1),(v_2,v_2),(v_3,v_3)\}$ is the identity relation.

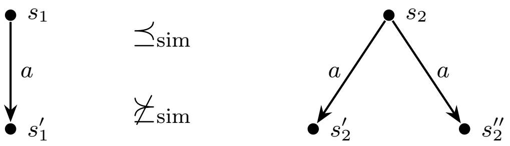
Definition 8.5 (Simulation)**.** For $s_1, s_2 \in S$, s_2 simulates s_1, i.e., $s_1 \preceq_{\text{sim}} s_2$, iff $\exists r \in S \times S$ such that:

(i) $\forall s_1, s_2$ if $s_1 \preceq_{\text{sim}} s_2$ then, $\forall s_1' \forall a$ if $s_1 \xrightarrow{a} s_1'$ then $\exists s_2'$ such that $s_2 \xrightarrow{a} s_2'$ and $s_1' \preceq_{\text{sim}} s_2'$;

(ii) $(s_1, s_2) \in r$

Remark 8.5. Note that in Definition 8.5 the second condition in bisimilarity is not present in the definition of simulation. (Example on difference on blackboard) Hence, it is not symmetric.

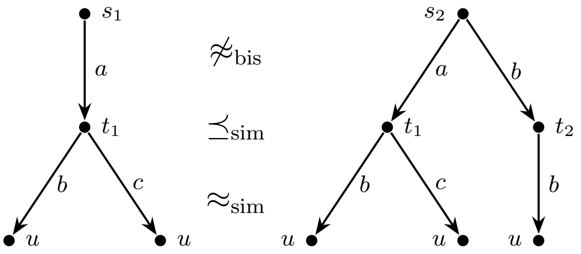
Example 8.6. The asymmetry of $\preceq_{\text{sim}}$ can be observed in the following example. The right structure simulates the left, but the inverse does not hold.



Definition 8.6 (Similarity)**.** Two states s_1, s_2 are called similar or $s_1 \approx_{\text{sim}} s_2$ iff (i) $s_1 \preceq_{\text{sim}} s_2$ and (ii) $s_2 \preceq_{\text{sim}} s_1$.

Remark 8.6. Bisimilarity implies similarity, but the opposite direction is not generally true. Similarity is a greatest fixed-point and can be computed via approximation sequence $\approx^0_{\text{sim}}, \approx^1_{\text{sim}}, \ldots$ in $\mathcal{O}(m \cdot n)$.

Example 8.7. The following demonstrates that similartiy is a weaker form of equivalence than bisimulation.



in this transition system we have

$$\begin{aligned}
\approx_{\text{bis}} &= \{(s,s) \mid s \in S\} = \text{id} \\
\preceq_{\text{sim}} &= \text{id} \cup \{(t_2,t_1),(s_1,s_2),(s_2,s_1)\} \\
\approx_{\text{sim}} &= \text{id} \cup \{(s_1,s_2),(s_2,s_1)\}
\end{aligned}$$

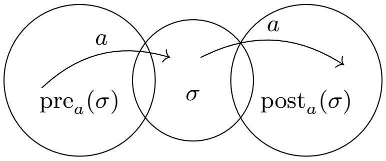
Theorem 8.3. *Bisimilarity is a congruence for CCS, i.e., (i) bisimilarity is an equivalence and (ii) for all CCS processes P, Q, R, if $P \approx_{\text{bis}} Q$ then $R[P] \approx_{\text{bis}} R[Q]$, e.g., $R + P \approx_{\text{bis}} R + Q$ or $R \parallel P \approx_{\text{bis}} R \parallel Q$*

Exercise 8.2. Proof (ii) in Theorem 8.3.

Definition 8.7. We define the predecessor and successor of a subset $\sigma \subseteq S$ for some actions $a \in A$ respectively as

$$\begin{aligned}
\text{pre}_a(\sigma) &\coloneqq \{s \mid \exists s' \in \sigma.\ s \xrightarrow{a} s'\} \\
\text{post}_a(\sigma) &\coloneqq \{s' \mid \exists s \in \sigma.\ s \xrightarrow{a} s'\}
\end{aligned}$$

Remark 8.7. The set $\text{pre}_a(\sigma)$ is the set of states that can reach σ by action a and the set $\text{post}_a(\sigma)$ is the set of states that can be reached from σ by action a.



Theorem 8.4.

$$s \preceq_{\text{sim}} t \iff \exists R \subseteq S \times S.\ (s,t) \in R \Rightarrow \forall a\ \forall s' \in \text{post}_a(s)\ \exists t' \in post_a(t).(s',t') \in R$$

$$s \approx_{\text{sim}} t \iff s \preceq_{\text{sim}} t \wedge t \preceq_{\text{sim}} s$$

$$s \approx_{\text{bis}} t \iff \exists R \subseteq S \times S.\ (s,t) \in R \Rightarrow \forall a \begin{bmatrix} (\forall s' \in \text{post}_a(s)\ \exists t' \in \text{post}_a(t).(s',t') \in R) \wedge \\ (\forall t' \in \text{post}_a(t)\ \exists s' \in \text{post}_a(s).(s',t') \in R) \end{bmatrix}$$

8.1 Characterization

Theorem 8.5 (Game theoretic characterization of $\preceq_{\text{sim}}$)**.** *The game theoretic characterization $\preceq_{\text{sim}}$ is given be the following game.*

1. *Put red token on s, and blue token on t.*
2. *Repeat:*
   - (a) *Player* 1 *moves the* red *token to a successor state.*
   - (b) *Player* 2 *moves the* blue *button to a successor state, matching the action of the player* 1 *move.*

If in the following game player 1 *has a strategy s.t. at some point, player* 2 *cannot match player* 1*'s move, then* $s \not\preceq_{\text{sim}} t$*, otherwise* $s \preceq_{\text{sim}} t$*.*

Theorem 8.6 (Game theoretic characterization of $\approx_{\text{bis}}$)**.** *The game theoretic characterization* $\preceq_{\text{sim}}$ *is given be the following game.*

1. *Put red token on* s*, and blue token on* t*.*
2. *Repeat:*
   - (a) *Player* 1 *chooses either the red or blue token.*
   - (b) *Player* 1 *moves the* chosen *token to a successor state.*
   - (c) *Player* 2 *moves the* other *button to a successor state, matching the action of the player* 1 *move.*

If in the following game player 1 *has a strategy s.t. at some point, player* 2 *cannot match player* 1*'s move, then* $s \not\approx_{\text{bis}} t$*, otherwise* $s \approx_{\text{bis}} t$*.*

Algorithm 2 $\mathcal{A}_{\approx_{\text{bis}}}$

```
$X \leftarrow \{S\}$
$X' \leftarrow \{\emptyset\}$
while $X' \neq X$ do
    $X' \leftarrow X$
    $X \leftarrow X \cup \{\sigma \cup \text{pre}_a(\sigma') | \sigma, \sigma' \in X,\ a \in A\} \cup \{\sigma \setminus \text{pre}_a(\sigma') | \sigma, \sigma' \in X,\ a \in A\}$
end while
if $\forall \sigma \in X.\ s, t \in \sigma \vee s, t \notin \sigma$ then
    return true
end if
return false
```

Theorem 8.7 (Fixpoint characterization of $\approx_{\text{bis}}$)**.** *Algorithm* $\mathcal{A}_{\approx_{\text{bis}}}$ *provides a fixed point characterization of* $\approx_{\text{bis}}$*, i.e.,* $\forall s, t \in S$ *we have that* $s \approx_{\text{bis}} t$ *iff* $\mathcal{A}_{\approx_{\text{bis}}}$ *terminates with* **true***.*

Theorem 8.8 (Fixpoint characterization of $\approx_{\text{sim}}$)**.** *Algorithm* $\mathcal{A}_{\approx_{\text{sim}}}$ *provides a fixed point characterization of* $\approx_{\text{sim}}$*, i.e.,* $\forall s, t \in S$ *we have that* $s \approx_{\text{sim}} t$ *iff* $\mathcal{A}_{\approx_{\text{sim}}}$ *terminates with* **true***.*

Definition 8.8 (Hennessy-Milner Logic (HML) Syntax)**.** The syntax of Hennessy-Milner Logic (HML) is given by the following grammar.

$$\varphi ::= \textbf{true} \mid \textbf{false} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi,\ a \in A \mid [a]\varphi,\ a \in A$$

Definition 8.9 (Hennessy-Milner Logic (HML) Semantics)**.** The semantics of Hennessy-Milner Logic (HML) is defined as follows. Let $T := (S, \rightarrow)$ be a labeled transition system, let φ a HML formula, let $s \in S$, then we define $s \models \varphi$ inductively:

Algorithm 3 $\mathcal{A}_{\approx_{\text{sim}}}$

```
$X \leftarrow \{S\}$
$X' \leftarrow \{\emptyset\}$
while $X' \neq X$ do
    $X' \leftarrow X$
    $X \leftarrow X \cup \{\sigma \cup \text{pre}_a(\sigma') | \sigma, \sigma' \in X,\ a \in A\}$
end while
if $\forall \sigma \in X.\ s, t \in \sigma \vee s, t \notin \sigma$ then
    return true
end if
return false
```

- if $\varphi = \varphi_1 \wedge \varphi_2$, then $s \models_T \varphi_1 \wedge \varphi_2$ iff $s \models_T \varphi_1$ and $\models_T \varphi_2$;
- if $\varphi = \varphi_1 \vee \varphi_2$, then $s \models_T \varphi_1 \vee \varphi_2$ iff $s \models_T \varphi_1$ and $\models_T \varphi_2$;
- if $\varphi = \langle a \rangle \varphi$, then $s \models_T \langle a \rangle \varphi$ iff $\exists s' \in \text{post}_a(s).\ s' \models_T \varphi$ (equivalent to $\exists s'.\ s \xrightarrow{a} s' \wedge \varphi$);
- if $\varphi = [a]\varphi$, then $s \models_T [a]\varphi$ iff $\forall s' \in \text{post}_a(s).\ s' \models_T \varphi$ (equivalent to $\forall s'.\ s \xrightarrow{a} s' \Rightarrow \varphi$);

Moreover, we define $[\![\varphi]\!]_T \subseteq S$ s.t. $[\![\mathbf{true}]\!]_T \coloneqq S$, $[\![\mathbf{false}]\!]_T \coloneqq \emptyset$, $[\![\varphi_1 \wedge \varphi_2]\!]_T \coloneqq [\![\varphi_1]\!]_T \cap [\![\varphi_2]\!]_T$, and $[\![\varphi_1 \vee \varphi_2]\!]_T \coloneqq [\![\varphi_1]\!]_T \cup [\![\varphi_2]\!]_T$.

Definition 8.10 (HML Fragments)**.** The logic $\text{HML}^{\exists}$ is the HML without $[a]$. The logic $\text{HML}^{\forall}$ is the HML without $\langle a \rangle$. The logic $\text{HML}_{\neg}$ adds negation, i.e., for any HML formula we add $\neg\varphi$.

Lemma 8.9. *In* $\text{HML}_{\neg}$ *the following equivalences hold*

$$\begin{aligned} \neg\mathbf{true} &\equiv \mathbf{false} \\ \neg(\varphi_1 \wedge \varphi_2) &\equiv \neg\varphi_1 \vee \neg\varphi_2 \\ \neg\langle a \rangle \varphi &\equiv [a]\neg\varphi \\ \neg[a]\varphi &\equiv \langle a \rangle \neg\varphi \end{aligned}$$

Theorem 8.10 (Logical Characterization of $\approx_{\text{bis}}$)**.** *The* $\approx_{\text{bis}}$ *is logically characterized by* HML*, i.e.,* $s \approx_{\text{bis}} t$ *iff* $\forall \varphi \in \text{HML}$*, either* $s, t \models \varphi$ *or* $s, t \not\models \varphi$*.*

Theorem 8.11 (Logical Characterization of $\approx_{\text{sim}}$)**.** *The* $\approx_{\text{sim}}$ *is logically characterized by* $\text{HML}^{\exists}$ *(or* $\text{HML}^{\forall}$*), i.e.,* $s \approx_{\text{sim}} t$ *iff* $\forall \varphi \in \text{HML}^{\exists}$*, either* $s, t \models \varphi$ *or* $s, t \not\models \varphi$*. (Alternatively,* $s \approx_{\text{sim}} t$ *iff* $\forall \varphi \in \text{HML}^{\forall}$*, either* $s, t \models \varphi$ *or* $s, t \not\models \varphi$*).*

8.2 State equivalence language (or trace) equivalence

Definition 8.11. Let $T = (S, \rightarrow)$ be an LTS, let $s \in S$, then we define

$$L_T(s) \coloneqq \{a_1 a_2 \cdots a_n | \exists s_0 \cdots s_n.\ s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \wedge s_0 = s\}$$

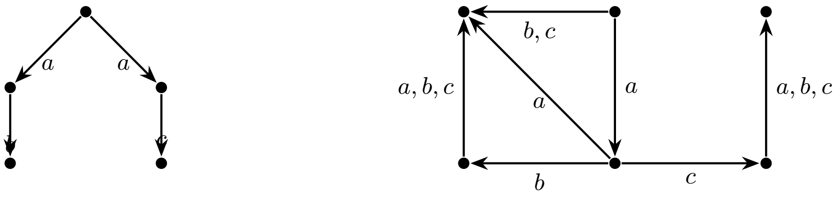
Lemma 8.12. *$L_T(s)$ contains ε and is prefix-closed.*

Definition 8.12 (Linear Equivalence)**.** $s_1 \approx_{\text{lin}} s_2$ iff $L_T(s_1) = L_T(s_2)$.

Remark 8.8. $s_1 \approx_{\text{sim}} s_2$ implies $s_1 \approx_{\text{lin}} s_2$ (but not the other direction). The problem of $s_1 \overset{?}{\approx}_{\text{lin}} s_2$ reduces to $L_T(s_1) \overset{?}{=} L_T(s_2)$. The latter is known as the language inclusion problem and corresponds to the equivalence problem for finite non-deterministic automata (NFA).

Remark 8.9. To decide whether $L_{T_1}(s) = L_{T_2}(s)$, we decide whether $L_{T_1}(s) \subseteq L_{T_2}(s)$ and $L_{T_2}(s) \subseteq L_{T_1}(s)$. However, for two languages L_1 and L_2 accepted by NFAs we have $L_1 \subseteq L_2$ iff $L_1 \cap A^* \setminus L_2 =$. Hence, the language inclusion problem is PSPACE-hard. To elaborate while the intersection of two languages is polynomial and the emptiness check for NFAs is linear, the step $A^* \setminus L_2$ requires the determinization of an NFA which is exponential.

Example 8.8. Below an example of the determinization of a NFA.

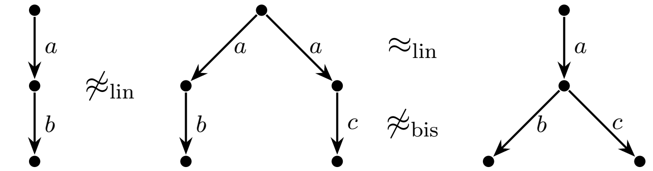


Definition 8.13 (HML$_{\text{lin}}$)**.** The fragment $\text{HML}_{\text{lin}}{}^{\exists}$ of HML is characterized by the grammar

$$\varphi ::= \mathit{true} \mid \langle a \rangle \varphi$$

The fragment $\text{HML}_{lin}^{\forall}$ is defined analogously.

Example 8.9. Below an example of comparing bisimulation and linear equivalence



Remark 8.10. To summarize:

$\approx_{\text{iso}}$	NP	–
$\approx_{\text{bis}}$	$O(mlogn)$	HML
$\approx_{\text{sim}}$	$O(m \cdot n)$	HML$^{\exists}$
$\approx_{\text{lin}}$	PSPACE	HML$_{\text{lin}}{}^{\exists}$

9 Modal μ-calculus

Definition 9.1 (Modal μ-calculus Syntax)**.** The syntax of the modal μ-calculus is defined by the following grammar. Let X be a set of variables

$$\varphi ::= \text{HML} \mid x \in X \mid \underbrace{\mu\, x.\varphi}_{\text{lfp}} \mid \underbrace{\nu\, x.\varphi}_{\text{gfp}}$$

Definition 9.2 (Modal μ-calculus Semantics)**.** The semantics of the modal μ-calculus is defined as follows. Let $T := (S, \to)$ be a LTS over a finite action set (alphabet) A. The we define: (i) $[\![\varphi]\!]_T^{\mathcal{I}} \subseteq S$ where $\mathcal{I} \colon X \to 2^S$; (ii) $[\![x]\!]_T^{\mathcal{I}} = \mathcal{I}(x)$; (iii) $[\![\mu\, x.\varphi]\!]_T^{\mathcal{I}} = \bigcup_{\sigma \subseteq S} [\![\varphi]\!]_T^{\mathcal{I}[x \to \sigma]}$; (iv) $[\![\nu\, x.\varphi]\!]_T^{\mathcal{I}} = \bigcap_{\sigma \subseteq S} [\![\varphi]\!]_T^{\mathcal{I}[x \to \sigma]}$.

Remark 9.1. Let $F := \lambda\sigma \subseteq S.[\![\varphi]\!]_T^{\mathcal{I}[x \to \sigma]}$ then $[\![\mu\, x.\varphi]\!]_T^{\mathcal{I}} = \text{lfp}F$ and $[\![\nu\, x.\varphi]\!]_T^{\mathcal{I}} = \text{gfp}F$. That is, lfp$F$ is the limit (infinite union) of

$$\sigma_0 = \emptyset \subseteq \sigma_1 = [\![\varphi]\!]_T^{\mathcal{I}[x \to \sigma_0]} \subseteq \sigma_2 = [\![\varphi]\!]_T^{\mathcal{I}[x \to \sigma_1]} \subseteq \cdots$$

and lfp$F$ is the limit (infinite intersection) of

$$\sigma_0 = S \supseteq \sigma_1 = [\![\varphi]\!]_T^{\mathcal{I}[x \to \sigma_0]} \supseteq \sigma_2 = [\![\varphi]\!]_T^{\mathcal{I}[x \to \sigma_1]} \supseteq \cdots$$

Example 9.1. Consider the expression $(\mu\, x)(\langle a\rangle\mathbf{true} \vee \langle\bar{a}\rangle x)$ where $\bar{a} := A \backslash \{a\}$. By evaluating the expression we obtain

$$\begin{aligned}
\sigma_0 &= \mathbf{false}\\
\sigma_1 &= \langle a\rangle\mathbf{true}\\
\sigma_2 &= \langle a\rangle\mathbf{true} \vee \langle\bar{a}\rangle\langle a\rangle\mathbf{true}\\
\sigma_3 &= \langle a\rangle\mathbf{true} \vee \langle\bar{a}\rangle(\langle a\rangle\mathbf{true} \vee \langle\bar{a}\rangle\langle a\rangle\mathbf{true})\\
&= \langle a\rangle\mathbf{true} \vee \langle\bar{a}\rangle\langle a\rangle\mathbf{true} \vee \langle\bar{a}\rangle\langle\bar{a}\rangle\langle a\rangle\mathbf{true}\\
\sigma_4 &= \cdots
\end{aligned}$$

It expresses that there exists a path somewhere along the path there exists an a-transition, i.e., $\exists\Diamond a$.

Example 9.2. Consider the expression $(\nu\, x)([\bar{a}]\mathbf{false} \wedge \langle a\rangle x)$. By evaluating the expression we obtain

$$\begin{aligned}
\sigma_0 &= \mathbf{true}\\
\sigma_1 &= [\bar{a}]\mathbf{false}\\
\sigma_2 &= [\bar{a}]\mathbf{false} \wedge [a][\bar{a}]\mathbf{false}\\
\sigma_3 &= \cdots
\end{aligned}$$

It expresses that on all paths there are only a-transition, i.e., $\forall\Box a$.

Remark 9.2. The the equivalence $\neg\mu\, x.\varphi(x) = \nu\, x.\neg\varphi(\neg x)$ where left has even number of negations.

Definition 9.3 (μ-calculus)**.** We can add propositions $p \in P$ as atomic formulas. Syntactically we can define the grammar

$$\varphi ::= p \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu\, x.\varphi \mid \nu\, x.\varphi x$$

Semantically we evaluate over Kripke structures, i.e., $K = (S, \rightarrow, [\![\cdot]\!] \colon S \rightarrow 2^P)$, e.g., $[\![p]\!]_K \subseteq S$

Remark 9.3. We can formulate the following abbreviations:

$$\begin{aligned}
\exists\Diamond\varphi &= \mu\, x.(\varphi \lor \exists \bigcirc x) \\
\exists\Diamond p &= \mu\, x.(p \lor \exists \bigcirc x) && \text{(p is reachable)} \\
\forall\Box\varphi &= \nu\, x.(\varphi \land \forall \bigcirc x) \\
\forall\Box p &= \nu\, x.(p \land \forall \bigcirc x) && \text{(p is an invariant)} \\
\forall\Diamond\varphi &= \mu\, x.(\varphi \lor \forall \bigcirc x) && \text{(on all paths φ is eventually true)} \\
\exists\Diamond\varphi &= \nu\, x.(\varphi \land \exists \bigcirc x) && \text{(there exists an inf. path where φ is always true)}
\end{aligned}$$

9.1 Computational Tree Logic (CTL)

Definition 9.4 (CTL Syntax)**.** The syntax of the Computation Tree Logic (CTL) is given by the following grammar

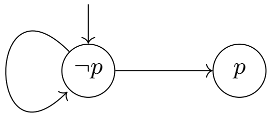
$$\varphi ::= p \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \overbrace{\exists \bigcirc \varphi}^{EX} \mid \overbrace{\forall \bigcirc \varphi}^{AX} \mid \overbrace{\exists\Diamond\varphi}^{EF} \mid \overbrace{\forall\Diamond\varphi}^{AF} \mid \overbrace{\exists\Box\varphi}^{EG} \mid \overbrace{\forall\Box\varphi}^{AG}$$

Remark 9.4. The following equivalences hold: $\exists \bigcirc \varphi = \neg\forall \bigcirc \neg\varphi$ and $\exists\Diamond\varphi = \neg\forall\Box\neg\varphi$.

Remark 9.5. The full μ-calculus is strictly more expressive than CTL, in two ways: (i) $\mu\, x.\exists \bigcirc \exists \bigcirc x$ means p is reachable in even number of steps. (ii) $\nu\, x.\exists\Diamond(p \land \exists \bigcirc x) = \nu\, x.\, \mu\, y((p \land \exists \bigcirc x) \lor \exists \bigcirc y)$ means there exists a path with infinitely many p clauses. It is not expressible in CTL.

Exercise 9.1. Proof that $\mu\, x.\exists \bigcirc \exists \bigcirc x$ is not expressible in CTL.

Remark 9.6. We have that $\forall\Box\forall\Diamond p = \forall\Box\Diamond p$ expresses that on all infinite paths, we have p infinitely often. By contrast, while $\exists\Box\exists\Diamond p \Leftarrow \exists\Box\Diamond p$ the reverse does not hold, e.g.,



Moreover, CTL characterizes $\approx_{\text{bis}}$ and $CTL^{\exists}$ characterizes $\approx_{\text{sim}}$. Next we define a logic that characterizes $\approx_{\text{lin}}$.

9.2 Linear Temporal Logic (LTL)

Remark 9.7. LTL characterizes $\approx_{\text{lin}}$.

Definition 9.5 (LTL Snytax)**.** The syntax of Linear Temporal Logic (LTL) is given by the following grammar

$$phi ::= p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi$$

Definition 9.6 (LTL Semantics)**.** LTL is interpreted over inf traces $t = s_0 s_1 s_2 s_3 \ldots$ s.t $s_i \to s_{i+1}$ for all $i \geq 0$ the we define

$$\begin{array}{lcll}
t \models p & \iff & p \in [\![s_0]\!] & \\
t \models \bigcirc\varphi & \iff & s_0 s_1 s_2 s_3 \ldots \models \varphi & \text{next } \varphi \\
t \models \Diamond\varphi & \iff & \exists i \geq 0 s_i s_{i+1} s_{i+2} s_{i+3} \ldots \models \varphi & \text{eventually } \varphi \\
t \models \Box\varphi & \iff & \forall i \geq 0 s_i s_{i+1} s_{i+2} s_{i+3} \ldots \models \varphi & \text{always } \varphi
\end{array}$$

We say $s \models \varphi$ iff there exists trace t from s s.t. $t \models \varphi$.

Remark 9.8. Now you can compare LTL to CLT etc. For example $\Box\Diamond p$ is infinitely often p. It is incomparable to CTL (fragment of of μ-calculus).

Definition 9.7 (Until)**.** We define the until operator $\mathcal{U}$ as

$$\varphi \mathcal{U} \psi \iff \exists s_i s_{i+1} \ldots \models \psi \wedge \forall j < i . s_j s_{j+1} \ldots \models \varphi$$

10 Concurrency Theory

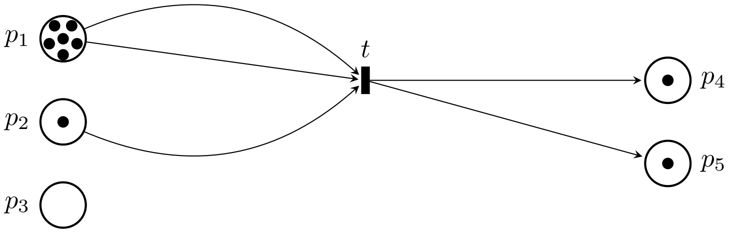
Remark 10.1. Calculus of Communicating Systems (CCS) can be given two semantics.

- *Sequential semantics:* Labeled transition systems and logics (e.g., Hennessy-Milner logic (HML) and temporal logic)
- *Truly concurrent semantics:* Petri Nets (because parallelism is different from nondeterminism)

Definition 10.1 (Petri nets)**.** A *petri net* is a tuple $N = (P, T)$ where P is a set of places and T a set of transitions. The state of a petri net is given by a marking $M \colon P \to \mathbb{N}$ and a transition is a function that takes a marking and an action and returns another marking, i.e., $t \colon M \to M$. Equivalently, considering a set of names for each transition (actions) A, we have that $T \colon M \times A \to M$.

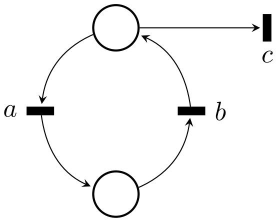
Remark 10.2 (Petri nets as lebeled transition systems). A Petri net $N = (P, T)$ can be interpreted as the following labeled transition system (LTS) $LTS_N = (M, \to)$. The transition $\to$ is given by $m \to m'$ if and only if there exists $t \in T$ such that: (i) the transformation t decomposes in precodition $^*t \in M$, action $a \in A$ and postcondition $t^* \in M$, i.e., $t = (^\bullet t, a, t^\bullet)$; (ii) the precodition $^\bullet t$ is contained in m, i.e., $^\bullet t \subseteq m$; (iii) the transformation t converts m into m', i.e., $m' = m -^\bullet t + t^\bullet$.

Example 10.1 (Petri net)**.** Consider the Petri net $N = (P, T)$ where $P = \{p_1, p_2, p_3, p_4, p_5\}$, $T = \{t\}$, and $t\colon \{(p_1, 2), (p_2, 1)\} \mapsto \{(p_4, 1), (p_5, 1)\}$. In other words, t can fire a transition if there are 2 markings in p_1 and 1 marking in p_2, and adds 1 marking in p_4 and 1 marking in p_5.



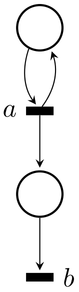
Definition 10.2 (Sequential net)**.** A Petri net is *sequential* if all transformations t have a precondition $^*t$ such that $|^*t| \leq 1$. Sequential nets are equivalent to regular languages for finite strings.

Example 10.2 (Sequential net)**.** Below an example of a sequential net.



Definition 10.3 (Basic parallel process)**.** A Petri net is a *basic parallel processes* if all transformations t have a precondition $^*t$ such that $|^*t| = 1$. Basic parallel processes are equivalent to context-free languages for finite strings.

Example 10.3 (Basic parallel process)**.** Below an example of a basic parallel process.



Definition 10.4 (Chomsky hierarchy)**.** The *Chomsky hierarchy* is a hierarchy of sequential computation given by

$$FA \subseteq PDA \subseteq LBA \subseteq TM\,,$$

where the decidabilities are as follows.

- FA. Fully decidable.
- PDA. Universality is undecidable, i.e., $L(A) \overset{?}{=} \Sigma^*$.
- LBA. Emptyness is undecidable, i.e., $L(A) \overset{?}{=} \emptyset$.
- TM. Membership is undecidable, i.e., $w \overset{?}{\in} L(A)$.

Definition 10.5 (Concurrency hierarchy)**.** The *concurrency hierarchy* is a hierarchy of concurrent computation given by

$$FA \subseteq BPP \subseteq CCS \subseteq TM\,,$$

where, in CCS, reachability is decidable but non-elementary.

Exercise 10.1 (Ortogonal Hierarchies)**.** Solve the following tasks.

(i) Show that $X = a(X|b) + c(X|d) + e$ is not context-free. Hint: use the pumping lemma.

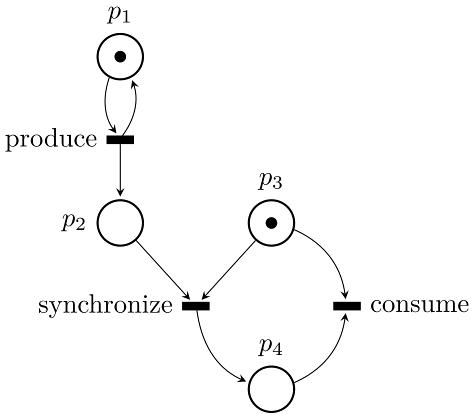
(ii) Present a context-free language that is not in BPP, and try to prove it.

(iii) Give a Petri net that generates $\{a^n b^n | n > 0\}$.

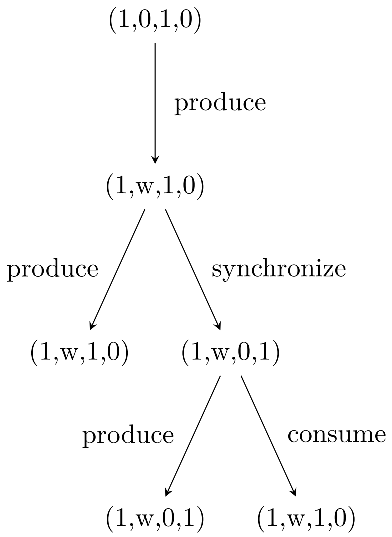
Definition 10.6 (Coverage problem for Petri nets)**.** Given a Petrin net N and two marking m, m', the problem is to decide the existence of a sequence of transitions that take m to a marking that covers m', i.e., if there exists m'' such that $m \to^* m''$ and $m' \subseteq m''$. As opposed to the reachability question, the coverage problem asks for $m' \subseteq m''$ instead of $m' = m''$.

Remark 10.3. The coverage problem is a finite problem, since every finite Petri net N has a finite coverage tree of size $\mathcal{O}(n^n)$ where $n = |N|$. The coverage tree can be obtained by the Karp-Miller algorithm, which generates a tree of depth at most $\mathcal{O}(n \log n)$.

Example 10.4. Consider the Petri net with an initial marking is given below.



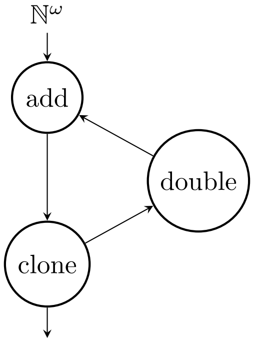
The corresponding coverage tree follows. Note that m' is covered by m if there is a node in the tree that covers it.



Definition 10.7 (Control-flow models)**.** Graphs whose nodes represent control states and whose edges represent control flow. For example, labeled transition systems and Petri nets.

Definition 10.8 (Data-flow models)**.** Also called circuits, they are graphs whose nodes represent actors (gates) and whose edges represent data flow.

Example 10.5. Below an example of a data-flow model.



Definition 10.9 (Combinatorial circuits)**.** A data-flow model is a *combinatorial circuit* if there are no loops. For combinatorial circuits, every tuple of inputs produces a tuple of outputs.

Definition 10.10 (Sequential circuits)**.** A data-flow model is a *sequential circuit* if there are loops. For sequential circuits, every sequence of inputs produces a sequence of outputs.

Remark 10.4. To deal with loops in sequential circuits, there are two approaches, i.e., synchronous (clocked) and asynchronous (queues as channels – Kahn networks).