Formal Techniques for Software Engineering: Introduction and Preliminaries

Rocco De Nicola

IMT Institute for Advanced Studies, Lucca
rocco.denicola@imtlucca.it

April 2013

Lesson 1

Programs Semantics

Formal Definitions

Each language comes equipped with **syntax** & **semantics**

- **Syntax**: defines legal programs (grammar based)
- **Semantics**: defines meaning, behavior, errors (formally)

Rôle of Formal Semantics

- Language **design**
- Language **implementation**
- Program/Model **correctness**
- Program/Model **equivalence**
- Program/Model **refinement**
- ...

Warmup Motivations

Consider a `C`-like language, say with `x` having initial value `1`

- `y = x++ + x++;`
  What is the value of `x` and of `y`?
- `z = x++ - x++;`
  What is the value of `z`?
- `g(x)=g(x-1) with f(x)=1;`
  What is the value of `f(g(42))`?

The situation is even more critical when concurrency enters the game. Then, we need not only to worry about the choices of compiler designers but also of the internal nondeterminism triggered by the parallel evaluation of programs that may collaborate on tasks but compete for resources (cpu, memory,data, ...). Examples later.

Warmup Motivations

Consider a `C`-like language, say with `x` having initial value `1`

- `y = x++ + x++;`
  What is the value of `x` and of `y`?
- `z = x++ - x++;`
  What is the value of `z`?
- `g(x)=g(x-1) with f(x)=1;`
  What is the value of `f(g(42))`?

The situation is even more critical when concurrency enters the game. Then, we need not only to worry about the choices of compiler designers but also of the internal nondeterminism triggered by the parallel evaluation of programs that may collaborate on tasks but compete for resources (cpu, memory,data, . . .). Examples later.

Course structure and objectives

Course Structure

1. Part 1 will focus on denotational and operational semantics of sequential programming languages
2. Part 2 will introduce formalisms to specify, design, implement, analyze, and prove properties of concurrent and distributed systems.

Objective

- To appreciate the inner meaning of programming languages
- To learn to design and analyze simple concurrent systems
- To acquire the skills necessary to verify systems correctness through software tools.

Course structure and objectives

Course Structure

1. Part 1 will focus on denotational and operational semantics of sequential programming languages
2. Part 2 will introduce formalisms to specify, design, implement, analyze, and prove properties of concurrent and distributed systems.

Objective

- To appreciate the inner meaning of programming languages
- To learn to design and analyze simple concurrent systems
- To acquire the skills necessary to verify systems correctness through software tools.

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, . . .)

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, ...)

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, . . .)

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, . . .)

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, ...)

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, . . .)

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, . . .)

Main Contents

- A refresh of discrete mathematics and proof techniques.
- Finite state automata and labeled transition systems.
- Denotational and operational semantics of simple programming languages.
- Domain theory and fixed points.
- Process Algebras and their models as transition systems.
- Behavioral equivalences as tools for abstracting from unwanted details and for minimizing systems.
- Temporal and modal logics, and verification techniques of systems properties based on model checking.
- Quantitative (probabilistic, stochastic) variants of process calculi and their equivalences.
- Recent developments on Network Aware Programming and Autonomic Computing (Klaim, SCEL, . . .)

Readings

Books/notes:

1. H.R. Nielson, F. Nielson: **Semantics with Applications: an Appetizer**, Springer, 2007 (old edition available online).
2. G. Winskel: **The Formal Semantics of Programming Languages**, MIT Press, 1993.
3. L. Aceto, A. Ingolfsdottir, K.G. Larsen and J. Srba: **Reactive Systems: Modelling, Specification and Verification**, Cambridge University Press, 2007.

For Part 1, see also:

1. G. Plotkin: **A Structural Approach to Operational Semantics**, University of Aarhus, tech.rep. DAIMI-FN-19 Available online
2. M. Hennessy: **Semantics of Programming Languages**, Wiley, 1990. Out of print, online copy available

For Part 2, see . . . later.

Readings

Books/notes:

1. H.R. Nielson, F. Nielson: **Semantics with Applications: an Appetizer**, Springer, 2007 (old edition available online).
2. G. Winskel: **The Formal Semantics of Programming Languages**, MIT Press, 1993.
3. L. Aceto, A. Ingolfsdottir, K.G. Larsen and J. Srba: **Reactive Systems: Modelling, Specification and Verification**, Cambridge University Press, 2007.

For Part 1, see also:

1. G. Plotkin: **A Structural Approach to Operational Semantics**, University of Aarhus, tech.rep. DAIMI-FN-19 Available online
2. M. Hennessy: **Semantics of Programming Languages**, Wiley, 1990. Out of print, online copy available

For Part 2, see . . . later.

Course Material & Exam

Course Material: On the web site; it will, e.g.,:

1. latest version of the slides
2. pdf of relevant papers
3. ...

Exam: Some ideas to discuss with you

Outline of the first part

1. Preliminaries
2. Formal semantics of regular expressions
3. A simple `while` language
4. Operational semantics of `while`
5. Denotational semantics of `while`
6. A taste of Domain Theory
7. A less simple programming language and its semantics

Outline of the second part

1. Headaches of Concurrent Programming
2. Operators for Concurrent Processes and Their Semantics
3. Behavioural Equivalences
4. Process Calculi
5. CCS: An Exemplar Process Algebra
6. Temporal Logics and Model Checking
7. Extensions of Process Calculi for Quantitative Analysis
8. Extensions of Process Calculi for Network Aware Programming and/or Autonomic Computing.

The Hard Life of Programmers (and students)

Thanks

Many of the slides that will be used for the first part of the course have been drafted by two colleagues at IMT:

Francesco Tiezzi and Valerio Senni

that are currently lecturing on the same topic, by relying on old notes of mine (unfortunately in Italian).

Some preliminary math

Set Notation

$A \subseteq B$ every element of A is in B

$A \subset B$ if $A \subseteq B$ and there is one element of B not in A

$A \subseteq B$ and $B \subseteq A$ implies $A = B$

$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ $\quad (\bigcup_{i \in I} A_i)$

$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$ $\quad (\bigcap_{i \in I} A_i)$

$A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$

$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$ *ordered pairs* $\quad (\times_{i=1}^{n} A_i)$

$2^A = \{X \mid X \subseteq A\}$ *powerset*

Set Notation

$A \subseteq B$ every element of A is in B

$A \subset B$ if $A \subseteq B$ and there is one element of B not in A

$A \subseteq B$ and $B \subseteq A$ implies $A = B$

$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ $\quad(\bigcup_{i \in I} A_i)$

$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$ $\quad(\bigcap_{i \in I} A_i)$

$A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$

$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$ *ordered pairs* $\quad(\times_{i=1}^{n} A_i)$

$2^A = \{X \mid X \subseteq A\}$ *powerset*

Relations

$R \subseteq A \times B$ is a relation on sets A and B $\qquad (R \subseteq \times_{i=1}^{n} A_i)$

$(a, b) \in R \;\equiv\; R(a, b) \;\equiv\; aRb$ *infix notation*

$Id_A = \{(a, a) \mid a \in A\}$ (identity)

$R^{-1} = \{(y, x) \mid (x, y) \in R\} \subseteq B \times A$ (inverse)

$R_1 \cdot R_2 = \{(x, z) \mid \exists y \in B.\ (x, y) \in R_1 \wedge (y, z) \in R_2\} \subseteq A \times C$ (composition)

Some basic constructions:

$$
\begin{array}{lcl}
R^0 & = & Id_A \\
R^{n+1} & = & R \cdot R^n \\
R^* & = & \bigcup_{n \geq 0} R^n \\
R^+ & = & \bigcup_{n \geq 1} R^n
\end{array}
$$

Note that: $\quad R^1 = R \cdot R^0 = R, \qquad R^* = Id_A \cup R^+$ and

$R^+ = \{(x, y) \mid \exists n, \exists x_1, \ldots, x_n$ with $x_i R x_{i+1}\ (1 \leq i \leq n-1),\ x_1 = x,\ x_n = y\}$

Relations

$R \subseteq A \times B$ is a relation on sets A and B $\qquad (R \subseteq \times_{i=1}^{n} A_i)$

$(a, b) \in R \;\equiv\; R(a, b) \;\equiv\; aRb$ *infix notation*

$Id_A = \{(a, a) \mid a \in A\}$ (identity)

$R^{-1} = \{(y, x) \mid (x, y) \in R\} \subseteq B \times A$ (inverse)

$R_1 \cdot R_2 = \{(x, z) \mid \exists y \in B.\ (x, y) \in R_1 \wedge (y, z) \in R_2\} \subseteq A \times C$ (composition)

Some basic constructions:

$$
\begin{array}{lcl}
R^0 & = & Id_A \\
R^{n+1} & = & R \cdot R^n \\
R^* & = & \bigcup_{n \geq 0} R^n \\
R^+ & = & \bigcup_{n \geq 1} R^n
\end{array}
$$

Note that: $R^1 = R \cdot R^0 = R$, $\quad R^* = Id_A \cup R^+$ and

$R^+ = \{(x, y) \mid \exists n, \exists x_1, \ldots, x_n$ with $x_i R x_{i+1}\ (1 \leq i \leq n-1),\ x_1 = x,\ x_n = y\}$

Relations

$R \subseteq A \times B$ is a relation on sets A and B $\qquad (R \subseteq \times_{i=1}^{n} A_i)$

$(a, b) \in R \;\equiv\; R(a, b) \;\equiv\; aRb$ *infix notation*

$Id_A = \{(a, a) \mid a \in A\}$ (identity)

$R^{-1} = \{(y, x) \mid (x, y) \in R\} \subseteq B \times A$ (inverse)

$R_1 \cdot R_2 = \{(x, z) \mid \exists y \in B.\ (x, y) \in R_1 \wedge (y, z) \in R_2\} \subseteq A \times C$ (composition)

Some basic constructions:

$$
\begin{array}{lcl}
R^0 & = & Id_A \\
R^{n+1} & = & R \cdot R^n \\
R^* & = & \bigcup_{n \geq 0} R^n \\
R^+ & = & \bigcup_{n \geq 1} R^n
\end{array}
$$

Note that: $R^1 = R \cdot R^0 = R$, $\quad R^* = Id_A \cup R^+$ and

$R^+ = \{(x, y) \mid \exists n, \exists x_1, \ldots, x_n \text{ with } x_i R x_{i+1}\ (1 \leq i \leq n-1),\ x_1 = x,\ x_n = y\}$

Properties of Relations

Binary Relations

A binary relation $R \subseteq A \times A$ is (same set A)

reflexive: if $\forall x \in A, (x, x) \in R$,
symmetric: if $\forall x, y \in A, (x, y) \in R \Rightarrow (y, x) \in R$,
antisymmetric: if $\forall x, y \in A, (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$;
transitive: if $\forall x, y, z \in A, (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Closure of Relations

$S = R \cup Id_A$ the reflexive closure of R
$S = R \cup R^{-1}$ the symmetric closure of R
$S = R^+$ the transitive closure of R
$S = R^*$ the reflexive and transitive closure of R

Properties of Relations

Binary Relations

A binary relation $R \subseteq A \times A$ is (same set A)

reflexive:	if $\forall x \in A, (x,x) \in R$,
symmetric:	if $\forall x, y \in A, (x,y) \in R \Rightarrow (y,x) \in R$,
antisymmetric:	if $\forall x, y \in A, (x,y) \in R \wedge (y,x) \in R \Rightarrow x = y$;
transitive:	if $\forall x, y, z \in A, (x,y) \in R \wedge (y,z) \in R \Rightarrow (x,z) \in R$

Closure of Relations

$S = R \cup Id_A$	the reflexive closure of R
$S = R \cup R^{-1}$	the symmetric closure of R
$S = R^+$	the transitive closure of R
$S = R^*$	the reflexive and transitive closure of R

Special Relations

A relation R is

- an **order** if it is reflexive, antisymmetric and transitive
- an **equivalence** if it is reflexive, symmetric and transitive
- a **preorder** if it is reflexive and transitive

Kernel relation

- Given a preorder R its kernel, defined as $K = R \cap R^{-1}$, is an equivalence relation

Special Relations

A relation R is

- an **order** if it is reflexive, antisymmetric and transitive
- an **equivalence** if it is reflexive, symmetric and transitive
- a **preorder** if it is reflexive and transitive

Kernel relation

- Given a preorder R its kernel, defined as $K = R \cap R^{-1}$, is an equivalence relation

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$R(7, 7)$, (R,)

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$R(7, 7)$, (R,)

$1 = 1$

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$R(7,7)$, $R(7{,}1)$, $R(1,7)$, (R,S,)

$1 = 1$

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$R(7,7)$, $R(7,1)$, $R(1,7)$, (R,S,)

1=1 1 1

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$R(7,7)$, $R(7{,}1)$, $R(1,7)$, $R(7{,}10)$, $R(1{,}10)$ (R,S,T)

1=1 1 1

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \text{ mod } 3\}$

$R(7, 7)$, $R(7,1)$, $R(1, 7)$, $R(7,10)$, $R(1,10)$ (R,S,T)

1=1 1 1 1 1 1

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$R(7,7)$, $R(7,1)$, $R(1,7)$, $R(7,10)$, $R(1,10)$ (R,S,T)

1=1 1 1 1 1 1

$[0] = \{0, 3, 6, 9, \ldots\}$

$[1] = \{1, 4, 7, 10, \ldots\}$

$[2] = \{2, 5, 8, 11, \ldots\}$

equivalence classes:

- have a representative

- are disjoint

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N}\times\mathbb{N} \mid (x = y) \bmod 3\}$

$R(7,7)$, $R(7,1)$, $R(1,7)$, $R(7,10)$, $R(1,10)$ (R,S,T)

1=1 1 1 1 1 1

$[0] = \{0, 3, 6, 9, \ldots\}$ *equivalence classes:*

$[1] = \{1, 4, 7, 10, \ldots\}$ *- have a representative*

$[2] = \{2, 5, 8, 11, \ldots\}$ *- are disjoint*

An **equivalence class** is a subset C of A such that

$x, y \in C \Rightarrow (x, y) \in R$ *consistent* and

$x \in C \wedge (x, y) \in R \Rightarrow y \in C$ *saturated*

Equivalence Classes and Quotient Set

Examples of **equivalence relations**: $R \subseteq A \times A$ (reflexive, symmetric, transitive)

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$R(7,7)$, $R(7{,}1)$, $R(1,7)$, $R(7{,}10)$, $R(1{,}10)$ (R,S,T)

1=1 1 1 1 1 1

$[0] = \{0, 3, 6, 9, \ldots\}$ *equivalence classes:*

$[1] = \{1, 4, 7, 10, \ldots\}$ *- have a representative*

$[2] = \{2, 5, 8, 11, \ldots\}$ *- are disjoint*

The **quotient set** Q_A^R of A modulo R is a partition of A is the set of equivalence classes induced by R on A

Example: $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid (x = y) \bmod 3\}$

$Q_{\mathbb{N}}^R = \{\ [0],\ [1],\ [2]\ \}$

Functions

Partial Functions

A *partial function* is a relation $f \subseteq A \times B$ such that

$$\forall x, y, z.\ (x, y) \in f \wedge (x, z) \in f \Rightarrow y = z$$

We denote partial function by $\quad f : A \rightharpoondown B$

Total Functions

A (total) *function* is a partial function $f : A \rightharpoondown B$ such that

$$\forall x\ \exists y.\ (x, y) \in f$$

We denote total function by $\quad f : A \rightarrow B$

Functions (total or partial) can be *monotone*, *continuous*, *injective*, *surjective*, *bijective*, *invertible*...

Functions

Partial Functions

A *partial function* is a relation $f \subseteq A \times B$ such that

$$\forall x, y, z.\ (x, y) \in f \wedge (x, z) \in f \Rightarrow y = z$$

We denote partial function by $f : A \rightharpoondown B$

Total Functions

A (total) *function* is a partial function $f : A \rightharpoondown B$ such that

$$\forall x\ \exists y.\ (x, y) \in f$$

We denote total function by $f : A \rightarrow B$

Functions (total or partial) can be *monotone*, *continuous*, *injective*, *surjective*, *bijective*, *invertible*...

Functions

Partial Functions

A *partial function* is a relation $f \subseteq A \times B$ such that

$$\forall x, y, z.\ (x, y) \in f \wedge (x, z) \in f \Rightarrow y = z$$

We denote partial function by $f : A \rightharpoondown B$

Total Functions

A (total) *function* is a partial function $f : A \rightharpoondown B$ such that

$$\forall x\ \exists y.\ (x, y) \in f$$

We denote total function by $f : A \rightarrow B$

Functions (total or partial) can be *monotone*, *continuous*, *injective*, *surjective*, *bijective*, *invertible*...

Induction Principle

Mathematical Induction

To prove that $P(n)$ holds for every natural number $n \in \mathbb{N}$, prove

1. $P(0)$
2. for any $k \in \mathbb{N}$, $P(k)$ implies $P(k+1)$

Example: Show that $sum(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for every $n \in \mathbb{N}$

Induction Principle

Mathematical Induction

To prove that $P(n)$ holds for every natural number $n \in \mathbb{N}$, prove

1. $P(0)$
2. for any $k \in \mathbb{N}$, $P(k)$ implies $P(k+1)$

Example: Show that $sum(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for every $n \in \mathbb{N}$

Induction Principle

Mathematical Induction

To prove that $P(n)$ holds for every natural number $n \in \mathbb{N}$, prove

1. $P(0)$
2. for any $k \in \mathbb{N}$, $P(k)$ implies $P(k+1)$

Example: Show that $sum(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for every $n \in \mathbb{N}$

(1) $sum(0) = \frac{0(0+1)}{2} = 0$ *base case*

Induction Principle

Mathematical Induction

To prove that $P(n)$ holds for every natural number $n \in \mathbb{N}$, prove

1. $P(0)$
2. for any $k \in \mathbb{N}$, $P(k)$ implies $P(k+1)$

Example: Show that $sum(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for every $n \in \mathbb{N}$

(1) $sum(0) = \frac{0(0+1)}{2} = 0$ *base case*

(2) to show: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ implies $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$

assume $sum(n) = \frac{n(n+1)}{2}$, for a generic n

$sum(n+1) = sum(n) + (n+1) =$ *properties of summation*

$= \frac{n(n+1)}{2} + (n+1)$ *inductive hypothesis*

$= \frac{(n+1)(n+2)}{2}$ *qed*

Inductively Defined Sets

basis: the set I of initial elements of S
induction: rules R for constructing elements in S from elements in S
closure: S is the least set containing I and closed w.r.t. R

Natural numbers

$I = \{0\}, \quad R_1: \text{ if } X \in S \text{ then } s(X) \in S$

$S = \{0, s(0), s(s(0)), \ldots\}$

$S = Lists(\mathbb{N})$, lists of numbers in $\mathbb{N}$

$I = \{[\,]\}, \quad R_1: \text{ if } X \in S \text{ and } n \in \mathbb{N} \text{ then } [n|X] \in S$

$S = \{[\,], [0], [1], [2], \ldots, [0,0], [0,1], [0,2], \ldots, [1,0], [1,1], [1,2], \ldots\}$

n-ary trees

$I = \{\varepsilon\}, \quad R_1: \text{ if } X_1, \ldots, X_n \in S \text{ then } t(X_1, \ldots, X_n) \in S$

$S = \{\varepsilon, t(\varepsilon), t(\varepsilon, \varepsilon), \ldots, t(t(\varepsilon)), \ldots, t(\varepsilon, t(t(\varepsilon), \varepsilon), t(\varepsilon, \varepsilon, \varepsilon)), \ldots\}$

Inductively Defined Sets

basis: the set I of initial elements of S
induction: rules R for constructing elements in S from elements in S
closure: S is the least set containing I and closed w.r.t. R

Natural numbers

$I = \{0\}, \quad R_1: \text{ if } X \in S \text{ then } s(X) \in S$

$S = \{0, s(0), s(s(0)), \ldots\}$

$S = Lists(\mathbb{N})$, lists of numbers in $\mathbb{N}$

$I = \{[\,]\}, \quad R_1: \text{ if } X \in S \text{ and } n \in \mathbb{N} \text{ then } [n|X] \in S$

$S = \{[\,], [0], [1], [2], \ldots, [0,0], [0,1], [0,2], \ldots, [1,0], [1,1], [1,2], \ldots\}$

n-ary trees

$I = \{\varepsilon\}, \quad R_1: \text{ if } X_1, \ldots, X_n \in S \text{ then } t(X_1, \ldots, X_n) \in S$

$S = \{\varepsilon, t(\varepsilon), t(\varepsilon, \varepsilon), \ldots, t(t(\varepsilon)), \ldots, t(\varepsilon, t(t(\varepsilon), \varepsilon), t(\varepsilon, \varepsilon, \varepsilon)), \ldots\}$

Inductively Defined Sets

basis: the set I of initial elements of S
induction: rules R for constructing elements in S from elements in S
closure: S is the least set containing I and closed w.r.t. R

Natural numbers

$I = \{0\}, \quad R_1: \text{ if } X \in S \text{ then } s(X) \in S$

$S = \{0, s(0), s(s(0)), \ldots\}$

$S = Lists(\mathbb{N})$, lists of numbers in $\mathbb{N}$

$I = \{[\,]\}, \quad R_1: \text{ if } X \in S \text{ and } n \in \mathbb{N} \text{ then } [n|X] \in S$

$S = \{[\,], [0], [1], [2], \ldots, [0,0], [0,1], [0,2], \ldots, [1,0], [1,1], [1,2], \ldots\}$

n-ary trees

$I = \{\varepsilon\}, \quad R_1: \text{ if } X_1, \ldots, X_n \in S \text{ then } t(X_1, \ldots, X_n) \in S$

$S = \{\varepsilon, t(\varepsilon), t(\varepsilon,\varepsilon), \ldots, t(t(\varepsilon)), \ldots, t(\varepsilon, t(t(\varepsilon),\varepsilon), t(\varepsilon,\varepsilon,\varepsilon)), \ldots\}$

Inductively Defined Sets

basis: the set I of initial elements of S
induction: rules R for constructing elements in S from elements in S
closure: S is the least set containing I and closed w.r.t. R

Natural numbers

$I = \{0\}, \quad R_1: \text{ if } X \in S \text{ then } s(X) \in S$

$S = \{0, s(0), s(s(0)), \ldots\}$

$S = Lists(\mathbb{N})$, lists of numbers in $\mathbb{N}$

$I = \{[\,]\}, \quad R_1: \text{ if } X \in S \text{ and } n \in \mathbb{N} \text{ then } [n|X] \in S$

$S = \{[\,], [0], [1], [2], \ldots, [0,0], [0,1], [0,2], \ldots, [1,0], [1,1], [1,2], \ldots\}$

n-ary trees

$I = \{\varepsilon\}, \quad R_1: \text{ if } X_1, \ldots, X_n \in S \text{ then } t(X_1, \ldots, X_n) \in S$

$S = \{\varepsilon, t(\varepsilon), t(\varepsilon, \varepsilon), \ldots, t(t(\varepsilon)), \ldots, t(\varepsilon, t(t(\varepsilon), \varepsilon), t(\varepsilon, \varepsilon, \varepsilon)), \ldots\}$

Structural Induction

Let us consider a set S inductively defined by a set $C = \{c_1, \ldots, c_n\}$ of constructors of arity $\{a_1, \ldots, a_n\}$ with

- $I = \{c_i(\,) \mid a_i = 0\}$
- $R_i: \quad \text{if} \quad X_1, \ldots, X_{a_i} \in S \quad \text{then} \quad c_i(X_1, \ldots, X_{a_i}) \in S$

To prove that $P(x)$ holds for every $x \in S$, it is sufficient to prove that

$$P(s_1), \ldots, P(s_k) \implies P\big(c_k(s_1, \ldots, s_k)\big)$$

- for every constructor $c_k \in C$ and
- for every $s_1, \ldots, s_k \in S$, where k is the arity of c_k

Notice that the base case is the one dealing with constructors of arity 0 i.e. with constants.

Structural Induction – exercise

Prove that $\mathit{sum}(\ell) \leq \mathit{max}(\ell) * \mathit{len}(\ell)$, for every $\ell \in \mathit{Lists}(\mathbb{N})$

where

- $\mathit{sum}(\ell)$ is the sum of the elements in the list
- ℓ, $\mathit{max}(\ell)$ is the greatest element in ℓ (with $\mathit{max}([\,]) = 0$)
- $\mathit{len}(\ell)$ is the number of elements in ℓ.

Structural Induction – exercise

Exercise: prove $sum(\ell) \leq max(\ell) * len(\ell)$, for every $\ell \in Lists(\mathbb{N})$

$$
\begin{array}{ll}
sum([\,]) = 0 & len([\,]) = 0 \\
sum([n|X]) = n + sum(X) & len([n|X]) = 1 + len(X) \\
max([]) = 0 & \\
max([n|X]) = n & \text{if } max(X) \leq n \\
max([n|X]) = max(X) & \text{if } n < max(X)
\end{array}
$$

Structural Induction – exercise

Exercise: prove $sum(\ell) \leq max(\ell) * len(\ell)$, for every $\ell \in Lists(\mathbb{N})$

$$
\begin{array}{ll}
sum([\,]) = 0 & len([\,]) = 0 \\
sum([n|X]) = n + sum(X) & len([n|X]) = 1 + len(X) \\
max([]) = 0 & \\
max([n|X]) = n & \text{if } max(X) \leq n \\
max([n|X]) = max(X) & \text{if } n < max(X)
\end{array}
$$

(1) $sum([\,]) \leq max([\,]) * len([\,])$

$0 \leq 0 * 0$ *applying definitions*

Structural Induction – exercise

Exercise: prove $sum(\ell) \leq max(\ell) * len(\ell)$, for every $\ell \in Lists(\mathbb{N})$

$$
\begin{array}{ll}
sum([\,]) = 0 & len([\,]) = 0 \\
sum([n|X]) = n + sum(X) & len([n|X]) = 1 + len(X) \\
max([]) = 0 & \\
max([n|X]) = n & \text{if } max(X) \leq n \\
max([n|X]) = max(X) & \text{if } n < max(X)
\end{array}
$$

(1) $sum([\,]) \leq max([\,]) * len([\,])$

$0 \leq 0 * 0$ *applying definitions*

(2) assume $sum(\ell) \leq max(\ell) * len(\ell)$ *inductive hyp.*

prove $sum([n|\ell]) \leq max([n|\ell]) * len([n|\ell])$ for any $n \in \mathbb{N}$

Structural Induction – exercise

Exercise: prove $sum(\ell) \leq max(\ell) * len(\ell)$, for every $\ell \in Lists(\mathbb{N})$

$$
\begin{array}{lll}
sum([\,]) = 0 & len([\,]) = 0 & \\
sum([n|X]) = n + sum(X) & len([n|X]) = 1 + len(X) & \\
max([]) = 0 & & \\
max([n|X]) = n & \text{if } max(X) \leq n & \text{(a)} \\
max([n|X]) = max(X) & \text{if } n < max(X) &
\end{array}
$$

(1) $sum([\,]) \leq max([\,]) * len([\,])$

$0 \leq 0 * 0$ *applying definitions*

(2) assume $sum(\ell) \leq max(\ell) * len(\ell)$ *inductive hyp.*

prove $sum([n|\ell]) \leq max([n|\ell]) * len([n|\ell])$ for any $n \in \mathbb{N}$

(a) $n + sum(\ell) \leq n * (1 + len(\ell))$ if $max(\ell) \leq n$ *applying definitions*

$sum(\ell) \quad \leq_{hyp} \quad max(\ell) * len(\ell) \quad \leq_{(a)} \quad n * len(\ell)$ *QED*

Structural Induction – exercise

Exercise: prove $sum(\ell) \leq max(\ell) * len(\ell)$, for every $\ell \in Lists(\mathbb{N})$

$$
\begin{array}{ll}
sum([\,]) = 0 & len([\,]) = 0 \\
sum([n|X]) = n + sum(X) & len([n|X]) = 1 + len(X)
\end{array}
$$

$$
\begin{array}{lll}
max([]) = 0 & & \\
max([n|X]) = n & \text{if } max(X) \leq n & \text{(a)} \\
max([n|X]) = max(X) & \text{if } n < max(X) & \text{(b)}
\end{array}
$$

(1) $sum([\,]) \leq max([\,]) * len([\,])$

$0 \leq 0 * 0$ *applying definitions*

(2) assume $sum(\ell) \leq max(\ell) * len(\ell)$ *inductive hyp.*

prove $sum([n|\ell]) \leq max([n|\ell]) * len([n|\ell])$ for any $n \in \mathbb{N}$

(a) $n + sum(\ell) \leq n * (1 + len(\ell))$ if $max(\ell) \leq n$ *applying definitions*

$sum(\ell) \leq_{hyp} max(\ell) * len(\ell) \leq_{(a)} n * len(\ell)$ *QED*

(b) $n + sum(\ell) \leq max(\ell) + max(\ell) * len(\ell))$ if $n < max(\ell)$ *applying definitions*

$A \leq B$ and $C \leq D$ imply $A + C \leq B + D$ *QED*

Inference Systems

1. I can be written as $\dfrac{}{t}$ (for any $t \in I$)
2. R_i can be written as $\dfrac{p_1 \quad \cdots \quad p_n}{q}$

Meaning: $\vdash t$ and if $\vdash p_1, \ldots, \vdash p_n$ then $\vdash q$

Example: rational numbers $\mathbb{Q}$

$$\frac{}{0 \in N} \qquad \frac{}{1 \in D} \qquad \frac{k \in N}{k+1 \in N} \qquad \frac{k \in D}{k+1 \in D} \qquad \frac{k \in N, \quad h \in D}{k/h \in \mathbb{Q}}$$

A derivation:

$$\dfrac{\dfrac{\dfrac{}{0 \in N}}{1 \in N} \quad \dfrac{\dfrac{}{1 \in D}}{2 \in D}}{1/2 \in \mathbb{Q}}$$

$\boxed{\vdash 1/2 \in \mathbb{Q}}$

Question: why do we need the rules in Red?

Inference Systems

1. I can be written as $\dfrac{}{t}$ (for any $t \in I$)
2. R_i can be written as $\dfrac{p_1 \quad \cdots \quad p_n}{q}$

Meaning: $\vdash t$ and if $\vdash p_1, \ldots, \vdash p_n$ then $\vdash q$

Example: rational numbers $\mathbb{Q}$

$$\frac{}{0 \in N} \qquad \frac{}{1 \in D} \qquad \frac{k \in N}{k+1 \in N} \qquad \frac{k \in D}{k+1 \in D} \qquad \frac{k \in N, \quad h \in D}{k/h \in \mathbb{Q}}$$

A derivation:
$$\dfrac{\dfrac{\dfrac{}{0 \in N}}{1 \in N} \quad \dfrac{\dfrac{}{1 \in D}}{2 \in D}}{1/2 \in \mathbb{Q}}$$

$\boxed{\vdash 1/2 \in \mathbb{Q}}$

Question: why do we need the rules in Red?

Inference Systems

1. I can be written as $\dfrac{}{t}$ (for any $t \in I$)
2. R_i can be written as $\dfrac{p_1 \quad \cdots \quad p_n}{q}$

Meaning: $\vdash t$ and if $\vdash p_1, \ldots, \vdash p_n$ then $\vdash q$

Example: rational numbers $\mathbb{Q}$

$$\frac{}{0 \in N} \qquad \frac{}{1 \in D} \qquad \frac{k \in N}{k+1 \in N} \qquad \frac{k \in D}{k+1 \in D} \qquad \frac{k \in N, \quad h \in D}{k/h \in \mathbb{Q}}$$

A derivation:

$$\dfrac{\dfrac{\dfrac{}{0 \in N}}{1 \in N} \qquad \dfrac{\dfrac{}{1 \in D}}{2 \in D}}{1/2 \in \mathbb{Q}}$$

$$\boxed{\vdash 1/2 \in \mathbb{Q}}$$

Question: why do we need the rules in Red?

More on Inductively Defined Sets

- $S_{I,R} = \{x \mid \vdash x\}$ *the set of all* *finitely* *derivable elements*
- $R(X) = \{y \mid \dfrac{x_1 \quad \cdots \quad x_n}{y} \text{ and } x_1, \ldots x_n \in X\}$ *one step derivation*

X is **closed** under R if $R(X) \subseteq X$ *called a (pre-)fixed point*

R is **monotonic** if $A \subseteq B \;\Rightarrow\; R(A) \subseteq R(B)$

$$
\begin{array}{lclcl}
S^0 & = & R^0(\emptyset) & = & \emptyset \\
S^1 & = & R^1(\emptyset) & = & R(\emptyset) \\
S^2 & = & R^2(\emptyset) & = & R(R(\emptyset)) \\
 & \vdots & & & \\
S & \triangleq & \bigcup_{i \in \mathbb{N}} S^i & &
\end{array}
$$

$S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots$

$\boxed{S \text{ closed under } R \qquad R(S) = S \qquad S \text{ least } R\text{-closed set}}$

more on fixpoints to come...

More on Inductively Defined Sets

- $S_{I,R} = \{x \mid \vdash x\}$ *the set of all finitely derivable elements*
- $R(X) = \{y \mid \dfrac{x_1 \ \cdots \ x_n}{y} \text{ and } x_1, \ldots x_n \in X\}$ *one step derivation*

X is **closed** under R if $R(X) \subseteq X$ *called a (pre-)fixed point*

R is **monotonic** if $A \subseteq B \Rightarrow R(A) \subseteq R(B)$

$$\begin{array}{lclcl} S^0 & = & R^0(\emptyset) & = & \emptyset \\ S^1 & = & R^1(\emptyset) & = & R(\emptyset) \\ S^2 & = & R^2(\emptyset) & = & R(R(\emptyset)) \\ & \vdots & & & \\ S & \triangleq & \bigcup_{i \in \mathbb{N}} S^i & & \end{array}$$

$$S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots$$

$$\boxed{S \text{ closed under } R \qquad R(S) = S \qquad S \text{ least } R\text{-closed set}}$$

more on fixpoints to come...

Constructing Inductively Defined Sets – an example

$fib(0) = 0$
$fib(1) = 1$
$fib(n+2) = fib(n+1) + fib(n)$

$fib : \mathbb{N} \to \mathbb{N}$

$$\frac{}{(0,0) \in Fib} \qquad \frac{}{(1,1) \in Fib} \qquad \frac{(n+1,a) \in Fib \quad (n,b) \in Fib}{(n+2, a+b) \in Fib}$$

$$R(X) = \{y \mid \frac{x_1 \ \cdots \ x_n}{y} \text{ and } x_1, \ldots x_n \in X\}$$

one step derivation

$$\begin{array}{lllll}
S^0 & = & R(\emptyset) & = & \emptyset \\
S^1 & = & R(S^0) & = & \{(0,0),(1,1)\} \\
S^2 & = & R(S^1) & = & \{(0,0),(1,1),(2,1)\} \\
S^3 & = & R(S^2) & = & \{(0,0),(1,1),(2,1),(3,2)\} \\
S^4 & = & R(S^3) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3)\} \\
S^5 & = & R(S^4) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5)\} \\
S^6 & = & R(S^5) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8)\} \\
S^7 & = & R(S^6) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8),(7,13)\} \\
 & \vdots & & &
\end{array}$$

$\boxed{S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots}$

a sequence of partial functions (under-) approximating *fib*

Constructing Inductively Defined Sets – an example

$fib(0) = 0$
$fib(1) = 1$
$fib(n+2) = fib(n+1) + fib(n)$

$fib : \mathbb{N} \rightarrow \mathbb{N}$

$$\frac{}{(0,0) \in Fib} \qquad \frac{}{(1,1) \in Fib} \qquad \frac{(n+1,a) \in Fib \quad (n,b) \in Fib}{(n+2, a+b) \in Fib}$$

$$R(X) = \{y \mid \frac{x_1 \quad \cdots \quad x_n}{y} \text{ and } x_1, \ldots x_n \in X\}$$

one step derivation

$$\begin{array}{lclcl}
S^0 & = & R(\emptyset) & = & \emptyset \\
S^1 & = & R(S^0) & = & \{(0,0),(1,1)\} \\
S^2 & = & R(S^1) & = & \{(0,0),(1,1),(2,1)\} \\
S^3 & = & R(S^2) & = & \{(0,0),(1,1),(2,1),(3,2)\} \\
S^4 & = & R(S^3) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3)\} \\
S^5 & = & R(S^4) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5)\} \\
S^6 & = & R(S^5) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8)\} \\
S^7 & = & R(S^6) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8),(7,13)\} \\
 & \vdots & & &
\end{array}$$

$$\boxed{S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots}$$

a sequence of partial functions (under-) approximating *fib*

Constructing Inductively Defined Sets – an example

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n+2) = fib(n+1) + fib(n)$$

$$fib : \mathbb{N} \to \mathbb{N}$$

$$\frac{}{(0,0) \in Fib} \qquad \frac{}{(1,1) \in Fib} \qquad \frac{(n+1, a) \in Fib \quad (n, b) \in Fib}{(n+2, a+b) \in Fib}$$

$$R(X) = \{y \mid \frac{x_1 \ \cdots \ x_n}{y} \text{ and } x_1, \ldots x_n \in X\}$$

one step derivation

$$\begin{array}{lllll}
S^0 & = & R(\emptyset) & = & \emptyset \\
S^1 & = & R(S^0) & = & \{(0,0), (1,1)\} \\
S^2 & = & R(S^1) & = & \{(0,0), (1,1), (2,1)\} \\
S^3 & = & R(S^2) & = & \{(0,0), (1,1), (2,1), (3,2)\} \\
S^4 & = & R(S^3) & = & \{(0,0), (1,1), (2,1), (3,2), (4,3)\} \\
S^5 & = & R(S^4) & = & \{(0,0), (1,1), (2,1), (3,2), (4,3), (5,5)\} \\
S^6 & = & R(S^5) & = & \{(0,0), (1,1), (2,1), (3,2), (4,3), (5,5), (6,8)\} \\
S^7 & = & R(S^6) & = & \{(0,0), (1,1), (2,1), (3,2), (4,3), (5,5), (6,8), (7,13)\} \\
 & \vdots & & &
\end{array}$$

$$\boxed{S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots}$$

a sequence of partial functions (under-) approximating *fib*

Constructing Inductively Defined Sets – an example

$fib(0) = 0$
$fib(1) = 1$
$fib(n+2) = fib(n+1) + fib(n)$

$fib : \mathbb{N} \to \mathbb{N}$

$$\frac{}{(0,0) \in Fib} \qquad \frac{}{(1,1) \in Fib} \qquad \frac{(n+1,a) \in Fib \quad (n,b) \in Fib}{(n+2,a+b) \in Fib}$$

$$R(X) = \{y \mid \frac{x_1 \ \cdots \ x_n}{y} \text{ and } x_1,\ldots x_n \in X\}$$

one step derivation

$$\begin{array}{lllll}
S^0 & = & R(\emptyset) & = & \emptyset \\
S^1 & = & R(S^0) & = & \{(0,0),(1,1)\} \\
S^2 & = & R(S^1) & = & \{(0,0),(1,1),(2,1)\} \\
S^3 & = & R(S^2) & = & \{(0,0),(1,1),(2,1),(3,2)\} \\
S^4 & = & R(S^3) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3)\} \\
S^5 & = & R(S^4) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5)\} \\
S^6 & = & R(S^5) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8)\} \\
S^7 & = & R(S^6) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8),(7,13)\} \\
 & \vdots & & &
\end{array}$$

a sequence of partial functions (under-) approximating *fib*

$\boxed{S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots}$

Constructing Inductively Defined Sets – an example

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n+2) = fib(n+1) + fib(n)$$

$$fib : \mathbb{N} \to \mathbb{N}$$

$$\frac{}{(0,0) \in Fib} \qquad \frac{}{(1,1) \in Fib} \qquad \frac{(n+1,a) \in Fib \quad (n,b) \in Fib}{(n+2, a+b) \in Fib}$$

$$R(X) = \{y \mid \frac{x_1 \ \cdots \ x_n}{y} \text{ and } x_1, \ldots x_n \in X\}$$

one step derivation

$$\begin{array}{llll}
S^0 & = & R(\emptyset) & = & \emptyset \\
S^1 & = & R(S^0) & = & \{(0,0),(1,1)\} \\
S^2 & = & R(S^1) & = & \{(0,0),(1,1),(2,1)\} \\
S^3 & = & R(S^2) & = & \{(0,0),(1,1),(2,1),(3,2)\} \\
S^4 & = & R(S^3) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3)\} \\
S^5 & = & R(S^4) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5)\} \\
S^6 & = & R(S^5) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8)\} \\
S^7 & = & R(S^6) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8),(7,13)\} \\
 & \vdots & & &
\end{array}$$

a sequence of partial functions (under-) approximating *fib*

$$\boxed{S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots}$$

Constructing Inductively Defined Sets – an example

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n+2) = fib(n+1) + fib(n)$$

$$fib : \mathbb{N} \to \mathbb{N}$$

$$\frac{}{(0,0) \in Fib} \qquad \frac{}{(1,1) \in Fib} \qquad \frac{(n+1,a) \in Fib \quad (n,b) \in Fib}{(n+2, a+b) \in Fib}$$

$$R(X) = \{y \mid \frac{x_1 \ \cdots \ x_n}{y} \text{ and } x_1, \ldots x_n \in X\}$$

one step derivation

$$\begin{array}{llll}
S^0 & = & R(\emptyset) & = & \emptyset \\
S^1 & = & R(S^0) & = & \{(0,0),(1,1)\} \\
S^2 & = & R(S^1) & = & \{(0,0),(1,1),(2,1)\} \\
S^3 & = & R(S^2) & = & \{(0,0),(1,1),(2,1),(3,2)\} \\
S^4 & = & R(S^3) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3)\} \\
S^5 & = & R(S^4) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5)\} \\
S^6 & = & R(S^5) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8)\} \\
S^7 & = & R(S^6) & = & \{(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8),(7,13)\} \\
 & \vdots & & &
\end{array}$$

$$\boxed{S^0 \subseteq S^1 \subseteq S^2 \subseteq \ldots}$$

a sequence of partial functions (under-) approximating *fib*

$$S \triangleq \bigcup_{i \in \mathbb{N}} S^i$$

this limit is exactly the (total) function *fib*

Languages

Strings over an alphabet

Let Γ be an alphabet (a finite nonempty set of symbols). The set *Strings*(Γ) is inductively defined as follows:

- $I = \Gamma \cup \{\varepsilon\}$,
- R_1 : if $x, y \in Strings(\Gamma)$ then $xy \in Strings(\Gamma)$
- xy is the concatenation of the strings x and y $(\varepsilon x = x\varepsilon = x)$
- Notation: $\Gamma^* = Strings(\Gamma)$ star closure of an alphabet

An example

$\Gamma = \{a, b\}, \quad Strings(\Gamma) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$

Languages

- A **language** on Γ is any subset $L \subseteq \Gamma^*$
- Languages can be defined inductively through formal grammars

Languages

Strings over an alphabet

Let Γ be an alphabet (a finite nonempty set of symbols). The set *Strings*(Γ) is inductively defined as follows:

- $I = \Gamma \cup \{\varepsilon\}$,
- R_1 : if $x, y \in Strings(\Gamma)$ then $xy \in Strings(\Gamma)$
- xy is the concatenation of the strings x and y $(\varepsilon x = x\varepsilon = x)$
- Notation: $\Gamma^* = Strings(\Gamma)$ star closure of an alphabet

An example

$\Gamma = \{a, b\}, \quad Strings(\Gamma) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$

Languages

- A **language** on Γ is any subset $L \subseteq \Gamma^*$
- Languages can be defined inductively through formal grammars

Languages

Strings over an alphabet

Let Γ be an alphabet (a finite nonempty set of symbols). The set $\mathit{Strings}(\Gamma)$ is inductively defined as follows:

- $I = \Gamma \cup \{\varepsilon\}$,
- R_1 : if $x, y \in \mathit{Strings}(\Gamma)$ then $xy \in \mathit{Strings}(\Gamma)$
- xy is the concatenation of the strings x and y $(\varepsilon x = x\varepsilon = x)$
- Notation: $\Gamma^* = \mathit{Strings}(\Gamma)$ star closure of an alphabet

An example

$\Gamma = \{a, b\}$, $\mathit{Strings}(\Gamma) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$

Languages

- A **language** on Γ is any subset $L \subseteq \Gamma^*$
- Languages can be defined inductively through formal grammars

Grammars

A **grammar** is a 4-tuple $G = \langle T, NT, S, P \rangle$ where

1. **terminals** T
2. **nonterminals** NT $(T \cap NT = \emptyset)$
3. **start symbol** $S \in NT$
4. **productions** $P \subseteq (T \cup NT)^* \times (T \cup NT)^*$

   if $(u, v) \in P$ then u has at least a nonterminal symbol

Grammars

A **grammar** is a 4-tuple $G = \langle T, NT, S, P \rangle$ where

1. **terminals** T
2. **nonterminals** NT $(T \cap NT = \emptyset)$
3. **start symbol** $S \in NT$
4. **productions** $P \subseteq (T \cup NT)^* \times (T \cup NT)^*$
   if $(u, v) \in P$ then u has at least a nonterminal symbol

(u, v) is also written as $u \rightarrow v$

Grammars

A **grammar** is a 4-tuple $G = \langle T, NT, S, P \rangle$ where

1. **terminals** T
2. **nonterminals** NT $\quad (T \cap NT = \emptyset)$
3. **start symbol** $S \in NT$
4. **productions** $P \subseteq (T \cup NT)^* \times (T \cup NT)^*$
   if $(u, v) \in P$ then u has at least a nonterminal symbol

(u, v) is also written as $u \rightarrow v$

$(u, v_1), (u, v_2), \ldots, (u, v_n) \in P$ also written as

$u \rightarrow v_1 \mid v_2 \mid \ldots \mid v_n$

or

$u ::= v_1 \mid v_2 \mid \ldots \mid v_n$ *Backus-Naur Normal Form (BNF)*

Grammars – derivation relation

$G = \langle T, N, S, P \rangle$

$$\frac{s = lur \quad t = lvr \quad u \rightarrow v}{s \Rightarrow t}$$

for any production $u \rightarrow v$ in P

$\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$

Grammars and Languages

The language generated by G is the following set of string of terminal symbols

$$L(G) = \{ w \in T^* \mid S \Rightarrow^* w \}$$

Grammars – example

$T = \{a, b, c\}$ $\quad N = \{S, B\}$ $\quad$ start symbol: S

$S \to aBSc \mid abc \qquad Ba \to aB \qquad Bb \to bb$

Grammars – example

$T = \{a, b, c\}$ $\quad N = \{S, B\}$ $\quad$ start symbol: S

$S \to aBSc \mid abc \qquad Ba \to aB \qquad Bb \to bb$

A derivation:

$\underline{S}$

Grammars – example

$T = \{a, b, c\}$ $\quad N = \{S, B\}$ $\quad$ start symbol: S

$S \rightarrow aBSc \mid abc \qquad Ba \rightarrow aB \qquad Bb \rightarrow bb$

A derivation:

$\underline{S} \Rightarrow aB\underline{S}c$

Grammars – example

$T = \{a, b, c\}$ $N = \{S, B\}$ start symbol: S

$S \rightarrow aBSc \mid abc \qquad Ba \rightarrow aB \qquad Bb \rightarrow bb$

A derivation:

$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc$

Grammars – example

$T = \{a, b, c\} \quad N = \{S, B\}$ start symbol: S

$S \rightarrow aBSc \mid abc \qquad Ba \rightarrow aB \qquad Bb \rightarrow bb$

A derivation:

$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc \Rightarrow a\underline{Ba}Babccc$

Grammars – example

$T = \{a, b, c\} \quad N = \{S, B\}$ start symbol: S

$S \rightarrow aBSc \mid abc \qquad Ba \rightarrow aB \qquad Bb \rightarrow bb$

A derivation:

$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc \Rightarrow a\underline{Ba}Babccc \Rightarrow$

$\Rightarrow aaB\underline{Ba}bccc$

Grammars – example

$T = \{a, b, c\} \quad N = \{S, B\}$ start symbol: S

$S \to aBSc \mid abc \qquad Ba \to aB \qquad Bb \to bb$

A derivation:

$$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc \Rightarrow a\underline{Ba}Babccc \Rightarrow$$

$$\Rightarrow aaB\underline{Ba}bccc \Rightarrow aa\underline{Ba}Bbccc$$

Grammars – example

$T = \{a, b, c\} \quad N = \{S, B\}$ start symbol: S

$S \rightarrow aBSc \mid abc \qquad Ba \rightarrow aB \qquad Bb \rightarrow bb$

A derivation:

$$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc \Rightarrow a\underline{Ba}Babccc \Rightarrow$$
$$\Rightarrow aaB\underline{Ba}bccc \Rightarrow aa\underline{Ba}Bbccc \Rightarrow aaaB\underline{Bb}ccc$$

Grammars – example

$T = \{a, b, c\}$ $N = \{S, B\}$ start symbol: S

$S \rightarrow aBSc \mid abc$ $Ba \rightarrow aB$ $Bb \rightarrow bb$

A derivation:

$$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc \Rightarrow a\underline{Ba}Babccc \Rightarrow$$
$$\Rightarrow aaB\underline{Ba}bccc \Rightarrow aa\underline{Ba}Bbccc \Rightarrow aaaB\underline{Bb}ccc \Rightarrow$$
$$\Rightarrow aaa\underline{Bb}bccc$$

Grammars – example

$T = \{a, b, c\} \quad N = \{S, B\}$ start symbol: S

$S \to aBSc \mid abc \qquad Ba \to aB \qquad Bb \to bb$

A derivation:

$$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc \Rightarrow a\underline{Ba}Babccc \Rightarrow$$

$$\Rightarrow aaB\underline{Ba}bccc \Rightarrow aa\underline{Ba}Bbccc \Rightarrow aaaB\underline{Bb}ccc \Rightarrow$$

$$\Rightarrow aaa\underline{Bb}bccc \Rightarrow aaabbbccc \in \{a, b, c\}^*$$

Grammars – example

$T = \{a, b, c\}$ $N = \{S, B\}$ start symbol: S

$S \rightarrow aBSc \mid abc$ $Ba \rightarrow aB$ $Bb \rightarrow bb$

A derivation:

$$\underline{S} \Rightarrow aB\underline{S}c \Rightarrow aBaB\underline{S}cc \Rightarrow a\underline{Ba}Babccc \Rightarrow$$
$$\Rightarrow aaB\underline{Ba}bccc \Rightarrow aa\underline{Ba}Bbccc \Rightarrow aaaB\underline{Bb}ccc \Rightarrow$$
$$\Rightarrow aaa\underline{Bb}bccc \Rightarrow aaabbbccc \in \{a, b, c\}^*$$

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}$$

Abstract and Concrete Syntax

When providing the syntax of programming languages we need to worry about precedence of operators or grouping of statements to distinguish, e.g., between:

$$(3+4)*5 \quad \text{and} \quad 3+(4*5),$$

$$\textbf{while } p \textbf{ do } (c_1; c_2) \quad \text{and} \quad (\textbf{while } p \textbf{ do } c_1); c_2$$

Thus, e.g., for arithmetic expressions we have grammars with parenthesis:

$$E \quad ::= \quad n \mid (E) \mid E+E \mid E-E \mid E*E \mid E/E$$

or more elaborate grammars specifying the precedence of operators (like the next one . . .).

Abstract and Concrete Syntax

E	$::=$	$E + T \mid E - T \mid T$	(expressions)
T	$::=$	$T * P \mid T/P \mid P$	(terms)
P	$::=$	$N \mid (E)$	(atomic expressions)
N	$::=$	$D\,N \mid D$	(numbers)
D	$::=$	$0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$	(digits)

- When defining the semantics of programming languages, we are only concerned with the meaning of their constructs, not with the theory of how to write programs.
- We thus resort to abstract syntax that leaves us the task of adding enough parentheses to programs to ensure they can be built-up in a unique way.

Abstract syntax specifies the parse trees of a language; it is the job of concrete syntax to provide enough information through parentheses or precedence rules for a string to parse uniquely.

Abstract and Concrete Syntax

$$
\begin{array}{lcll}
E & ::= & E + T \mid E - T \mid T & \text{(expressions)} \\
T & ::= & T * P \mid T/P \mid P & \text{(terms)} \\
P & ::= & N \mid (E) & \text{(atomic expressions)} \\
N & ::= & D\,N \mid D & \text{(numbers)} \\
D & ::= & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 & \text{(digits)}
\end{array}
$$

- When defining the semantics of programming languages, we are only concerned with the meaning of their constructs, not with the theory of how to write programs.
- We thus resort to abstract syntax that leaves us the task of adding enough parentheses to programs to ensure they can be built-up in a unique way.

Abstract syntax specifies the parse trees of a language; it is the job of concrete syntax to provide enough information through parentheses or precedence rules for a string to parse uniquely.

From Parsing to Execution

Concrete Syntax $\xrightarrow{\textit{defines}}$ $\boxed{\text{Statements}}$ | $2 + (3 * 4)$

From Parsing to Execution

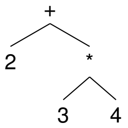
Concrete Syntax $\xrightarrow{\textit{defines}}$ $\boxed{\text{Statements}}$

$\downarrow$ Parse

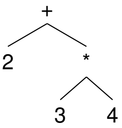
Abstract Syntax $\xrightarrow{\textit{defines}}$ $\boxed{\text{Syntax Trees}}$

$2 + (3 * 4)$

$\downarrow$



From Parsing to Execution

Concrete Syntax	$\xrightarrow{\textit{defines}}$	$\boxed{\text{Statements}}$	$2 + (3 * 4)$
		$\downarrow$ Parse	$\downarrow$
Abstract Syntax	$\xrightarrow{\textit{defines}}$	$\boxed{\text{Syntax Trees}}$	
		$\downarrow$ Execute	$\downarrow$
Semantics	$\xrightarrow{\textit{defines}}$	$\boxed{\text{Meaning of Syntax Trees}}$	14

Labelled Transition Systems

A labelled transition system is a 4-tuple $S = \langle Q, A, \rightarrow, q_0 \rangle$ such that

1. **states** Q
2. **actions** A
3. **transitions** $\rightarrow \subseteq Q \times A \times Q$

   $q \xrightarrow{a} q'$ denotes $(q, a, q') \in \rightarrow$
4. **initial state** $q_0 \in Q$

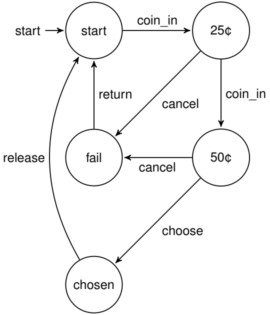
Labelled Transition Systems

A labelled transition system is a 4-tuple $S = \langle Q, A, \rightarrow, q_0 \rangle$ such that

1. **states** Q
2. **actions** A
3. **transitions** $\rightarrow \subseteq Q \times A \times Q$

   $q \xrightarrow{a} q'$ denotes $(q, a, q') \in \rightarrow$
4. **initial state** $q_0 \in Q$

Vending machine:



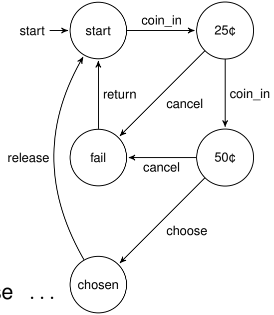
Labelled Transition Systems

A labelled transition system is a 4-tuple $S = \langle Q, A, \rightarrow, q_0 \rangle$ such that

1. **states** Q
2. **actions** A
3. **transitions** $\rightarrow \subseteq Q \times A \times Q$

   $q \xrightarrow{a} q'$ denotes $(q, a, q') \in \rightarrow$
4. **initial state** $q_0 \in Q$

Vending machine:



Semantics: traces

$\tau : a_0 a_1 a_2 a_3 a_4 a_5 a_6 \ldots$

τ : coin_in cancel return coin_in coin_in choose release ...

LTS-based Semantics of Arithmetic Expressions

$$\frac{m \circ n = k}{m \circ n \xrightarrow{\circ} k}\ \text{(op)} \qquad \frac{E_1 \xrightarrow{\circ'} E_1'}{E_1 \circ E_2 \xrightarrow{\circ'} E_1' \circ E_2}\ \text{(rl)} \qquad \frac{E_2 \xrightarrow{\circ'} E_2'}{E_1 \circ E_2 \xrightarrow{\circ'} E_1 \circ E_2'}\ \text{(rr)}$$

LTS-based Semantics of Arithmetic Expressions

$$\frac{m \circ n = k}{m \circ n \xrightarrow{\circ} k} \text{ (op)} \qquad \frac{E_1 \xrightarrow{\circ'} E_1'}{E_1 \circ E_2 \xrightarrow{\circ'} E_1' \circ E_2} \text{ (rl)} \qquad \frac{E_2 \xrightarrow{\circ'} E_2'}{E_1 \circ E_2 \xrightarrow{\circ'} E_1 \circ E_2'} \text{ (rr)}$$

$$(4 + (7 * 3))/(6 - 1) \quad \xrightarrow{*} \quad (4 + 21)/(6 - 1) \quad \xrightarrow{+} \quad 25/(6 - 1) \quad \xrightarrow{-} \quad 25/5 \quad \xrightarrow{/} \quad 5$$

LTS-based Semantics of Arithmetic Expressions

$$\frac{m \circ n = k}{m \circ n \xrightarrow{\circ} k} \text{ (op)} \qquad \frac{E_1 \xrightarrow{\circ'} E_1'}{E_1 \circ E_2 \xrightarrow{\circ'} E_1' \circ E_2} \text{ (rl)} \qquad \frac{E_2 \xrightarrow{\circ'} E_2'}{E_1 \circ E_2 \xrightarrow{\circ'} E_1 \circ E_2'} \text{ (rr)}$$

$$(4 + (7 * 3))/(6 - 1) \xrightarrow{*} (4 + 21)/(6 - 1) \xrightarrow{+} 25/(6 - 1) \xrightarrow{-} 25/5 \xrightarrow{/} 5$$

$$\frac{\dfrac{\dfrac{7 * 3 = 21}{7 * 3 \xrightarrow{*} 21}}{4 + (7 * 3) \xrightarrow{*} 4 + 21}}{(4 + (7 * 3))/(6 - 1) \xrightarrow{*} (4 + 21)/(6 - 1)} \qquad \frac{\dfrac{4 + 21 = 25}{4 + 21 \xrightarrow{+} 25}}{(4 + 21)/(6 - 1) \xrightarrow{+} 25/(6 - 1)}$$

similarly for $-$ and $/$

Finite State Automata – as language recognizers

A *finite state automaton* M is a 5-tuple $M = \langle Q, \Gamma, \rightarrow, q_0, F \rangle$ s.t.

1. **states** Q **finite !**
2. **alphabet** Γ
3. **transitions** $\rightarrow \subseteq Q \times \Gamma \times Q$

   $q \xrightarrow{a} q'$ denotes $(q, a, q') \in \rightarrow$
4. **initial state** $q_0 \in Q$
5. **accepting states** $F \subseteq Q$

Finite State Automata – as language recognizers

A *finite state automaton* M is a 5-tuple $M = \langle Q, \Gamma, \rightarrow, q_0, F \rangle$ s.t.

1. **states** Q **finite !**
2. **alphabet** Γ
3. **transitions** $\rightarrow \subseteq Q \times \Gamma \times Q$

   $q \xrightarrow{a} q'$ denotes $(q, a, q') \in \rightarrow$
4. **initial state** $q_0 \in Q$
5. **accepting states** $F \subseteq Q$

$$p \stackrel{w}{\Longrightarrow} q \quad \text{iff} \quad p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n = q \qquad\qquad w = a_1 \cdots a_n$$

Finite State Automata – as language recognizers

A *finite state automaton* M is a 5-tuple $M = \langle Q, \Gamma, \rightarrow, q_0, F\rangle$ s.t.

1. **states** Q **finite !**
2. **alphabet** Γ
3. **transitions** $\rightarrow \subseteq Q \times \Gamma \times Q$

   $q \xrightarrow{a} q'$ denotes $(q, a, q') \in \rightarrow$
4. **initial state** $q_0 \in Q$
5. **accepting states** $F \subseteq Q$

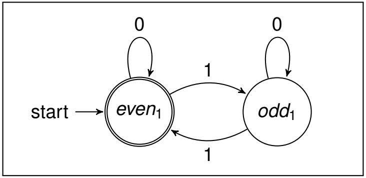
$p \stackrel{w}{\Longrightarrow} q$ iff $p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n = q$ $\quad w = a_1 \cdots a_n$

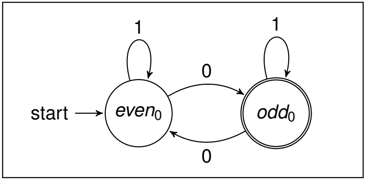
Semantics of Finite State Automata

The language accepted by a Finite State Automata is the set:

$$L(M) = \{w \in \Gamma^* \mid q_0 \stackrel{w}{\Longrightarrow} q \text{ and } q \in F\}$$

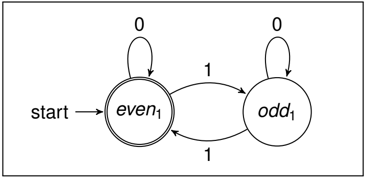
Some Regular Bit-Strings – $\Gamma = \{0, 1\}$



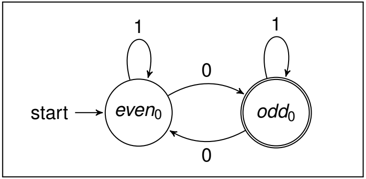


$L(A_1) = \{w \mid \text{even number of 1's}\}$ $\quad$ $L(A_2) = \{w \mid \text{odd number of 0's}\}$

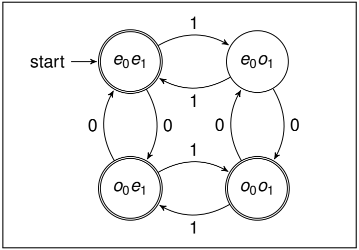
Some Regular Bit-Strings – $\Gamma = \{0, 1\}$



$L(A_1) = \{w \mid \text{even number of 1's}\}$

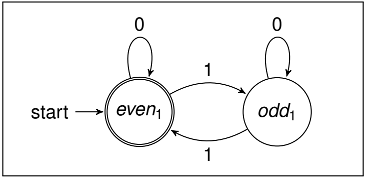


$L(A_2) = \{w \mid \text{odd number of 0's}\}$

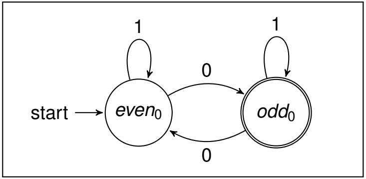


$L(A_1) \cup L(A_2)$

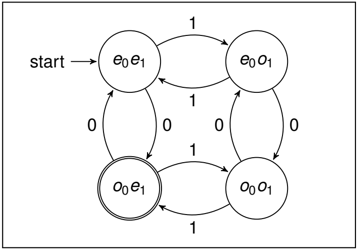
Some Regular Bit-Strings – $\Gamma = \{0, 1\}$



$L(A_1) = \{w \mid \text{even number of 1's}\}$

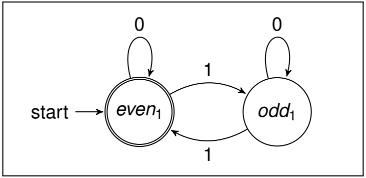


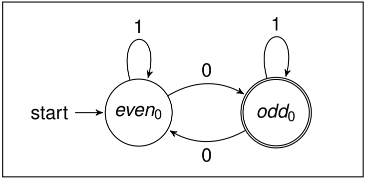
$L(A_2) = \{w \mid \text{odd number of 0's}\}$



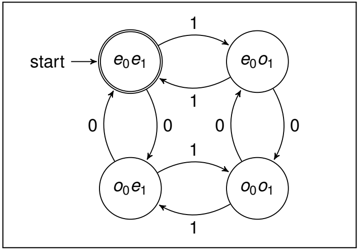
$L(A_1) \cap L(A_2)$

Some Regular Bit-Strings – $\Gamma = \{0, 1\}$



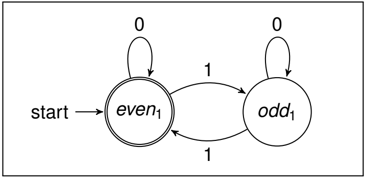


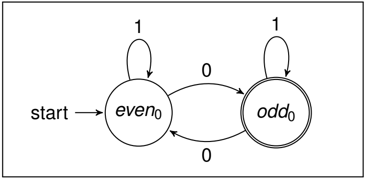
$L(A_1) = \{w \mid \text{even number of 1's}\}$ $L(A_2) = \{w \mid \text{odd number of 0's}\}$



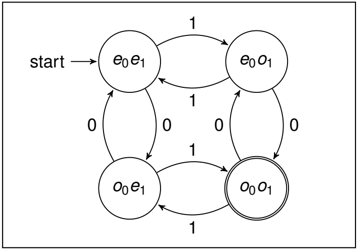
$L(A_1) \setminus L(A_2)$

Some Regular Bit-Strings – $\Gamma = \{0, 1\}$



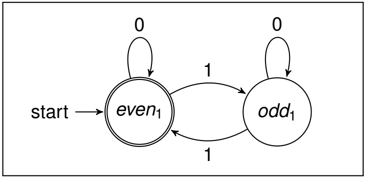


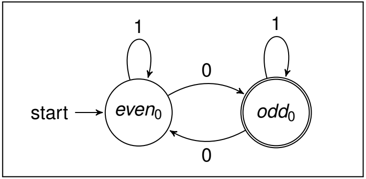
$L(A_1) = \{w \mid \text{even number of 1's}\}$ $L(A_2) = \{w \mid \text{odd number of 0's}\}$



$L(A_2) \setminus L(A_1)$

Some Regular Bit-Strings – $\Gamma = \{0, 1\}$





$L(A_1) = \{w \mid \text{even number of 1's}\}$

$L(A_2) = \{w \mid \text{odd number of 0's}\}$

regular languages are closed w.r.t. the operations of $\cap$, $\cup$, $\setminus$, complement, reversal, concatenation, star closure, ...

Regular Languages

Chomsky Hierarchy	Grammar Restriction	Language	Abstract Machine
Type 0	unrestricted	recursively enumerable	Turing machines
Type 1	$\alpha A \beta \rightarrow \alpha \gamma \beta$	context sensitive	linear bounded automata
Type 2	$A \rightarrow \gamma$	context free	nondeterministic pushdown automata
Type 3	$A \rightarrow a \quad A \rightarrow aB$	regular	finite state automata

with $A,\ B \in NT$, and $a \in T$ and $\alpha,\ \beta,\ \gamma \in (T \cup NT)^*$