Lectures Notes on Selected Mathematical Preliminaries of Computer Science

Florian Rabe

2017

Contents

Chapter 1

Mathematical Preliminaries

1.1 Binary Relations

A binary relation is on A is a subset $\# \subseteq A \times A$. We usually write $(x, y) \in \#$ as $x\#y$.

Definition 1.1 (Properties of Binary Relations)**.** We say that $\#$ is ... if the following holds:

- reflexive: for all x, $x\#x$
- transitive: for all x, y, z, if $x\#y$ and $y\#z$, then $x\#z$
- a preorder: reflexive and transitive
- anti-symmetric: for all x, y, if $x\#y$ and $y\#x$, then $x = y$
- symmetric: for all x, y, if $x\#y$, then $y\#x$
- an order: preorder and anti-symmetric
- an equivalence: preorder and symmetric
- a total order: order and for all x, y, $x\#y$ or $y\#x$

An element $a \in A$ is called ... of $\#$ if the following holds:

- least element: for all x, $a\#x$
- greatest element: for all x, $x\#a$
- least upper bound for x, y: $x\#a$ and $y\#a$ and for all z, if $x\#z$ and $y\#z$, then $a\#z$
- greatest lower bound for x, y: $a\#x$ and $a\#y$ and for all z, if $z\#x$ and $z\#y$, then $z\#a$

Theorem 1.2. *If $\#$ is an order, then least element, greatest element, least upper bound of x, y, and greatest lower bound of x, y are unique whenever they exist.*

1.2 Binary Functions

A binary function on A is a function $\circ : A \times A \to A$. We usually write $\circ(x, y)$ as $x \circ y$.

Definition 1.3 (Properties of Binary Functions)**.** We say that $\circ$ is ... if the following holds:

- associative: for all x, y, z, $x \circ (y \circ z) = (x \circ y) \circ z$
- commutative: for all x, y, $x \circ y = y \circ x$
- idempotent: for all x, $x \circ x = x$

An element $a \in A$ is called a ... element of $\circ$ if the following holds:

- left-neutral: for all x, $a \circ x = x$
- right-neutral: for all x, and $x \circ a = x$
- neutral: left-neutral and right-neutral
- left-absorbing: for all x, $a \circ x = a$
- right-absorbing: for all x, $x \circ a = a$
- absorbing: left-absorbing and right-absorbing

Theorem 1.4. *Neutral and absorbing element of $\circ$ are unique whenever they exist.*

1.3 The Integer Numbers

1.3.1 Divisibility

Definition 1.5 (Divisibility)**.** For $x, y \in \mathbb{Z}$, we write $x|y$ iff there is a $k \in \mathbb{Z}$ such that $x * k = y$.
We say that y is divisible by x or that x divides y.

Remark 1.6 (Divisible by 0 and 1). Even though division by 0 is forbidden, the case $x = 0$ is perfectly fine. But it is boring: $0|x$ iff $x = 0$.
Similarly, the case $x = 1$ is trivial: $1|x$ for all x.

Theorem 1.7 (Divisibility)**.** *Divisibility has the following properties for all $x, y, z \in Z$*

- *reflexive:* $x|x$
- *transitive: if* $x|y$ *and* $y|z$ *then* $x|z$
- *anti-symmetric for natural numbers* $x, y \in \mathbb{N}$*: if* $x|y$ *and* $y|x$*, then* $x = y$
- 1 *is a least element:* $1|x$
- 0 *is a greatest element:* $x|0$
- $\gcd(x, y)$ *is a greatest lower bound of* x, y
- $\mathrm{lcm}(x, y)$ *is a least upper bound of* x, y

Thus, $|$ *is a preorder on* $\mathbb{Z}$ *and an order on* $\mathbb{N}$.

Divisibility is preserved by arithmetic operations: If $x|m$ *and* $y|m$*, then*

- *preserved by addition:* $x + y|m$
- *preserved by subtraction:* $x - y|m$
- *preserved by multiplication:* $x * y|m$
- *preserved by division if* $x/y \in Z$*:* $x/y|m$
- *preserved by negation of any argument:* $-x|m$ *and* $x| - m$

gcd *has the following properties for all* $x, y \in \mathbb{N}$*:*

- *associative:* $\gcd(\gcd(x, y), z) = \gcd(x, \gcd(y, z))$
- *commutative:* $\gcd(x, y) = \gcd(y, x)$
- *idempotence:* $\gcd(x, x) = x$
- 0 *is a neutral element:* $\gcd(0, x) = x$
- 1 *is an absorbing element:* $\gcd(1, x) = 1$

lcm *has the same properties as* gcd *except that* 1 *is neutral and* 0 *is absorbing.*

Theorem 1.8. *For all $x, y \in \mathbb{Z}$, there are numbers $a, b \in \mathbb{Z}$ such that $ax + by = \gcd(x, y)$.*
a and b can be computed using the extended Euclidean algorithms.

Definition 1.9. If $\gcd(x, y) = 1$, we call x and y **coprime**.
For $x \in \mathbb{N}$, the number of coprime $y \in \{0, \ldots, x - 1\}$ is called $\varphi(x)$. φ is called Euler's **totient function**.

We have $\varphi(0) = 0$, $\varphi(1) = \varphi(2) = 1$, $\varphi(3) = 2$, $\varphi(4) = 1$, and so on. Because $\gcd(x, 0) = x$, we have $\varphi(x) \leq x - 1$. x is prime iff $\varphi(x) = x - 1$.

1.3.2 Equivalence Modulo

Definition 1.10 (Equivalence Modulo)**.** For $x, y, m \in \mathbb{Z}$, we write $x \equiv_m y$ iff $m|x - y$.

Theorem 1.11 (Relationship between Divisibility and Modulo)**.** *The following are equivalent:*

- $m|n$
- $\equiv_m \supseteq \equiv_n$ *(i.e., for all x, y we have that $x \equiv_n y$ implies $x \equiv_m y$)*
- $n \equiv_m 0$

Remark 1.12 (Modulo 0 and 1). In particular, the cases $m = 0$ and $m = 1$ are trivial again:

- $x \equiv_0 y$ iff $x = y$,
- $x \equiv_1 y$ always

Thus, just like 0 and 1 are greatest and least element for $|$, we have that $\equiv_0$ and $\equiv_1$ are the smallest and the largest equivalence relation on $\mathbb{Z}$.

Theorem 1.13 (Modulo)**.** *The relation $\equiv_m$ has the following properties*

- *reflexive: $x \equiv_m x$*
- *transitive: if $x \equiv_m y$ and $y \equiv_m z$ then $x \equiv_m z$*
- *symmetric: if $x|y$ then $y|x$*

Thus, it is an equivalence relation.

It is also preserved by arithmetic operations: If $x \equiv_m x'$ and $y \equiv_m y'$, then

- *preserved by addition: $x + y \equiv_m x' + y'$*
- *preserved by subtraction: $x - y \equiv_m x' - y'$*
- *preserved by multiplication: $x * y \equiv_m x' * y'$*
- *preserved by division if $x/y \in Z$ and $x'/y' \in \mathbb{Z}$: $x/y \equiv_m x'/y'$*
- *preserved by negation of both arguments: $-x \equiv_m -x'$*

1.3.3 Arithmetic Modulo

Definition 1.14 (Modulus)**.** We write $x \bmod m$ for the smallest $y \in \mathbb{N}$ such that $x \equiv_m y$.
We also write $modulus_m$ for the function $x \mapsto x \bmod m$. We write $\mathbb{Z}_m$ for the image of $modulus_m$.

Remark 1.15 (Modulo 0 and 1). The cases $m = 0$ and $m = 1$ are trivial again:

- $x \bmod 0 = x$ and $\mathbb{Z}_0 = \mathbb{Z}$
- $x \bmod 1 = 0$ and $\mathbb{Z}_1 = \{0\}$

Remark 1.16 (Possible Values). For $m \neq 0$, we have $x \bmod m \in \{0, \ldots, m-1\}$. In particular, there are m possible values $m \bmod x$.
For example, we have $x \bmod 1 \in \{0\}$. And we have $x \bmod 2 = 0$ if x is even and $x \bmod 2 = 1$ if x is odd.

Definition 1.17 (Arithmetic Modulo m)**.** For $x, y \in \mathbb{Z}$, we define arithmetic operations modulo m by

$$x \circ_m y = (x \circ y) \bmod m \qquad \text{for} \qquad \circ \in \{+, -, \cdot\}$$

Moreover, if there is a unique $q \in \mathbb{Z}_m$ such that $q \cdot x \equiv_m y$, we define $x/_m y = q$.

Note that the condition $y|x$ is neither necessary nor sufficient for $x/_m y$ to de defined. For example, $2/_4 2$ is undefined because $1 \cdot 2 \equiv_4 3 \cdot 2 \equiv_4 2$. Conversely, $2/_4 3$ is defined, namely 2.

Theorem 1.18 (Arithmetic Modulo m)**.** *For $x, y \in \mathbb{Z}$,* mod *commutes with arithmetic operations in the sense that*

$$(x \circ y) \bmod m = (x \bmod m) \circ_m (y \bmod m) \qquad \text{for} \qquad \circ \in \{+, -, \cdot\}$$

Moreover, $x/_m y$ *is defined iff* $\gcd(y, m) = 1$ *and*

$$(x/y) \bmod m = (x \bmod m)/_m (y \bmod m) \qquad \text{if} \qquad y|x$$

$$x/_m y = x \cdot_m a \qquad \text{if } ay + bm = 1 \text{ as in see Thm. 1.8}$$

Theorem 1.19 (Fermat's Little Theorem)**.** *For all prime numbers* p *and* $x \in \mathbb{Z}$*, we have that* $x^p \equiv_p x$*. If* x *and* p *are coprime, that is equivalent to* $x^{p-1} \equiv 1$*.*

1.3.4 Digit-Base Representations

Fix $m \in \mathbb{N} \setminus \{0\}$, which we call the base.

Theorem 1.20 (Div-Mod Representation)**.** *Every* $x \in \mathbb{Z}$ *can be uniquely represented as* $a \cdot m + b$ *for* $a \in \mathbb{Z}$ *and* $b \in \mathbb{Z}_m$*.*
Moreover, $b = x \bmod m$*. We write* $b \operatorname{div} m$ *for* a*.*

Definition 1.21 (Base-m-Notation)**.** For $d_i \in \mathbb{Z}_m$, we define $(d_k \ \ldots \ d_0)_m = d_k \cdot m^k + \ldots + d_1 \cdot k + d_0$.
The d_i are called digits.

Theorem 1.22 (Base-m Representation)**.** *Every* $x \in \mathbb{N}$ *can be uniquely represented as* $(0)_m$ *or* $(d_k \ \ldots \ d_0)_m$ *such that* $d_k \neq 0$*.*
Moreover, we have $k = \lfloor \log_m x \rfloor$ *and* $d_0 = x \bmod m$*,* $d_1 = (x \operatorname{div} m) \bmod m$*,* $d_2 = ((x \operatorname{div} m) \operatorname{div} m) \bmod m$ *and so on.*

Example 1.23 (Important Bases)*.* We call $(d_k \ \ldots \ d_0)_m$ the binary/octal/decimal/hexadecimal representation if $m = 2, 8, 10, 16$, respectively.
In case $m = 16$, we write the elements of $\mathbb{Z}_m$ as $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}$

1.3.5 Finite Fields

In this section, let $m = p$ be prime.

Construction Then $x/_p y$ is defined for all $x, y \in \mathbb{Z}_p$ with $y \neq 0$. Consequently, $\mathbb{Z}_p$ is a field.
Up to isomorphism, all finite fields are obtained as an n-dimensional vector space $\mathbb{Z}_p^n$ for $n \geq 1$. This field is usually called F_{p^n} because it has p^n elements. From now on, let $q = p^n$.
All elements of F_q are vectors $(a_0, \ldots, a_{n-1})$ for $a_i \in \mathbb{Z}_p$. Addition and subtraction are component-wise, the 0-element is $(0, \ldots, 0)$, the 1-elements is $(1, 0, \ldots, 0)$.
However, multiplication in F_q is tricky. To multiply two elements, we think of the vectors $(a_0, \ldots, a_{n-1})$ as polynomials $a_{n-1}X^{n-1} + \ldots + a_1 X + a_0$, and multiply the polynomials. This can introduce powers X^n and higher, which we eliminate using $X^n = k_{n-1}X^{n-1} + \ldots + k_1 X + k_0$. The resulting polynomial has degree at most $n - 1$, and its coefficient (modulo p) yield the result.
The values k_i always exists but are non-trivial to find. They must be such that the polynomial $X^n - k_{n-1}X^{n-1} - \ldots - k_1 X - k_0$ has no roots in $\mathbb{Z}_p$. There may be multiple polynomials, which may lead to different multiplication operations. However, all of them yield isomorphic fields.

Binary Fields The operations become particularly easy if $p = 2$. The elements of F_{2^n} are just the bit strings of length n. Addition and subtraction are the same operation and can be computed by component-wise XOR. Multiplication is a bit more complex but can be obtained as a sequence of bit-shifts and XORs.

Exponentiation and Logarithm Because F_q has multiplication, we can define natural powers in the usual way:

Definition 1.24. For $x \in F^q$ and $l \in \mathbb{N}$, we define $x^l \in F_q$ by $x^0 = 1$ and $x^{l+1} = x \cdot x^l$.
If l is the smallest number such that $x^l = y$, we write $l = \log_x y$ and call n the **discrete q-logarithm** of y with base x.

The powers $1, x, x^2, \ldots \in F_q$ of x can take only $q-1$ different values because F_q has only q elements and x^l can never be 0 (unless $x = 0$). Therefore, they must be periodic:

Theorem 1.25. *For every $x \in F_q$, we have $x^q = x$ or equivalently $x^{q-1} = 1$ for $x \neq 0$.*

For some x, the period is indeed $q-1$, i.e., we have $\{1, x, x^2, \ldots, x^{q-1} = F_q \setminus \{0\}$. Those x are called primitive elements of F_q. But the period may be smaller. For example, the powers of 1 are $1, \ldots, 1$, i.e., 1 has period 1. For a non-trivial example consider $p = 5$, $n = 1$, (i.e., $q = 5$): The powers of 4 are $4^0 = 1$, $4^1 = 4$, $4^2 = 16 \bmod 5 = 1$, and $4^3 = 4$.
If the period is smaller, x^l does not take all possible values in F_q. Therefore, $\log_x y$ is not defined for all $y \in F_q$.

Computing x^l is straightforward and can be done efficiently. (If $n > 1$, we first have to find the values k_i needed to do the multiplication, but we can precompute them once and for all.)
Determining whether $\log_x y$ is defined and computing its value is also straightforward: We can enumerate all powers $1, x, x^2, \ldots$ until we find 1 or y. However, no efficient algorithm is known.

1.4 Size of Sets

The size $|S|$ of a set S is a very complex topic of mathematics because there are different degrees of infinity. Specifically, we have that $|\mathcal{P}(S)| > |S|$, i.e., we have infinitely many degrees of infinity.
In computer science, we are only interested in countable sets. We use a very simple definition that writes C for countable and merges all greater sizes into uncountable sets, whose size we write as U.

Definition 1.26 (Size of sets)**.** The size $|S| \in \mathbb{N} \cup \{C, U\}$ of a set S is defined by:

- if S is finite: $|S|$ is the number of elements of S
- if S is infinite and bijective to $\mathbb{N}$: $|S| = C$, and we say that S is countable
- if S is infinite and not bijective to $\mathbb{N}$: $|S| = U$, and we say that S is uncountable

We can compute with set sizes as follows:

Definition 1.27 (Computing with Sizes)**.** For two sizes $s, t \in \mathbb{N} \cup \{C, U\}$, we define addition, multiplication, and exponentiation by the following tables:

	$s+t$	t: $n \in \mathbb{N}$	C	U
	$m \in \mathbb{N}$	$m+n$	C	U
s	C	C	C	U
	U	U	U	U

	$s*t$	t: $n \in \mathbb{N}$	C	U
	$m \in \mathbb{N}$	$m*n$	C	U
s	C	C	C	U
	U	U	U	U

	s^t	t: 0	1	$n \in \mathbb{N} \setminus \{0\}$	C	U
	0	1	0	0	0	0
	1	1	1	1	1	1
s	$m \in \mathbb{N} \setminus \{0\}$	1	m	m^n	U	U
	C	1	C	C	U	U
	U	1	U	U	U	U

Because exponentiation s^t is not commutative, the order matters: s is given by the row and t by the column.

The intuition behind these rules is given by the following:

Theorem 1.28. *For all sets S, T, we have for the size of the*

- *disjoint union:*
$$|S \uplus T| = |S| + |T|$$
- *Cartesian product:*
$$|S \times T| = |S| * |T|$$
- *set of functions from T to S:*
$$|S^T| = |S|^{|T|}$$

Thus, we can understand the rules for exponentiation as follows. Let us first consider the 4 cases where one of the arguments has size 0 or 1: For every set A

1. there is exactly one function from the empty set (namely the empty function): $|A^{\varnothing}| = 1$,
2. there are as many functions from a singleton set as there are elements of A: $|A^{\{x\}}| = |A|$,
3. there are no functions to the empty set (unless A is empty): $|\varnothing^A| = 0$ if $A \neq \varnothing$,
4. there is exactly one function into a singleton set (namely the constant function): $|\{x\}^A| = 1$,

Now we need only one more rule: The set of functions from a non-empty finite set to a finite/countable/uncountable set is again finite/countable/uncountable. In all other cases, the set of functions is uncountable.

1.5 Important Sets and Functions

The meaning and purpose of a data structure is to describe a set in the sense of mathematics. Similarly, the meaning and purpose of an algorithm is to describe a function between two sets.

Thus, it is helpful to collect some sets and functions as examples. These are typically among the first data structures and algorithms implemented in any programming language and they serve as test cases for evaluating our languages.

1.5.1 Base Sets

When building sets, we have to start somewhere with some sets that are assumed to exist. These are called the *bases sets* or the *primitive sets*.

The following table gives an overview, where we also list the size of each set according to Def. 1.26:

set	description/definition	size
typical base sets of mathematics[1]		
$\varnothing$	empty set	0
$\mathbb{N}$	natural numbers	C
$\mathbb{Z}$	integers	C
$\mathbb{Z}_m$ for $m > 0$	integers modulo m, $\{0, \ldots, m-1\}$ [2]	m
$\mathbb{Q}$	rational numbers	C
$\mathbb{R}$	real numbers	U
additional or alternative base sets used in computer science		
unit	unit type, $\{()\}$, equivalent to $\mathbb{Z}_1$	1
$\mathbb{B}$	booleans, $\{false, true\}$, equivalent to $\mathbb{Z}_2$	2
int	primitive integers, $-2^{n-1}, \ldots, 2^{n-1} - 1$ for machine-dependent n, equivalent to $\mathbb{Z}_{2^n}$ [3]	2^n
float	IEEE floating point approximations of real numbers	C
char	characters	finite[4]
string	lists of characters	C

[1] All of mathematics can be built by using $\varnothing$ as the only base set because the others are definable. But it is common to assume at least the number sets as primitives.

[2] $\mathbb{Z}_0$ also exists but is trivial: $\mathbb{Z}_0 = \mathbb{Z}$.

[3] Primitive integers are the 2^n possible values for a sequence of n bits. Old machines used $n = 8$ (and the integers were called "bytes"), later machines used $n = 16$ (called "words"). Modern machines typically use 32-bit or 64-bit integers. Modern programmers usually—but dangerously—assume that 2^n is much bigger than any number that comes up in practice so that essentially $int = \mathbb{Z}$.

[4] The ASCII standard defined 2^7 or 2^8 characters. Nowadays, we use Unicode characters, which is a constantly growing set containing

1.5.2 Functions on the Base Sets

For every base set, we can define some basic operations. These are usually built-in features of programming languages whenever the respective base set is built-in.
We only list a few examples here.

Numbers

For all number sets, we can define addition, subtraction, multiplication, and division in the usual way.
Some care must be taken when subtracting or dividing because the result may be in a different set. For example, the difference of two natural numbers is not in general a natural number but only an integer (e.g., $3 - 5 \notin \mathbb{N}$). Moreover, division by 0 is always forbidden.

Quotients of the Integers

The function $modulus_m$ (see Sect. 1.3.3) for $m \in \mathbb{N}$ maps $x \in \mathbb{Z}$ to $x \bmod m \in \mathbb{Z}_m$.
In programming languages, the set $\mathbb{Z}_m$ is usually not provided. Instead, $x \bmod y$ is built-in as a functions on *int*.

Booleans

On booleans, we can define the usual boolean operations conjunction (usually written & or &&), disjunction (usually written | or ||), and negation (usually written !).
Moreover, we have the equality and inequality functions, which take two objects x, y and return a boolean. These are usually written $x == y$ and $x! = y$ in text files languages and $x = y$ and $x \neq y$ on paper.

1.5.3 Set Constructors

From the base sets, we build all other sets by applying set constructors. Those are operations that take sets and return new sets.
The following table gives an overview, where we also list the size of each set according to Def. 1.27:

the characters of virtually any writing system, many scientific symbols, emojis, etc. Many programming languages assume that there is one character for every primitive integers, e.g., typically 2^{32} characters.

set	description/definition	size
	typical constructors in mathematics	
$A \uplus B$	disjoint union	$\lvert A\rvert + \lvert B\rvert$
$A \times B$	(Cartesian) product	$\lvert A\rvert * \lvert B\rvert$
A^n for $n \in \mathbb{N}$	n-dimensional vectors over A	$\lvert A\rvert^n$
B^A or $A \to B$	functions from A to B	$\lvert B\rvert^{\lvert A\rvert}$
$\mathcal{P}(A)$	power set, equivalent to $\mathbb{B}^A$	$2^{\lvert A\rvert} = \begin{cases} 2^n & \text{if } \lvert A\rvert = n \\ U & \text{otherwise} \end{cases}$
$\{x \in A \mid P(x)\}$	subset of A given by property P	$\leq \lvert A\rvert$
$\{f(x) : x \in A\}$	image of function f when applied to elements of A	$\leq \lvert A\rvert$
A/r	quotient set for an equivalence relation r on A	$\leq \lvert A\rvert$
	selected additional constructors often used in computer science	
A^*	lists over A	$\begin{cases} 1 & \text{if } A = \varnothing \\ U & \text{if } \lvert A\rvert = U \\ C & \text{otherwise} \end{cases}$
$A^?$	optional element[5] of A	$1 + \lvert A\rvert$
	for new names $l_1, \ldots, l_n$	
$enum\{l_1, \ldots, l_n\}$	enumeration: like $\mathbb{Z}_n$ but also introduces named elements l_i of the enumeration	n
$l_1(A_1) \mid \ldots \mid l_n(A_n)$	labeled union: like $A_1 \uplus \ldots \uplus A_n$ but also introduces named injections l_i from A_i into the union	$\lvert A_1\rvert + \ldots + \lvert A_n\rvert$
$\{l_1 : A_1, \ldots, l_n : A_n\}$	record: like $A_1 \times \ldots \times A_n$ but also introduces named projections l_i from the record into A_i	$\lvert A_1\rvert * \ldots * \lvert A_n\rvert$
inductive data types[6]		C
classes[7]		U

1.5.4 Characteristic Functions of the Set Constructors

Every set constructor comes systematically with characteristic functions into and out of the constructed sets C. These functions allow building elements of C or using elements of C for other computations.

For some sets, these functions do not have standard notations in mathematics. In those cases, different programming languages may use slightly different notations.

The following table gives an overview:

set C	build an element of C	use an element x of C
$A_1 \uplus A_2$	$inj_1(a_1)$ or $inj_2(a_2)$ for $a_i \in A_i$	pattern-matching
$A_1 \times A_2$	(a_1, a_2) for $a_i \in A_i$	$x.i \in A_i$ for $i = 1, 2$
A^n	$(a_1, \ldots, a_n)$ for $a_i \in A$	$x.i \in A$ for $i = 1, \ldots, n$
B^A	$(a \in A) \mapsto b(a)$	$x(a)$ for $a \in A$
A^*	$[a_0, \ldots, a_{l-1}]$[8] for $a_i \in A$	pattern-matching
$A^?$	$None$ or $Some(a)$ for $a \in A$	pattern-matching
$enum\{l_1, \ldots, l_n\}$	l_1 or $\ldots$ or l_n	switch statement or pattern-matching
$l_1(A_1) \mid \ldots \mid l_n(A_n)$	$l_1(a_1)$ or $\ldots$ or $l_n(a_n)$ for $a_i \in A_i$	pattern-matching
$\{l_1 : A_1, \ldots, l_n : A_n\}$	$\{l_1 = a_1, \ldots, l_n = a_n\}$ for $a_i \in A_i$	$x.l_i \in A_i$
inductive data type A	$l(u_1, \ldots, u_n)$ for a constructor l of A	pattern-matching
class A	**new** A	$x.l(u_1, \ldots, u_n)$ for a field l of A

[5] An optional element of A is either absent or an element of A.

[6] These are too complex to define at this point. They are a key feature of functional programming languages like SML.

[7] These are too complex to define at this point. They are a key feature of object-oriented programming languages like Java.

[8] Mathematicians start counting at 1 and would usually write a list of length n as $[a_1, \ldots, a_n]$. However, computer scientists always start counting at 0 and therefore write it as $[a_0, \ldots, a_{n-1}]$. We use the computer science numbering here.