



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 1 of 709

Go Back

Full Screen

Close

Quit

Introduction to Computer Science

S. Arun-Kumar
sak@cse.iitd.ernet.in

Department of Computer Science and Engineering
I. I. T. Delhi, Hauz Khas, New Delhi 110 016.

March 19, 2007

Contents

1	Computing: The Functional Way	3
1.1	Introduction to Computing	3
1.2	Our Computing Tool	26
1.3	Primitives: Integer & Real	53
1.4	Example: Fibonacci	80
1.5	Primitives: Booleans	103
2	Algorithms: Design & Refinement	114
2.1	Technical Completeness & Algorithms	114
2.2	Algorithm Refinement	148
2.3	Variations: Algorithms & Code	180
2.4	Names, Scopes & Recursion	208
3	Introducing Reals	242
3.1	Floating Point	242
3.2	Root Finding, Composition and Recursion	264
4	Correctness, Termination & Complexity	294
4.1	Termination and Space Complexity	294
4.2	Efficiency Measures and Speed-ups	333
4.3	Invariance & Correctness	364
5	Compound Data	390
5.1	Tuples, Lists & the Generation of Primes	390
5.2	Compound Data & Lists	429
5.3	Compound Data & List Algorithms	457
6	Higher Order Functions & Structured Data	486
6.1	Higher Order Functions	486
6.2	Structured Data	513



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 2 of 709

Go Back

Full Screen

Close

Quit

6.3	User Defined Structured Data Types	547
7	Imperative Programming: An Introduction	570
7.1	Introducing a Memory Model	570
7.2	Imperative Programming:	599
7.3	Arrays	626
8	A large Example: Tautology Checking	640
8.1	Large Example: Tautology Checking	640
8.2	Tautology Checking Contd.	665
9	Lecture-wise Index to Slides	680



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 3 of 709

Go Back

Full Screen

Close

Quit

1. Computing: The Functional Way

1.1. Introduction to Computing

1. Introduction
2. Computing tools
3. Ruler and Compass
4. Computing and Computers
5. Primitives
6. Algorithm
7. Problem: Doubling a Square
8. Solution: Doubling a Square
9. Execution: Step 1
10. Execution: Step 2
11. Doubling a Square: Justified
12. Refinement: Square Construction
13. Refinement 2: Perpendicular at a point
14. Solution: Perpendicular at a point
15. Perpendicular at a point: Justification

Next: [Our Computing Tool](#)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 4 of 709

Go Back

Full Screen

Close

Quit



Introduction

- This course is about **computing**
- **Computing** as a process is nearly as fundamental as **arithmetic**
- **Computing** as a mental process
- **Computing** may be done with a variety of **tools** which may or may not assist the mind

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 5 of 709

Go Back

Full Screen

Close

Quit



Computing tools

- Sticks and stones (counting)
- Paper and pencil (an aid to mental computing)
- Abacus (still used in Japan!)
- Slide rules (ask a retired engineer!)
- Ruler and compass

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 6 of 709

Go Back

Full Screen

Close

Quit



Ruler and Compass

Actually it is a **computing** tool!

- Construct a length that is **half** of a given length
- **Bisect** an angle
- Construct a square that is **twice** the area of a given square
- Construct $\sqrt{10}$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 7 of 709

Go Back

Full Screen

Close

Quit



Computing and Computers

- **Computing** is much more fundamental
- **Computing** may be done without a **computer** too!
- But a **Computer** cannot do much besides **computing**.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 8 of 709

Go Back

Full Screen

Close

Quit

Primitives

- Each tool has a set of capabilities called **primitive operations** or **primitives**

Ruler: Can specify **lengths**, lines

Compass: Can define **arcs** and **circles**

- The primitives may be combined in various ways to perform a **computation**.
- **Example** Constructing a right bisector of a given line segment.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 9 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 10 of 709

Go Back

Full Screen

Close

Quit

Algorithm

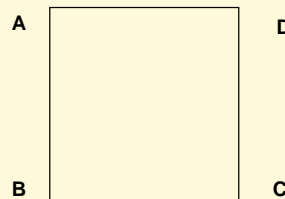
Given a **problem** to be solved with a given **tool**, the attempt is to evolve a combination of **primitives** of the tool **in a certain order** to solve the problem.

An explicit statement of this combination along with the order is an **algorithm**



Problem: Doubling a Square

Given a square, construct another square of twice the area of the original square.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 11 of 709

Go Back

Full Screen

Close

Quit



Solution: Doubling a Square

Assume given a square $\square ABCD$ of side $a > 0$.

1. Draw the diagonal \overline{AC} .
2. Complete the square $\square ACEF$ on side \overline{AC} .

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 12 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 13 of 709

Go Back

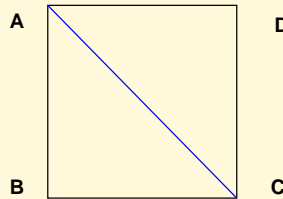
Full Screen

Close

Quit

Execution: Step 1

Draw the diagonal \overline{AC} .





IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 14 of 709

Go Back

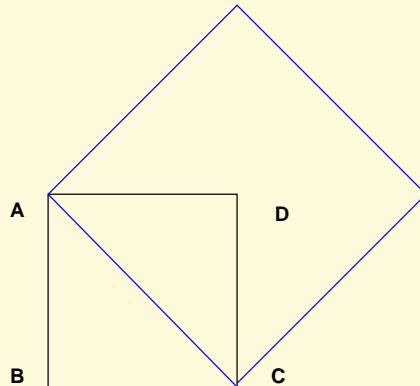
Full Screen

Close

Quit

Execution: Step 2

Complete the square $\square ACEF$ on side \overline{AC} .





IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 15 of 709

Go Back

Full Screen

Close

Quit

Doubling a Square: Justified

Assume given a square $\square ABCD$ of side $a > 0$.

1. Draw the diagonal \overline{AC} . $AC = \sqrt{2}a$
2. Complete the square $\square ACEF$ on side \overline{AC} . Area of $\square ACEF = 2a^2$.

Refinement: Square

Given a line segment of length $b > 0$ construct a square of side b .

Assume given a line segment \overline{PQ} of length b .

1. Construct two lines l_1 and l_2 perpendicular to \overline{PQ} passing through P and Q respectively
2. On the same side of \overline{PQ} mark points R on l_1 and S on l_2 such that $PR = PQ = QS$.
3. Draw \overline{RS} . $\square PQSR$ is a square of side b



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 16 of 709

Go Back

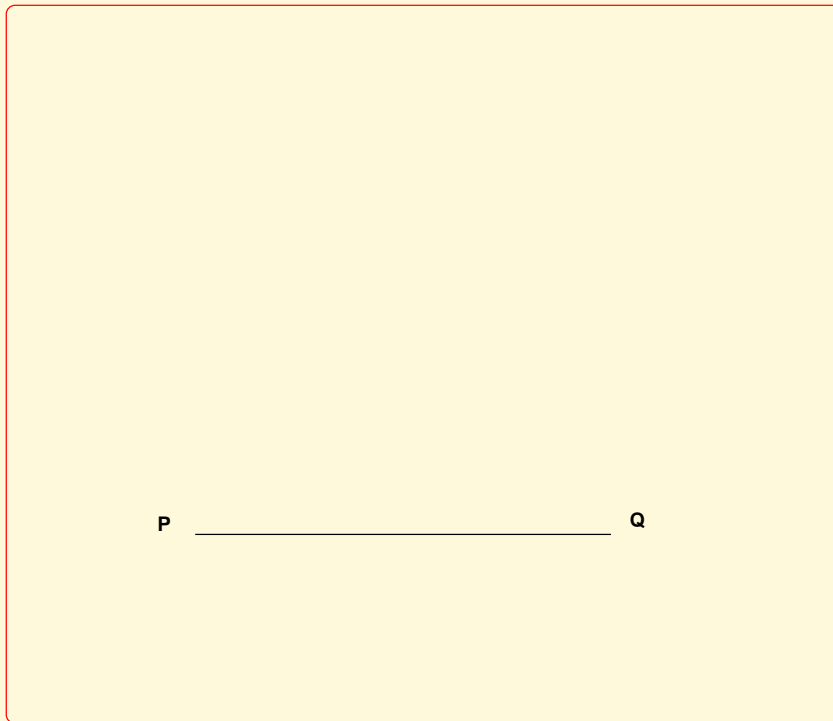
Full Screen

Close

Quit

Square on Segment: 0

Assume given a line segment \overline{PQ} of length b .



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 17 of 709

Go Back

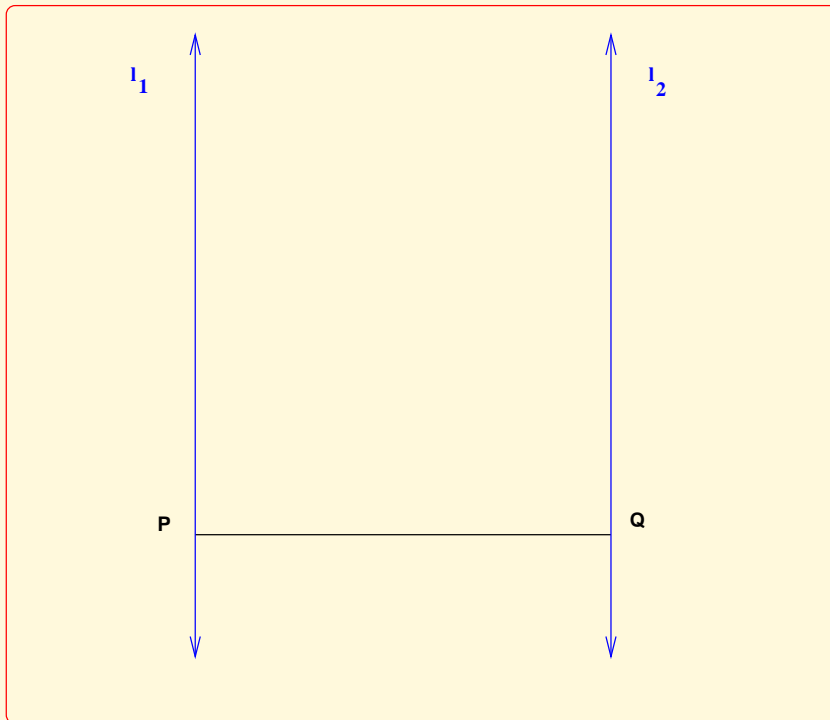
Full Screen

Close

Quit

Square on Segment: 1

Construct two lines l_1 and l_2 perpendicular to \overline{PQ} passing through P and Q respectively



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 18 of 709

Go Back

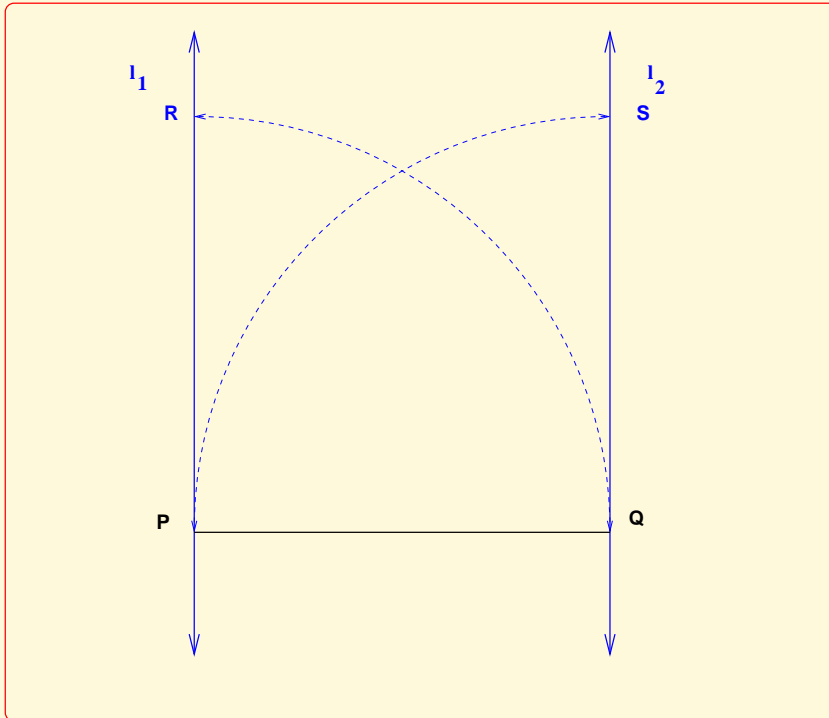
Full Screen

Close

Quit

Square on Segment: 2

On the same side of \overline{PQ} mark points R on l_1 and S on l_2 such that $PR = PQ = QS$.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 19 of 709

Go Back

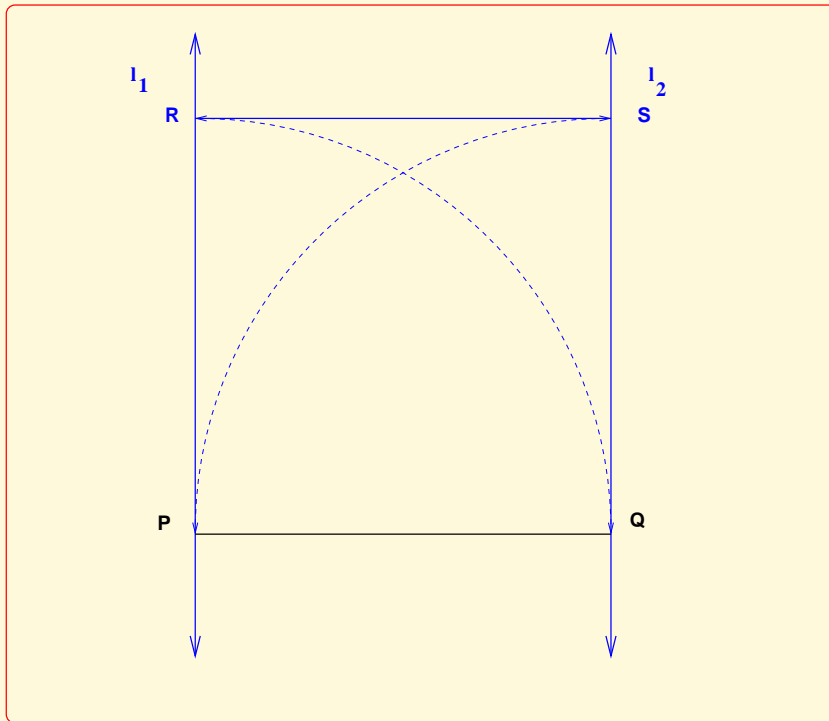
Full Screen

Close

Quit

Square on Segment: 3

Draw \overline{RS} . $\square PQSR$ is a square of side b



Square Construction algorithm



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 20 of 709

Go Back

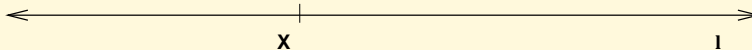
Full Screen

Close

Quit

Perpendicular at a point

Given a line, draw a perpendicular to it passing through a given point on it.
Assume given a line l containing a point X .



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 21 of 709

Go Back

Full Screen

Close

Quit

Solution: Perpendicular at a point

1. Choose a length $c > 0$. With X as centre mark off points Y and Z on l on either side of X , such that $YX = c = XZ$. $YZ = 2c$.
2. Draw Circles $C_1(Y, 2c)$ and $C_2(Z, 2c)$ respectively.
3. Join the points of intersection of the two circles.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 22 of 709

Go Back

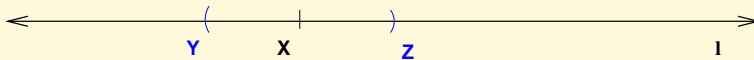
Full Screen

Close

Quit

Perpendicular at a Point: 1

Choose a length $c > 0$. With X as centre mark off points Y and Z on l on either side of X , such that $YX = c = XZ$. $YZ = 2c$.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 23 of 709

Go Back

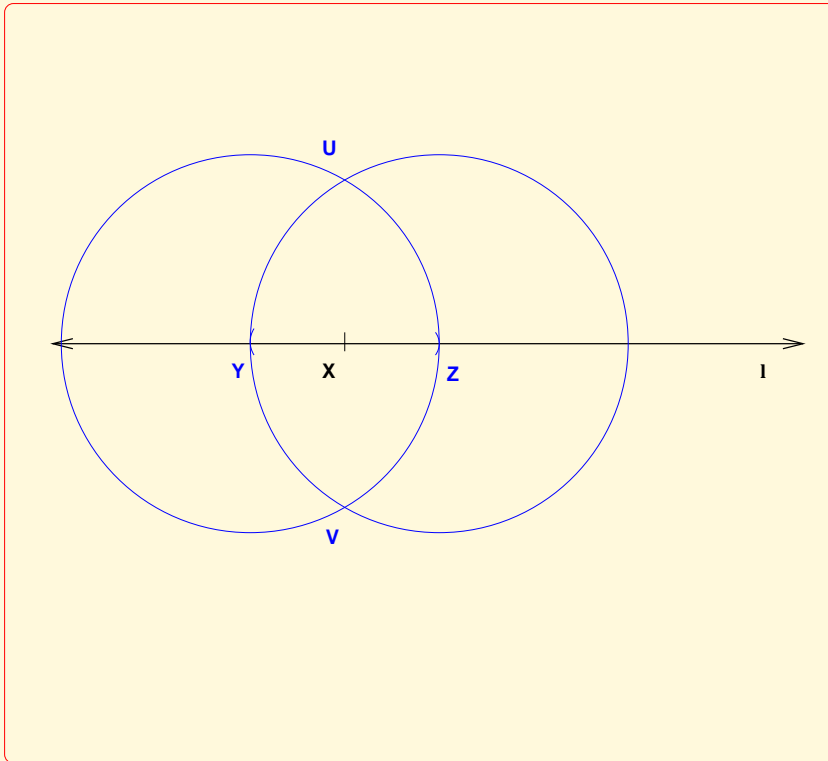
Full Screen

Close

Quit

Perpendicular at a Point: 2

Draw Circles $C_1(Y, 2c)$ and $C_2(Z, 2c)$ respectively.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 24 of 709

Go Back

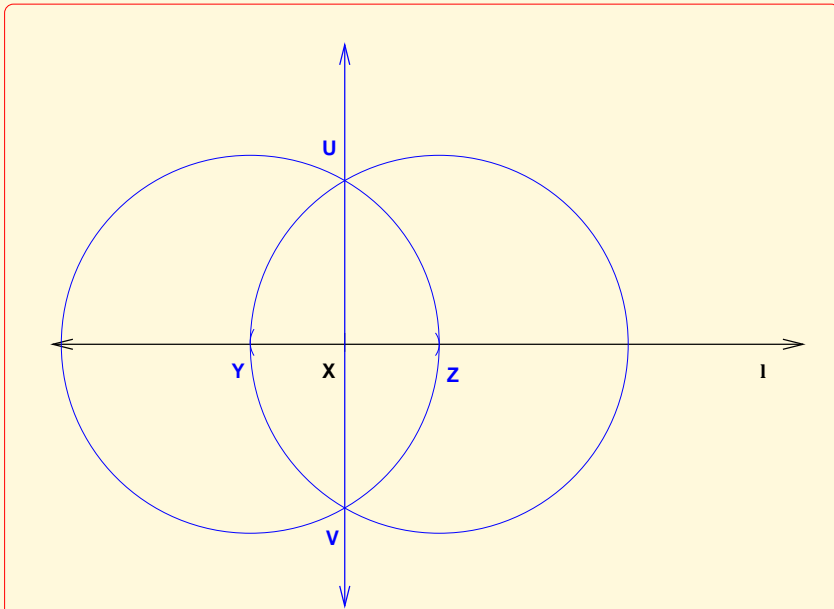
Full Screen

Close

Quit

Perpendicular at a Point: 3

Choose a length $c > 0$. With X as centre mark off points Y and Z on l on either side of X , such that $YX = c = XZ$. $YZ = 2c$.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 25 of 709

Go Back

Full Screen

Close

Quit

Perpendicular at a point: Justification

1. The two circles intersect at points U and V on either side of line l .
2. \overleftrightarrow{UV} is a perpendicular bisector of \overline{YZ} .
3. Since $YX = c = XZ$ and $YZ = 2c$, \overleftrightarrow{UV} is perpendicular to l and passes through X .

Back to square 1



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 26 of 709

Go Back

Full Screen

Close

Quit



1.2. Our Computing Tool

Previous: [Introduction to Computing](#)

1. [The Digital Computer: Our Computing Tool](#)
2. [Algorithms](#)
3. [Programming Language](#)
4. [Programs and Languages](#)
5. [Programs](#)
6. [Programming](#)
7. [Computing Models](#)
8. [Primitives](#)
9. [Primitive expressions](#)
10. [Methods of combination](#)
11. [Methods of abstraction](#)
12. [The Functional Model](#)
13. [Mathematical Notation 1: Factorial](#)
14. [Mathematical Notation 2: Factorial](#)
15. [Mathematical Notation 3: Factorial](#)
16. [A Functional Program: Factorial](#)

[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)



[Page 27 of 709](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

17. **A Computation: Factorial**
18. **A Computation: Factorial**
19. **A Computation: Factorial**
20. **A Computation: Factorial**
21. **A Computation: Factorial**
22. **A Computation: Factorial**
23. **A Computation: Factorial**
24. **Standard ML**
25. **SML: Important Features**

Next: **Primitives: Integer & Real**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 28 of 709

Go Back

Full Screen

Close

Quit

The Digital Computer: Our Computing Tool

Algorithm: A **finite** specification of the solution to a given problem using the primitives of the computing tool.

- It specifies a definite input and output
- It is unambiguous
- It specifies a solution as a **finite** process i.e. the number of steps in the computation is **finite**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 29 of 709

Go Back

Full Screen

Close

Quit

Algorithms

An **algorithm** will be written in a mixture of English and standard mathematical notation. Usually,

- algorithms written in a natural language are often ambiguous
- mathematical notation is not ambiguous, but still cannot be understood by machine
- algorithms written by us use various mathematical properties. We know them, but the machine doesn't.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 30 of 709

Go Back

Full Screen

Close

Quit

Programming Language

- Require a way to communicate with a machine which has essentially no intelligence or understanding.
- Translate the algorithm into a form that may be “understood” by a machine
- This “form” is usually a **program**

Program: An **algorithm** written in a programming language.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 31 of 709

Go Back

Full Screen

Close

Quit

Programs and Languages

- Every programming language has a well defined **vocabulary** and a well defined **grammar**
- Each **program** has to be written following rigid **grammatical** rules
- A programming language and the computer together form our single **computing tool**
- Each program uses *only* the primitives of the computing tool



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 32 of 709

Go Back

Full Screen

Close

Quit

Programs

Program: An **algorithm** written in the grammar of a **programming language**.

A **grammar** is a set of rules for forming sentences in a language.

Each programming language also has its own **vocabulary** and grammar just as in the case of natural languages.

We will learn the grammar of the language as we go along.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 33 of 709

Go Back

Full Screen

Close

Quit

Programming

The act of writing programs and testing them is **programming**

Even though most programming languages use essentially the same computing primitives, each programming language needs to be learned.

Programming languages differ from each other in terms of the convenience and facilities they offer even though they are all equally powerful.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 34 of 709

Go Back

Full Screen

Close

Quit

Computing Models

We consider mainly two models.

- **Functional**: A program is specified simply as a **mathematical expression**
- **Imperative**: A program is specified by a sequence of **commands** to be executed.

Programming languages also come mainly in these two flavours. We will often identify the computing model with the programming language.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 35 of 709

Go Back

Full Screen

Close

Quit

Primitives

Every programming language offers the following capabilities to define and use:

- Primitive expressions and data
- Methods of combination of expressions and data
- Methods of abstraction of both expressions and data

The functional model



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 36 of 709

Go Back

Full Screen

Close

Quit

Primitive expressions

The simplest objects and operations in the computing model. These include

- **basic data elements**: numbers, characters, truth values etc.
- **basic operations on the data elements**: addition, subtraction, multiplication, division, boolean operations, string operations etc.
- a **naming mechanism** for various quantities and expressions to be used without repeating definitions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 37 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 38 of 709

Go Back

Full Screen

Close

Quit

Methods of combination

Means of combining simple expressions and objects to obtain more complex expressions and objects.

Examples: composition of functions, inductive definitions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 39 of 709

Go Back

Full Screen

Close

Quit

Methods of abstraction

Means of naming and using groups of objects and expressions as a single unit

Examples: functions, data structures, modules, classes etc.

The Functional Model

The **functional** model is very convenient and easy to use:

- **Programs** are written (more or less) in **mathematical notation**
- It is like using a hand-held calculator
- Very interactive and so answers are immediately available
- Very convenient for developing, testing and proving algorithms

Standard ML



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 40 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 41 of 709

Go Back

Full Screen

Close

Quit

Mathematical Notation 1: Factorial

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n \times (n - 1)! & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 42 of 709

Go Back

Full Screen

Close

Quit

Mathematical Notation 2: Factorial

Or more informally,

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ 1 \times 2 \times \dots \times n & \text{otherwise} \end{cases}$$



Mathematical Notation 3: Factorial

How about this?

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ (n + 1)! / (n + 1) & \text{otherwise} \end{cases}$$

Mathematically correct but computationally incorrect!

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 43 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 44 of 709

Go Back

Full Screen

Close

Quit

A Functional Program: Factorial

```
fun fact n = if n < 1 then 1  
           else n * fact (n-1)
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 45 of 709

Go Back

Full Screen

Close

Quit

A Computation: Factorial

sml

Standard ML of New Jersey,

—



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 46 of 709

Go Back

Full Screen

Close

Quit

A Computation: Factorial

```
sml
```

```
Standard ML of New Jersey,
```

```
- fun fact n =
```

```
=
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 47 of 709

Go Back

Full Screen

Close

Quit

A Computation: Factorial

```
sml
```

```
Standard ML of New Jersey,
```

```
- fun fact n =
```

```
= if n < 1 then 1
```

```
=
```



A Computation: Factorial

sml

Standard ML of New Jersey,

```
- fun fact n =
```

```
= if n < 1 then 1
```

```
= else n * fact (n-1);
```

```
val fact = fn : int -> int
```

```
-
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 48 of 709

Go Back

Full Screen

Close

Quit

A Computation: Factorial

sml

Standard ML of New Jersey,

```
- fun fact n =  
= if n < 1 then 1  
= else n * fact (n-1);  
val fact = fn : int -> int  
- fact 8;  
val it = 40320 : int  
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 49 of 709

Go Back

Full Screen

Close

Quit

A Computation: Factorial

sml

Standard ML of New Jersey,

```
- fun fact n =  
= if n < 1 then 1  
= else n * fact (n-1);  
val fact = fn : int -> int  
- fact 8;  
val it = 40320 : int  
- fact 9;  
val it = 362880 : int  
-
```



[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)

[◀](#) [▶](#)

[◀](#) [▶](#)

[Page 50 of 709](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

A Computation: Factorial

sml

Standard ML of New Jersey,

```
- fun fact n =  
= if n < 1 then 1  
= else n * fact (n-1);  
val fact = fn : int -> int  
- fact 8;  
val it = 40320 : int  
- fact 9;  
val it = 362880 : int  
-
```

More SML



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 51 of 709

Go Back

Full Screen

Close

Quit



Standard ML

- Originated as part of a theorem-proving development project
- Runs on both Windows and UNIX environments
- Is **free!**
- <http://www.smlnj.org>

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 52 of 709

Go Back

Full Screen

Close

Quit

SML: Important Features

- Has a small vocabulary of just a few short words
- Far more “intelligent” than currently available languages:
 - automatically finds out what various names mean and
 - their correct usage
- Haskell, Miranda and Caml are a few other such languages.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 53 of 709

Go Back

Full Screen

Close

Quit



1.3. Primitives: Integer & Real

Previous: [Primitives: Integer & Real](#)

1. Algorithms & Programs
2. SML: Primitive Integer Operations 1
3. SML: Primitive Integer Operations 1
4. SML: Primitive Integer Operations 1
5. SML: Primitive Integer Operations 1
6. SML: Primitive Integer Operations 1
7. SML: Primitive Integer Operations 1
8. SML: Primitive Integer Operations 2
9. SML: Primitive Integer Operations 2
10. SML: Primitive Integer Operations 2
11. SML: Primitive Integer Operations 2
12. SML: Primitive Integer Operations 2
13. SML: Primitive Integer Operations 3
14. SML: Primitive Integer Operations 3
15. SML: Primitive Integer Operations 3
16. SML: Primitive Integer Operations 3

[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)



[Page 54 of 709](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

17. SML: Primitive Integer Operations 3
18. Quotient & Remainder
19. SML: Primitive Real Operations 1
20. SML: Primitive Real Operations 1
21. SML: Primitive Real Operations 1
22. SML: Primitive Real Operations 2
23. SML: Primitive Real Operations 3
24. SML: Primitive Real Operations 4
25. SML: Precision
26. Fibonacci Numbers
27. Euclidean Algorithm

Next: **Technical Completeness & Algorithms**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 55 of 709

Go Back

Full Screen

Close

Quit

Algorithms & Programs

- Algorithm
- Need for a formal notation
- Programs
- Programming languages
- Programming
- Functional Programming
- Standard ML

Factorial



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 56 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 57 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 1

`sml`

`Standard ML of New Jersey,`

`—`



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 58 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 1

```
sml
```

```
Standard ML of New Jersey,
```

```
- val x = 5;
```

```
val x = 5 : int
```

```
-
```



SML: Primitive Integer Operations 1

```
sml
```

```
Standard ML of New Jersey,
```

```
- val x = 5;
```

```
val x = 5 : int
```

```
- val y = 6;
```

```
val y = 6 : int
```

```
-
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 59 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 1

```
sml
```

```
Standard ML of New Jersey,
```

```
- val x = 5;
```

```
val x = 5 : int
```

```
- val y = 6;
```

```
val y = 6 : int
```

```
- x+y;
```

```
val it = 11 : int
```

```
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 60 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 1

```
sml
```

```
Standard ML of New Jersey,
```

```
- val x = 5;
```

```
val x = 5 : int
```

```
- val y = 6;
```

```
val y = 6 : int
```

```
- x+y;
```

```
val it = 11 : int
```

```
- x-y;
```

```
val it = ~1 : int
```

```
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 61 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 1

Standard ML of New Jersey,

```
- val x = 5;
```

```
val x = 5 : int
```

```
- val y = 6;
```

```
val y = 6 : int
```

```
- x+y;
```

```
val it = 11 : int
```

```
- x-y;
```

```
val it = ~1 : int
```

```
- it + 5;
```

```
val it = 4 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 62 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 2

```
val x = 5 : int
- val y = 6;
val y = 6 : int
- x+y;
val it = 11 : int
- x-y;
val it = ~1 : int
- it + 5;
val it = 4 : int
- x * y;
val it = 30 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 63 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 2

```
val y = 6 : int
- x+y;
val it = 11 : int
- x-y;
val it = ~1 : int
- it + 5;
val it = 4 : int
- x * y;
val it = 30 : int
- val a = 25;
val a = 25 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 64 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 2

```
val it = 11 : int
- x-y;
val it = ~1 : int
- it + 5;
val it = 4 : int
- x * y;
val it = 30 : int
- val a = 25;
val a = 25 : int
- val b = 7;
val b = 7 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 65 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 2

```
val it = ~1 : int
- it + 5;
val it = 4 : int
- x * y;
val it = 30 : int
- val a = 25;
val a = 25 : int
- val b = 7;
val b = 7 : int
- val q = a div b;
val q = 3 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 66 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 2

```
- x * y;
```

```
val it = 30 : int
```

```
- val a = 25;
```

```
val a = 25 : int
```

```
- val b = 7;
```

```
val b = 7 : int
```

```
- val q = a div b;
```

```
val q = 3 : int
```

```
- val r = a mod b;
```

```
GC #0.0.0.0.2.45:      (0 ms)
```

```
val r = 4 : int
```

```
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 67 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 3

```
- val a = 25;  
val a = 25 : int  
- val b = 7;  
val b = 7 : int  
- val q = a div b;  
val q = 3 : int  
- val r = a mod b;  
GC #0.0.0.0.2.45:      (0 ms)  
val r = 4 : int  
- a = b*q + r;  
val it = true : bool
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 68 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 3

```
- val b = 7;  
val b = 7 : int  
- val q = a div b;  
val q = 3 : int  
- val r = a mod b;  
GC #0.0.0.0.2.45:      (0 ms)  
val r = 4 : int  
- a = b*q + r;  
val it = true : bool  
- val c = ~7;  
val c = ~7 : int
```



[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)



[Page 69 of 709](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

SML: Primitive Integer Operations 3

```
- val q = a div b;
```

```
val q = 3 : int
```

```
- val r = a mod b;
```

```
GC #0.0.0.0.2.45:      (0 ms)
```

```
val r = 4 : int
```

```
- a = b*q + r;
```

```
val it = true : bool
```

```
- val c = ~7;
```

```
val c = ~7 : int
```

```
- val q1 = a div c;
```

```
val q1 = ~4 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 70 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 3

```
- val r = a mod b;  
GC #0.0.0.0.2.45:      (0 ms)  
val r = 4 : int  
- a = b*q + r;  
val it = true : bool  
- val c = ~7;  
val c = ~7 : int  
- val q1 = a div c;  
val q1 = ~4 : int  
- val r1 = a mod c;  
val r1 = ~3 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 71 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Integer Operations 3

```
val r = 4 : int
- a = b*q + r;
val it = true : bool
- val c = ~7;
val c = ~7 : int
- val q1 = a div c;
val q1 = ~4 : int
- val r1 = a mod c;
val r1 = ~3 : int
- a = c*q1 + r1;
val it = true : bool
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 72 of 709

Go Back

Full Screen

Close

Quit



Quotient & Remainder

For any two integers **a** and **b**, the **quotient q** and **remainder r** are uniquely determined to satisfy

-

$$a = b \times q + r$$

-

$$\begin{cases} 0 \leq r < b & \text{when } b > 0 \\ b < r \leq 0 & \text{when } b < 0 \end{cases}$$

So $0 \leq |r| < |b|$ always.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 73 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 74 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Real Operations 1

```
sml
```

```
Standard ML of New Jersey,
```

```
- val real_a = real a;
```

```
val real_a = 25.0 : real
```

```
-
```

SML: Primitive Real Operations 1

sml

Standard ML of New Jersey,

```
- val real_a = real a;
```

```
val real_a = 25.0 : real
```

```
- real_a + b;
```

```
stdIn:40.1-40.11 Error: operator and
```

```
operator domain: real * real
```

```
operand:          real * int
```

```
in expression:
```

```
  real_a + b
```

```
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 75 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Real Operations 1



```
stdIn:40.1-40.11 Error: operator and  
operator domain: real * real  
operand:          real * int  
in expression:  
    real_a + b  
- b + real_a;  
stdIn:1.1-2.6 Error: operator and ope  
operator domain: int * int  
operand:          int * real  
in expression:  
    b + real_a
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 76 of 109

Go Back

Full Screen

Close

Quit

SML: Primitive Real Operations 2

```
- val a = 25.0;  
val a = 25.0 : real  
- val b = 7.0;  
val b = 7.0 : real  
- a/b;  
val it = 3.57142857143 : real  
- a div b;  
stdIn:49.3-49.6 Error: overloaded val  
  symbol: div  
  type: real  
GC #0.0.0.0.3.98:      (0 ms)  
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 77 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Real Operations 3

```
- val c = a/b;  
val c = 3.57142857143 : real  
- trunc(c);  
val it = 3 : int  
- trunc (c + 0.5);  
val it = 4 : int  
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 78 of 709

Go Back

Full Screen

Close

Quit

SML: Primitive Real Operations 4

```
- val d = 3.0E10;  
val d = 300000000000.0 : real  
- val pi = 0.314159265E1;  
val pi = 3.14159265 : real  
- - d+pi;  
val it = 300000000003.1 : real  
- d-pi;  
val it = 299999999996.9 : real  
- pi + d;  
val it = 300000000003.1 : real  
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 79 of 709

Go Back

Full Screen

Close

Quit

SML: Precision

```
- pi + d*10.0;  
val it = 30000000000003.0 : real  
- pi + d*100.0;  
val it = 3E12 : real  
- d*100.0 + pi;  
val it = 3E12 : real  
- d*100.0 - pi;  
val it = 3E12 : real  
- d*10.0 - pi;  
val it = 29999999999997.0 : real  
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 80 of 709

Go Back

Full Screen

Close

Quit

1.4. Example: Fibonacci

1. Fibonacci Numbers: 1
2. Fibonacci Numbers: 2
3. Fibonacci Numbers: 3
4. Fibonacci Numbers: 4
5. Fibonacci Numbers: 5
6. Is $F(n, 1, 1) = F(n)$?
7. Trial & Error
8. Generalization
9. Proof
10. Another Generalization
11. Try Proving it!
12. Another Generalization
13. Try Proving it!
14. Complexity
15. Complexity
16. Time Complexity: \mathcal{R}
17. Time Complexity
18. Time Complexity: \mathcal{R}



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 81 of 709

Go Back

Full Screen

Close

Quit

19. Bound on \mathcal{R}

20. Other Bounds: \mathcal{C}_F

21. Other Bounds: \mathcal{A}_F



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 82 of 709

Go Back

Full Screen

Close

Quit



Fibonacci Numbers: 1

$$\begin{cases} \mathbf{F}(0) = 1 \\ \mathbf{F}(1) = 1 \\ \mathbf{F}(n) = \mathbf{F}(n - 1) + \mathbf{F}(n - 2) \text{ if } n > 1 \end{cases}$$

```
fun fib (n) =  
  if (n = 0) orelse (n = 1)  
  then 1  
  else fib (n-1) + fib (n-2) ;
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 83 of 709

Go Back

Full Screen

Close

Quit



Fibonacci Numbers: 2

$$\begin{cases} \mathbf{F}(0) = 1 \\ \mathbf{F}(1) = 1 \\ \mathbf{F}(n) = \mathbf{F}(n - 1) + \mathbf{F}(n - 2) \quad \text{if } n > 1 \end{cases}$$

Alternatively,

$$\mathbf{F}(n) = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \mathbf{F}(n - 1) + \mathbf{F}(n - 2) & \text{if } n > 1 \end{cases}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 84 of 709

Go Back

Full Screen

Close

Quit

Fibonacci Numbers: 3

```
fun fib (n) =  
  if (n = 0) orelse (n = 1)  
  then 1  
  else fib (n-1) + fib (n-2) ;
```

Alternatively,

```
fun fib (n) =  
  if (n = 0) then 1  
  else if (n = 1) then 1  
  else fib (n-1) + fib (n-2) ;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 85 of 709

Go Back

Full Screen

Close

Quit

Fibonacci Numbers: 4

$$\begin{cases} \mathbf{F}(0) = 1 \\ \mathbf{F}(1) = 1 \\ \mathbf{F}(n) = \mathbf{F}(n - 1) + \mathbf{F}(n - 2) \quad \text{if } n > 1 \end{cases}$$

Alternatively,

$$\mathbf{F}(n) = \mathbf{Fa}(n, 1, 1)$$

where

$$\mathbf{Fa}(n, a, b) = \begin{cases} a & \text{if } n = 0 \\ b & \text{if } n = 1 \\ \mathbf{Fa}(n - 1, b, a + b) & \text{if } n > 1 \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 86 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 87 of 709

Go Back

Full Screen

Close

Quit

Fibonacci Numbers: 5

```
fun fib_a (n, a, b) =  
  if (n = 0) then a  
  else if (n = 1) then b  
  else fib_a (n, b, a+b);  
  
fun fib (n) = fib_a (n, 1, 1);
```



Is $F_a(n, 1, 1) = F(n)$?

Intuition F_a is a generalization of F .

Question 1 What does it actually generalize to?

Question 2 Does the generalization have a **non-recursive** form?

Trial & Error Can we use trial and error to get a **non-recursive** form?

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 88 of 709

Go Back

Full Screen

Close

Quit



Trial & Error

$$\mathbf{Fa}(2, a, b) = a + b$$

$$\begin{aligned}\mathbf{Fa}(3, a, b) &= \mathbf{Fa}(2, b, a + b) \\ &= a + 2b\end{aligned}$$

$$\begin{aligned}\mathbf{Fa}(4, a, b) &= \mathbf{Fa}(3, b, a + b) \\ &= \mathbf{Fa}(2, a + b, a + 2b) \\ &= 2a + 3b\end{aligned}$$

$$\begin{aligned}\mathbf{Fa}(5, a, b) &= \mathbf{Fa}(2, a + 2b, 2a + 3b) \\ &= 3a + 5b\end{aligned}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 89 of 709

Go Back

Full Screen

Close

Quit

Generalization

- $\mathbf{Fa}(0, a, b) = a$
- $\mathbf{Fa}(1, a, b) = b$
- $\mathbf{Fa}(n, a, b) = a\mathbf{F}(n - 2) + b\mathbf{F}(n - 1)$
- When $a = 1$ and $b = 1$, for all $n \geq 0$,
 $\mathbf{Fa}(n, a, b) = \mathbf{F}(n)$

Theorem 1 *For all integers a, b and $n > 1$,*

$$\mathbf{Fa}(n, a, b) = a\mathbf{F}(n - 2) + b\mathbf{F}(n - 1)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 90 of 709

Go Back

Full Screen

Close

Quit

$$n > 1$$

Proof:

Basis For $n = 2$, $\mathbf{Fa}(2, a, b) = a + b = a\mathbf{F}(0) + b\mathbf{F}(1)$

Induction hypothesis (IH) Assume

$\mathbf{Fa}(k, a, b) = a\mathbf{F}(k - 2) + b\mathbf{F}(k - 1)$,
for some $k > 1$ and all integers a, b

Induction Step

$$\begin{aligned} & \mathbf{Fa}(k + 1, a, b) \\ &= \mathbf{Fa}(k, b, a + b) \quad \text{Definition of } \mathbf{Fa} \\ &= b\mathbf{F}(k - 2) + (a + b)\mathbf{F}(k - 1) \quad \text{IH} \\ &= a\mathbf{F}(k - 1) + b(\mathbf{F}(k - 2) + \mathbf{F}(k - 1)) \\ &= a\mathbf{F}(k - 1) + b\mathbf{F}(k) \quad \text{Definition of } \mathbf{F} \end{aligned}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 91 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 92 of 709

Go Back

Full Screen

Close

Quit

Another Generalization

Try to prove a different and more direct theorem.

Theorem 2 For all integers $n > 1$,

$$\mathbf{Fa}(n, 1, 1) = \mathbf{F}(n)$$

Try Proving it!

Proof: By induction on $n > 1$.

Basis For $n = 0$ and $n = 1$, $\mathbf{Fa}(0, 1, 1) = 1 = \mathbf{Fa}(1)$

Induction hypothesis (IH) Assume

$\mathbf{Fa}(k, 1, 1) = \mathbf{F}(k)$, for some $k > 1$

Induction Step

$$\begin{aligned} & \mathbf{Fa}(k + 1, 1, 1) \\ &= \mathbf{Fa}(k, 1, 2) \quad \text{Definition of Fa} \\ &= ??? \quad \text{IH} \end{aligned}$$

STUCK!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 93 of 709

Go Back

Full Screen

Close

Quit

Another Generalization

Try to prove a different and more direct theorem.

Theorem 3 For all integers $n \geq 1$ and $j \geq 1$,

$$\mathbf{Fa}(n, \mathbf{F}(j - 1), \mathbf{F}(j)) = \mathbf{F}(n + j - 1)$$

Proof: By induction on $n \geq 1$, for all values of $j \geq 1$. \square

Corollary 4 For all integers $n \geq 1$,

$$\mathbf{Fa}(n, \mathbf{F}(0), \mathbf{F}(1)) = \mathbf{F}(n)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 94 of 709

Go Back

Full Screen

Close

Quit

Try Proving it!

Basis For $n = 1$, $\mathbf{Fa}(1, \mathbf{F}(j - 1), \mathbf{F}(j)) = \mathbf{F}(j)$

Induction hypothesis (IH) For some $k > 1$ and all $j \geq 1$,

$$\mathbf{Fa}(k, \mathbf{F}(j - 1), \mathbf{F}(j)) = \mathbf{F}(k + j - 1)$$

Induction Step We need to prove $\mathbf{Fa}(k + 1, \mathbf{F}(j - 1), \mathbf{F}(j)) = \mathbf{F}(k + j)$.

$$\begin{aligned} & \mathbf{Fa}(k + 1, \mathbf{F}(j - 1), \mathbf{F}(j)) \\ &= \mathbf{Fa}(k, \mathbf{F}(j), \mathbf{F}(j - 1) + \mathbf{F}(j)) \\ &= \mathbf{Fa}(k, \mathbf{F}(j), \mathbf{F}(j + 1)) \\ &= \mathbf{Fa}(k + (j + 1) - 1) \quad \text{IH} \\ &= \mathbf{Fa}(k + j) \end{aligned}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 95 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 96 of 709

Go Back

Full Screen

Close

Quit

Complexity

- Time complexity:
 - No of additions: $A_F(n)$
 - No of comparisons: $C_F(n)$
 - No of recursive calls to F : $R_F(n)$
- Space complexity:



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 97 of 709

Go Back

Full Screen

Close

Quit

Complexity

- Time complexity:
- Space complexity:
 - left-to-right evaluation: $\mathcal{LR}_F(n)$
 - arbitrary evaluation: $\mathcal{U}_F(n)$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 98 of 709

Go Back

Full Screen

Close

Quit

Time Complexity: \mathcal{R}

- **Hardware** operations like **addition** and **comparisons** are usually very fast compared to **software** operations like **recursion unfolding**
- The number of **recursion unfoldings** also includes **comparisons** and **additions**.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 99 of 709

Go Back

Full Screen

Close

Quit

Time Complexity

- It is enough to put **bounds on the number of recursion unfoldings** and not worry about individual hardware operations.
- Similar theorems may be proved for **any operation** by counting and induction.

So we concentrate on \mathcal{R} .



Time Complexity: \mathcal{R}

- $\mathcal{R}_F(0) = \mathcal{R}_F(1) = 0$
- $\mathcal{R}_F(n) = 2 + \mathcal{R}_F(n - 1) + \mathcal{R}_F(n - 2)$
for $n > 1$

To solve the equation as initial value problem and obtain an upper bound we guess the following theorem.

Theorem 5 $\mathcal{R}_F(n) \leq 2^{n-1}$ for all $n > 2$

Proof: By induction on $n > 2$. \square

Bound on \mathcal{R}

Basis $n = 3$. $\mathcal{R}_{\mathbf{F}}(3) = 2 + 2 + 0 \leq 2^{3-1}$

Induction hypothesis (IH) For some $k > 2$, $\mathcal{R}_{\mathbf{F}}(k) \leq 2^{k-1}$

Induction Step If $n = k + 1$ then $n > 3$

$$\begin{aligned} & \mathcal{R}_{\mathbf{F}}(n) \\ &= 2 + \mathcal{R}_{\mathbf{F}}(n-2) + \mathcal{R}_{\mathbf{F}}(n-1) \\ &\leq 2 + 2^{n-3} + 2^{n-2} \quad (\text{IH}) \\ &\leq 2 \cdot 2^{n-3} + 2^{n-2} \quad \text{for } n > 3, 2^{n-3} \geq 2 \\ &= 2^{n-2} + 2^{n-2} \\ &= 2^{n-1} \end{aligned}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 101 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 102 of 709

Go Back

Full Screen

Close

Quit

Other Bounds: \mathcal{C}_F

One comparison for each call.

- $\mathcal{C}_F(0) = \mathcal{C}_F(1) = 1$
- $\mathcal{C}_F(n) = 1 + \mathcal{C}_F(n-1) + \mathcal{C}_F(n-2)$ for $n > 1$

Theorem 6 $\mathcal{C}_F(n) \leq 2^n$ for all $n \geq 0$.



Other Bounds: \mathcal{A}_F

No additions for the basis and **one** addition in each call.

- $\mathcal{A}_F(0) = \mathcal{A}_F(1) = 0$
- $\mathcal{A}_F(n) = 1 + \mathcal{A}_F(n - 1) + \mathcal{A}_F(n - 2)$
for $n > 1$

Theorem 7 $\mathcal{A}_F(n) \leq 2^{n-1}$ for all $n > 0$.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 104 of 709

Go Back

Full Screen

Close

Quit

1.5. Primitives: Booleans

1. Boolean Conditions
2. Booleans in SML
3. Booleans in SML
4. \wedge vs. `andalso`
5. \vee vs. `orelse`
6. SML: `orelse`
7. SML: `andalso`
8. `and`, `andalso`, \perp
9. `or`, `orelse`, \perp
10. Complex Boolean Conditions



Boolean Conditions

- Two (truth) value set : `{true, false}`
- Boolean conditions are those statements or names which can take only truth values.

Examples: `n < 0`, `true`,

- Negation operator: `not`

Examples: `not (n < 0)`, `not true`, `not false`

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 105 of 709

Go Back

Full Screen

Close

Quit

Booleans in SML

Standard ML of New Jersey,

```
- val tt = true;
```

```
val tt = true : bool
```

```
- not(tt);
```

```
val it = false : bool
```

```
- val n = 10;
```

```
val n = 10 : int
```

```
- n < 10;
```

```
val it = false : bool
```

```
- not (n < 10);
```

```
val it = true : bool
```

```
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 106 of 709

Go Back

Full Screen

Close

Quit



Booleans in SML

Examples:

```
- (n >= 10) andalso (n=10);  
val it = true : bool  
- n < 0 orelse n >= 10;  
val it = true : bool  
- not ((n >= 10)  
= andalso (n=10))  
= orelse n < 0 orelse n >= 10;  
val it = true : bool  
-
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 107 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 108 of 709

Go Back

Full Screen

Close

Quit

\wedge VS. andalso

p	q	$p \wedge q$	p andalso q
true	true	true	true
true	false	false	false
false	true	false	false
false	false	false	false



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 109 of 709

Go Back

Full Screen

Close

Quit

\vee VS. or else

p	q	$p \vee q$	p or else q
true	true	true	true
true	false	true	true
false	true	true	true
false	false	false	false

SML: orelse

Standard ML of New Jersey,

```
- val tt = true;
```

```
val tt = true : bool
```

```
- val ff = false;
```

```
val ff = false : bool
```

```
- fun gtz n = if n=1 then true
```

```
=           else gtz (n-1);
```

```
val gtz = fn : int -> bool
```

```
- tt orelse (gtz 0);
```

```
val it = true : bool
```

```
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 110 of 709

Go Back

Full Screen

Close

Quit



SML: andalso

```
- (gtz 0) orelse tt;
```

Interrupt

```
- ff andalso (gtz 0);  
val it = false : bool  
- (gtz 0) andalso ff;
```

Interrupt

```
-
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 111 of 709

Go Back

Full Screen

Close

Quit



and, andalso, \perp

p	q	$p \wedge q$	$p \text{ andalso } q$
true	\perp	\perp	\perp
\perp	true	\perp	\perp
false	\perp	false	false
\perp	false	false	\perp

\wedge is commutative whereas andalso is not.



or, orelse, \perp

p	q	$p \vee q$	$p \text{ orelse } q$
true	\perp	true	true
\perp	true	true	\perp
false	\perp	\perp	\perp
\perp	false	\perp	\perp

\vee is commutative whereas `orelse` is not.



Complex Boolean Conditions

Assume p and q are boolean conditions

$$p \text{ or else } q \equiv \text{if } p \text{ then true else } q$$
$$p \text{ and also } q \equiv \text{if } p \text{ then } q \text{ else false}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 114 of 709

Go Back

Full Screen

Close

Quit



2. Algorithms: Design & Refinement

2.1. Technical Completeness & Algorithms

1. Recapitulation: Integers & Real
2. Recap: Integer Operations
3. Recapitulation: Real Operations
4. Recapitulation: Simple Algorithms
5. More Algorithms
6. Powering: Math
7. Powering: SML
8. Technical completeness
9. What SML says
10. Technical completeness
11. What SML says ... *contd*
12. Powering: Math 1
13. Powering: SML 1
14. Technical Completeness
15. What SML says

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 115 of 709

Go Back

Full Screen

Close

Quit

16. Powering: Integer Version
17. Exceptions: A new primitive
18. Integer Power: SML
19. Integer Square Root 1
20. Integer Square Root 2
21. An analysis
22. Algorithmic idea
23. Algorithm: isqrt
24. Algorithm: shrink
25. SML: shrink
26. SML: intsqrt
27. Run it!
28. SML: Reorganizing Code
29. Intsqrt: Reorganized
30. shrink: Another algorithm
31. Shrink2: SML
32. Shrink2: SML ... *contd*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 116 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 117 of 709

Go Back

Full Screen

Close

Quit

Recapitulation: Integers & Real

- Primitive Integer Operations
- Primitive Real Operations
- Some algorithms

Forward



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 118 of 709

Go Back

Full Screen

Close

Quit

Recap: Integer Operations

- Primitive Integer Operations
 - Naming, $+$, $-$, \sim
 - Multiplication, division
 - Quotient & remainder
- Primitive Real Operations
- Some algorithms

Back

Recapitulation: Real Operations

- Primitive Integer Operations
- Primitive Real Operations
 - Integer to Real
 - Real to Integer
 - Real addition & subtraction
 - Real division
 - Real Precision
- Some algorithms

Back



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 119 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 120 of 709

Go Back

Full Screen

Close

Quit

Recapitulation: Simple Algorithms

- Primitive Integer Operations
- Primitive Real Operations
- Some algorithms
 - Factorial
 - Fibonacci
 - Euclidean GCD

Back



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 121 of 709

Go Back

Full Screen

Close

Quit

More Algorithms

- Powering
- Integer square root
- Combinations ${}^n C_k$



Powering: Math

For any integer or real number $x \neq 0$
and **non-negative integer** n

$$x^n = \underbrace{x \times x \times \cdots \times x}_{n \text{ times}}$$

Noting that $x^0 = 1$ we give an inductive definition:

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x^{n-1} \times x & \text{otherwise} \end{cases}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 122 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 123 of 709

Go Back

Full Screen

Close

Quit

Powering: SML

```
fun power (x:real, n) =  
  if n = 0  
  then 1.0  
  else power (x, n-1) * x
```

Is it **technically complete**?



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 124 of 709

Go Back

Full Screen

Close

Quit

Technical completeness

Can it be always guaranteed that

- x will be **real**?
- n will be **integer**?
- n will be **non-negative**?
- $x \neq 0$?

If $x = 0$ what is 0.0^0 ?

What SML says

sml

Standard ML of New Jersey

- use `"/tmp/power.sml"`;

[opening `/tmp/power.sml`]

```
val power = fn : real * int ->
              real
```

```
val it = () : unit
```

Can it be always guaranteed that

- x will be real? **YES**
- n will be integer? **YES**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 125 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 126 of 709

Go Back

Full Screen

Close

Quit

Technical completeness

Can it be always guaranteed that

- n will be non-negative? **NO**
- $x \neq 0$? **NO**

If $x = 0$ what is 0.0^0 ?

```
- power(0.0, 0);  
val it = 1.0 : real
```

What SML says ...

contd

sml

Standard ML of New Jersey

```
val power = fn : real * int -> real
```

```
val it = () : unit
```

```
- power (~2.5, 0);
```

```
val it = 1.0 : real
```

```
- power (0.0, 3);
```

```
val it = 0.0 : real
```

```
- power (2.5, ~3)
```

Goes on forever!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

real

Contents



Page 127 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 128 of 709

Go Back

Full Screen

Close

Quit

Powering: Math 1

For any real number x and integer n

$$x^n = \begin{cases} 1.0/x^{-n} & \text{if } n < 0 \\ 1 & \text{if } n = 0 \\ x^{n-1} \times x & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 129 of 709

Go Back

Full Screen

Close

Quit

Powering: SML 1

```
fun power (x, n) =  
  if n < 0  
  then 1.0/power(x, ~n)  
  else if n = 0  
  then 1.0  
  else power(x, n-1) * x
```

Is this definition technically complete?



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 130 of 709

Go Back

Full Screen

Close

Quit

Technical Completeness

- $0.0^0 = 1.0$ whereas $0.0^n = 0$ for positive n
- What if $x = 0.0$ and $n = -m < 0$?

Then

$$\begin{aligned} & 0.0^n \\ &= 1.0 / (0.0^m) \\ &= 1.0 / 0.0 \end{aligned}$$

Division by zero!

What SML says

```
- power (2.5, ~2);  
val it = 0.16 : real  
- power (~2.5, ~2);  
val it = 0.16 : real  
- power (0.0, 2);  
val it = 0.0 : real  
- power (0.0, ~2);  
val it = inf : real  
-
```

SML is somewhat more understanding than most languages



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 131 of 709

Go Back

Full Screen

Close

Quit

Powering: Integer Version

$$x^n = \begin{cases} \text{undefined} & \text{if } n < 0 \\ \text{undefined} & \text{if } x = 0 \& n = 0 \\ 1 & \text{if } x \neq 0 \& n = 0 \\ x^{n-1} \times x & \text{otherwise} \end{cases}$$

Technical completeness requires us to consider the case $n < 0$. Otherwise, the computation can go on **forever**

Notation: \perp denotes the *undefined*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 132 of 709

Go Back

Full Screen

Close

Quit

Exceptions: A new primitive

```
exception negExponent;  
exception zeroPowerZero;  
fun intpower (x, n) =  
  if n < 0  
  then raise negExponent  
  else if n = 0  
  then if x=0  
        then raise zeroPowerZero  
        else 1  
  else intpower (x, n-1) * x
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 133 of 709

Go Back

Full Screen

Close

Quit

Integer Power: SML

```
- intpower(3, 4);  
val it = 81 : int  
- intpower(~3, 5);  
val it = ~243 : int  
- intpower(3, ~4);
```

```
uncaught exception negExponent  
  raised at: intpower.sml:4.16-4.32  
- intpower(0, 0);
```

```
uncaught exception zeroPowerZero  
  raised at: stdIn:24.26-24.39
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 13 of 709

Go Back

Full Screen

Close

Quit

[Back to More Algos](#)

Integer Square Root 1

$$\text{isqrt}(n) = \lfloor \sqrt{n} \rfloor$$

```
- fun isqrt n =  
    trunc (Real.Math.sqrt  
           (real (n)));  
val isqrt = fn : int -> int  
- isqrt (38);  
val it = 6 : int  
- isqrt (~38);  
uncaught exception domain error  
  raised at: boot/real64.sml:89:32-89:32  
-
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 135 of 709

Go Back

Full Screen

Close

Quit

Integer Square Root 2

Suppose `Real.Math.sqrt` were not available to us!

$isqrt(n)$ of a non-negative integer n is the integer $k \geq 0$ such that $k^2 \leq n < (k + 1)^2$

That is,

$$isqrt(n) = \begin{cases} \perp & \text{if } n < 0 \\ k & \text{otherwise} \end{cases}$$

where $0 \leq k^2 \leq n < (k + 1)^2$.

This value of k is **unique**!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 136 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 137 of 709

Go Back

Full Screen

Close

Quit

An analysis

$$\begin{aligned}0 &\leq k^2 \leq n < (k + 1)^2 \\ \Rightarrow 0 &\leq k \leq \sqrt{n} < k + 1 \\ \Rightarrow 0 &\leq k \leq n\end{aligned}$$

Strategy. Use this fact to close in on the value of k . Start with the interval $[l, u] = [0, n]$ and try to **shrink** it till it collapses to the interval $[k, k]$ which contains a single value.

Algorithmic idea

If $n = 0$ then $isqrt(n) = 0$.

Otherwise with $[l, u] = [0, n]$ and

$$l^2 \leq n < u^2$$

use one or both of the following to shrink the interval $[l, u]$.

- if $(l + 1)^2 \leq n$ then try $[l + 1, u]$
otherwise $l^2 \leq n < (l + 1)^2$ and $k = l$
- if $u^2 > n$ then try $[l, u - 1]$
otherwise $(u - 1)^2 \leq n < u^2$ and
 $k = u - 1$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 138 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 139 of 709

Go Back

Full Screen

Close

Quit

Algorithm: isqrt

$$\text{isqrt}(n) = \begin{cases} \perp & \text{if } n < 0 \\ 0 & \text{if } n = 0 \\ \text{shrink}(n, 0, n) & \text{if } n > 0 \end{cases}$$

where

Algorithm: shrink

$shrink(n, l, u) =$

$$\left\{ \begin{array}{ll} l & \text{if } l = u \\ shrink(n, l + 1, u) & \text{if } l < u \\ & \text{and } (l + 1)^2 \leq n \\ l & \text{if } (l + 1)^2 > n \\ shrink(n, l, u - 1) & \text{if } l < u \\ & \text{and } u^2 > n \\ u - 1 & \text{if } l < u \\ & \text{and } (u - 1)^2 \leq n \\ \perp & \text{if } l > u \end{array} \right.$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 140 of 709

Go Back

Full Screen

Close

Quit

SML: shrink

```
exception intervalError;  
fun shrink (n, l, u) =  
  if l > u orelse  
    l * l > n orelse  
    u * u < n  
  then raise intervalError  
  else if (l+1) * (l+1) <= n  
  then shrink (n, l+1, u)  
  else l;
```

intsqrt



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 141 of 709

Go Back

Full Screen

Close

Quit



SML: intsqrt

```
exception negError;  
fun intsqrt n =  
  if n < 0  
  then raise negError  
  else if n = 0  
  then 0  
  else shrink (n, 0, n)
```

shrink

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 142 of 709

Go Back

Full Screen

Close

Quit

Run it!

```
exception intervalError
val shrink =
fn : int * int * int -> int
exception negError
val intsqrt = fn : int -> int
val it = () : unit
- intsqrt 8;
val it = 2 : int
- intsqrt 16;
val it = 4 : int
- intsqrt 99;
val it = 9 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 143 of 709

Go Back

Full Screen

Close

Quit



SML: Reorganizing Code

- `shrink` was used to develop `intsqrt`
- Is `shrink` general-purpose enough to be kept separate?
- Shouldn't `shrink` be placed within `intsqrt`?

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 144 of 709

Go Back

Full Screen

Close

Quit

Intsqrt: Reorganized

```
exception negError;  
fun intsqrt n =  
  let fun shrink (n, l, u) =  
    in if n < 0  
       then raise negError  
       else if n = 0  
          then 0  
          else shrink (n, 0, n)  
    end  
  end
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 145 of 709

Go Back

Full Screen

Close

Quit

shrink: Another algorithm

Shrink

$shrink2(n, l, u) =$

$$\left\{ \begin{array}{ll} l & \text{if } l = u \text{ or } u = l + 1 \\ shrink2(n, m, u) & \text{if } l < u \\ & \text{and } m^2 \leq n \\ shrink2(n, l, m) & \text{if } l < u \\ & \text{and } m^2 > n \\ \perp & \text{if } l > u \end{array} \right.$$

where $m = (l + u) \text{ div } 2$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 146 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 147 of 709

Go Back

Full Screen

Close

Quit

Shrink2: SML

```
fun shrink2 (n, l, u) =  
  if l>u orelse  
    l*l > n orelse  
    u*u < n  
  then raise intervalError  
  else if l = u  
  then l
```



Shrink2: SML ... *contd*

```
else
let val m = (l+u) div 2;
    val msqr = m*m
in if msqr <= n
    then shrink (n, m, u)
    else shrink (n, l, m)
end;
```

[Back to More Algos](#)

[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)

[◀](#) [▶](#)

[◀](#) [▶](#)

Page 148 of 709

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 149 of 709

Go Back

Full Screen

Close

Quit

2.2. Algorithm Refinement

1. Recap: More Algorithms
2. Recap: Power
3. Recap: Technical completeness
4. Recap: More Algorithms
5. Intsqrt: Reorganized
6. Intsqrt: Reorganized
7. Some More Algorithms
8. Combinations: Math
9. Combinations: Details
10. Combinations: SML
11. Perfect Numbers
12. Refinement
13. Perfect Numbers: SML
14. $\sum_i^u \text{ifdivisor}(k)$
15. SML: sum_divisors
16. *ifdivisor* and ifdivisor
17. SML: Assembly 1

18. SML: Assembly 2
19. Perfect Numbers . . . *contd.*
20. Perfect Numbers . . . *contd.*
21. SML: Assembly 3
22. Perfect Numbers: Run
23. Perfect Numbers: Run
24. SML: Code variations
25. SML: Code variations
26. SML: Code variations
27. Summation: Generalizations
28. Algorithmic Improvements:
29. Algorithmic Variations
30. Algorithmic Variations



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 150 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 151 of 709

Go Back

Full Screen

Close

Quit

Recap: More Algorithms

- x^n for real and integer x
- Integer square root

Forward



Recap: Power

- x^n for real and integer x
 - Technical Completeness
 - * Undefinedness
 - * Termination
 - More complete definition for real x
 - Power of an integer
 - \perp and exceptions
- Integer square root

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 152 of 709

Go Back

Full Screen

Close

Quit

Recap: Technical completeness

Can it be always guaranteed that

- x will be **real**? **YES**
- n will be **integer**? **YES**
- n will be **non-negative**? **NO**
- $x \neq 0$? **NO**

If $x = 0$ what is 0.0^0 ?

INFINITE COMPUTATION



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 153 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 154 of 709

Go Back

Full Screen

Close

Quit

Recap: More Algorithms

- x^n for real and integer x
- Integer square root
 - Analysis
 - Algorithmic idea
 - Algorithm
 - where
 - and `let ...in ...end`

Intsqrt: Reorganized

```
exception negError;  
exception intervalError;  
fun intsqrt n =  
  let fun shrink (n, l, u) =  
        if l > u orelse  
          l * l > n orelse  
          u * u < n  
        then raise intervalError  
        else if (l + 1) * (l + 1) <= n  
            then shrink (n, l + 1, u)  
            else l;  
  in shrink (n, 0, n) end
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 155 of 709

Go Back

Full Screen

Close

Quit



Intsqr: Reorganized

```
in if n<0
  then raise negError
  else if n=0
    then 0
    else shrink (n, 0, n)
end
```

Back

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 156 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 157 of 709

Go Back

Full Screen

Close

Quit

Some More Algorithms

- **Combinations**
- **Perfect Numbers**

Combinations: Math

$$\begin{aligned} {}^n C_k &= \frac{n!}{(n-k)!k!} \\ &= \frac{n(n-1)\cdots(n-k+1)}{k!} \\ &= \frac{n(n-1)\cdots(k+1)}{(n-k)!} \\ &= {}^{n-1}C_{k-1} + {}^{n-1}C_k \end{aligned}$$

Since we already have the function `fact`, we may program ${}^n C_k$ using any of the above identities. Let's program it using the last one.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 158 of 709

Go Back

Full Screen

Close

Quit

Combinations: Details

Given a set of $n \geq 0$ elements, find the number of subsets of k elements, where $0 \leq k \leq n$

$${}^n C_k = \begin{cases} \perp & \text{if } n < 0 \text{ or} \\ & k < 0 \text{ or} \\ & k > n \\ 1 & \text{if } n = 0 \text{ or} \\ & k = 0 \text{ or} \\ & k = n \\ {}^{n-1} C_{k-1} + {}^{n-1} C_k & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 159 of 709

Go Back

Full Screen

Close

Quit

Combinations: SML

```
exception invalid_arg;  
fun comb (n, k) =  
    if n < 0 orelse  
      k < 0 orelse  
      k > n  
    then raise invalid_arg  
    else if n = 0 orelse  
          k = 0 orelse  
          n = k  
    then 1  
    else comb (n-1, k-1) +  
          comb (n-1, k);
```

[Back to Some More Algorithms](#)



[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)

[◀](#) [▶](#)

[◀](#) [▶](#)

[Page 160 of 709](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

Perfect Numbers

An integer $n > 0$ is **perfect** if it equals the sum of all its **proper divisors**.

A divisor $k|n$ is **proper** if $0 < k < n$

$$k|n \iff n \bmod k = 0$$

perfect(n)

$$\iff n = \sum \{k : 0 < k < n, k|n\}$$

$$\iff n = \sum_{k=1}^{n-1} \text{ifdivisor}(k)$$

where



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 161 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 162 of 709

Go Back

Full Screen

Close

Quit

Refinement

1. $ifdivisor(k)$ needs to be defined
2. $\sum_{k=1}^{n-1} ifdivisor(k)$ needs to be defined **algorithmically**.



Perfect Numbers: SML

```
exception nonpositive;  
fun perfect (n) =  
  if n <= 0  
  then raise nonpositive  
  else  
    n = sum_divisors (1, n-1)
```

where `sum_divisors` needs to be defined

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 163 of 709

Go Back

Full Screen

Close

Quit



$$\sum_l^u \text{ifdivisor}(k)$$

$$\sum_{k=l}^u \text{ifdivisor}(k) =$$

$$\begin{cases} 0 & \text{if } l > u \\ \text{ifdivisor}(l) + \sum_{k=l+1}^{n-1} \text{ifdivisor}(k) & \text{otherwise} \end{cases}$$

where $\text{ifdivisor}(k)$ needs to be defined



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 165 of 709

Go Back

Full Screen

Close

Quit

SML: `sum_divisors`

From the algorithmic definition of

$$\sum_{k=l}^u \textit{ifdivisor}(k)$$

```
fun sum_divisors (l, u) =  
  if l > u  
  then 0  
  else ifdivisor (l) +  
        sum_divisors (l+1, u)
```

where *ifdivisor*(*k*) still needs to be defined

ifdivisor and `ifdivisor`

$$\textit{ifdivisor}(k) = \begin{cases} k & \text{if } k|n \\ 0 & \text{otherwise} \end{cases}$$

```
fun ifdivisor (k) =  
  if n mod k = 0  
  then k  
  else 0
```

Not **technically complete!**
However ...



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 166 of 709

Go Back

Full Screen

Close

Quit

SML: Assembly 1

```
fun sum_divisors (l, u) =  
  if l > u then 0  
  else  
    let fun ifdivisor (k) =  
          if n mod k = 0  
          then k  
          else 0  
        in ifdivisor (l) +  
          sum_divisors (l+1, u)  
        end
```

Clearly $k \in [l, u]$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 167 of 709

Go Back

Full Screen

Close

Quit

SML: Assembly 2

```
exception nonpositive;  
fun perfect (n) =  
  if n <= 0  
  then raise nonpositive  
  else  
    let fun sum_divisors (l, u) =  
          ...  
        in n = sum_divisors (1, n-1)  
        end
```

Clearly $k \in [l, u] = [1, n - 1]$ whenever $n > 0$.
Technically complete!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 168 of 709

Go Back

Full Screen

Close

Quit



Perfect Numbers ... *contd.*

Clearly for all k , $n/2 < k < n$,
 $ifdivisor(k) = 0$.

$$\lfloor n/2 \rfloor = n \operatorname{div} 2 < n/2$$

Hence

$$\sum_{k=1}^{n-1} ifdivisor(k) = \sum_{k=1}^{n \operatorname{div} 2} ifdivisor(k)$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 169 of 709

Go Back

Full Screen

Close

Quit

Perfect Numbers ... *contd.*

Hence

perfect(n)

$$\iff n = \sum_{k=1}^{n-1} \text{ifdivisor}(k)$$

$$\iff n = \sum_{k=1}^n \text{div}^2 \text{ifdivisor}(k)$$

where

$$\text{ifdivisor}(k) = \begin{cases} k & \text{if } k|n \\ 0 & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 170 of 709

Go Back

Full Screen

Close

Quit

SML: Assembly 3

```
exception nonpositive;  
fun perfect (n) =  
  
  if n <= 0  
  then raise nonpositive  
  else  
    let fun sum_divisors (l, u) =  
          ...  
        in n = sum_divisors (1, n div 2)  
        end
```

Clearly $k \in [l, u] = [1, n \text{ div } 2]$ whenever $n > 0$.

Technically complete!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 171 of 709

Go Back

Full Screen

Close

Quit



Perfect Numbers: Run

```
exception nonpositive
val perfect = fn : int -> bool
val it = () : unit
- perfect ~8;
uncaught exception nonpositive
  raised at: perfect.sml:4.16-4.27
- perfect 5;
val it = false : bool
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 172 of 709

Go Back

Full Screen

Close

Quit



Perfect Numbers: Run

```
- perfect 6;  
val it = true : bool  
- perfect 23;  
val it = false : bool  
- perfect 28;  
GC #0.0.0.1.3.88:    (1 ms)  
val it = true : bool  
- perfect 30;  
val it = false : bool
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 173 of 709

Go Back

Full Screen

Close

Quit

SML: Code variations

```
exception nonpositive;  
fun perfect (n) =  
  if n <= 0  
  then raise nonpositive  
  else  
    let  
      fun ifdivisor (k) = ...;  
      fun sum_divisors (l, u)  
    in  
      n=sum_divisors (1, n div 2)  
    end
```

Technically complete though
ifdivisor, by itself is not!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 174 of 709

Go Back

Full Screen

Close

Quit

SML: Code variations

What about this?

```
exception nonpositive;  
fun perfect (n) =  
  let  
    fun ifdivisor (k) = ...;  
    fun sum_divisors (l, u) = ...  
  in if n <= 0  
     then raise nonpositive  
     else  
n=sum_divisors (1, n div 2)  
  end
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 175 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 176 of 709

Go Back

Full Screen

Close

Quit

SML: Code variations

What about this?

```
exception nonpositive;  
fun ifdivisor (k) = ...;  
fun sum_divisors (l, u) = ...;  
fun perfect (n) =  
  if n <= 0  
  then raise nonpositive  
  else  
    n=sum_divisors (1, n div 2)
```

Technically **incomplete!**

Summation: Generalizations

Need a method to compute summations in **general**.

For any function $f : \mathbb{Z} \rightarrow \mathbb{Z}$ and integers l and u ,

$$\sum_{i=l}^u f(i) = \begin{cases} 0 & \text{if } l > u \\ f(l) + \sum_{i=l+1}^u f(i) & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 177 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 178 of 709

Go Back

Full Screen

Close

Quit

Algorithmic Improvements:

1. *perfect2*
2. *shrink2*



Algorithmic Variations

1. For any $k|n$, $m = n \operatorname{div} k$ is also a divisor of n

2. 1 is a divisor of every positive number

3. For $n > 2$, $\lfloor \sqrt{n} \rfloor < n \operatorname{div} 2$

4. Hence $\sum_{k=1}^n \operatorname{div} 2 \text{ if divisor}(k) =$

$$1 + \sum_{k=2}^{\lfloor \sqrt{n} \rfloor} \operatorname{if divisor} 2(k)$$



Algorithmic Variations

perfect(n)

$$\iff n = 1 + \sum_{k=2}^{\lfloor \sqrt{n} \rfloor} \text{ifdivisor2}(k)$$

where

$$\text{ifdivisor2}(k) = \begin{cases} k + (n \operatorname{div} k) & \text{if } k|n \\ 0 & \text{otherwise} \end{cases}$$

Are there any glitches? Is it technically correct and complete?

2.3. Variations: Algorithms & Code



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 181 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 182 of 709

Go Back

Full Screen

Close

Quit

Recap

- Combinations
- Perfect Numbers
- Code Variations
- Algorithmic Variations

forward



Recap: Combinations

$$\begin{aligned} {}^n C_k &= \frac{n!}{(n-k)!k!} \\ &= \frac{n(n-1)\cdots(n-k+1)}{k!} \\ &= \frac{n(n-1)\cdots(k+1)}{(n-k)!} \\ &= {}^{n-1} C_{k-1} + {}^{n-1} C_k \end{aligned}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 183 of 709

Go Back

Full Screen

Close

Quit

Combinations 1

```
use "fact.sml";  
exception invalid_arg;  
fun comb_wf (n, k) =  
    if n < 0 orelse  
       k < 0 orelse  
       k > n  
    then raise invalid_arg  
    else fact (n) div  
         (fact (n-k) * fact (k));
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 184 of 709

Go Back

Full Screen

Close

Quit

Combinations 2

```
exception invalid_arg;  
fun comb (n, k) =  
  if n < 0 orelse  
    k < 0 orelse  
    k > n  
  then raise invalid_arg  
  else if n = 0 orelse  
        k = 0 orelse  
        n = k  
  then 1  
  else (* 0 < k < n *)  
    prod (n, n-k+1) div  
    fact (k)
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 185 of 709

Go Back

Full Screen

Close

Quit

Combinations 3

```
exception invalid_arg;  
fun comb (n, k) =  
  if n < 0 orelse  
    k < 0 orelse  
    k > n  
  then raise invalid_arg  
  else if n = 0 orelse  
        k = 0 orelse  
        n = k  
  then 1  
  else (* 0 < k < n *)  
    prod (n, k+1) div  
    fact (n-k)
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 186 of 709

Go Back

Full Screen

Close

Quit

Perfect 2

perfect(n)

$$\iff n = 1 + \sum_{k=2}^{\lfloor \sqrt{n} \rfloor} \text{ifdivisor2}(k)$$

where

ifdivisor2(k) =

$$\begin{cases} k + m & \text{if } k|n \text{ and } k \neq m \\ k & \text{if } k|n \text{ and } k = m \\ 0 & \text{otherwise} \end{cases}$$

where $m = (n \operatorname{div} k)$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 187 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 188 of 709

Go Back

Full Screen

Close

Quit

Power 2

power

The previous inductive definition used

$$x^n = \underbrace{(x \times x \times \cdots \times x)}_{n-1 \text{ times}} \times x$$

We could associate the product differently

A Faster Power

$$x^n = \underbrace{(x \times x \times \cdots \times x)}_{n/2 \text{ times}} \times \underbrace{(x \times x \times \cdots \times x)}_{n/2 \text{ times}}$$

when n is **even** and

$$x^n = \underbrace{(x \times x \times \cdots \times x)}_{\lfloor n/2 \rfloor \text{ times}} \times \underbrace{(x \times x \times \cdots \times x)}_{\lfloor n/2 \rfloor \text{ times}} \times x$$

when n is **odd**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 189 of 709

Go Back

Full Screen

Close

Quit



Power2: Complete

$power2(x, n) =$

$$\begin{cases} 1.0/power2(x, n) & \text{if } n < 0 \\ 1.0 & \text{if } n = 0 \\ (power2(x, \lfloor n/2 \rfloor))^2 & \text{if } even(n) \\ (power2(x, \lfloor n/2 \rfloor))^2 \times x & \text{otherwise} \end{cases}$$

where $even(n) \iff n \bmod 2 = 0$.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 191 of 709

Go Back

Full Screen

Close

Quit

Power2: SML

```
fun power2 (x, n) =  
  if n < 0  
  then 1.0/power2 (x, ~n)  
  else if n = 0  
  then 1.0  
  else
```

Power2: SML

```
let fun even m =
      (m mod 2 = 0) ;
    fun square y = y * y ;
    val pwr_n_by_2 =
        power2 (x, n div 2) ;
    val sq_pwr_n_by_2 =
        square (pwr_n_by_2)
in if even (n)
   then sq_pwr_n_by_2
   else x * sq_pwr_n_by_2
end
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 192 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 193 of 709

Go Back

Full Screen

Close

Quit

Computation: Issues

1. Correctness

- (a) General correctness
- (b) Technical Completeness
- (c) Termination



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 194 of 709

Go Back

Full Screen

Close

Quit

General Correctness

1. **Mathematical correctness** should be established for all **algorithmic variations**.
2. Program Correctness: Mathematically developed code should not be moved around arbitrarily.
 - **Code variations** should also be **mathematically** proven



Code: Justification

- How does one justify the **correctness** of
 - **this version** and
 - **this version**?
- Can one **correct this version**?
- But first of all, **what is incorrect** about **this version**?

incorrectness

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 195 of 709

Go Back

Full Screen

Close

Quit

Recall

- A **program** is an
 - **explicit**,
 - **unambiguous** and
 - **technically complete**translation of an algorithm written in **mathematical notation**.
- Moreover, **mathematical notation** is more **concise** than a program.
- Hence **mathematical notation** is easier to analyse and diagnose.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 196 of 709

Go Back

Full Screen

Close

Quit

Features: Definition before Use

incorrect version

Definition of a name before use:

- $ifdivisor(k)$ is **defined** first.
- $idivisor(k)$ **uses** the name n without defining it.
- k has been **defined** (as an argument of $ifdivisor(k)$) before being **used**.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 197 of 709

Go Back

Full Screen

Close

Quit



Run ifdivisor

Standard ML of New Jersey,

```
- fun ifdivisor(k) =  
= if n mod k = 0  
= then k  
= else 0  
;
```

```
stdIn:18.8 Error:  
unbound variable  
or constructor: n
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 198 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 199 of 709

Go Back

Full Screen

Close

Quit

Diagnosis: Features of programs

incorrect version

- So both $sum_divisors(l, u)$ and $perfect(n)$ may use $ifdivisor(k)$.
- $sum_divisors(l, u)$ is **defined** before $perfect(n)$.
- So $perfect(n)$ may **use** both $ifdivisor(k)$ and $sum_divisors(l, u)$

Back to Math

incorrect version

Let

$$ifdivisor(k) = \begin{cases} k & \text{if } k|n \\ 0 & \text{otherwise} \end{cases}$$

and $sum_divisors(l, u) =$

$$\begin{cases} 0 & \text{if } l > u \\ ifdivisor(l) + \\ sum_divisors(l + 1, u) & \text{otherwise} \end{cases}$$

and $perfect(n) \iff$

$$n = sum_divisors(1, \lfloor n/2 \rfloor)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 200 of 709

Go Back

Full Screen

Close

Quit



Incorrectness

incorrect version

- *if divisor*(k) has a single argument k
- But it actually depends upon n too!
- But that is not made **explicit** in its definition.

Let's make it **explicit**!

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 201 of 709

Go Back

Full Screen

Close

Quit

ifdivisor3

Let

$$\text{ifdivisor3}(n, k) = \begin{cases} k & \text{if } k|n \\ 0 & \text{otherwise} \end{cases}$$

and $\text{sum_divisors}(l, u) =$

$$\begin{cases} 0 & \text{if } l > u \\ \text{ifdivisor3}(n, l) + & \text{otherwise} \\ \text{sum_divisors}(l + 1, u) & \end{cases}$$

and $\text{perfect}(n) \iff$

$$n = \text{sum_divisors}(1, \lfloor n/2 \rfloor)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 202 of 709

Go Back

Full Screen

Close

Quit



Run it!

Standard ML of New Jersey

```
- fun ifdivisor3 (n, k)
= = if (n mod k = 0)
= then k
= else 0;
val ifdivisor3 =
fn : int * int -> int
-
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 203 of 709

Go Back

Full Screen

Close

Quit

Try it!

```
- fun sum_divisors (l, u) =  
= if l > u  
= then 0  
= else ifdivisor3 (n, l)+  
= sum_divisors (l+1, u);  
stdIn:40.18 Error: unbound  
variable or constructor: n  
-
```

Now *sum_divisors* also depends on *n*!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 204 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 205 of 709

Go Back

Full Screen

Close

Quit

Hey! Wait a minute!

But n was defined in `ifdivisor3` (n, k)!

So then where is the problem?

Let's ignore it for the moment and come back to it later

The n 's

Let

$$ifdivisor3(\underline{n}, k) = \begin{cases} k & \text{if } k|n \\ 0 & \text{otherwise} \end{cases}$$

and $sum_divisors2(\underline{n}, l, u) =$

$$\begin{cases} 0 & \text{if } l > u \\ ifdivisor3(n, l) + \\ sum_divisors(l + 1, u) & \text{otherwise} \end{cases}$$

and $perfect(\underline{n}) \iff$

$$n = sum_divisors2(n, 1, \lfloor n/2 \rfloor)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 206 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 207 of 709

Go Back

Full Screen

Close

Quit

Scope

- The scope of a name begins from its definition and ends where the corresponding scope ends
- Scopes end with definitions of functions
- Scopes end with the keyword `end` in any `let ... in ...end`



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 208 of 709

Go Back

Full Screen

Close

Quit

Scope Rules

- Scopes may be disjoint
- Scopes may be nested one completely within another
- A scope cannot span two disjoint scopes
- Two scopes cannot (partly) overlap

forward



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 209 of 709

Go Back

Full Screen

Close

Quit

2.4. Names, Scopes & Recursion

1. Disjoint Scopes
2. Nested Scopes
3. Overlapping Scopes
4. Spanning
5. Scope & Names
6. Names & References
7. Names & References
8. Names & References
9. Names & References
10. Names & References
11. Names & References
12. Names & References
13. Names & References
14. Names & References
15. Names & References
16. Definition of Names
17. Use of Names

18. `local...in...end`
19. `local...in...end`
20. `local...in...end`
21. `local...in...end`
22. **Scope & local**
23. **Computations: Simple**
24. **Simple computations**
25. **Computations: Composition**
26. **Composition: Alternative**
27. **Compositions: Compare**
28. **Compositions: Compare**
29. **Computations: Composition**
30. **Recursion**
31. **Recursion: Left**
32. **Recursion: Right**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 210 of 709

Go Back

Full Screen

Close

Quit

Disjoint Scopes



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 211 of 709

Go Back

Full Screen

Close

Quit

```
let
  val x = 10;
  fun fun1
    y =
      let
        ...
      in
        ...
      end
  fun fun2
    z =
      let
        ...
      in
        ...
      end
in
  fun1 (fun2 x)
end
```

Nested Scopes



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 212 of 709

Go Back

Full Screen

Close

Quit

```
let
  val x = 10;
  fun fun1
    y =
      let
        val x = 15
      in
        x + y
      end
in
  fun1 x
end
```

Overlapping Scopes



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 213 of 709

Go Back

Full Screen

Close

Quit

```
let
  val x = 10;
  fun fun1
    y =
      ...
      ...
      ...
in
  fun1 (fun2 x)
end
```

A diagram illustrating overlapping scopes. A large light red rectangle represents the outer scope, containing the code from 'let' to 'end'. Inside it, a cyan rectangle represents the scope of 'fun1', containing 'y =', '...', and '...'. Inside the cyan rectangle, a yellow rectangle represents the scope of 'fun2', containing '...', '...', and '...'. A large red 'X' is drawn over the overlapping area between the cyan and yellow rectangles, indicating that this configuration is invalid due to overlapping scopes.

Spannning



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 214 of 709

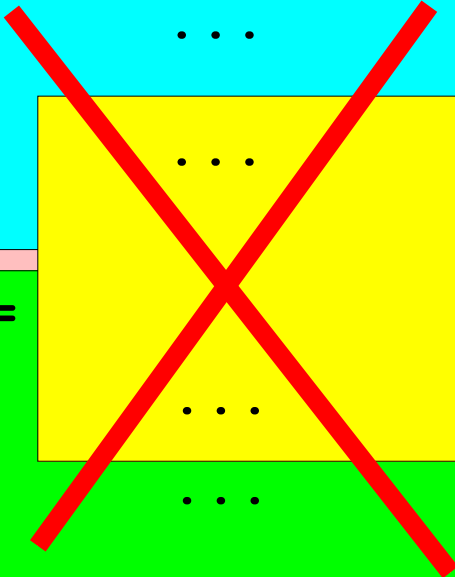
Go Back

Full Screen

Close

Quit

```
let
  val x = 10;
  fun fun1
    y =
      ...
  fun fun2
    z =
      ...
  in
    fun1 (fun2 x)
  end
```





Scope & Names

- A **name** may occur either as being **defined** or as a **use** of a previously defined name
- The same name may be used to refer to different objects.
- The **use** of a name refers to the **textually most recent definition in the innermost enclosing scope**

diagram

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 215 of 709

Go Back

Full Screen

Close

Quit

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 216 of 709

Go Back

Full Screen

Close

Quit

```
let
  val x = 10; val z = 5;
  fun fun1
    y =
      let
        val x = 15
      in
        x + y * z
      end
in
  fun1 x
end
```

Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 217 of 709

Go Back

Full Screen

Close

Quit

```
let
  val x = 10; val z = 5;
  fun fun1 y =
    let
      val x = 15
    in
      x + y * z
    end
in
  fun1 x
end
```

Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 218 of 709

Go Back

Full Screen

Close

Quit

```
let
```

```
  val x = 10; val z = 5;
```

```
  fun fun1 y =
```

```
    let
```

```
      val x = 15
```

```
    in
```

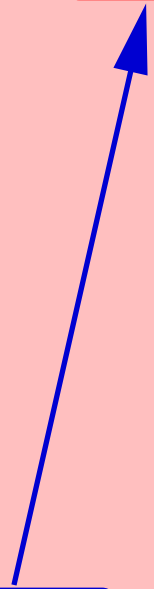
```
      x + y * z
```

```
    end
```

```
in
```

```
  fun1 x
```

```
end
```



Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



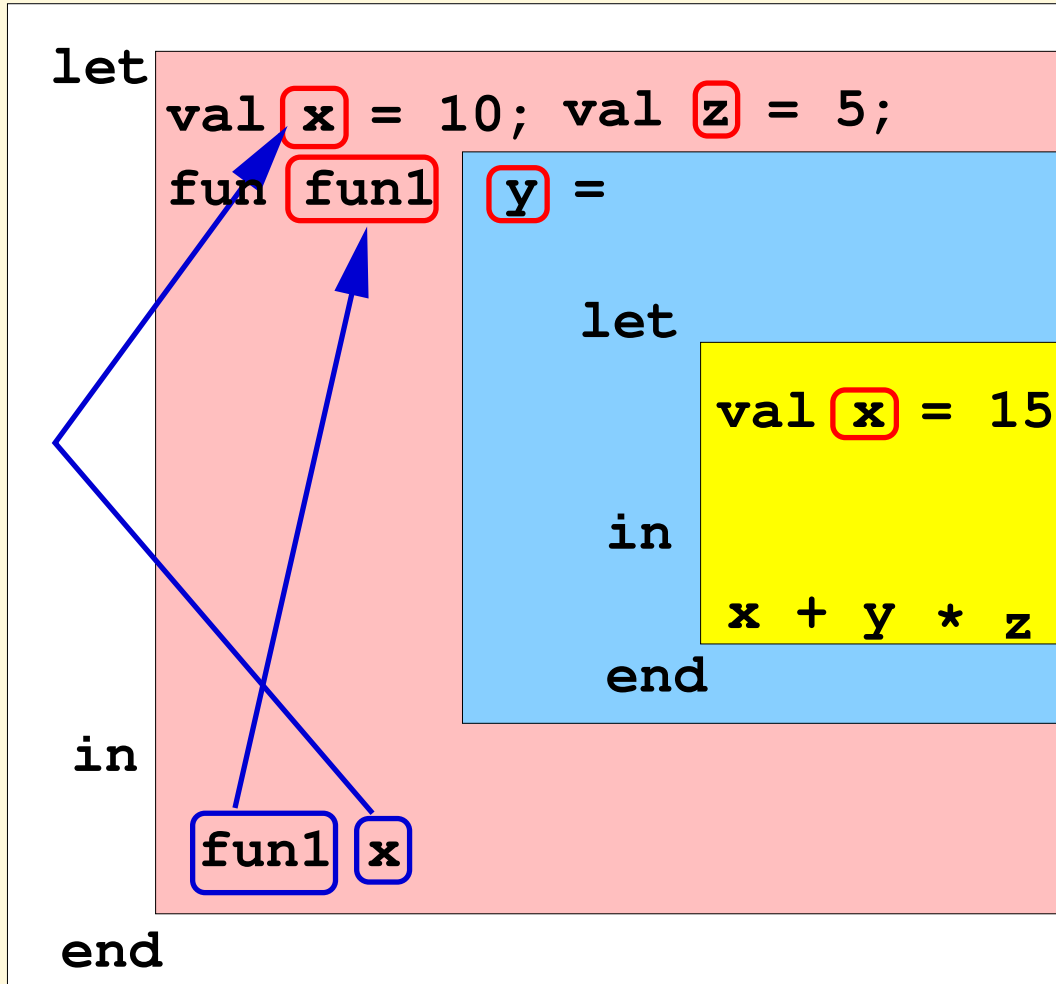
Page 219 of 709

Go Back

Full Screen

Close

Quit



Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



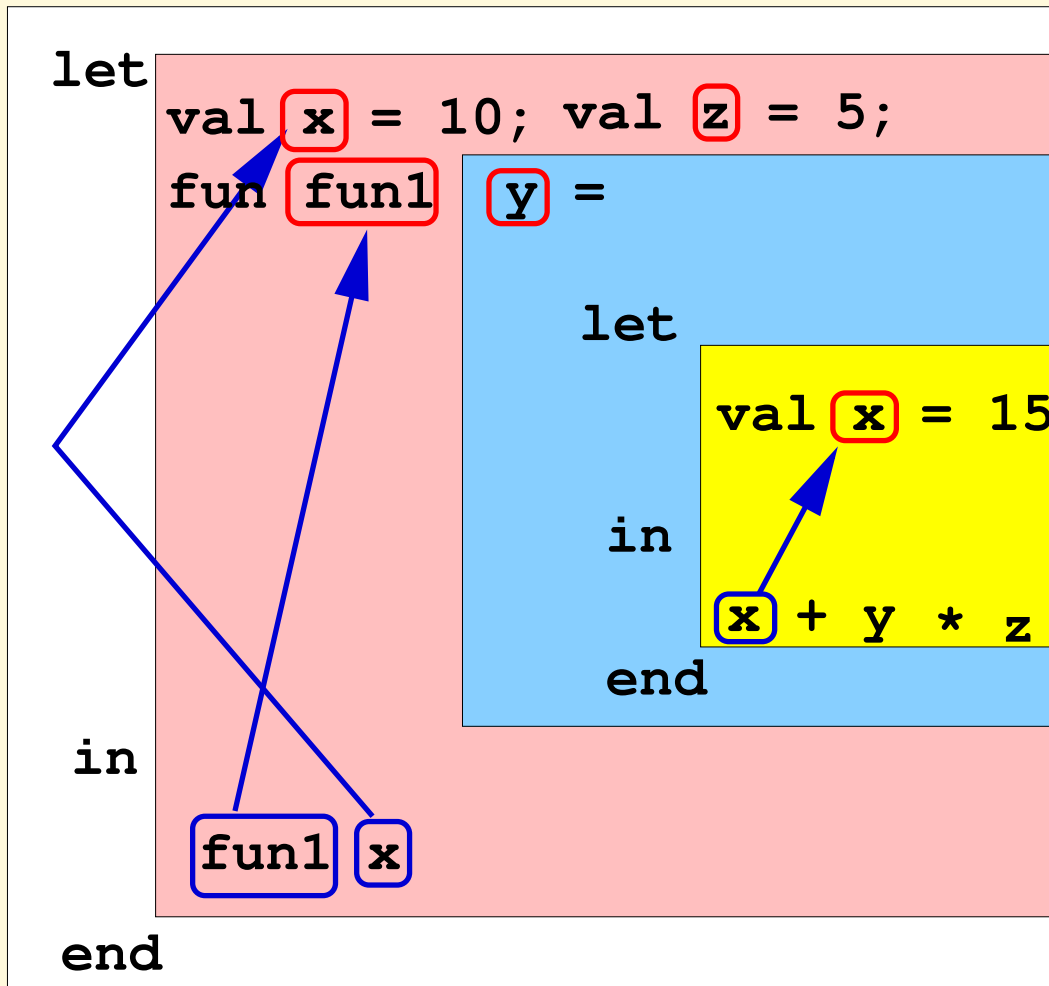
Page 220 of 709

Go Back

Full Screen

Close

Quit



Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



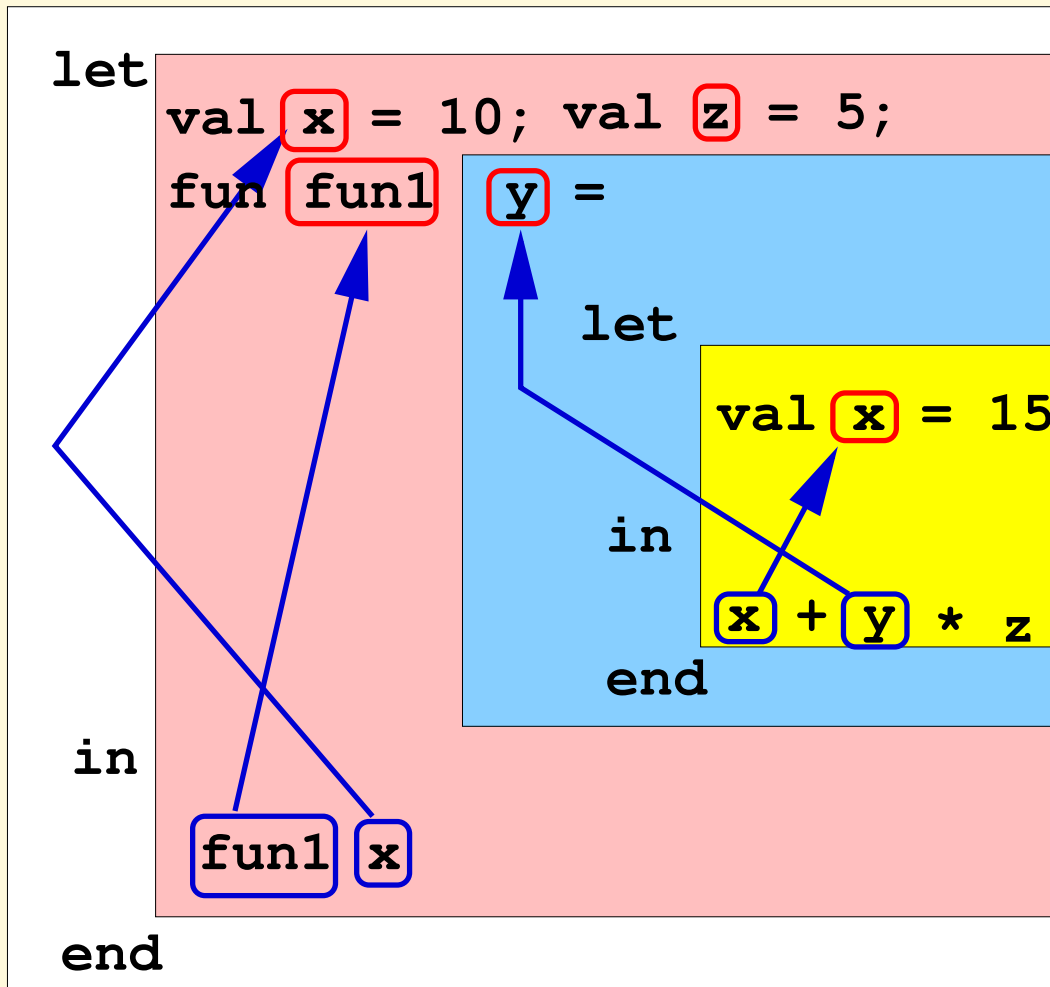
Page 221 of 709

Go Back

Full Screen

Close

Quit



Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



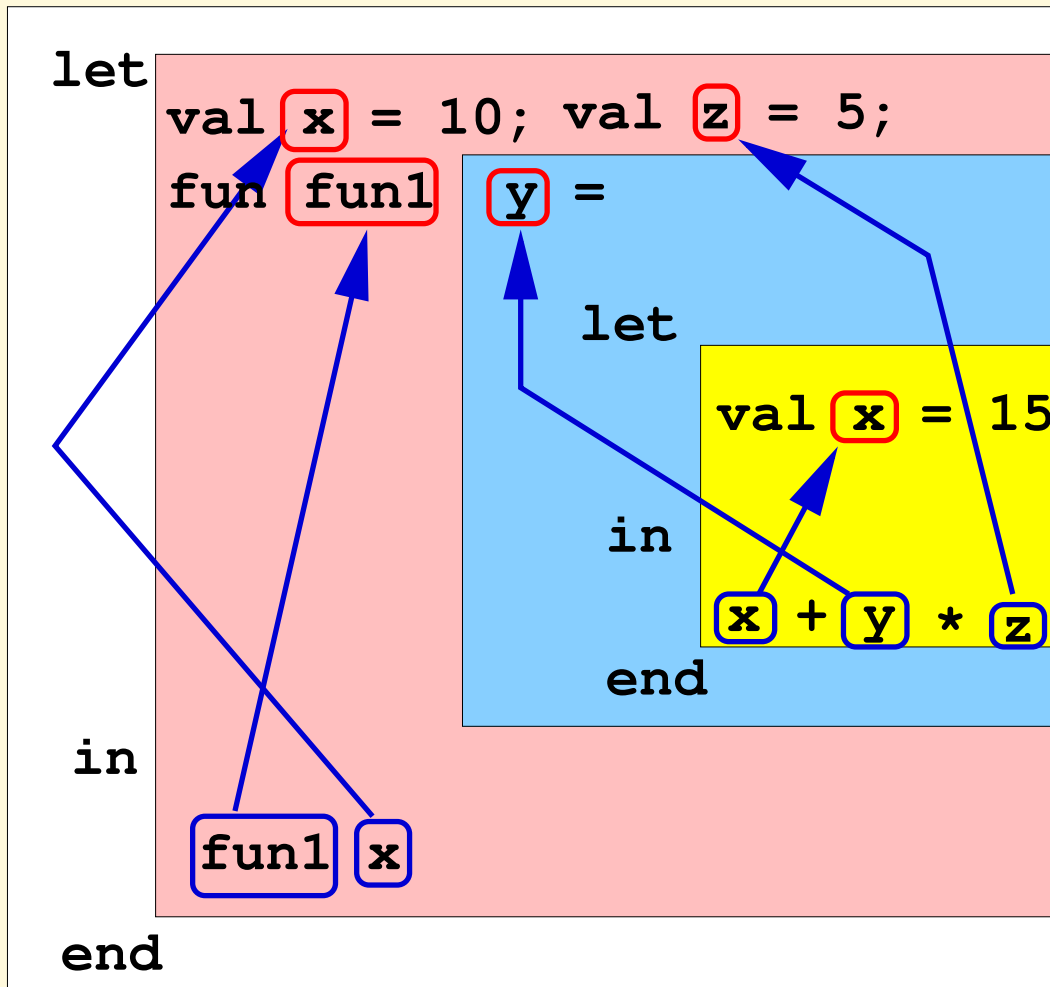
Page 222 of 709

Go Back

Full Screen

Close

Quit



Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 223 of 709

Go Back

Full Screen

Close

Quit

```
let
  val x = 10; val x = x - 5;
  fun fun1
    y =
      let
        ...
      in
        ...
      end
    fun fun2
      z =
        let
          ...
        in
          ...
        end
    in fun1 (fun2 x)
end
```

Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 224 of 709

Go Back

Full Screen

Close

Quit

```
let
```

```
  val x = 10; val x = x - 5;
```

```
  fun fun1
```

```
    y =
```

```
      let
```

```
        ...
```

```
      in
```

```
        ...
```

```
      end
```

```
  fun fun2
```

```
    z =
```

```
      let
```

```
        ...
```

```
      in
```

```
        ...
```

```
      end
```

```
  in fun1 (fun2 x)
```

```
end
```

Back to scope names

Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 225 of 709

Go Back

Full Screen

Close

Quit

```
let
```

```
  val x = 10; val x = x - 5;
```

```
  fun fun1
```

```
    y =
```

```
      let
```

```
        ...
```

```
      in
```

```
        ...
```

```
      end
```

```
  fun fun2
```

```
    z =
```

```
      let
```

```
        ...
```

```
      in
```

```
        ...
```

```
      end
```

```
  in fun1 (fun2 x)
```

```
end
```

Back to scope names



Definition of Names

Definitions are of the form

qualifier *name* ... = *body*

- `val` *name* =
- `fun` *name* (*argnames*) =
- `local` *definitions*
in definition
`end`



Use of Names

Names are used in expressions.
Expressions may occur

- by themselves – to be evaluated
- as the *body* of a definition
- as the *body* of a **let**-expression

```
let definitions  
in expression  
end
```

use of local

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 227 of 709

Go Back

Full Screen

Close

Quit

local...in...end

perfect

```
local
```

```
    exception invalidArg;
```

```
fun ifdivisor3 (n, k) =
```

```
    if n <= 0 orelse
```

```
        k <= 0 orelse
```

```
            n < k
```

```
    then raise invalidArg
```

```
    else if n mod k = 0
```

```
        then k
```

```
        else 0;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 228 of 709

Go Back

Full Screen

Close

Quit

local...in...end

perfect

```
fun sum_div2 (n, l, u) =  
  if n <= 0 orelse  
    l <= 0 orelse  
    l > n orelse  
    u <= 0 orelse  
    u > n  
  then raise invalidArg  
  else if l > u  
  then 0  
  else ifdivisor3 (n, l)  
    + sum_div2 (n, l+1, u)
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 229 of 709

Go Back

Full Screen

Close

Quit

```
local...in...end
```

perfect

```
in
```

```
  fun perfect n =
```

```
    if n <= 0
```

```
    then raise invalidArg
```

```
    else
```

```
      let
```

```
        val nby2 = n div 2
```

```
      in
```

```
        n = sum_div2 (n, 1, nby2)
```

```
      end
```

```
end
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 230 of 709

Go Back

Full Screen

Close

Quit

local...in...end

perfect

```
Standard ML of New Jersey,  
- use "perfect2.sml";  
[opening perfect2.sml]  
GC #0.0.0.0.1.10:      (1 ms)  
val perfect = fn : int -> bool  
val it = () : unit  
- perfect 28;  
val it = true : bool  
- perfect 6;  
val it = true : bool  
- perfect 8128;  
val it = true : bool
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 231 of 709

Go Back

Full Screen

Close

Quit

Scope & local

local

fun fun1

y =
...

fun fun2

z = ...
fun1

in

fun fun3

x =
fun2 ...
fun1 ...

end



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 232 of 709

Go Back

Full Screen

Close

Quit

Computations: Simple

For most simple expressions it is

- left to right, and
- top to bottom

except when

- presence of brackets
- precedence of operators

determine otherwise.

Hence



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 233 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 234 of 709

Go Back

Full Screen

Close

Quit

Simple computations

$$\begin{aligned} & 4 + 6 - (4 + 6) \operatorname{div} 2 \\ = & 10 - (4 + 6) \operatorname{div} 2 \\ = & 10 - 10 \operatorname{div} 2 \\ = & 10 - 5 \\ = & 5 \end{aligned}$$

Computations: Composition

$$f(x) = x^2 + 1$$

$$g(x) = 3 * x + 2$$

Then for any value $a = 4$

$$\begin{aligned} & f(g(a)) \\ = & f(3 * 4 + 2) \\ = & f(14) \\ = & 14^2 + 1 \\ = & 196 + 1 \\ = & 197 \end{aligned}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 235 of 709

Go Back

Full Screen

Close

Quit

Composition: Alternative

$$f(x) = x^2 + 1$$

$$g(x) = 3 * x + 2$$

Why not

$$\begin{aligned} & f(g(a)) \\ = & g(4)^2 + 1 \\ = & (3 * 4 + 2)^2 + 1 \\ = & (12 + 2)^2 + 1 \\ = & 14^2 + 1 \\ = & 196 + 1 \\ = & 197 \end{aligned}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 236 of 709

Go Back

Full Screen

Close

Quit

Compositions: Compare

$$\begin{aligned} & f(g(a)) && f(g(a)) \\ = & f(3 * 4 + 2) && = g(4)^2 + 1 \\ = & f(14) && = (3 * 4 + 2)^2 + 1 \\ = & && = (12 + 2)^2 + 1 \\ = & 14^2 + 1 && = 14^2 + 1 \\ = & 196 + 1 && = 196 + 1 \\ = & 197 && = 197 \end{aligned}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 237 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 238 of 709

Go Back

Full Screen

Close

Quit

Compositions: Compare

Question 1: Which is more correct?
Why?

Question 2: Which is easier to implement?

Question 3: Which is more efficient?



Computations: Composition

A computation of $f(g(a))$ proceeds thus:

- $g(a)$ is evaluated to some value, say b
- $f(b)$ is next evaluated

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 239 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 240 of 709

Go Back

Full Screen

Close

Quit

Recursion

$$factL(n) = \begin{cases} 1 & \text{if } n = 0 \\ factL(n - 1) * n & \text{otherwise} \end{cases}$$

$$factR(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * factR(n - 1) & \text{otherwise} \end{cases}$$



Recursion: Left

$$\begin{aligned} & factL(4) \\ = & (factL(4 - 1) * 4) \\ = & (factL(3) * 4) \\ = & ((factL(3 - 1) * 3) * 4) \\ = & ((factL(2) * 3) * 4) \\ = & (((factL(2 - 1) * 2) * 3) * 4) \\ = & \dots \end{aligned}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 241 of 709

Go Back

Full Screen

Close

Quit



Recursion: Right

$$\begin{aligned} & factR(4) \\ = & (4 * factR(4 - 1)) \\ = & (4 * factR(3)) \\ = & (4 * (3 * factR(3 - 1))) \\ = & (4 * (3 * factR(2))) \\ = & (4 * (3 * (2 * factR(2 - 1)))) \\ = & \dots \end{aligned}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 242 of 709

Go Back

Full Screen

Close

Quit



3. Introducing Reals

3.1. Floating Point

1. So Far-1: Computing
2. So Far-2: Algorithms & Programs
3. So far-3: Top-down Design
4. So Far-4: Algorithms to Programs
5. So far-5: Caveats
6. So Far-6: Algorithmic Variations
7. So Far-7: Computations
8. Floating Point
9. Real Operations
10. Real Arithmetic
11. Numerical Methods
12. Errors
13. Errors
14. Infinite Series
15. Truncation Errors

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 243 of 709

Go Back

Full Screen

Close

Quit

16. Equation Solving

17. Root Finding-1

18. Root Finding-2

19. Root Finding-3

20. Root Finding-4



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 244 of 709

Go Back

Full Screen

Close

Quit



So Far-1: Computing

- The general nature of computation
- The notion of primitives, composition & induction
- The notion of an **algorithm**
- The digital computer & programming language

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 245 of 709

Go Back

Full Screen

Close

Quit



So Far-2: Algorithms & Programs

- **Algorithms:** Finite mathematical processes
- **Programs:** Precise, unambiguous explications of algorithms
- Standard ML: Its primitives
- Writing **technically complete** specifications

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 246 of 709

Go Back

Full Screen

Close

Quit

So far-3: Top-down Design

integer Square Root

- Begin with the function you need to design
- Write a
 - small compact technically complete definition of the function
 - perhaps using other functions that have not yet been defined
- Each function in turn is defined in a top-down manner

Perfect Numbers



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 247 of 709

Go Back

Full Screen

Close

Quit



So Far-4: Algorithms to Programs

- Perform top development till you require only the available primitives
- Directly translate the algorithm into a Program
- Use scope rules to localize or generalize

SML code for perfect

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 248 of 709

Go Back

Full Screen

Close

Quit

So far-5: Caveats

- Don't arbitrarily vary code from your algorithmic development
 - It might **work** or
 - It might **not work**
 - unless properly **justified**
- May destroy technical completeness
- May create **scope** violations.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 249 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 250 of 709

Go Back

Full Screen

Close

Quit

So Far-6: Algorithmic Variations

Algorithmic Variations

- **Are safe if developed from first principles.** Thus ensuring their
 - mathematical correctness
 - technical completeness
 - termination properties



So Far-7: Computations

- Work within the notion of mathematical equality
 - Simple **expressions**
 - **Composition of functions**
 - **Recursive** computations
- But are generally **irreversible**

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 251 of 709

Go Back

Full Screen

Close

Quit

Floating Point

- Each real number $3E11$ is represented by a **pair** of integers
 1. **Mantissa**: 3 or 30 or 300 or ...
 2. **Exponent**: the power of 10 which the mantissa has to be multiplied by
- What is displayed is not necessarily the same as the internal representation.
- There is no **unique** representation of a real number



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 252 of 709

Go Back

Full Screen

Close

Quit



Real Operations

Depending upon the operations involved

- Each real number is first converted into a **suitable representation**
- The operation is performed
- The result is converted into a suitable representation for display.

skip to Numerical methods

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 253 of 709

Go Back

Full Screen

Close

Quit

Real Arithmetic

- for **addition** and **subtraction** the two numbers should have the same exponent for ease of **integer operations** to be performed
- the conversion may involve loss of precision
- for **multiplication** and **division** the exponents may have to be adjusted so as not to cause an **integer overflow or underflow**.

back



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 254 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 255 of 709

Go Back

Full Screen

Close

Quit

Numerical Methods

- Finite (limited) precision
- Accuracy depends upon available precision
- Whereas **integer** arithmetic is exact, **real** arithmetic is not.
- Numerical solutions are a finite approximation of the result



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 256 of 709

Go Back

Full Screen

Close

Quit

Errors

- Hence an estimate of the **error** is necessary.
- If a is the “correct” value and a^* is the computed value,

$$\text{absolute error} = a^* - a$$

$$\text{relative error} = \frac{a^* - a}{a}$$



Errors

Errors in floating point computations are mainly due

finite precision Round-off errors

fnite process It is impossible to compute the value of a (convergent) infinite series because computations are themselves finite processes. **Infinite series**

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 257 of 709

Go Back

Full Screen

Close

Quit

Infinite Series

cannot be computed to ∞

$$e^x = \sum_{m=0}^{\infty} \frac{x^m}{m!}$$

$$\cos x = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{(2m)!}$$

$$\sin x = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m+1}}{(2m+1)!}$$

Truncation



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 258 of 709

Go Back

Full Screen

Close

Quit

Truncation Errors

and hopefully it is good enough to restrict it to appropriate values of n

$$e^x = \sum_{m=0}^n \frac{x^m}{m!}$$

$$\cos x = \sum_{m=0}^n \frac{(-1)^m x^{2m}}{(2m)!}$$

$$\sin x = \sum_{m=0}^n \frac{(-1)^m x^{2m+1}}{(2m+1)!}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 259 of 709

Go Back

Full Screen

Close

Quit

back to Errors



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 260 of 709

Go Back

Full Screen

Close

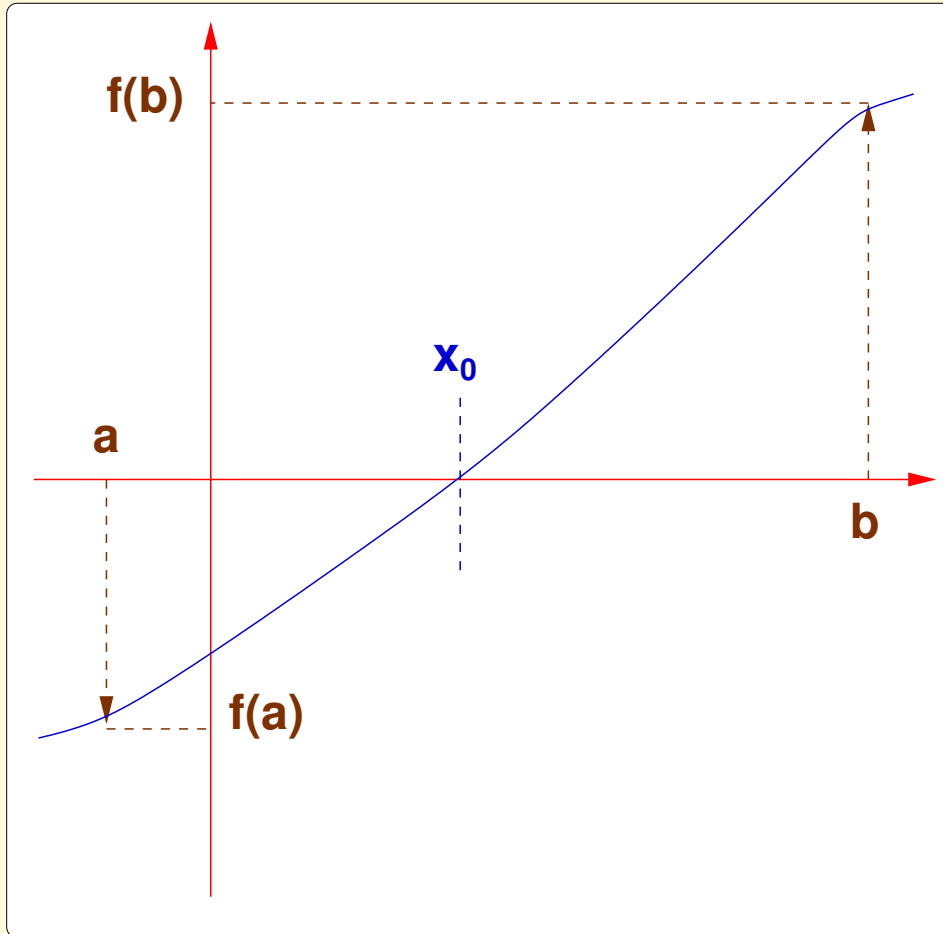
Quit

Equation Solving

- The fifth most basic **operation**
- Root finding: A particular form of equation solving

$$f(x) = 0$$

Root Finding-1



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 261 of 709

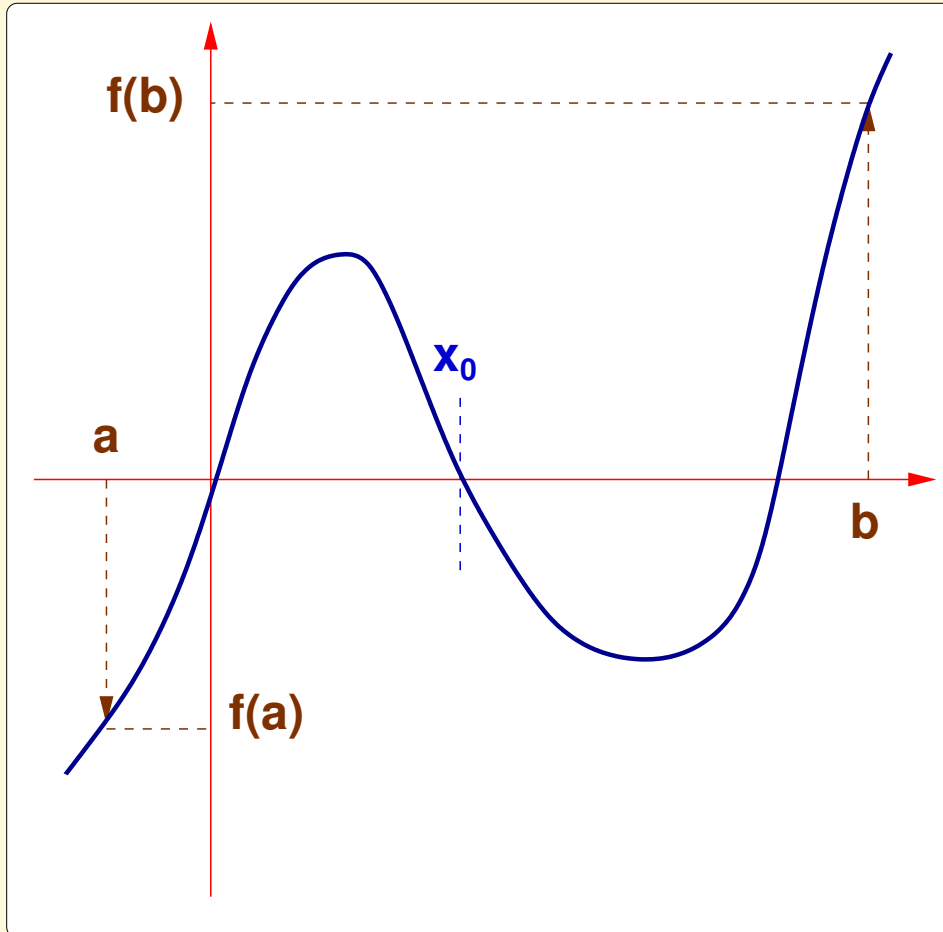
Go Back

Full Screen

Close

Quit

Root Finding-2



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 262 of 709

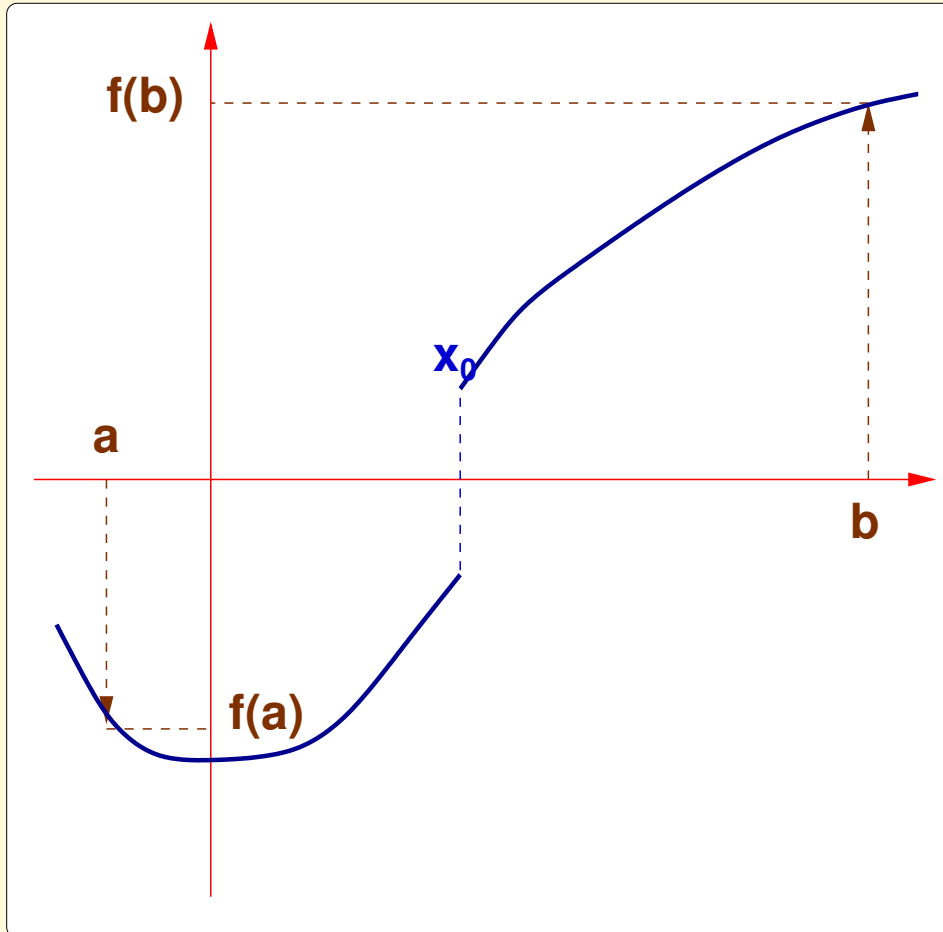
Go Back

Full Screen

Close

Quit

Root Finding-3



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 263 of 709

Go Back

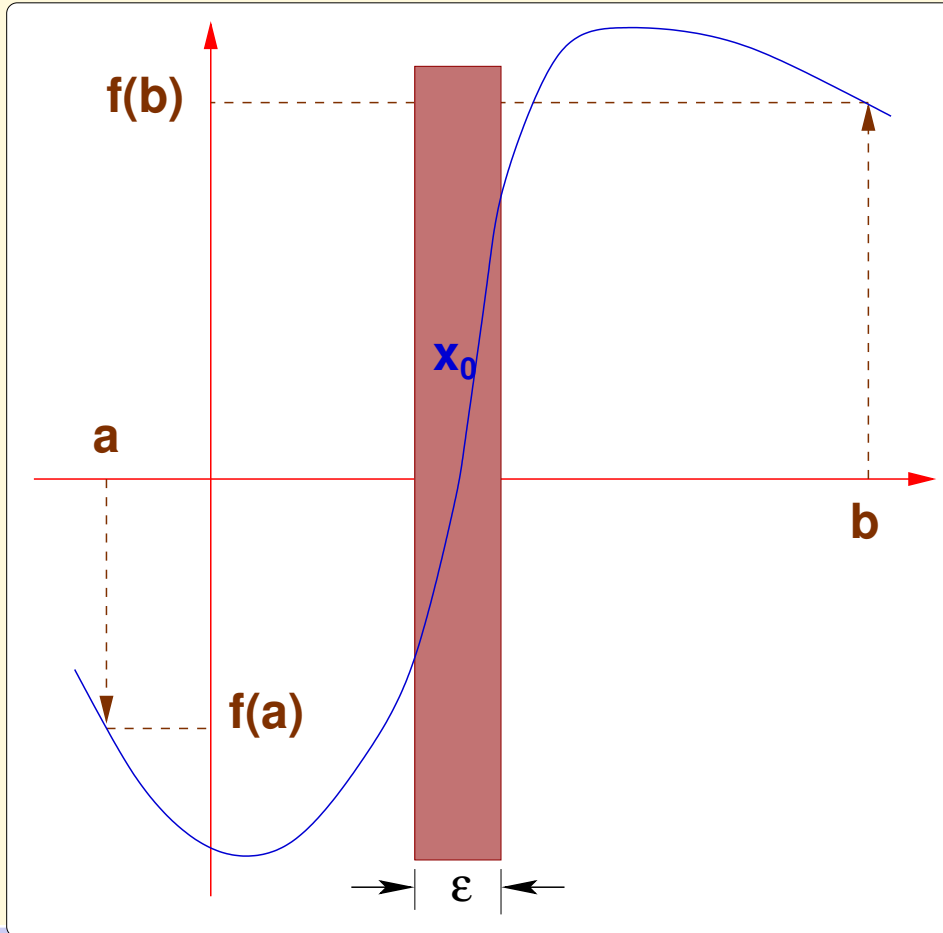
Full Screen

Close

Quit

Root Finding-4

Rather **steep** isn't it?



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 264 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 265 of 709

Go Back

Full Screen

Close

Quit

3.2. Root Finding, Composition and Recursion

1. Root Finding: Newton's Method
2. Root Finding: Newton's Method
3. Root Finding: Newton's Method
4. Root Finding: Newton's Method
5. Root Finding: Newton's Method
6. Root Finding: Newton's Method
7. Newton's Method: Basis
8. Newton's Method: Basis
9. Newton' Method: Algorithm
10. What can go wrong!-1
11. What can go wrong!-2
12. What can go wrong!-2
13. What can go wrong!-3
14. What can go wrong!-4
15. Real Computations & Induction: 1
16. Real Computations & Induction: 2
17. What's it good for? 1

18. What's it good for? 2
19. *newton*: Computation
20. Generalized Composition
21. Two Computations of $h(1)$
22. Two Computations of $h(-1)$
23. Recursive Computations
24. Recursion: Left
25. Recursion: Right
26. Recursion: Nonlinear
27. Some Practical Questions
28. Some Practical Questions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 266 of 709

Go Back

Full Screen

Close

Quit



Root Finding: Newton's Method

Consider a function $f(x)$

- **smooth** and **continuously differentiable** over $[a, b]$
- with a **non-zero** derivative $f'(x)$ everywhere in $[a, b]$
- the **signs** of $f(a)$ and $f(b)$ are different

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 267 of 709

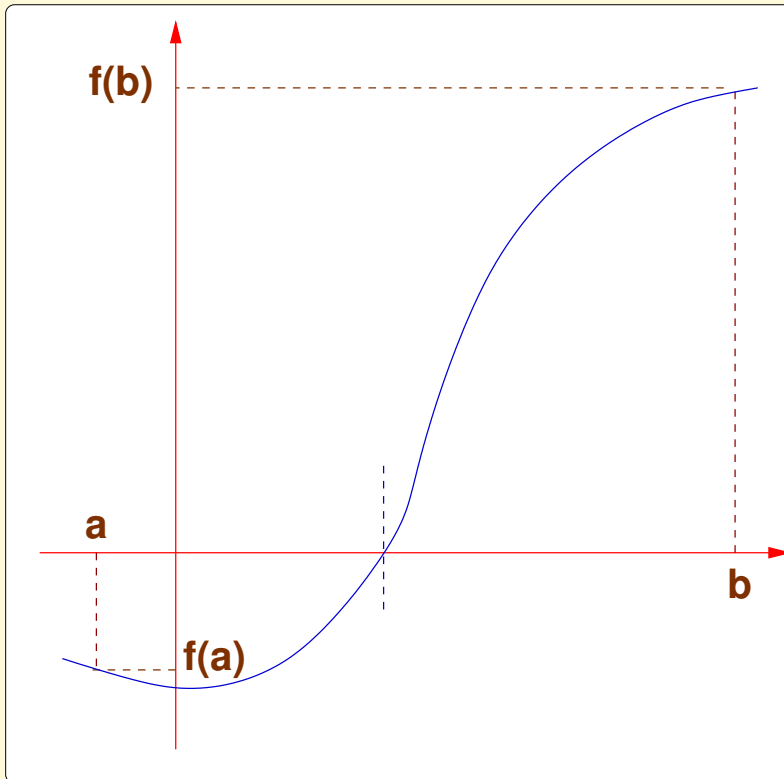
Go Back

Full Screen

Close

Quit

Root Finding: Newton's Method



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 268 of 709

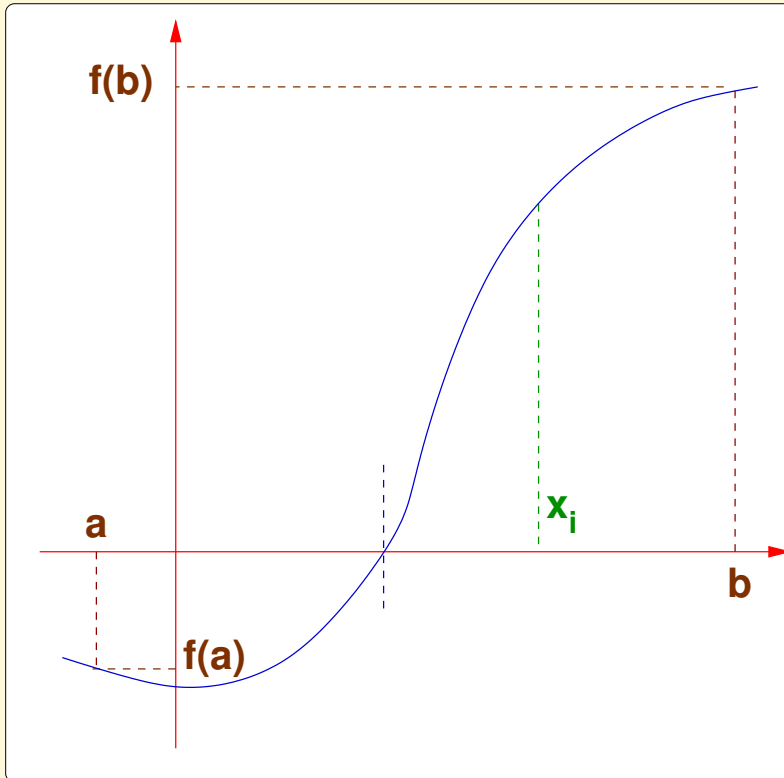
Go Back

Full Screen

Close

Quit

Root Finding: Newton's Method



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 269 of 709

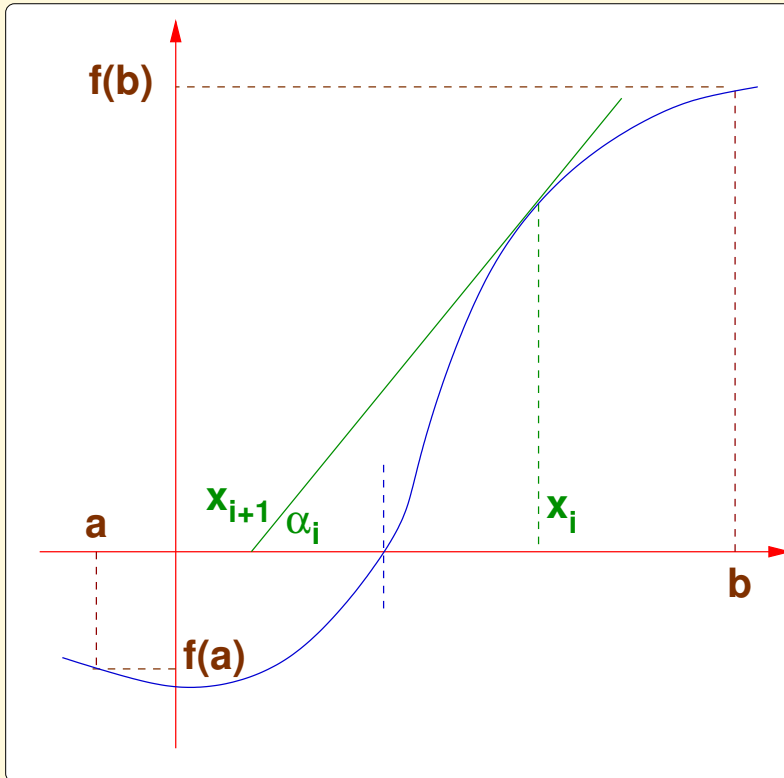
Go Back

Full Screen

Close

Quit

Root Finding: Newton's Method



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 270 of 709

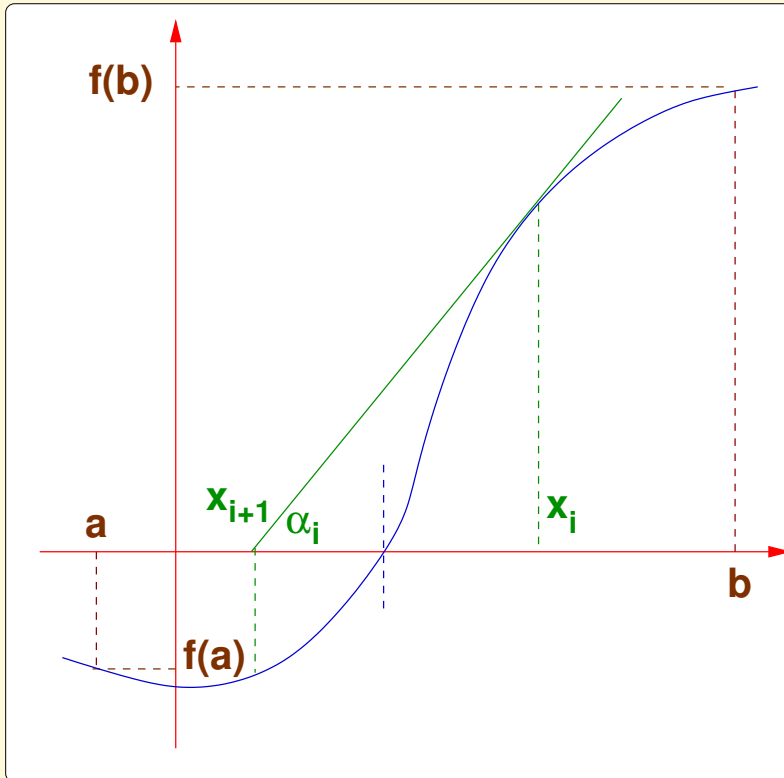
Go Back

Full Screen

Close

Quit

Root Finding: Newton's Method



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 271 of 709

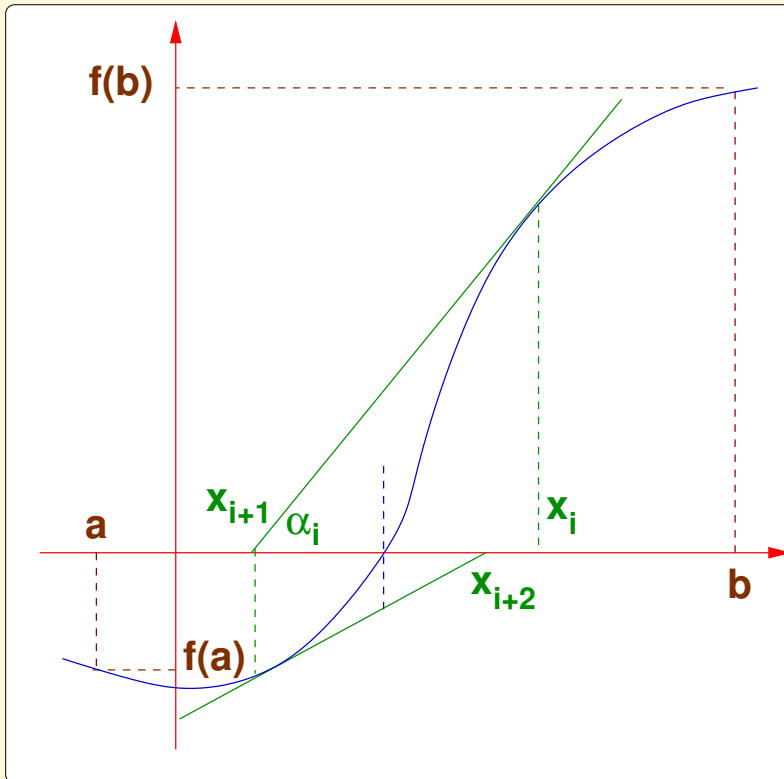
Go Back

Full Screen

Close

Quit

Root Finding: Newton's Method



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 272 of 709

Go Back

Full Screen

Close

Quit

Newton's Method: Basis

$$\tan \alpha_i = f'(x_i) = \frac{f(x_i)}{x_i - x_{i+1}}$$

whence

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Starting from an initial value $x_0 \in [a, b]$,
if the sequence $f(x_i)$ converges to 0
i.e

$$f(x_0), f(x_1), f(x_2), \dots \rightarrow 0$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 273 of 709

Go Back

Full Screen

Close

Quit

Newton's Method: Basis

$$\text{i.e. } \lim_{n \rightarrow \infty} |f(x_n)| = 0$$

$$\text{i.e. } \forall \varepsilon > 0 : \exists N \geq 0 : \forall n > N :$$

$$|f(x_n)| < \varepsilon$$

then the sequence

$$x_0, x_1, x_2, \dots$$

converges to a root of f .



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 274 of 709

Go Back

Full Screen

Close

Quit

Newton' Method: Algorithm

Select a small enough $\varepsilon > 0$ and x_0 .

Then

$$\text{newton}(f, f', a, b, \varepsilon, x_i) =$$

$$\begin{cases} x_i & \text{if } |f(x_i)| < \varepsilon \\ \text{newton}(f, f', a, b, \varepsilon, x_{i+1}) & \text{otherwise} \end{cases}$$

where

$$x_0 \in [a, b]$$

and

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \in [a, b]$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 275 of 709

Go Back

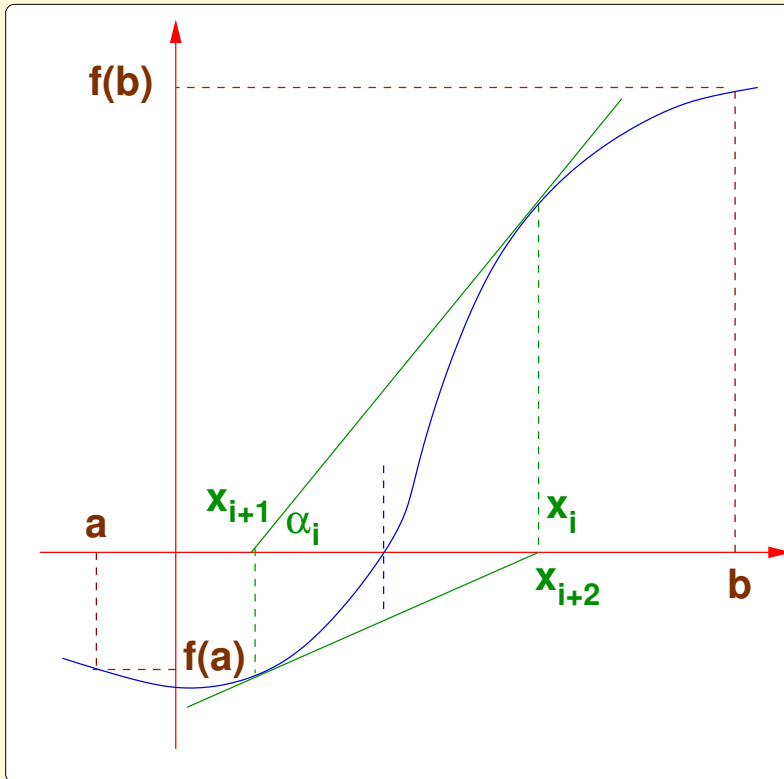
Full Screen

Close

Quit

What can go wrong!-1

Oscillations!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 276 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

⏪ ⏩

◀ ▶

Page 277 of 709

Go Back

Full Screen

Close

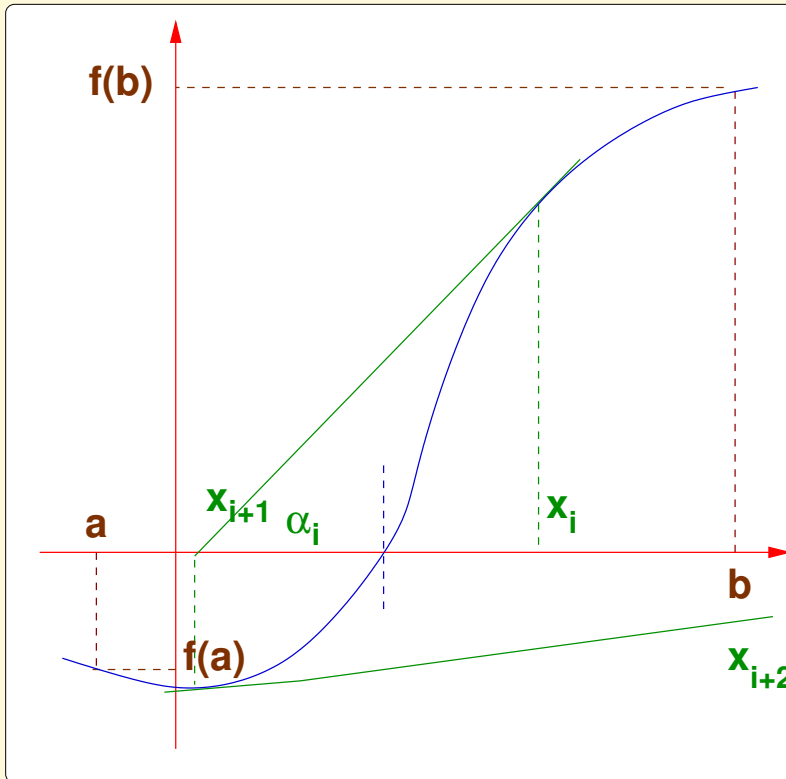
Quit

What can go wrong!-2

An intermediate point may lie **outside** $[a, b]$! The function may not satisfy all the assumptions outside $[a, b]$. There are then no guarantees about the behaviour of the function.

What can go wrong!-2

Interval bounds error!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 278 of 709

Go Back

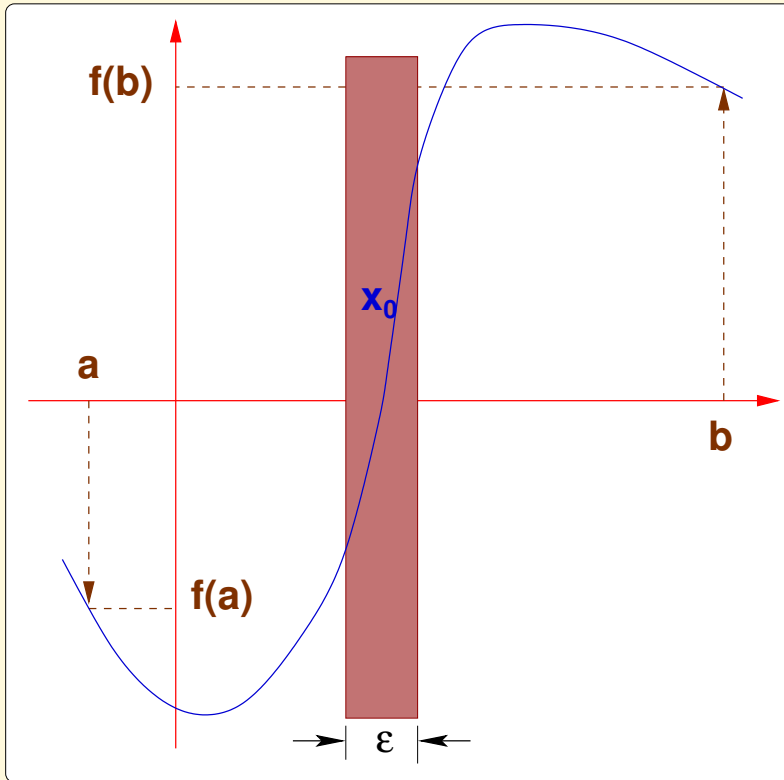
Full Screen

Close

Quit

What can go wrong!-3

The function may be too steep



for the available precision.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 279 of 709

Go Back

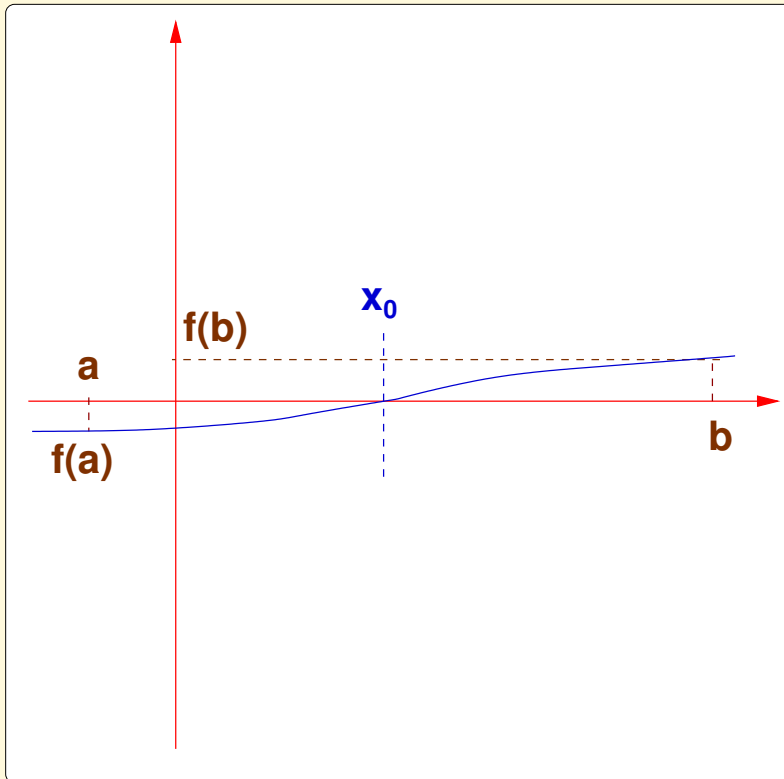
Full Screen

Close

Quit

What can go wrong!-4

Or too shallow!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 280 of 709

Go Back

Full Screen

Close

Quit

Real Computations & Induction: 1

Newton's method (**when it does work!**) computes a sequence

$$x_0, x_1, x_2, \dots, x_n$$

of essentially **discrete** values such that even if the sequence is not totally ordered, there is some discrete convergence **measure** viz.

$$|f(x_i) - 0|$$

which is **well-founded**.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 281 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 282 of 709

Go Back

Full Screen

Close

Quit

Real Computations & Induction: 2

That is, for some decreasing sequence of integers $k_i \geq 0$,

$$k_0 > k_1 > k_2 > \cdots > k_n = 0$$

we have

$$k_i \varepsilon \leq |f(x_i) - 0| < (k_i + 1) \varepsilon$$

and therefore inductive on **integer multiples of ε**

What's it good for? 1

Finding the positive n -th root $\sqrt[n]{c}$ of a $c > 0$ and $n > 1$ amounts to solving the equation

$$x^n = c$$

which is equivalent to finding the root of $f(x)$ with

$$\begin{aligned} f(x) &= x^n - c \\ f'(x) &= nx^{n-1} \end{aligned}$$

with $[a, b] = [0, c]$ or $[0, \sqrt[n]{c}]$ and an appropriately chosen ε .



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 283 of 709

Go Back

Full Screen

Close

Quit

What's it good for? 2

Finding roots of polynomials.

$$f(x) = \sum_{i=0}^n c_i x^i$$

$$f'(x) = \sum_{i=1}^n i c_i x^{i-1}$$

and

- an appropriately chosen ε ,
- an appropriately chosen $[a, b]$ with one of the limits possibly being c_0 .



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 284 of 709

Go Back

Full Screen

Close

Quit

Generalized Composition

Computations

$$h(x) = f(x, g(x))$$

where

$$f(x, y) = \begin{cases} 0 & \text{if } x < 0 \\ y & \text{otherwise} \end{cases}$$

$$g(x) = \begin{cases} 0 & \text{if } x = 0 \\ g(x - 1) + 1 & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 286 of 709

Go Back

Full Screen

Close

Quit



Two Computations of $h(1)$

$h(1)$		$h(1)$
$\rightsquigarrow f(1, g(1))$		$\rightsquigarrow f(1, g(1))$
$\rightsquigarrow g(1)$		$\rightsquigarrow f(1, (g(0) + 1))$
$\rightsquigarrow g(0) + 1$		$\rightsquigarrow f(1, (0 + 1))$
$\rightsquigarrow 0 + 1$		$\rightsquigarrow f(1, 1)$
$\rightsquigarrow 1$		$\rightsquigarrow 1$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 288 of 709

Go Back

Full Screen

Close

Quit

Two Computations of $h(-1)$

$h(-1)$		$h(-1)$
$\rightsquigarrow f(-1, g(-1))$		$\rightsquigarrow f(-1, g(-1))$
$\rightsquigarrow 0$		$\rightsquigarrow f(-1, (g(-2) + 1))$
DONE!		$\rightsquigarrow f(-1, ((g(-3) + 1) + 1)$
		$\rightsquigarrow \dots$
		\rightsquigarrow FOREVER!



Recursive Computations

- Newton's method
- Factorial
 - $factL$
 - $factR$

skip to nonlinear recursion
skip to Recursion Revisited

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 289 of 709

Go Back

Full Screen

Close

Quit



Recursion: Left

$factL(4)$

$\rightsquigarrow (factL(4 - 1) * 4)$

$\rightsquigarrow (factL(3) * 4)$

$\rightsquigarrow ((factL(3 - 1) * 3) * 4)$

$\rightsquigarrow ((factL(2) * 3) * 4)$

$\rightsquigarrow (((factL(2 - 1) * 2) * 3) * 4)$

$\rightsquigarrow \dots$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 290 of 709

Go Back

Full Screen

Close

Quit



Recursion: Right

$$factR(4)$$

$$\rightsquigarrow (4 * factR(4 - 1))$$

$$\rightsquigarrow (4 * factR(3))$$

$$\rightsquigarrow (4 * (3 * factR(3 - 1)))$$

$$\rightsquigarrow (4 * (3 * factR(2)))$$

$$\rightsquigarrow (4 * (3 * (2 * factR(2 - 1))))$$

$$\rightsquigarrow \dots$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 291 of 709

Go Back

Full Screen

Close

Quit



Recursion: Nonlinear

Fibonacci

$$\begin{aligned} & fib(5) \\ \rightsquigarrow & fib(4) + fib(3) \\ \rightsquigarrow & (fib(3) + fib(2)) + fib(3) \\ \rightsquigarrow & ((fib(2) + fib(1)) + fib(2)) + fib(3) \\ \rightsquigarrow & (((fib(1) + fib(0)) + fib(1)) + fib(2)) + fib(3) \\ \rightsquigarrow & (((1 + fib(0)) + fib(1)) + fib(2)) + fib(3) \\ \rightsquigarrow & \dots \end{aligned}$$

contd ...

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 232 of 709

Go Back

Full Screen

Close

Quit

Some Practical Questions

- What is the essential **difference** between the computations of *newton* and the two factorial programs?

Answer: **Constant space** vs. **Linear space**

- What is the essential **similarity** between the computations of *factL* and *factR*? **Answer**

- Why can't we calculate beyond *fib*(43) using the definition **Fibonacci**, on *ccsun50* or a P-IV? **Answer**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 293 of 709

Go Back

Full Screen

Close

Quit



Some Practical Questions

- What does a computation of **Fi-bonacci** look like?
- What is the essential difference between the computations of Fibonacci and *newton* or *factL* or *factR*?

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 294 of 709

Go Back

Full Screen

Close

Quit

4. Correctness, Termination & Complexity

4.1. Termination and Space Complexity

1. Recursion Revisited
2. Linear Recursion: Waxing
3. Recursion: Waning
4. Nonlinear Recursions
5. Fibonacci: *contd*
6. Recursion: Waxing & Waning
7. Unfolding Recursion
8. Non-termination
9. Termination
10. Proofs of termination
11. Proofs of termination: Induction
12. Proof of termination: Factorial
13. Proof of termination: Factorial
14. Fibonacci: Termination
15. GCD computations
16. Well-foundedness: GCD
17. Well-foundedness



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 295 of 709

Go Back

Full Screen

Close

Quit

18. Induction is Well-founded
19. Induction is Well-founded
20. Where it doesn't work
21. Well-foundedness is inductive
22. Well-foundedness is inductive
23. GCD: Well-foundedness
24. Newton: Well-foundedness
25. Newton: Well-foundedness
26. Example: Zero
27. Questions
28. The Collatz Problem
29. Questions
30. Space Complexity
31. Newton & Euclid: Absolute
32. Newton & Euclid: Relative
33. Deriving space requirements
34. GCD: Space
35. Factorial: Space
36. Fibonacci: Space
37. Fibonacci: Space



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 296 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 297 of 709

Go Back

Full Screen

Close

Quit

Recursion Revisited

- Linear recursions
 - Waxing
 - Waning
- Non-linear recursion



Linear Recursion: Waxing

$$\begin{aligned} & factL(4) \\ \rightsquigarrow & (factL(3) * 4) \\ \rightsquigarrow & ((factL(2) * 3) * 4) \\ \rightsquigarrow & (((factL(1) * 2) * 3) * 4) \\ \rightsquigarrow & (((((factL(0) * 1) * 2) * 3) * 4) \end{aligned}$$

contrast with newton

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 298 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 299 of 709

Go Back

Full Screen

Close

Quit

Recursion: Waning

$$\rightsquigarrow (((((1 * 1) * 2) * 3) * 4)$$

$$\rightsquigarrow (((1 * 2) * 3) * 4)$$

$$\rightsquigarrow ((2 * 3) * 4)$$

$$\rightsquigarrow (6 * 4)$$

$$\rightsquigarrow 24$$

contrast with newton



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 300 of 709

Go Back

Full Screen

Close

Quit

Nonlinear Recursions

Fibonacci

- Each computation of *fib* has its own waxing and waning
- There is still an “envelope” which shows waxing and waning.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 301 of 709

Go Back

Full Screen

Close

Quit

Fibonacci: *contd*

$$\rightsquigarrow (((1 + 1) + fib(1)) + fib(2)) + fib(3)$$

$$\rightsquigarrow (2 + fib(1)) + fib(2)) + fib(3)$$

$$\rightsquigarrow ((2 + 1) + fib(2)) + fib(3)$$

$\rightsquigarrow \dots$

Recursion: Waxing & Waning

- **Waning**: Occurs when an expression is **simplified** without requiring replacement of names by definitions.
- **Waxing**: Occurs when a *name* is replaced by its *definition*.
 - name by value replacements
 - occurs in **generalized composition** but just once if it is not recursively defined
 - **Unfolding recursion**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 302 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 303 of 709

Go Back

Full Screen

Close

Quit

Unfolding Recursion

- may occur several times (terminating), or
- even an infinite number of times leading to nontermination



Non-termination

Algorithm

- Simple expressions never lead to nontermination
- (Generalized) composition never leads to nontermination
- Recursion may lead to non-termination or infinite computations, unless proved **otherwise**

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 304 of 709

Go Back

Full Screen

Close

Quit



Termination

Since **recursion** may lead to nontermination

- **Termination** needs to be **proved** for **recursive definitions**, and
- for expressions and definitions that use **recursively defined names** as components.

[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)

[◀](#) [▶](#)

[◀](#) [▶](#)

Page 305 of 709

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 306 of 709

Go Back

Full Screen

Close

Quit

Proofs of termination

A recursive definition guarantees termination

- if it is **inductive**, or
- it is **well-founded**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 307 of 709

Go Back

Full Screen

Close

Quit

Proofs of termination: Induction

A recursive definition guarantees termination

- if it is **inductive**,

Examples:

- **Factorial**
- **Fibonacci**

- it is **well-founded**, though not obviously inductive



Proof of termination: Factorial

Factorial

Consider *factL* defined only for non-negative integers. We prove that it is an algorithm i.e. that it terminates

Basis : For $n = 0$, $factL(n) = 1$ which is not a recursive definition. Hence it does indeed terminate in a single step.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 308 of 709

Go Back

Full Screen

Close

Quit



Proof of termination: Factorial

Induction hypothesis . For some $n > 0$, $\forall k : 0 \leq k \leq n : \text{factL}(k)$ terminates in $\propto k$ steps.

Induction step . Then $\text{factL}(n + 1) = \text{factL}(n) * (n + 1)$ is guaranteed to terminate in $\propto (n + 1)$ steps, since $\text{factL}(n)$ does so in $\propto n$ steps.

back

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 309 of 709

Go Back

Full Screen

Close

Quit

Fibonacci: Termination

Fibonacci

The proof is similar to that of *factL*.

Basis For $n = 0$ or $n = 1$ $fib(n) = 1$.

Induction hypothesis For some $n > 0$,
 $\forall k : 0 \leq k \leq n : fib(k)$ terminates in
 $\propto f(k)$ steps

Induction Step Then since each of
 $fib(n)$ and $fib(n - 1)$ is guaranteed
to terminate in $\propto f(n)$ and $\propto f(n - 1)$
steps $fib(n + 1) = fib(n) + fib(n - 1)$
is also guaranteed to terminate in
 $f(n + 1) \propto f(n) + f(n - 1)$ steps.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 310 of 709

Go Back

Full Screen

Close

Quit



GCD computations

Euclidean GCD

$$\begin{aligned} & \gcd(12, 64) \\ \rightsquigarrow & \gcd(64, 12) \\ \rightsquigarrow & \gcd(12, 4) \\ \rightsquigarrow & \gcd(4, 0) \\ \rightsquigarrow & 4 \end{aligned}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 311 of 709

Go Back

Full Screen

Close

Quit

Well-foundedness: GCD

Euclidean GCD

For $x, y > 0$, $0 \leq x \bmod y < y$. Hence the sequence of remainders obtained is

- a sequence of *non-negative integers*, and
- is *strictly decreasing*

$$r_1 > r_2 > \cdots > r_{n-1} > r_n = 0$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 312 of 709

Go Back

Full Screen

Close

Quit

Well-foundedness

A definition is **well-founded** if it is possible to define a **measure** (i.e. a function w of its arguments) called the **well-founded function** such that

1. the **well-founded function** takes only **non-negative integer** values
2. with each successive recursive call the value of the **well-founded function** is **guaranteed to decrease by at least 1**.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 313 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 314 of 709

Go Back

Full Screen

Close

Quit

Induction is Well-founded

The **well-founded function** usually is a measure of the number of computation steps that the algorithm will take to terminate

- **Factorial** $w(n) \propto n$
- **Fibonacci** $w(n) \propto f(n)$

Then



Induction is Well-founded

- $w(n)$ is always non-negative if $factL$ and fib are defined only for non-negative integers
- The argument to $factL$ and fib in each recursive unfolding is **strictly decreasing**.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 315 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 316 of 709

Go Back

Full Screen

Close

Quit

Where it doesn't work

Such proofs do not work for

- *fact* arbitrarily extended to include negative integers. (since $w(n)$ no longer strictly non-negative)
- $fact(n) = fact(n+1) \text{ div } (n+1)$, even if n is non-negative (since $w(n)$ is no longer decreasing)

since the function is no longer well-founded.



Well-foundedness is inductive

But the induction variable is

- **hidden** or
- too complex to worry about, or
- it serves no useful purpose for the algorithm, except as a counter.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 317 of 709

Go Back

Full Screen

Close

Quit

Well-foundedness is inductive

Given any well-founded function $w(\vec{x})$ whose values form a decreasing sequence in some algorithm

$$y_0 > y_1 > \cdots > y_{n-1} > y_n \geq 0$$

it is possible to put this sequence in 1-1 correspondence with the set $\{0, \dots, n\}$ via a function ind such that

$$ind(w(\vec{x})) = n - i$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 318 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 319 of 709

Go Back

Full Screen

Close

Quit

GCD: Well-foundedness

GCD

Well-founded function for *gcd*

$$w(a, b) = b$$

Newton: Well-foundedness

Newton's Method

Convergence condition

$$f(x_0), f(x_1), f(x_2), \dots \rightarrow 0$$

Compute the discrete value sequence

$$x_0, x_1, x_2, \dots, x_n$$

such that

$$k_0 > k_1 > k_2 > \dots > k_n = 0$$

where



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 320 of 709

Go Back

Full Screen

Close

Quit



Newton: Well-foundedness

Newton's Method

$$k_i \varepsilon \leq |f(x_i) - 0| < (k_i + 1) \varepsilon$$

and therefore inductive on **integer multiples of ε** Hence

$$w(x) = \left\lfloor \frac{|f(x)|}{\varepsilon} \right\rfloor$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 321 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 322 of 709

Go Back

Full Screen

Close

Quit

Example: Zero

A peculiar way to define the *zero* function

$$\text{zero}(x) =$$

$$\begin{cases} \text{zero}(x + 1.0) & \text{if } x \leq -1.0 \\ 0.0 & \text{if } -1.0 < x < 1.0 \\ \text{zero}(x - 1.0) & \text{if } x \geq 1.0 \end{cases}$$

$w(x) = \lceil |x| \rceil$ is the well-founded function



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 323 of 709

Go Back

Full Screen

Close

Quit

Questions

Q: Is it always possible to find a well-founded function for each algorithm?

A: Unfortunately not! However if we can't then we cannot call it an algorithm!. But if we can then we are guaranteed that the algorithm will terminate.

The Collatz Problem



The Collatz Problem

Does the following algorithm terminate?

$$\text{collatz}(m) =$$

$$\begin{cases} 1 & \text{if } m \leq 1 \\ \text{collatz}(m \text{ div } 2) & \text{if } m \text{ is even} \\ \text{collatz}(3 * m + 1) & \text{otherwise} \end{cases}$$

Unproven Claim. $\text{collatz}(m) \rightsquigarrow 1$ for all m .

Questions

Q: what other uses can well-founded functions be put to?

A: They can be used to estimate the complexity of your algorithm in order of magnitude terms.

Space Complexity : The amount of memory space required, as a function of the input

Time Complexity : The amount of time (number of computation steps) as a function of the input



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 325 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 326 of 709

Go Back

Full Screen

Close

Quit

Space Complexity

What is the space complexity of

- Newton's method
- Euclidean GCD
- Factorial
- Fibonacci

Newton & Euclid: Absolute

Newton's Method
Computation

Newton's method (wherever and whenever it works well) requires space to compute

- the value of f at each point x_i
- the value of f' at each point x_i
- the value of x_{i+1} from the above

Their **absolute** space requirements could be different. But ...

Euclidean GCD
Computation



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 327 of 709

Go Back

Full Screen

Close

Quit



Newton & Euclid: Relative

Newton's Method
Computation

GCD and Newton's method (wherever and whenever it works well) require the same amount of space for each **recursive unfolding** since each fresh unfolding can reuse the space used by the previous one.

Euclidean GCD
Computation

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 328 of 709

Go Back

Full Screen

Close

Quit



Deriving space requirements

We may use the algorithm itself to derive the space required as follows:

Assume that memory *proportional to calculating* and *outputting* the answer is a *unit*. Then space as a function of the input is given by

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 329 of 709

Go Back

Full Screen

Close

Quit



GCD: Space

$$S_{gcd(a,b)} = \begin{cases} 1 & \text{if } b = 0 \\ S_{gcd(b, a \bmod b)} & \text{otherwise} \end{cases}$$

This implies (from well-foundedness) that the entire computation ends with the space of a **unit**.

$$S_{gcd(a,b)} \propto 1$$

A similar analysis and result holds for *newton*

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 330 of 709

Go Back

Full Screen

Close

Quit

Factorial: Space

factL

$$S_{factL}(n) = \begin{cases} 1 & \text{if } n = 0 \\ S_{factL}(n-1) + 1 & \text{otherwise} \end{cases}$$

The **1** is for output and the **+1** is because one needs to store space proportional to remembering “*multiply by n*”.

$$S_{factL}(n) \propto n.$$

A similar analysis and result holds for *factR*.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 331 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 332 of 709

Go Back

Full Screen

Close

Quit

Fibonacci: Space

Fibonacci

$$S_{fib(n)} = \begin{cases} 1 & \text{if } n \leq 1 \\ S_{fib(n-1)} + S_{fib(n-2)} & \text{if } n > 1 \end{cases}$$

Fibonacci: Space

Fibonacci

It is easy to see prove by induction that for $n > 1$,

$$S_{fib(n-1)} < S_{fib(n)} \leq 2S_{fib(n-1)}$$

That is, as the value of n increases by 1 the space requirement approximately doubles. Further, it is easy to show by induction that

$$2^{n-2} < S_{fib(n)} \leq 2^{n-1}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 333 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 334 of 709

Go Back

Full Screen

Close

Quit

4.2. Efficiency Measures and Speed-ups

1. Recapitulation
2. Recapitulation
3. Time & Space Complexity
4. *isqrt*: Space
5. Time Complexity
6. *isqrt*: Time Complexity
7. *isqrt2*: Time
8. *shrink* vs. *shrink2*: Times
9. Factorial: Time Complexity
10. Fibonacci: Time Complexity
11. Comparative Complexity
12. Comparisons
13. Comparisons
14. Efficiency Measures: Time
15. Efficiency Measures: Space
16. Speeding Up: 1
17. Speeding Up: 2

18. Factoring out calculations
19. Tail Recursion: 1
20. Tail Recursion: 2
21. Factorial: Tail Recursion
22. Factorial: Tail Recursion
23. A Computation
24. Factorial: Issues
25. Fibonacci: Tail Recursion
26. Fibonacci: Tail Recursion
27. fibTR: SML
28. State in Tail Recursion
29. Invariance



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 335 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 336 of 709

Go Back

Full Screen

Close

Quit

Recapitulation

- Recursion & nontermination
- Termination & well-foundedness
- Well-foundedness proofs
- Well-foundedness & Complexity

Recapitulation

- Recursion & nontermination
- Termination & well-foundedness
- Well-foundedness proofs
 - By induction
 - well-founded functions
 - By well-founded functions
 - induction as well-foundedness
 - Well-foundedness as induction
- Well-foundedness & Complexity



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 337 of 709

Go Back

Full Screen

Close

Quit

Time & Space Complexity

Questions

An order of magnitude estimate of the **time** or **space (memory)** required (in terms of some large computation steps).

- **Newton & Euclid's GCD**
- **Deriving space requirements**
 - **Integer Sqrt**
 - **Factorial**
 - **Fibonacci**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 338 of 709

Go Back

Full Screen

Close

Quit

isqrt: Space

Integer Sqrt

shrink

$S_{isqrt}(n) = S_{shrink}(n,0,n)$ for large n .

$S_{shrink}(n,l,u) =$

$$\begin{cases} 1 & \text{if } l = u \\ S_{shrink}(n,l+1,u) & \text{if } l < u \dots \\ S_{shrink}(n,l,u-1) & \text{if } l < u \dots \end{cases}$$

Assuming 1 unit of space for output.

By induction on $||[l, u]||$

$$S_{isqrt}(n) = S_{shrink}(n,0,n) \propto 1$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 339 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 340 of 709

Go Back

Full Screen

Close

Quit

Time Complexity

As in the case of **space** we may use the algorithm itself to derive the **time** complexity.

- Integer sqrt
- Factorial
- Fibonacci

forward

isqrt: Time Complexity

Integer Sqrt
shrink

Assume **condition-checking** (along with $+1$ or -1) takes a unit of time.

Then $T_{shrink}(n, l, u) =$

$$\begin{cases} 0 & \text{if } l = u \\ 1 + T_{shrink}(n, l+1, u) & \text{if } l < u \dots \\ 1 + T_{shrink}(n, l, u-1) & \text{if } l < u \dots \end{cases}$$

Then $T_{shrink}(n, l, u) \propto |[l, u]| - 1$ and

$$T_{isqrt}(n) = T_{shrink}(n, 0, n) \propto n$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 341 of 709

Go Back

Full Screen

Close

Quit

isqrt2. Time

shrink

Assume **condition-checking** (along with $(l + u) \text{ div } 2$) takes a unit of time.

Then $T_{shrink2}(n, l, u) =$

$$\begin{cases} 0 & \text{if } u \leq l \leq u \\ 1 + T_{shrink2}(n, m, u) & \text{if } m^2 \leq n \dots \\ 1 + T_{shrink2}(n, l, u-1) & \text{if } m^2 > n \end{cases}$$

If $2^{k-1} \leq |[l, u]| - 1 < 2^k$ then the algorithm terminates in at most k steps.

Since $k = \lceil \log_2 |[l, u]| - 1 \rceil$,

$$T_{shrink2}(n, l, u) \propto \lceil \log_2 |[l, u]| - 1 \rceil$$

$$T_{shrink2}(n) \propto \lceil \log_2 n \rceil$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 342 of 709

Go Back

Full Screen

Close

Quit

shrink VS. *shrink2*: Times

shrink

shrink2

1. The time units are **different**,
2. But they differ by a **constant** factor at most.
3. So clearly, for **large** n , *shrink2* is **faster** than *shrink*
4. But for **small** n , it depends on the **constant factor**.
5. **Implicitly** assume that the actual unit of time includes the time required to **unfold** the recursion.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 343 of 709

Go Back

Full Screen

Close

Quit



Factorial: Time Complexity

factL

Here we assume **multiplication** takes unit time.

$$T_{factL}(n) = \begin{cases} 0 & \text{if } n = 0 \\ T_{factL}(n-1) + 1 & \text{otherwise} \end{cases}$$

Then

$$T_{factL}(n) \propto n$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 344 of 709

Go Back

Full Screen

Close

Quit

Fibonacci: Time Complexity

Fibonacci

Assuming addition and condition-checking together take a unit of time, we have

$$T_{fib(n)} = \begin{cases} 0 & \text{if } n \leq 1 \\ T_{fib(n-1)} + T_{fib(n-2)} & \text{if } n > 1 \end{cases}$$

It follows that

$$2^{n-2} < T_{fib(n)} \leq 2^{n-1}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 345 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 346 of 709

Go Back

Full Screen

Close

Quit

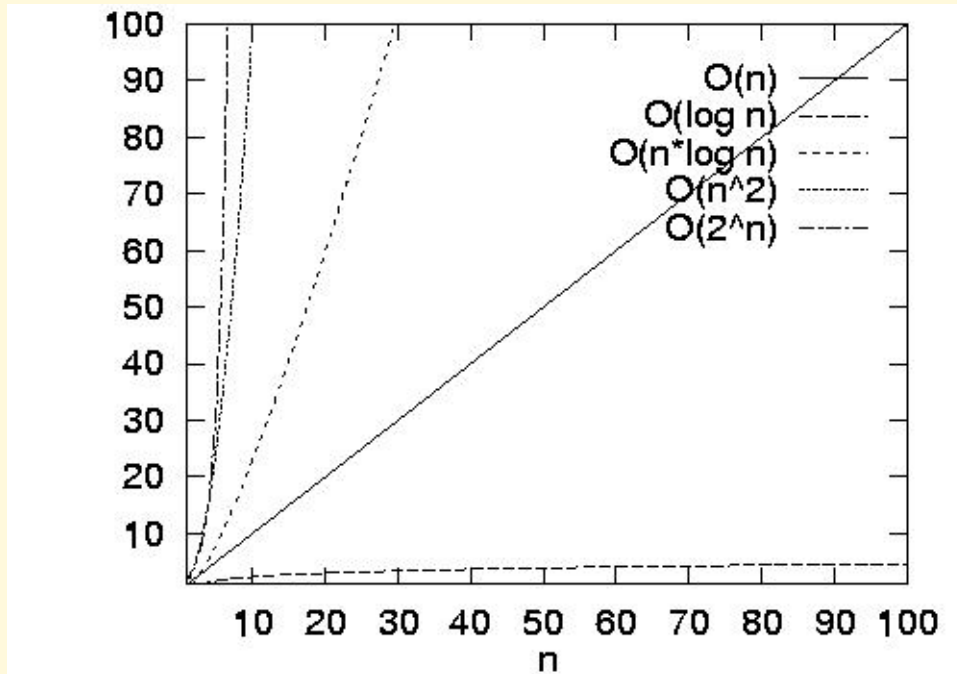
Comparative Complexity

Algorithm	Space	Time
$isqrt(n)$	$O(1)$	$O(n)$
$isqrt2(n)$	$O(1)$	$O(\log_2 n)$
$factL(n)$	$O(n)$	$O(n)$
$fib(n)$	$O(2^n)$	$O(2^n)$



Comparisons

For smaller values



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 347 of 709

Go Back

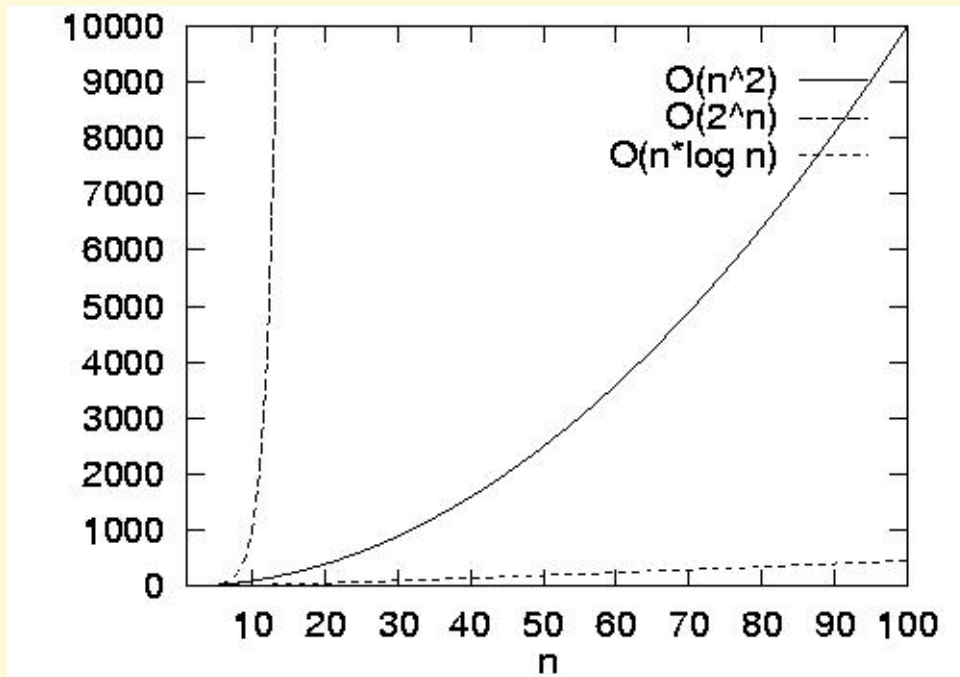
Full Screen

Close

Quit

Comparisons

For **large** values



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 348 of 709

Go Back

Full Screen

Close

Quit



Efficiency Measures: Time

An algorithm for a problem is asymptotically faster or asymptotically more time-efficient than another for the same problem if its time complexity is bounded by a function that has a slower growth rate as a function of the value of its arguments.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 349 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 350 of 709

Go Back

Full Screen

Close

Quit

Efficiency Measures: Space

Similarly an algorithm is asymptotically more space efficient than another if its **space complexity** is bounded by a function that has a *slower growth rate*.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 351 of 709

Go Back

Full Screen

Close

Quit

Speeding Up: 1

Q: Can fibonacci be speeded up or made more space efficient?

A: Perhaps by studying the nature of the function e.g. *isqrt2* vs. *isqrt* and attempting more efficient algorithmic variations.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 352 of 709

Go Back

Full Screen

Close

Quit

Speeding Up: 2

Q: Are there general methods of speeding up or saving space?

A: Take inspiration from *gcd*, *newton*, *shrink*

Factoring out calculations

$$\gcd(a_0, b_0)$$

compute a_1, b_1

$$\rightsquigarrow \gcd(a_1, b_1)$$

compute a_2, b_2

$$\rightsquigarrow \gcd(a_2, b_2)$$

$\rightsquigarrow \dots$

$$\rightsquigarrow \gcd(a_n, b_n)$$

$$\rightsquigarrow a_n$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 353 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 354 of 709

Go Back

Full Screen

Close

Quit

Tail Recursion: 1

- Factor out calculations and remember only those values that are required for the next recursive call.
- Create a vector of state variables and include them as arguments of the function



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 355 of 709

Go Back

Full Screen

Close

Quit

Tail Recursion: 2

- Try to **reorder** the computation using the **state** variables so as to get the **next state completely defined**.
- Redefine the function entirely in terms of the state variables so that the recursive call is the **outermost** operation.

Factorial: Tail Recursion

factL Waxing

factL Waning

- The recursive call **precedes** the multiplication operation. *Change it!*
- Define a **state** variable *p* which contains the product of all the values that one must remember
- **Reorder** the computation so that the computation of *p* is performed before the recursive call.
- For that **redefine** the function in terms of *p*.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 356 of 709

Go Back

Full Screen

Close

Quit

Factorial: Tail Recursion

Factorial

$$factL2(n) = \begin{cases} \perp & \text{if } n < 0 \\ 1 & \text{if } n = 0 \\ factL_tr(n, 1) & \text{otherwise} \end{cases}$$

where

$$factL_tr(n, p) = \begin{cases} p & \text{if } n = 0 \\ factL_tr(n - 1, np) & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 357 of 709

Go Back

Full Screen

Close

Quit



A Computation

$factL2(4)$
 $\rightsquigarrow factL_tr(4, 1)$
 $\rightsquigarrow factL_tr(3, 4)$
 $\rightsquigarrow factL_tr(2, 12)$
 $\rightsquigarrow factL_tr(1, 24)$
 $\rightsquigarrow factL_tr(0, 24)$
 $\rightsquigarrow 24$

Reminiscent of **gcd** and **newton**!

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 358 of 709

Go Back

Full Screen

Close

Quit

Factorial: Issues

- **Correctness:** Prove (by induction on n) that for all $n \geq 0$, $factL2(n) = n!$.
- **termination:** Prove by induction on n that **every** computation of $factL2$ terminates.
- **Space complexity:** Prove that $S_{factL2(n)} = O(1)$ (as against $S_{factL(n)} \propto n$).
- **Time complexity:** Prove that $T_{factL2(n)} = O(n)$

Complexity table



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 359 of 709

Go Back

Full Screen

Close

Quit

Fibonacci: Tail Recursion

- Remove **duplicate** computations by defining appropriate state variables
- Let a and b be the consecutive fibonacci numbers $fib(m - 2)$ and $fib(m - 1)$ required for the computation of $fib(m)$.
- The **state** consists of the variables n, a, b, m .



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 360 of 709

Go Back

Full Screen

Close

Quit

Fibonacci: Tail Recursion

$$fibTR(n) =$$

$$\begin{cases} \perp & \text{if } n < 0 \\ 1 & \text{if } 0 \leq n \leq 1 \\ fib_iter(n, 1, 1, 1) & \text{otherwise} \end{cases}$$

where

$$fib_iter(n, a, b, m) =$$

$$\begin{cases} b & \text{if } m \geq n \\ fib_iter(n, b, a + b, m + 1) & \text{otherwise} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 361 of 709

Go Back

Full Screen

Close

Quit

fibTR: SML

```
local
  fun fib_iter (n, a, b, m) =
    (* fib (m) = b , fib (m-1) = a *)
    if m >= n then b
    else fib_iter (n, b, a+b, m+1)
in
  fun fibTR (n) =
    if n < 0 then raise negativeArgument
    else if (n <= 1) then 1
    else fib_iter (n, 1, 1, 1)
end;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 362 of 709

Go Back

Full Screen

Close

Quit



State in Tail Recursion

- The variables that make up the *state* bear a definite relation to each other.
- **INVARIANCE**. That relationship between the state variables does not change throughout the computation of the function.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 363 of 709

Go Back

Full Screen

Close

Quit



Invariance

- The **invariant** property of a tail-recursive function must hold

Initially when it is first invoked, and **Continues** to hold before every successive invocation

- The **invariant** property characterizes the entire computation and the algorithm and is crucial to the proof of correctness

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 364 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 365 of 709

Go Back

Full Screen

Close

Quit

4.3. Invariance & Correctness

1. Recap
2. Recursion Transformation
3. Tail Recursion: Examples
4. Comparisons
5. Transformation Issues
6. Correctness Issues 1
7. Correctness Issues 2
8. Correctness Theorem
9. Invariants & Correctness 1
10. Invariants & Correctness 2
11. Invariance Lemma: *factL.tr*
12. Invariance: Example
13. Invariance: Example
14. Proof
15. Invariance Lemma: *fib.iter*
16. Proof
17. Correctness: Fibonacci

- 18. Variants & Invariants
- 19. Variants & Invariants
- 20. More Invariants
- 21. Fast Powering 1
- 22. Fast Powering 2
- 23. Root Finding: Bisection
- 24. Advantage Bisection



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 366 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 367 of 709

Go Back

Full Screen

Close

Quit

Recap

- Asymptotic Complexity:
 - Space
 - Time
- Comparative Complexity
- Comparisons:
 - Small inputs
 - Large inputs

Recursion Transformation

- To achieve **constant space** and **linear time**, if possible
- Speeding up using **tail recursion**
 - **Factor** out calculations
 - **Reorder** the computations with state variables
 - Recursion as the **outermost** operation



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 368 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 369 of 709

Go Back

Full Screen

Close

Quit

Tail Recursion: Examples

- Factorial vs. Factorial:
factL vs. *factL2* vs.
- Fibonacci vs. Fibonacci:
fib vs. *fibTR*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 370 of 709

Go Back

Full Screen

Close

Quit

Comparisons

Algorithm	Space	Time
$isqrt(n)$	$O(1)$	$O(n)$
$isqrt2(n)$	$O(1)$	$O(\log_2 n)$
$factL(n)$	$O(n)$	$O(n)$
$factL2(n)$	$O(1)$	$O(n)$
$fib(n)$	$O(2^n)$	$O(2^n)$
$fibTR(n)$	$O(1)$	$O(n)$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 371 of 709

Go Back

Full Screen

Close

Quit

Transformation Issues

- **Correctness**: Prove that the **new** algorithm computes the same function as the **original simple** algorithm
- **Termination**: Prove by induction on n that **every** computation is finite.
- **Space complexity**: Compute it.
- **Time complexity**: Compute it.



Correctness Issues 1

- **Absolute** correctness: For any function f , that an algorithm A that claims to implement it, prove that

$$f(\vec{x}) = A(\vec{x})$$

for all argument values \vec{x} for which f is defined.

- **Transformation** correctness:



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 373 of 709

Go Back

Full Screen

Close

Quit

Correctness Issues 2

- **Absolute** correctness:
- **Transformation** correctness: For any algorithm A and a transformed algorithm B prove that

$$A(\vec{x}) = B(\vec{x})$$

for all argument values \vec{x} for which A is defined. Then B is **absolutely** correct provided A is **absolutely** correct.



Correctness Theorem

Invariant properties *factL2*

Theorem 8 For all $n \geq 0$,

$$\text{factL2}(n) = n!$$

Proof: For $n = 0$, it is clear that $\text{factL2}(0) = 1 = 0!$. For $n > 0$, $\text{factL2}(n) = \text{factL_tr}(n, 1)$. The proof is *done* **provided** we can show that $\text{factL_tr}(n, 1) = n!$. \square



Invariants & Correctness 1

Invariant properties *factL2*

- To prove the **absolute** or **transformation** correctness of a tail-recursion transformation usually requires an **invariant** property to be proven about the tail-recursive function.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 375 of 709

Go Back

Full Screen

Close

Quit



Invariants & Correctness 2

Invariant properties *factL2*

- This allows the independent proof of the properties of the tail-recursive function without reference to the function that uses it.
- It reflects the design of the algorithm and its division into sub-problems.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 376 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 377 of 709

Go Back

Full Screen

Close

Quit

Invariance Lemma:

factL_tr

Invariant properties *factL2*

Lemma 9 For all $n \geq 0$ and p

$$\text{factL_tr}(n, p) = (n!)p$$

Proof: By induction on n . □

Back to theorem

Invariance: Example

factL2

$$\begin{aligned} & \text{factL_tr}(4, 7) \\ \rightsquigarrow & \text{factL_tr}(3, 28) \\ \rightsquigarrow & \text{factL_tr}(2, 84) \\ \rightsquigarrow & \text{factL_tr}(1, 168) \\ \rightsquigarrow & \text{factL_tr}(0, 168) \\ \rightsquigarrow & 168 \end{aligned}$$

Contrast with a *factL2(4)* computation



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 378 of 709

Go Back

Full Screen

Close

Quit



Invariance: Example

factL2

So what exactly is invariant?

$$\begin{aligned} & \text{factL_tr}(4, 7) \quad 168 = 4! \times 7 \\ \rightsquigarrow & \text{factL_tr}(3, 28) \quad 168 = 3! \times 28 \\ \rightsquigarrow & \text{factL_tr}(2, 84) \quad 168 = 2! \times 84 \\ \rightsquigarrow & \text{factL_tr}(1, 168) \quad 168 = 1! \times 168 \\ \rightsquigarrow & \text{factL_tr}(0, 168) \quad 168 = 0! \times 168 \\ \rightsquigarrow & 168 \end{aligned}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 379 of 709

Go Back

Full Screen

Close

Quit

Proof

Basis For $n = 0$, $factL_tr(0, p) = p = (0!)p$.

Induction hypothesis (IH) For all k ,
 $0 < k \leq n$,

$$factL_tr(k, p) = p = (k!)p$$

Induction Step

$$\begin{aligned} & factL_tr(n + 1, p) \\ = & factL_tr(n, (n + 1)p) \\ = & (n!)(n + 1)p \quad (IH) \\ = & (n + 1)!p \end{aligned}$$

Back to lemma



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 380 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 381 of 709

Go Back

Full Screen

Close

Quit

Invariance Lemma:

fib_iter

fib_iter **F**

Lemma 10 For all $n > 1, a, b, m : 1 \leq m \leq n$, if $a = \mathbf{F}(m - 1)$ and $b = \mathbf{F}(m)$, then

$$\boxed{\text{INV} : \text{fib_iter}(n, a, b, m) = \mathbf{F}(n)}$$

Proof: By induction on $k = n - m$ \square

Proof

Basis For $k = 0$, $n = m$, it follows that
$$\text{fib_iter}(n, a, b, m) = \mathbf{F}(n)$$

Induction hypothesis (IH) For all $n > 1$ and $1 \leq m \leq n$, with $n - m \leq k$,
INV holds

Induction Step Let $1 \leq m < n$ such that $n - m = k + 1$, $\mathbf{F}(m) = b$ and $\mathbf{F}(m - 1) = a$. Then $\mathbf{F}(m + 1) = a + b$ and

$$\begin{aligned} & \text{fib_iter}(n, a, b, m) \\ &= \text{fib_iter}(n, b, a + b, m + 1) \\ &= \mathbf{F}(n) \end{aligned} \quad (IH)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 382 of 709

Go Back

Full Screen

Close

Quit



Correctness: Fibonacci

fibTR

Fibonacci

Theorem 11 For all $n \geq 0$,
 $\text{fibTR}(n) = \mathbf{F}(n)$.

Proof: For $0 \leq n \leq 1$, it holds trivially. For $n > 1$, $\text{fibTR}(n) = \text{fib_iter}(n, 1, 1, 1) = \mathbf{F}(n)$, by the **invariance lemma**, with $m = 1$, $a = 1 = \mathbf{F}(m - 1)$ and $b = 1 = \mathbf{F}(m)$. \square



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

⏪ ⏩

◀ ▶

Page 384 of 709

Go Back

Full Screen

Close

Quit

Variants & Invariants

factL2

$factL3(n) =$

$$\begin{cases} \perp & \text{if } n < 0 \\ 1 & \text{if } n = 0 \\ factL_tr2(n, 1, 1) & \text{else} \end{cases}$$

where



Variants & Invariants

factL2

$factL_tr2(n, p, m) =$

$$\begin{cases} p & \text{if } n = m \\ factL_tr2(n, (m + 1)p, m + 1) & \text{else} \end{cases}$$

$$factL_tr2(n, p, m) = (m!)p$$

for all $1 \leq m \leq n$.



More Invariants

- *shrink* For all $n > 0$, l , u , if $[l, u] \subseteq [0, n]$,

$$l \leq \lfloor \sqrt{n} \rfloor \leq u$$

- *shrink2*

For all $n > 0$, l , u , if $[l, u] \subseteq [0, n]$,

$$m = \lfloor (l + u) / 2 \rfloor \text{ and } l \leq \lfloor \sqrt{n} \rfloor \leq u$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 387 of 709

Go Back

Full Screen

Close

Quit

Fast Powering 1

power2

power3(x, n) =

$$\begin{cases} 1.0/\textit{power3}(x, -n) & \text{if } n < 0 \\ 1.0 & \text{if } n = 0 \\ \textit{powerTR}(x, n, 1) & \text{else} \end{cases}$$

where



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 388 of 709

Go Back

Full Screen

Close

Quit

Fast Powering 2

power2

$powerTR(x, n, p) =$

$$\begin{cases} p & \text{if } n = 0 \\ powerTR(x^2, n \operatorname{div} 2, p) & \text{if } even(n) \\ powerTR(x^2, n \operatorname{div} 2, xp) & \text{otherwise} \end{cases}$$

where $even(n) \iff n \bmod 2 = 0$.

$$powerTR(x, n, p) = x^n p$$



Root Finding: Bisection

Newton's Method

Algorithm

Select a small enough $\varepsilon > 0$ and x_0 . Then if $\text{sgn}(f(a)) \neq \text{sgn}(f(b))$,
 $\text{bisect}(f, a, b, \varepsilon) =$

$$\begin{cases} c & \text{if } |f(c)| < \varepsilon \\ \text{bisect}(f, c, b, \varepsilon) & \text{if } \text{sgn}(f(c)) \neq \text{sgn}(f(b)) \\ \text{bisect}(f, a, c, \varepsilon) & \text{otherwise} \end{cases}$$

where $c = (a + b)/2$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 389 of 709

Go Back

Full Screen

Close

Quit

Advantage Bisection

More **robust** than Newton's method

- Requires continuity and change of sign
- Does not require differentiability
- Could change the condition suitably to take care of very shallow curves
- **Oscillations** could occur only if the function is too **steep**.
- An intermediate point can never go **outside** the interval.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 390 of 709

Go Back

Full Screen

Close

Quit

5. Compound Data

5.1. Tuples, Lists & the Generation of Primes

1. Recap: Tail Recursion
2. Examples: Invariants
3. Tuples
4. Lists
5. New Lists
6. List Operations
7. List Operations: *cons*
8. Generating Primes upto n
9. More Properties
10. Composites
11. Odd Primes
12. *primesUpto(n)*
13. *generateFrom(P, m, n, k)*
14. *generateFrom*
15. *primeWRT(m, P)*
16. *primeWRT(m, P)*
17. *primeWRT*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 391 of 709

Go Back

Full Screen

Close

Quit

- 18. Density of Primes
- 19. The Prime Number Theorem
- 20. The Prime Number Theorem
- 21. Complexity
- 22. Diagnosis



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 392 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 393 of 709

Go Back

Full Screen

Close

Quit

Recap: Tail Recursion

- Asymptotic Complexity:

Time Linear

Space Constant

- Correctness: Capture the algorithm through

Invariant Invariance Lemma

Bound function Proof by induction



Examples: Invariants

factL_tr2

shrink & shrink2

$$l \leq \lfloor \sqrt{n} \rfloor \leq u$$

$$m = \lfloor (l + u) / 2 \rfloor \text{ and } l \leq \lfloor \sqrt{n} \rfloor \leq u$$

Fast Powering

$$\text{powerTR}(x, n, p) = x^n p$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 394 of 709

Go Back

Full Screen

Close

Quit

Tuples: Formation

Simplest form of compound data:
Cartesian products.

- Each element of a cartesian product is a **tuple**
- Tuples may be constructed as we do in mathematics, simply by enclosing the elements (separated by commas) in a pair of parentheses.

```
- val a = (2, 3.0, false);  
val a = (2, 3.0, false) :  
      int * real * bool
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 395 of 709

Go Back

Full Screen

Close

Quit



Tuples: Decomposition

- Individual components of a tuple may be taken out too.

```
- #1 a;
```

```
val it = 2 : int
```

```
- #2 (a);
```

```
val it = 3.0 : real
```

```
- #3 a;
```

```
val it = false : bool
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 396 of 709

Go Back

Full Screen

Close

Quit

Tuples: divmod

Standard ML of New Jersey, ...

```
- fun divmod (a, b) =  
    (a div b, a mod b);  
val divmod =  
fn : int * int -> int * int  
- val dm = divmod (24, 9);  
val dm = (2, 6) : int * int  
- #1 dm;  
val it = 2 : int  
- #2 dm;  
val it = 6 : int
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 397 of 709

Go Back

Full Screen

Close

Quit

Constructors & Destructors

Every way of constructing compound data from simpler data elements has

Constructors : Operators which construct compound data from simpler ones (for tuples it is simply `(, , and)`).

Destructors : Operators which allow us to extract the individual components of a compound data item (for tuples they are `#1, #2 ...` depending upon how many components it consists of).



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 398 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 399 of 709

Go Back

Full Screen

Close

Quit

Tuples: Identity

Every tuple that has been broken up into its components using its **destructors** can be put together back again using its **constructors**.

Given a tuple $\mathbf{a} \in A_1 \times A_2 \times \dots \times A_n$, we have

$$\mathbf{a} = (\#1 \mathbf{a}, \#2 \mathbf{a}, \dots, \#n \mathbf{a})$$



Lists

An α *list* represents a **sequence** of elements of a given type α .

Given a (nonempty) list

- A list is **ordered**
- There may be **more than one occurrence of an element** in the list
- only the **first** element (called the **head**) of the list is immediately accessible.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 400 of 709

Go Back

Full Screen

Close

Quit

New Lists

Given a (nonempty) list L ,

- A **new** list M may be created from an existing list L by the tl operation.
- New elements can be **added** (by the operation *cons*) to an existing list, one at a time to create **new** lists.
- the last element that was added becomes the head of the **new** list.
- Two lists are equal only if they have the same elements in the same order



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 401 of 709

Go Back

Full Screen

Close

Quit

List Operations

- The empty list: *nil* or $[]$
- Nonempty lists: Given a nonempty list L

$$L = [1, 2, 3, 4]$$

head : $hd : \alpha List \rightarrow \alpha$

$$hd(L) = 1$$

tail : $tl : \alpha List \rightarrow \alpha List$

$$tl(L) = [2, 3, 4]$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 402 of 709

Go Back

Full Screen

Close

Quit



List Operations: *cons*

- $L = [1, 2, 3, 4]$

cons : $cons : \alpha \times \alpha List \rightarrow \alpha List$

$$cons(0, nil) = [0]$$

$$cons(0, L) = 0 :: L = [0, 1, 2, 3, 4]$$

$$1 :: (0 :: L) = [1, 0, 1, 2, 3, 4]$$

back to lists Recap

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 403 of 709

Go Back

Full Screen

Close

Quit

Polynomial Evaluation

Evaluating a polynomial

$$p(x) = \sum_{i=0}^n a_i x^i$$

given

- its coefficients as a list $[a_n, \dots, a_0]$ of values from the highest degree term to the constant a_0 .
- a value for the variable x .

Assume an empty list of coefficients yields a value 0.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 404 of 709

Go Back

Full Screen

Close

Quit



Naive Solution

$poly0(L, x) =$

$$\begin{cases} 0 & \text{if } L = nil \\ hx^n + poly0(T, x) & \text{if } L = h :: T \end{cases}$$

where $n = |L| - 1$.

```
fun poly0 ([], x) = 0.0
  | poly0 (h::T, x) =
    h*power(x, n)+poly0(T, x)
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 405 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 406 of 709

Go Back

Full Screen

Close

Quit

Complexity of $\text{poly}()$

Space. $O(n)$ to store both the list and the intermediate computations.

Additions. $O(n)$ additions.

Multiplications. $n(n - 1)/2$ by the simplest powering algorithm.

Multiplications. $O(\log_2(n) + O(\log_2(n - 1) + \dots + O(\log_2(1))) \leq O(n \log_2(n))$ by the fast powering algorithm.

Arden's Rule

Factor out the multiplications to get

$$p(x) = (\dots((a_n x + a_{n-1})x + a_{n-2})x + \dots)x + a_0$$

and define a tail-recursive function which requires only n multiplications.

$$\text{poly1}(L, x) = \text{poly_TR}(0, L, x)$$

where

$$\text{poly_TR}(p, L, x) =$$

$$\begin{cases} p & \text{if } L = \text{nil} \\ \text{poly_TR}(px + h, T, x) & \text{if } L = h :: T \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 407 of 709

Go Back

Full Screen

Close

Quit

poly1 in SML

```
local
  fun poly_TR (p, [], x) = p
    | poly_TR (p, (h::T), x) =
      poly_TR (p*x + h, T, x)
in
  fun poly L x =
    poly_TR (0.0, L, x)
end;
```

Question 1. What is the right theorem to prove that *poly_TR* is the right generalization for the problem?

Ans.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 408 of 709

Go Back

Full Screen

Close

Quit



poly1 in SML

```
local
  fun poly_TR (p, [], x) = p
    | poly_TR (p, (h::T), x) =
      poly_TR (p*x + h, T, x)
in
  fun poly L x =
    poly_TR (0.0, L, x)
end;
```

$$\text{Ans. } \text{poly_TR}(p, L, x) = px^{n+1} + \sum_{i=0}^n a_i x^i$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 409 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 410 of 709

Go Back

Full Screen

Close

Quit

Reverse Input

Supposing the coefficients were given in reverse order $[a_0, \dots, a_n]$. Reversing this list will be an extra $O(n)$ time and space. Though the asymptotic complexity does not change much, it is more interesting to work directly with the given list.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 411 of 709

Go Back

Full Screen

Close

Quit

revpoly0

revpoly0(*L*, *x*) =

$$\begin{cases} 0 & \text{if } L = \text{nil} \\ h + x \times \text{revpoly0}(T, x) & \text{if } L = h :: T \end{cases}$$

```
fun revpoly0 ([], x) = 0.0
  | revpoly0 (h::T, x) =
    h + x * revpoly0 (T, x)
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 412 of 709

Go Back

Full Screen

Close

Quit

Tail Recursive

$$\text{revpoly1}(L, x) = \text{revpoly1_TR}(L, x, 1, x)$$

where

$$\text{revpoly1_TR}(L, x, p, s) =$$

$$\begin{cases} s & \text{if } L = \text{nil} \\ \text{revpoly1}(T, x, px, s + ph) & \text{if } L = h :: T \end{cases}$$



Tail Recursion: SML

```
local
  fun revpoly1_TR([], x, p, s) = s
    | revpoly1_TR((h::T), x, p, s) =
      revpoly1_TR(T, x, p*x, s+p*h)
in
  fun revpoly1 (L, x) =
    revpoly1_TR (L, x, 1.0, 0.0)
end
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

Page 413 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 414 of 709

Go Back

Full Screen

Close

Quit

Complexity of *revpoly1*

Space. $O(n)$ space to store the list

Multiplications. $2n$ multiplications

Additions. n additions.

Generating Primes upto n

Definition 12 A positive integer $n > 1$ is *composite* iff it has a *proper* divisor $d|n$ with $1 < d < n$. Otherwise it is *prime*.

- 2 is the smallest (first) prime.
- 2 is the only even prime.
- No other even number can be a prime.
- All other primes are odd



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 415 of 709

Go Back

Full Screen

Close

Quit



More Properties

- An odd number cannot have any even divisors.
- Every number may be expressed uniquely (upto order) as a product of prime factors.
- No divisor of a positive integer can be greater than itself.
- For each divisor $d|n$ such that $d \leq \lfloor \sqrt{n} \rfloor$, $n/d \geq \lfloor \sqrt{n} \rfloor$ is also a divisor.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 416 of 709

Go Back

Full Screen

Close

Quit



Composites

- If a number n is composite, then it has a proper divisor d , $2 \leq d \leq \lfloor \sqrt{n} \rfloor$.
- If a number n is composite, then it has a **prime** divisor p , $2 \leq p \leq \lfloor \sqrt{n} \rfloor$.
- An **odd** composite number n has an **odd** prime divisor p , $3 \leq p \leq \lfloor \sqrt{n} \rfloor$.



Odd Primes

- An odd number > 1 is a prime iff it has no proper odd divisors
- An odd number > 1 is a prime iff it is not divisible by any odd prime smaller than itself.
- An odd number $n > 1$ is a prime iff it is not divisible by any odd prime $\leq \lfloor \sqrt{n} \rfloor$.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 418 of 709

Go Back

Full Screen

Close

Quit



primesUpto(n)

primesUpto(n) =

$\left\{ \begin{array}{ll} [] & \text{if } n < 2 \\ [(1, 2)] & \text{if } n = 2 \\ \text{primesUpto}(n - 1) & \text{elseif } \text{even}(n) \\ \text{generateFrom} & \text{otherwise} \\ ((1, 2), 3, n, 2) & \end{array} \right.$

where



generateFrom(P, m, n, k)
bound function $n - m$

Invariant

$$2 < m \leq n \wedge \text{odd}(m)$$

implies

$$P = [(k - 1, p_{k-1}), \dots, (1, p_1)]$$

and

$$\forall q : p_{k-1} < q < m : \text{composite}(q)$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 420 of 709

Go Back

Full Screen

Close

Quit



generateFrom

generateFrom(P, m, n, k) =

$\left\{ \begin{array}{ll} P & \text{if } m > n \\ \text{generateFrom} & \text{elseif} \\ ((k, m) :: P), m + 2, n, k + 1) \text{ } pwr & \\ \text{generateFrom} & \text{else} \\ (P, m + 2, n, k) & \end{array} \right.$

where $pwr = \text{primeWRT}(m, P)$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 421 of 709

Go Back

Full Screen

Close

Quit



primeWRT(m, P)

Definition 13 A number m is *prime with respect to* a list L of numbers iff it is not divisible by any of them.

- A number is **prime** iff it is **prime with respect to** the list of all primes smaller than itself.
- From **properties of odd primes** it follows that a number n is **prime** iff it is **prime with respect to** the list of all primes $\leq \sqrt{n}$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 422 of 709

Go Back

Full Screen

Close

Quit



$primeWRT(m, P)$

bound function $length(P)$

Invariant If $P = [(i - 1, p_{i-1}), \dots, (1, p_1)]$,
for some $i \geq 1$ then

- $p_k \geq m > p_{k-1}$, and
- m is prime with respect to $[(k - 1, p_{k-1}), \dots, (i, p_i)]$
- m is a prime iff it is a prime with respect to P



primeWRT

primeWRT(m, P) =

$$\left\{ \begin{array}{ll} \textit{true} & \text{if } P = \textit{nil} \\ \textit{false} & \text{elseif } h|m \\ \textit{primeWRT} & \text{else} \\ (m, \textit{tl}(P)) & \end{array} \right.$$

where

$$(i, h) = \textit{hd}(P)$$

for some $i > 0$

Density of Primes

Let $\pi(n)$ denote the number of primes upto n . Then

n	$\pi(n)$	%
100	25	25.00%
1000	168	16.80%
10000	1229	12.29%
100000	9592	9.59%
1000000	78,498	7.85%
10000000	664579	6.65%
100000000	5761455	5.76%



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 425 of 709

Go Back

Full Screen

Close

Quit

The Prime Number Theorem

$$\lim_{n \rightarrow \infty} \frac{\pi(n)}{n / \ln n} = 1$$

Proved by Gauss.

- Shows that the primes get sparser at higher n
- A larger percentage of numbers as we go higher are composite.

from David Burton: *Elementary Number Theory*.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 426 of 709

Go Back

Full Screen

Close

Quit

The Prime Number Theorem



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 427 of 709

Go Back

Full Screen

Close

Quit

n	$\pi(n)$	%	$\lim_{n \rightarrow \infty} \frac{\pi(n)}{n / \ln n}$
100	25	25.00%	1.159
1000	168	16.80%	
10000	1229	12.29%	
100000	9592	9.59%	
1000000	78,498	7.85%	
10000000	664579	6.65%	
100000000	5761455	5.76%	

from David Burton: *Elementary Number Theory*.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 428 of 709

Go Back

Full Screen

Close

Quit

Complexity

function	calls
<i>primesUpto</i>	1
<i>generateFrom</i>	$n/2$
<i>primeWRT</i>	$\sum_{m=3, \text{odd}(m)}^n \pi(m)$

Diagnosis

For each $m \leq n$,

- P is in **descending** order of the primes
- m is checked for divisibility $\pi(m)$ times
- From **properties of odd primes** it should not be necessary to check each m more than $\pi(\lfloor \sqrt{m} \rfloor)$ times for divisibility.
- Organize P in **ascending** order instead of **descending**.

ascending-order



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 429 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 430 of 709

Go Back

Full Screen

Close

Quit

5.2. Compound Data & Lists

1. Compound Data
2. Recap: Tuples
3. Tuple: Formation
4. Tuples: Selection
5. Tuples: Equality
6. Tuples: Equality errors
7. Lists: Recap
8. Lists: Append
9. *cons* vs. @
10. Lists of Functions
11. Lists of Functions
12. Arithmetic Sequences
13. Tail Recursion
14. Tail Recursion Invariant
15. Tail Recursion
16. Another Tail Recursion: *AS3*
17. Another Tail Recursion: *AS3_iter*

18. AS3: Complexity
19. Generating Primes: 2
20. *primes2Upto(n)*
21. *generate2From(P, m, n, k)*
22. *generate2From*
23. *prime2WRT(m, P)*
24. *prime2WRT*
25. *primes2: Complexity*
26. *primes2: Diagnosis*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 431 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 432 of 709

Go Back

Full Screen

Close

Quit

Compound Data

- Forming (compound) data structures from simpler ones
- Breaking up compound data into its components.



Recap: Tuples

formation : Cartesian products of types

selection : Selection of individual components

equality : Equality checking

equality errors : Equality errors

forward to Lists

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 433 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 434 of 709

Go Back

Full Screen

Close

Quit

Tuple: Formation

Standard ML of New Jersey,

```
- val a = ("arun", 1<2, 2);
```

```
val a = ("arun", true, 2)
```

```
      : string * bool * int
```

```
- val b = ("arun", true, 2);
```

```
val b = ("arun", true, 2)
```

```
      : string * bool * int
```

back

Tuples: Selection

```
- #2 a;
```

```
val it = true : bool
```

```
- #1 a;
```

```
val it = "arun" : string
```

```
- #3 a;
```

```
val it = 2 : int
```

```
- #4 a;
```

```
stdin:1.1-1.5 Error: operator and operand
```

```
operator domain: {4:'Y; 'Z}
```

```
operand: string * bool * int
```

```
in expression:
```

```
(fn {4=4, ...} => 4) a
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 435 of 769

Go Back

Full Screen

Close

Quit

back



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 436 of 709

Go Back

Full Screen

Close

Quit

Tuples: Equality

```
- a = b;
```

```
val it = true : bool
```

```
- (1 < 2, true) = (1.0 < 2.0, true);
```

```
val it = true : bool
```

```
- (true, 1.0 < 2.4)  
  = (1.0 < 2.4, true);
```

```
val it = true : bool
```

back

Tuples: Equality errors

```
- ("arun", (1, true))
```

```
= ("arun", 1, true);
```

```
stdin:1.1-29.39 Error: operator and operand
```

```
operator domain: (string * (int * k
```

```
operand: (string * (int * k
```

```
in expression:
```

```
 ("arun", (1, true))
```

```
= ("arun", 1, true)
```

```
- ("arun", (1, true))
```

```
= (("arun", 1), true);
```

```
stdin:1.1-29.39 Error: operator and operand
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 437 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 438 of 709

Go Back

Full Screen

Close

Quit

Lists: Recap

formation : Sequence α List

selection : Selection of individual components

new lists : Making new lists from old

Lists: Append

```
- op @;  
val it = fn : 'a list * 'a list  
          -> 'a list  
- [1,2,3] @ [~1, ~3];  
val it = [1,2,3,~1,~3]  
          : int list  
- [[1,2,3], [~1, ~2]]  
@ [[1,2,3], [~1, ~2]];  
val it =  
[[1,2,3], [~1,~2],  
 [1,2,3], [~1,~2]]  
: int list list
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 439 of 709

Go Back

Full Screen

Close

Quit



cons VS. @

cons is a constant time = $O(1)$ operation. But @ is linear = $O(n)$ in the length n of the first list. @ is defined as

$$L@M =$$

$$\begin{cases} M & \text{if } L = \text{nil} \\ h :: (T@M) & \text{if } L = h :: T \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 441 of 709

Go Back

Full Screen

Close

Quit

Lists of Functions

```
- fun add1 x = x+1;  
val add1 = fn : int -> int  
- fun add2 x = x + 2;  
val add2 = fn : int -> int  
- fun add3 x = x + 3;  
val add3 = fn : int -> int
```



Lists of Functions

```
- val addthree
= [add1, add2, add3];
val addthree
= [fn, fn, fn] : (int -> int) list
- fun addall x = [(add1 x), (add2 x)]
val addall = fn : int -> int list
- addall 3;
val it = [4, 5, 6] : int list
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 442 of 709

Go Back

Full Screen

Close

Quit

Arithmetic Sequences

$$AS1(a, d, n) =$$

$$\left\{ \begin{array}{ll} \square & \text{if } n \leq 0 \\ AS1(a, d, n - 1) & \text{else} \\ @ [a + (n - 1) * d] & \end{array} \right.$$

function	calls	Order
$AS1$	n	$O(n)$
@	n	$O(n)$
::	$\sum_{i=0}^n i$	$O(n^2)$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 443 of 709

Go Back

Full Screen

Close

Quit



Tail Recursion

$AS2(a, d, n) =$

$$\begin{cases} [] & \text{if } n \leq 0 \\ AS2_iter(a, d, n - 1, 0, []) & \text{else} \end{cases}$$

where

for any initial L_0 and $n \geq k \geq 0$

$INV2 : L = L_0 @ [a] @ \dots @ [a + (k - 1) * d]$

Tail Recursion: Invariant

$$INV2 : L = L_0 @ [a] @ \dots @ [a + (k - 1) * d]$$

and bound function

$$n - k$$

$AS2_iter(a, d, n, k, L) =$

$$\begin{cases} L & \text{if } k \geq n \\ AS2_iter(a, d, n, k + 1, L @ [a + k * d]) & \text{else} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 445 of 709

Go Back

Full Screen

Close

Quit



Tail Recursion: Complexity

function	calls	Order
<i>AS2</i>	1	
<i>AS2_iter</i>	n	$O(n)$
@	n	$O(n)$
::	$\sum_{i=0}^n i$	$O(n^2)$

So tail recursion simply doesn't help!

Another Tail Recursion

$AS3$

$$AS3(a, d, n) =$$

$$\begin{cases} [] & \text{if } n \leq 0 \\ AS3_iter(a, d, n - 1, []) & \text{else} \end{cases}$$

where

for any initial L_0 , $n_0 \geq n > 0$, and

$$INV3 : L = (a + (n - 1) * d) :: \dots ::$$

$$\dots :: (a + (n_0 - 1) * d) :: L_0$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 447 of 709

Go Back

Full Screen

Close

Quit

Another Tail Recursion: *AS3_iter*

$$INV3 : L = (a + (n - 1) * d) :: \dots$$

$$\dots :: (a + (n_0 - 1) * d) :: L_0$$

and bound function n ,

$$AS3_iter(a, d, n, L) =$$

$$\begin{cases} L & \text{if } n \leq 0 \\ AS3_iter(a, d, n - 1, & \text{else} \\ (a + (n - 1) * d) :: L) \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 448 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 449 of 709

Go Back

Full Screen

Close

Quit

AS3: Complexity

function	calls	Order
<i>AS3</i>	1	
<i>AS3_iter</i>	n	$O(n)$
@	0	
::	$\sum_{i=0}^n 1$	$O(n)$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 450 of 709

Go Back

Full Screen

Close

Quit

Generating Primes: 2

composites

- *primesUpto*
- *invariant*
- *generateFrom*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 451 of 709

Go Back

Full Screen

Close

Quit

$primes2Upto(n)$

$primes2Upto(n) =$

$$\left\{ \begin{array}{ll} [] & \text{if } n < 2 \\ [(1, 2)] & \text{if } n = 2 \\ primes2Upto(n - 1) & \text{elseif } even(n) \\ generate2From & \text{otherwise} \\ (([1, 2]), 3, n, 2) & \end{array} \right.$$

where



generate2From(P, m, n, k)
bound function $n - m$

Invariant

$$2 < m \leq n \wedge \text{odd}(m)$$

implies

$$P = [(1, p_1), \dots, (k - 1, p_{k-1})]$$

and

$$\forall q : p_{k-1} < q < m : \text{composite}(q)$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 452 of 709

Go Back

Full Screen

Close

Quit



generate2From

generate2From(P, m, n, k) =

$\left\{ \begin{array}{ll} P & \text{if } m > n \\ \text{generate2From} & \text{elseif} \\ (P@[k, m], m + 2, n, k + 1) \text{ } pwr & \\ \text{generate2From} & \text{else} \\ (P, m + 2, n, k) & \end{array} \right.$

where $pwr = \text{prime2WRT}(m, P)$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 453 of 709

Go Back

Full Screen

Close

Quit



prime2WRT(m, P)

bound function *length(P)*

Invariant If $P = [(i, p_i), \dots, (k - 1, p_{k-1})]$,
for some $i \geq 1$ then

- $p_k \geq m > p_{k-1}$, and
- m is prime with respect to $[(1, p_1), \dots, (i - 1, p_{i-1})]$
- m is a prime iff it is a prime with respect to $[(1, p_1), \dots, (j, p_j)]$, where $p_j \leq \lfloor \sqrt{m} \rfloor < p_{j+1}$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 454 of 709

Go Back

Full Screen

Close

Quit



prime2WRT

prime2WRT(m, P) =

$$\left\{ \begin{array}{ll} \textit{true} & \text{if } P = \textit{nil} \\ \textit{true} & \text{if } h > m \text{ div } h \\ \textit{false} & \text{elseif } h|m \\ \textit{primeWRT} & \text{else} \\ (m, \textit{tl}(P)) & \end{array} \right.$$

where

$$(i, h) = \textit{hd}(P)$$

for some $i > 0$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 455 of 709

Go Back

Full Screen

Close

Quit



primes2: Complexity

function	calls
<i>primes2Upto</i>	1
<i>generate2From</i>	$n/2$
<i>prime2WRT</i>	$\sum_{m=3, \text{odd}}^n \pi(\lfloor \sqrt{m} \rfloor)$

primes2: Diagnosis

generate2From

- Uses @ to create an ascending sequence of primes
- For each new prime p_k this operation takes time $O(k)$.
- Can tail recursion be used to reduce the complexity due to @?
- Can a more efficient algorithm using :: instead of @ be devised (as in the case of AS3)?



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 457 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 458 of 709

Go Back

Full Screen

Close

Quit

5.3. Compound Data & List Algorithms

1. Compound Data: Summary
2. Records: Constructors
3. Records: Example 1
4. Records: Example 2
5. Records: Destructors
6. Records: Equality
7. Tuples & Records
8. Back to Lists
9. Lists: Correctness
10. Lists: Case Analysis
11. Lists: Correctness by Cases
12. List-functions: *length*
13. List Functions: *search*
14. List Functions: *search2*
15. List Functions: *ordered*
16. List Functions: *insert*
17. List Functions: *reverse*

18. List Functions: *reverse2*
19. List Functions:*merge*
20. List Functions:*merge*
21. List Functions:*merge contd.*
22. ML: *merge*
23. Sorting by Insertion
24. Sorting by Merging
25. Sorting by Merging
26. Functions as Data
27. Higher Order Functions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 459 of 709

Go Back

Full Screen

Close

Quit

Compound Data: Summary

- Compound Data:

Tuples: Cartesian products of different types (ordered)

Lists: Sequences of the same type of element

Records: Unordered named aggregations of elements of different types.

- Constructors & Destructors



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 460 of 709

Go Back

Full Screen

Close

Quit

Records: Constructors

- A **record** is a **set** of values drawn from various types such that each component (called a **field**) has a unique **name**.
- Each record has a type defined by **field names**
types of fieldnames
The order of presentation of the record fields does not affect its type in any way.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 461 of 709

Go Back

Full Screen

Close

Quit

Records: Example 1

Standard ML of New Jersey,

```
- val pinky =  
{ name = "Pinky",    age = 3,  
  fav_colour = "pink"};  
- val pinky = {age=3,  
  fav_colour="pink",  
  name="Pinky"}  
: {age:int,  
  fav_colour:string,  
  name:string  
}
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 462 of 709

Go Back

Full Screen

Close

Quit

Records: Example 2

```
- val billu =  
{ age = 1,  
  name = "Billu",  
  fav_colour = "blue"  
};  
  
- val billu =  
{age=1,fav_colour="blue",name="Billu"  
{age:int, fav_colour:string, name:string  
- pinky = billu;  
val it = false : bool
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 463 of 209

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 464 of 709

Go Back

Full Screen

Close

Quit

Records: Destructors

```
#age billu;  
val it = 1 : int  
- #fav_colour billu;  
val it = "blue" : string  
- #name billu;  
val it = "Billu" : string
```


Records: Equality

```
- val pinky2 =  
{ name = "Pinky",  
  fav_colour = "pink",  
  age = 3  
};  
  
- val pinky2 =  
{age=3, fav_colour="pink", name="Pinky"}  
{age:int, fav_colour:string, name:string}  
- pinky = pinky2;  
val it = true : bool
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 465 of 769

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 466 of 709

Go Back

Full Screen

Close

Quit

Tuples & Records

- A k -tuple may be thought of as a record whose fields are numbered #1 to #k instead of having names.
- A record may be thought of as a generalization of tuples whose components are named rather than being numbered.

back



Back to Lists

- Every $L : \alpha \text{ List}$ satisfies

$$L = []$$

XOR

$$L = hd(L) :: tl(L)$$

- Many functions on lists (L) are defined by induction on its length ($|L|$).

$$f(L) = \begin{cases} c & \text{if } L = [] \\ g(h, T) & \text{if } L = h :: T \end{cases}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 467 of 709

Go Back

Full Screen

Close

Quit



Lists: Correctness

Hence their properties (P) are proved by induction on the length of the list.

Basis $|L| = 0$. Prove $P(\boxed{[]})$

Induction hypothesis (IH) Assume for some $|T| = n > 0$, $P(\boxed{T})$ holds.

Induction Step Prove $P(\boxed{h :: T})$ for $L = h :: T$ with $|L| = n + 1$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 468 of 709

Go Back

Full Screen

Close

Quit

Lists: Case Analysis

inductive defns on lists

- Every list has **exactly one** of the following **forms** (**patterns**)

– `[]`

– `h::T`

- ML provides convenient case analysis based on **patterns**.

```
fun f [] = c
  | f (h::T) = g (h, T)
;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 469 of 709

Go Back

Full Screen

Close

Quit

Lists: Correctness by Cases

Lists-correctness

P is proved by case analysis.

Basis Prove

$$P([])$$

Induction hypothesis (IH) Assume

$$P(T)$$

Induction Step Prove

$$P(h :: T)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 470 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 471 of 709

Go Back

Full Screen

Close

Quit

List-functions: *length*

$$\begin{cases} \text{length } [] & = 0 \\ \text{length } (h :: T) & = 1 + (\text{length } T) \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 472 of 709

Go Back

Full Screen

Close

Quit

List Functions: *search*

To determine whether x occurs in a list L

$$\left\{ \begin{array}{l} \mathit{search}(x, []) = \mathit{false} \\ \mathit{search}(x, h :: T) = \mathit{true} \text{ if } x = h \\ \mathit{search}(x, h :: T) = \mathit{search}(x, T) \text{ else} \end{array} \right.$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 473 of 709

Go Back

Full Screen

Close

Quit

List Functions: *search2*

Or even more conveniently

$$\left\{ \begin{array}{l} \mathit{search2}(x, []) = \mathit{false} \\ \mathit{search2}(x, h :: T) = (x = h) \text{ or } \\ \mathit{search2}(x, T) \end{array} \right.$$

Time Complexity??



List Functions: *ordered*

Definition 14 A list $L = [a_0, \dots, a_{n-1}]$ is *ordered* by a relation \leq if consecutive elements are related by \leq , i.e. $a_i \leq a_{i+1}$, for $0 \leq i < n - 1$.

$\left\{ \begin{array}{l} \textit{ordered} [] \\ \textit{ordered} [h] \\ \textit{ordered} (h_0 :: h_1 :: T) \text{ if } h_0 \leq h_1 \text{ and } \\ \textit{ordered}(h_1 :: T) \end{array} \right.$

Time Complexity??

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 474 of 709

Go Back

Full Screen

Close

Quit

List Functions: *insert*

Given an **ordered** list $L : \alpha \text{ List}$, insert an element $x : \alpha$ at an appropriate position

$$\left\{ \begin{array}{l} \text{insert}(x, []) = [x] \\ \text{insert}(x, h :: T) = x :: (h :: T) \\ \quad \text{if } x \leq h \\ \text{insert}(x, h :: T) = h :: (\text{insert}(x, T)) \\ \quad \text{else} \end{array} \right.$$

Time Complexity??



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 475 of 709

Go Back

Full Screen

Close

Quit



List Functions: *reverse*

Reverse the elements of a list $L = [a_0, \dots, a_{n-1}]$ to obtain $M = [a_{n-1}, \dots, a_0]$.

$$\begin{cases} \text{reverse } [] & = [] \\ \text{reverse } (h :: T) & = (\text{reverse } T)@[h] \end{cases}$$

Time Complexity?? $O(n^2)$



List Functions: *reverse2*

$$\begin{cases} \text{reverse } [] & = [] \\ \text{reverse } (h :: T) & = \text{rev } ((h :: T), []) \end{cases}$$

where

$$\begin{cases} \text{rev } ([], N) & = N \\ \text{rev } (h :: T, N) & = \text{rev } (T, h :: N) \end{cases}$$

Correctness ??
Time Complexity?? $O(n)$



List Functions: *merge*

Merge two **ordered** lists $|L| = l$, $|M| = m$ to produce an **ordered** list $|N| = l + m$ containing exactly the elements of L and M . That is if

$L = [1, 3, 5, 9, 11]$ and

$M = [0, 3, 4, 4, 10]$, then

$merge(L, M) = N$, where

$N = [0, 1, 3, 3, 4, 4, 5, 9, 10, 11]$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 478 of 709

Go Back

Full Screen

Close

Quit

List Functions: *merge*

$$\left\{ \begin{array}{l} \mathit{merge}([], M) = M \\ \mathit{merge}(L, []) = L \\ \mathit{merge}(L, M) = \\ a :: (\mathit{merge}(S, M)) \quad \text{if } a \leq b \\ \mathit{merge}(L, M) = \\ b :: (\mathit{merge}(L, T)) \quad \text{else} \end{array} \right.$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 479 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 480 of 709

Go Back

Full Screen

Close

Quit

List Functions: *merge* *contd.*

where

$$\begin{cases} L = a :: S \\ M = b :: T \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 481 of 709

Go Back

Full Screen

Close

Quit

ML: *merge*

```
fun merge ([], M) = M
  | merge (L, []) = L
  | merge (L as a::S,
          M as b::T) =
    if a <= b
    then a::merge(S, M)
    else b::merge(L, T)
```

Sorting by Insertion

Given a list of elements to **reorder** them (i.e. with the same number of occurrences of each element as in the original list) to produce a new ordered list.

Hence $sort[10, 8, 3, 6, 9, 7, 4, 8, 1] = [1, 3, 4, 6, 7, 8, 8, 9, 10]$

$$\begin{cases} isort[] = [] \\ isort(h :: T) = insert(h, (isort T)) \end{cases}$$

Time Complexity??



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 482 of 709

Go Back

Full Screen

Close

Quit



Sorting by Merging

$$\left\{ \begin{array}{l} \mathit{msort} [] = [] \\ \mathit{msort} [a] = [a] \\ \mathit{msort} L = \mathit{merge} ((\mathit{msort} M), \\ (\mathit{msort} N)) \end{array} \right.$$

where

$$(M, N) = \mathit{split} L$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 483 of 709

Go Back

Full Screen

Close

Quit



Sorting by Merging

where

$$\begin{cases} \textit{split} [] & = ([], []) \\ \textit{split} [a] & = ([a], []) \\ \textit{split} (a :: b :: P) & = (a :: \textit{Left}, b :: \textit{Right}) \end{cases}$$

where

$$(\textit{Left}, \textit{Right}) = \textit{split} P$$

Time Complexity??

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 484 of 709

Go Back

Full Screen

Close

Quit



Functions as Data

list of functions

- Every **function** is **unary**. A function of many arguments may be thought of as a function of a single argument i.e. a tuple of appropriate type.
- Every **function** is a **value** of an appropriate type.
- Hence **functions** are also **data**.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 485 of 709

Go Back

Full Screen

Close

Quit



Higher Order Functions

Compound data may be constructed from functions as values using the constructors of the compound data structure.

Functions may be defined with other functions and/or data as arguments to produce new values or new functions.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 486 of 709

Go Back

Full Screen

Close

Quit

6. Higher Order Functions & Structured Data

6.1. Higher Order Functions

1. Summary: Compound Data
2. List: Examples
3. Lists: Sorting
4. Higher Order Functions
5. An Example
6. Currying
7. Currying: Contd
8. Generalization
9. Generalization: 2
10. Applying a list
11. Trying it out
12. Associativity
13. Apply to a list
14. Sequences
15. Further Generalization
16. Further Generalization
17. Sequences



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 487 of 709

Go Back

Full Screen

Close

Quit

18. Efficient Generalization
19. Sequences: 2
20. More Generalizations
21. More Summations
22. Or Maybe . . . Products
23. Or Some Other \otimes
24. Other \otimes
25. Examples of \otimes , e



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 488 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 489 of 709

Go Back

Full Screen

Close

Quit

Summary: Compound Data

- Records and tuples
- Lists
 - Correctness
 - Examples



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 490 of 709

Go Back

Full Screen

Close

Quit

List: Examples

- Length of a list
- Searching a list
- Checking whether a list is ordered
- Reversing a list
- Sorting of lists



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 491 of 709

Go Back

Full Screen

Close

Quit

Lists: Sorting

- Sorting by insertion
- Sorting by Divide-and-Conquer



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 492 of 709

Go Back

Full Screen

Close

Quit

Higher Order Functions

- Functions as data
- Higher order functions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 493 of 709

Go Back

Full Screen

Close

Quit

An Example

List of functions

- $add1\ x = x + 1$

- $add2\ x = x + 2$

- $add3\ x = x + 3$

Suppose we needed to define a long list of length n , where the i -th element is the function that adds $i + 1$ to the argument.

Currying

$$\text{addc } y \ x = x + y$$

ML's response :

```
val addc = fn :  
           int -> (int -> int)
```

Contrast with ML's response

```
- op +;  
val it = fn : int * int -> int
```

addc is the **curried** version of the binary operation **+**.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 494 of 709

Go Back

Full Screen

Close

Quit

Currying: Contd

$$f : (\alpha * \beta * \gamma) \rightarrow \delta \checkmark$$

$$f_c : \alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta \checkmark$$

$$f_c' : (\alpha * \beta) \rightarrow \gamma \rightarrow \delta \checkmark$$

$$f_c'' : \alpha \rightarrow (\beta * \gamma) \rightarrow \delta \checkmark$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 495 of 709

Go Back

Full Screen

Close

Quit



Generalization

Then

- $addc1 = (addc\ 1): int \rightarrow int$
- $addc2 = (addc\ 2): int \rightarrow int$
- $addc3 = (addc\ 3): int \rightarrow int$

and for any i ,

$$(addc\ i): int \rightarrow int$$

is the required function.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 497 of 709

Go Back

Full Screen

Close

Quit

Generalization: 2

list_adds $n =$

$$\begin{cases} [] & \text{if } n \leq 0 \\ (list_adds(n - 1)) @ [(addc\ n)] & \text{else} \end{cases}$$

ML's response :

```
val list_adds = fn :  
int -> (int -> int) list
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 498 of 709

Go Back

Full Screen

Close

Quit

Applying a list

addall

$$\begin{cases} \text{applyl } [] x & = [] \\ \text{applyl } (h :: T) x & = (h x) :: (\text{applyl } T x) \end{cases}$$

ML's response:

```
val applyl = fn :  
('a -> 'b) list ->  
'a -> 'b list
```



Trying it out

interval x n = applyl x (list_adds n)

ML's response:

```
val interval = fn :  
    int -> int -> int list  
- interval 53 5;  
val it = [54,55,56,57,58]  
: int list
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 499 of 709

Go Back

Full Screen

Close

Quit



Associativity

- Application associates to the left.

$$f \ x \ y = ((f \ x) \ y)$$

- \rightarrow associates to the right.

$$\alpha \rightarrow \beta \rightarrow \gamma = \alpha \rightarrow (\beta \rightarrow \gamma)$$

If $f : \alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta$

then $f \ a : \beta \rightarrow \gamma \rightarrow \delta$

and $f \ a \ b : \gamma \rightarrow \delta$

and $f \ a \ b \ c : \delta$

Apply to a list

Apply a list Transpose of a matrix

$$\begin{cases} \text{map } f [] & = [] \\ \text{map } f (h :: T) & = (f h) :: (\text{map } f T) \end{cases}$$

```
val it = fn : ('a -> 'b) ->
          'a list -> 'b list
- map addc3 [4, 6, ~1, 0];
val it = [7, 9, 2, 3] : int list
- map real [7, 9, 2, 3];
val it = [7.0, 9.0, 2.0, 3.0]
: real list
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 501 of 709

Go Back

Full Screen

Close

Quit



Sequences

Arithmetic sequences-1

Arithmetic sequences-2

Arithmetic sequences-3

$$AS4(a, d, n) =$$

$$\begin{cases} [] & \text{if } n \leq 0 \\ a :: (\text{map } (\text{addc } d) \\ (AS4(a, d, (n - 1)))) & \text{else} \end{cases}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 502 of 709

Go Back

Full Screen

Close

Quit

Further Generalization

Given

$$f : \alpha * \alpha \rightarrow \alpha$$

Then

$$\text{curry2 } f \ x \ y = f(x, y)$$

and

$$(\text{curry2 } f) : \alpha \rightarrow (\alpha \rightarrow \alpha)$$

and for any $d : \alpha$,

$$((\text{curry2 } f) \ d) : \alpha \rightarrow \alpha$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 503 of 709

Go Back

Full Screen

Close

Quit



Further Generalization

$seq(f, a, d, n) =$

$$\begin{cases} [] & \text{if } n \leq 0 \\ a :: (map ((curry2 f) d) \\ (seq (f, a, d, n - 1))) & \text{else} \end{cases}$$

is the sequence of length n generated with $((curry2 f) d)$, starting from a .



Sequences

Arithmetic: $AS5(a, d, n)$ =
 $seq(op+, a, d, n)$

Geometric: $GS1(a, r, n)$ =
 $seq(op*, a, r, n)$

Harmonic: $HS1(a, d, n)$ =
 $map\ reci\ (AS5(a, d, n))$

where
 $reci\ x = 1.0/(real\ x)$ gives the reciprocal of a (non-zero) integer.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 505 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 506 of 709

Go Back

Full Screen

Close

Quit

Efficient Generalization

Let's not use *map* repeatedly.

$seq2(f, g, a, d, n) =$

$$\begin{cases} [] & \text{if } n \leq 0 \\ (f\ a) :: (seq2\ (f, g(a, d), d, n - 1)) & \text{else} \end{cases}$$

is the sequence of length n generated with a unary f , a binary g starting from $f(a)$.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 507 of 709

Go Back

Full Screen

Close

Quit

Sequences: 2

- $AS6(a, d, n) = seq2(id, op+, a, d, n)$
- $GS2(a, r, n) = seq2(id, op*, a, r, n)$
- $HS2(a, d, n) = seq2(reci, op+, a, d, n)$

where $id\ x = x$ is the identity function.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 508 of 709

Go Back

Full Screen

Close

Quit

More Generalizations

Often interested in some particular **measure** related to a sequence, rather than in the sequence itself, e.g. **sum-mations** of

- arithmetic, geometric, harmonic sequences
- e^x , trigonometric functions upto some **n -th** term
- (Truncated) **Taylor** and **Maclaurin** series

More Summations

Wasteful to **first** generate the sequence and **then** compute the measure

$$\sum_{i=l}^u f(i)$$

where the range $[l, u]$ is defined by a unary *succ* function

$$\text{sum}(f, \text{succ}, l, u) =$$

$$\begin{cases} 0 & \text{if } [l, u] = \emptyset \\ f(l) + \text{sum}(f, \text{succ}, \text{succ}(l), u) & \text{else} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 509 of 709

Go Back

Full Screen

Close

Quit



Or Maybe . . . Products

Or may be interested in forming products of sequences.

$$\prod_{i=l}^u f(i)$$

$$\text{prod}(f, \text{succ}, l, u) =$$

$$\begin{cases} 1 & \text{if } [l, u] = \emptyset \\ f(l) * \text{prod}(f, \text{succ}, \text{succ}(l), u) & \text{else} \end{cases}$$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 510 of 709

Go Back

Full Screen

Close

Quit

Or Some Other \otimes

Or some other binary operation \otimes which has the following properties:

- $\otimes : (\alpha * \alpha) \rightarrow \alpha$ is **closed**

- \otimes is **associative** i.e.

$$a \otimes (b \otimes c) = (a \otimes b) \otimes c$$

- \otimes has an **identity** element e i.e

$$a \otimes e = a = e \otimes a$$

$$\bigotimes_{i=l}^u f(l)$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 511 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 512 of 709

Go Back

Full Screen

Close

Quit

Other \otimes

Then if $f, succ : \alpha \rightarrow \alpha$

$ser(\otimes, f, succ, l, u) =$

$$\begin{cases} e & \text{if } [l, u] = \emptyset \\ f(l) \otimes ser(\otimes, f, succ, succ(l), u) & \text{else} \end{cases}$$



Examples of \otimes , e

- `+`, `0` on integers and reals
- concatenation and the empty string on strings
- `and` also, `true` on booleans
- `or` else, `false` on booleans
- `+`, `0` on vectors and matrices
- `*`, `1` on vectors and matrices

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 513 of 709

Go Back

Full Screen

Close

Quit

6.2. Structured Data

1. Transpose of a Matrix
2. Transpose: 0
3. Transpose: 10
4. Transpose: 01
5. Transpose: 20
6. Transpose: 02
7. Transpose: 30
8. Transpose: 03
9. *trans*
10. *is2DMatrix*
11. User Defined Types
12. Enumeration Types
13. User Defined Structural Types
14. Functions vs. data
15. Data as 0-ary Functions
16. Data vs. Functions
17. Data vs. Functions: Recursion
18. Lists



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 514 of 709

Go Back

Full Screen

Close

Quit

19. Constructors
20. Shapes
21. Shapes: Triangle Inequality
22. Shapes: Area
23. Shapes: Area
24. ML: Try out
25. ML: Try out (contd.)
26. Enumeration Types
27. Recursive Data Types
28. Resistors: Datatype
29. Resistors: Equivalent
30. Resistors
31. Resistors: Example
32. Resistors: ML session



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 515 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 516 of 709

Go Back

Full Screen

Close

Quit

Transpose of a Matrix

Map

Assume a 2-D $r \times c$ matrix is represented by a list of lists of elements.

Then

transpose $L =$

$$\begin{cases} \textit{trans } L & \text{if } \textit{is2DMatrix}(L) \\ \perp & \text{else} \end{cases}$$

where

Transpose: 0

$$\begin{bmatrix} [& & & &] \\ [& 11 & 12 & 13 &] \\ [& 21 & 22 & 23 &] \\ [& 31 & 32 & 33 &] \\ [& 41 & 42 & 43 &] \\] & & & & \end{bmatrix}$$


IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 517 of 709

Go Back

Full Screen

Close

Quit

Transpose: 01

$$\begin{bmatrix} [& & & &] \\ [& 12 & 13 & &] \\ [& 22 & 23 & &] \\ [& 32 & 33 & &] \\ [& 42 & 43 & &] \\] \end{bmatrix}$$
$$\begin{bmatrix} [& & & &] \\ [& 11 & 21 & 31 & 41 &] \\ [& & & &] \\ [& & & &] \\ [& & & &] \\] \end{bmatrix}$$


IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 519 of 709

Go Back

Full Screen

Close

Quit

Transpose: 20

$$\begin{bmatrix} [& 12 & 13] \\ [& 22 & 23] \\ [& 32 & 33] \\ [& 42 & 43] \\] \end{bmatrix} \quad \begin{bmatrix} [11 & 21 & 31 & 41] \end{bmatrix}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 520 of 709

Go Back

Full Screen

Close

Quit

Transpose: 02

$$\begin{bmatrix} [& & & &] \\ [& & & & 13] \\ [& & & & 23] \\ [& & & & 33] \\ [& & & & 43] \\] \end{bmatrix}$$
$$\begin{bmatrix} [& & & &] \\ [11 & 21 & 31 & 41 &] \\ [12 & 22 & 32 & 42 &] \\ [& & & &] \\] \end{bmatrix}$$


IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 521 of 709

Go Back

Full Screen

Close

Quit

Transpose: 30

$$\begin{bmatrix} [& & & &] \\ [& & 13 & &] \\ [& & 23 & &] \\ [& & 33 & &] \\ [& & 43 & &] \\] \end{bmatrix}$$
$$\begin{bmatrix} [11 & 21 & 31 & 41] \\ [12 & 22 & 32 & 42] \\] \end{bmatrix}$$


IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 522 of 709

Go Back

Full Screen

Close

Quit

Transpose: 03

$$\begin{bmatrix} [&] \\ [&] \\ [&] \\ [&] \\] \end{bmatrix} \quad \begin{bmatrix} [11 & 21 & 31 & 41] \\ [12 & 22 & 32 & 42] \\ [13 & 23 & 33 & 43] \\] \end{bmatrix}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 523 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 524 of 709

Go Back

Full Screen

Close

Quit

trans

$$\left\{ \begin{array}{l} \mathit{trans} [] = [] \\ \mathit{trans} [] :: TL = [] \\ \mathit{trans} LL = (\mathit{map} \mathit{hd} LL) :: \\ \quad (\mathit{trans} (\mathit{map} \mathit{tl} LL)) \end{array} \right.$$

and

is2DMatrix = #1(*dimensions L*)
where



is2DMatrix

$$\left\{ \begin{array}{l} \text{dimensions } [] = (true, 0, 0) \\ \text{dimensions } [H] = \\ (true, 1, h) \\ \text{dimensions } (H :: TL) = \\ (b \text{ and } (h = c), r + 1, c) \end{array} \right.$$

where *dimensions* $TL = (b, r, c)$
and $h = \text{length } H$

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 525 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 526 of 709

Go Back

Full Screen

Close

Quit

User Defined Types

Many languages allow **user-defined** data types.

- **record types**: **Pinky** and **Billu**
- **Enumerations**: aggregates of **heterogeneous** data.
- other **structural** constructions (if desperate!)

Enumeration Types

Many languages allow **user-defined** data types.

- **record types**: **Pinky** and **Billu**
- Enumerations: aggregates of **heterogeneous** data.
 - days of the week
 - colours
 - geometrical shapes
- other **structural** constructions (if desperate!)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 527 of 709

Go Back

Full Screen

Close

Quit

User Defined Structural Types

Many languages allow **user-defined** data types.

- **record types**: **Pinky** and **Billu**
- **Enumerations**: aggregates of **heterogeneous** data.
- other structural constructions (if desperate!)
 - trees
 - graphs
 - symbolic expressions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 528 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 529 of 709

Go Back

Full Screen

Close

Quit

Functions vs. data

- Inspired by the list constructors, `nil` and `cons`
- Grand Unification of functions and data
 - Functions as data
 - Data as functions



Data as 0-ary Functions

- Every data element may be regarded as a function with 0 arguments

– **Caution:** A constant function

$$f(x) = 5, \text{ for all } x : \alpha$$

where

$$f : \alpha \rightarrow \text{int}$$

is not the same as a value

$$5 : \text{int}$$

. Their types are different.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 530 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 531 of 709

Go Back

Full Screen

Close

Quit

Data vs. Functions

Facilities	Functions	Data
primitive	operations	values
user-defined	functions	constructors
composition	application	alternative
recursion	recursion	recursion



Data vs. Functions: Recursion

Recursion
Basis
naming
composition
induction

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 532 of 709

Go Back

Full Screen

Close

Quit

Lists as Structured Data

```
datatype 'a list =  
    nil |  
    cons of 'a * 'a list
```

Every α *list* is either

`nil`: (**Basis**, **name**)

| : or (**alternative**)

`cons`: constructed **inductively** from
an element of type `'a` and another
list of type `'a list` using the con-
structor `cons`



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 533 of 709

Go Back

Full Screen

Close

Quit

Constructors

- Inspired by the list constructors

nil : α list

cons : $\alpha \times \alpha$ list \rightarrow α list

- combine **heterogeneous** types: α and α list
- allows recursive definition by a form of induction

Basis : *nil*

Induction : *cons*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 534 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 535 of 709

Go Back

Full Screen

Close

Quit

Shapes

A non-recursive data type

`datatype shape =`

`CIRCLE of real`

`| RECTANGLE of real * real`

`| TRIANGLE of`

`real * real * real`



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 536 of 709

Go Back

Full Screen

Close

Quit

Shapes: Triangle Inequality

```
fun isTriangle
  (TRIANGLE (a, b, c)) =
    (a+b>c) andalso
    (b+c>a) andalso
    (c+a>b)
| isTriangle _ = false
```




Shapes: Area

```
exception notShape;
```

```
fun area (CIRCLE (r)) =  
    3.14159 * r * r
```

```
| area (RECTANGLE (l,b)) =  
    l*b
```

```
| area (s as  
    TRIANGLE (a, b, c)) =
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 537 of 709

Go Back

Full Screen

Close

Quit



Shapes: Area

```
if isTriangle (s) then
  let val s = (a+b+c)/2.0
      in Math.sqrt
          (s*(s-a)*(s-b)*(s-c))
end
else raise notShape;
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 538 of 709

Go Back

Full Screen

Close

Quit

ML: Try out

```
- use "shapes.sml";  
[opening shapes.sml]  
datatype shape  
  = CIRCLE of real  
  | RECTANGLE of real * real  
  | TRIANGLE of  
    real * real * real  
val isTriangle =  
  fn : shape -> bool  
exception notShape  
val area = fn : shape -> real
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 539 of 709

Go Back

Full Screen

Close

Quit

ML: Try out (contd.)

```
val it = () : unit
- area (TRIANGLE (2.0, 1.0, 3.0)) ;
uncaught exception notShape
  raised at: shapes.sml:22.17-22.25
- area
  (TRIANGLE (3.0, 4.0, 5.0));
val it = 6.0 : real
-
```

Back to User defined types



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

Page 540 of 709

Go Back

Full Screen

Close

Quit

Enumeration Types

- Enumeration types are non-recursive datatypes with
- 0-ary constructors

```
datatype working = MON | TUE  
                | WED | THU | FRI;  
datatype weekends = SAT | SUN  
datatype weekdays = working  
                  | weekends;
```

[Back to User defined types](#)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 541 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 542 of 709

Go Back

Full Screen

Close

Quit

Recursive Data Types

- But the really interesting types are the **recursive** data types

Back to Lists

- As with lists proofs of correctness on recursive data types depend on a case-analysis of the structure (basis and inductive constructors)

Correctness on lists



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 543 of 709

Go Back

Full Screen

Close

Quit

Resistors: Datatype

```
datatype resist =  
  RES of real |  
  SER of resist * resist |  
  PAR of resist * resist
```

Resistors: Equivalent

```
fun value (RES (r)) = r
  | value (SER (R1, R2)) =
    value (R1) + value (R2)
  | value (PAR (R1, R2)) =
    let val r1 = value (R1);
        val r2 = value (R2)
    in (r1*r2) / (r1+r2)
    end;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 544 of 709

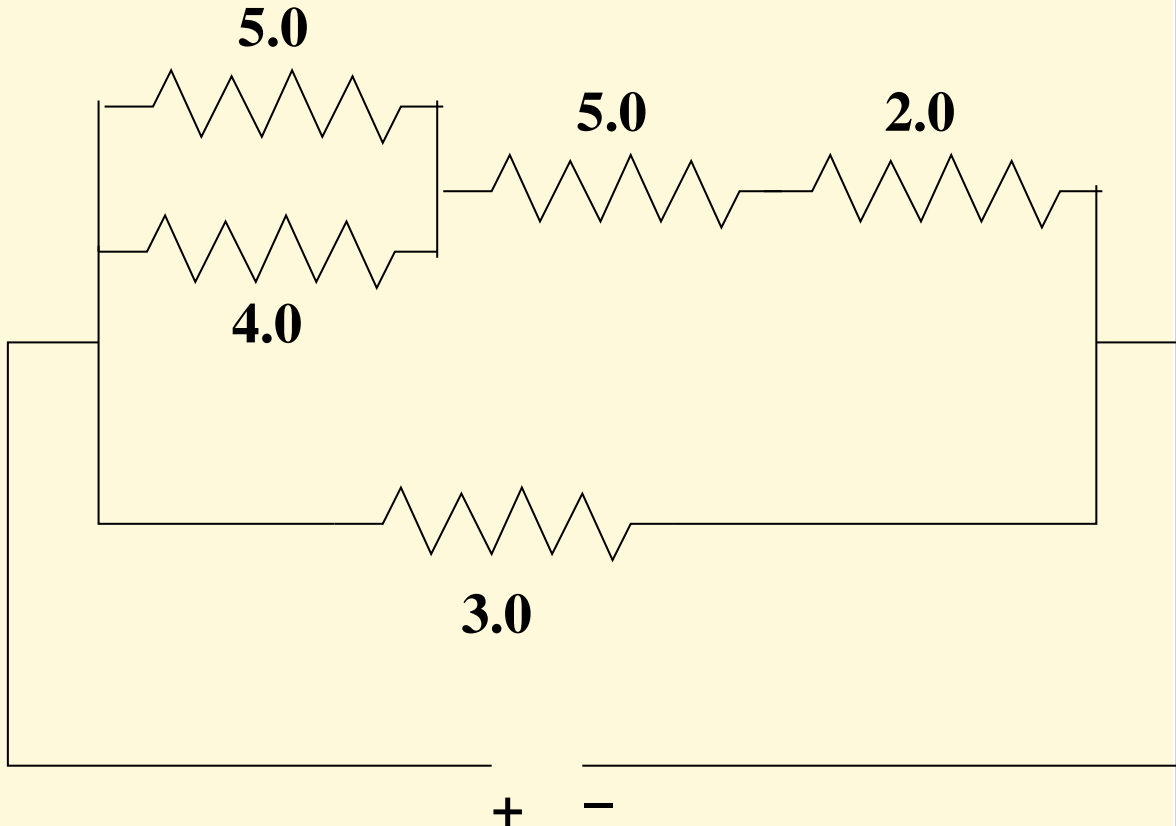
Go Back

Full Screen

Close

Quit

Resistors



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 545 of 709

Go Back

Full Screen

Close

Quit

Resistors: Example

```
val R = PAR (  
    SER (  
        PAR (  
            RES (5.0) ,  
            RES (4.0)  
        ) ,  
        SER (  
            RES (5.0) ,  
            RES (2.0)  
        )  
    ) ,  
    RES (3.0)  
);
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 546 of 709

Go Back

Full Screen

Close

Quit

Resistors: ML session

```
- use "resistors.sml";  
[opening resistors.sml]  
datatype resist = PAR of resist * resist  
                | RES of real  
                | SER of resist * resist  
val value = fn : resist -> real  
val R = PAR (SER (PAR #, SER #), RES 3)  
val it = () : unit  
- value R;  
val it = 2.2636363636364 : real  
-
```

Resistance Expressions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 547 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 548 of 709

Go Back

Full Screen

Close

Quit

6.3. User Defined Structured Data Types

1. User Defined Types
2. Resistors: Grouping
3. Resistors: In Pairs
4. Resistor: Values
5. Resistance Expressions
6. Resistance Expressions
7. Arithmetic Expressions
8. Arithmetic Expressions: 0
9. Arithmetic Expressions: 1
10. Arithmetic Expressions: 2
11. Arithmetic Expressions: 3
12. Arithmetic Expressions: 4
13. Arithmetic Expressions: 5
14. Arithmetic Expressions: 6
15. Arithmetic Expressions: 7
16. Arithmetic Expressions: 8
17. Binary Trees

- 18. Arithmetic Expressions: 0
- 19. Trees: Traversals
- 20. Recursive Data Types: Correctness
- 21. Data Types: Correctness



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 549 of 709

Go Back

Full Screen

Close

Quit



User Defined Types

- Records
- Structural Types
 - Constructors
 - * Non-recursive
 - * Enumeration Types
 - Recursive datatypes
 - * Resistance circuits

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 550 of 709

Go Back

Full Screen

Close

Quit

Resistors: Grouping



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



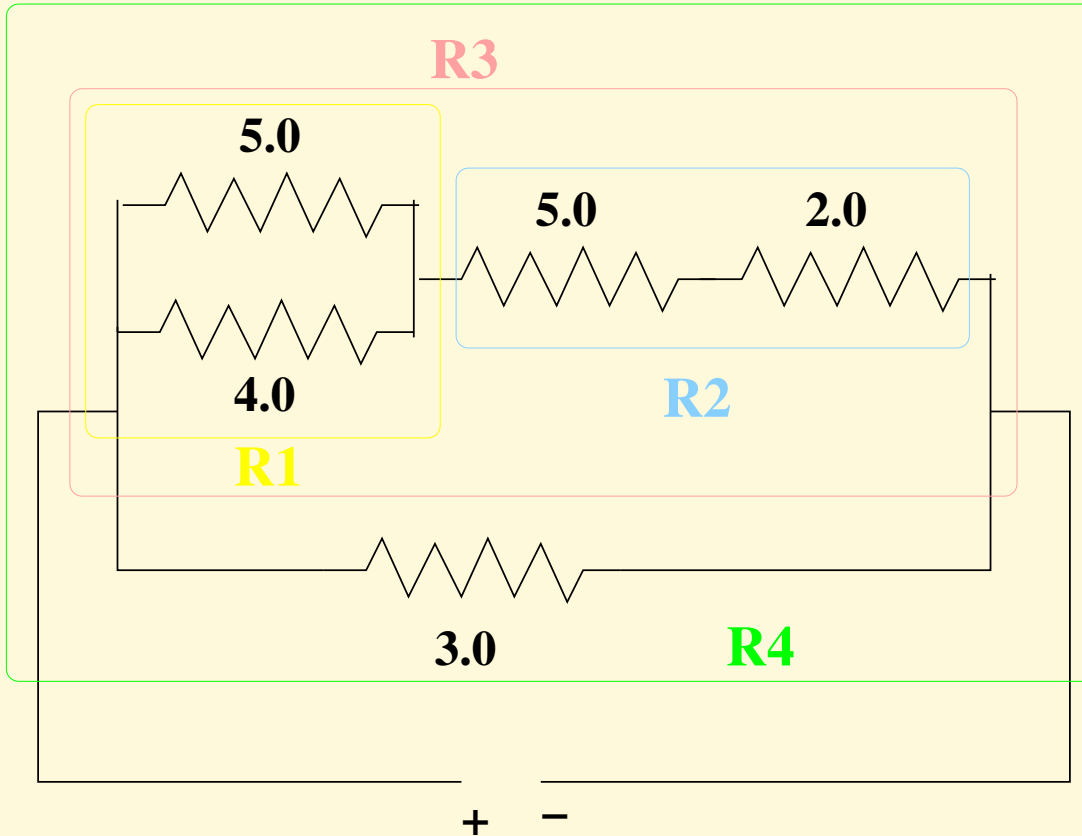
Page 551 of 709

Go Back

Full Screen

Close

Quit





IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 552 of 709

Go Back

Full Screen

Close

Quit

Resistors: In Pairs

```
val R1 = PAR (RES 5.0, RES 4.0)
val R2 = SER (RES 5.0, RES 2.0)
val R3 = SER (R1, R2);
val R4 = PAR (R3, RES (3.0));
```




Resistor: Values

```
- value R1;  
val it = 2.22222222222222 : real  
- value R2;  
val it = 7.0 : real  
- value R3;  
val it = 9.22222222222222 : real  
- value R4;  
val it = 2.2636363636364 : real  
-
```

[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)

[◀](#) [▶](#)

[◀](#) [▶](#)

Page 553 of 709

[Go Back](#)

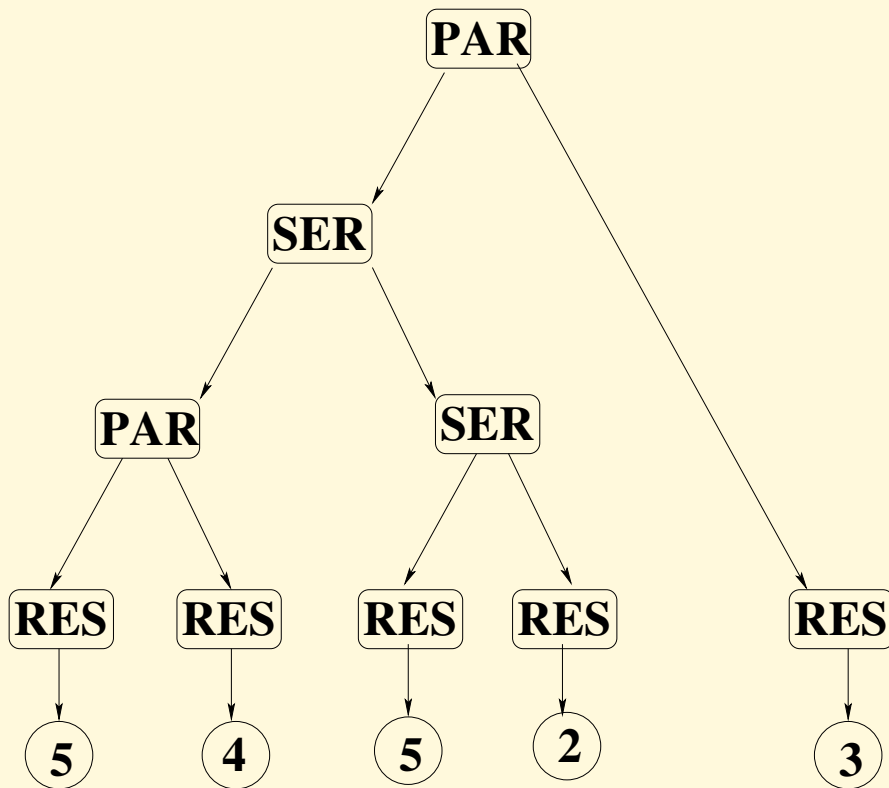
[Full Screen](#)

[Close](#)

[Quit](#)

Resistance Expressions

A resistance expression



Circuit Diagram



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 554 of 709

Go Back

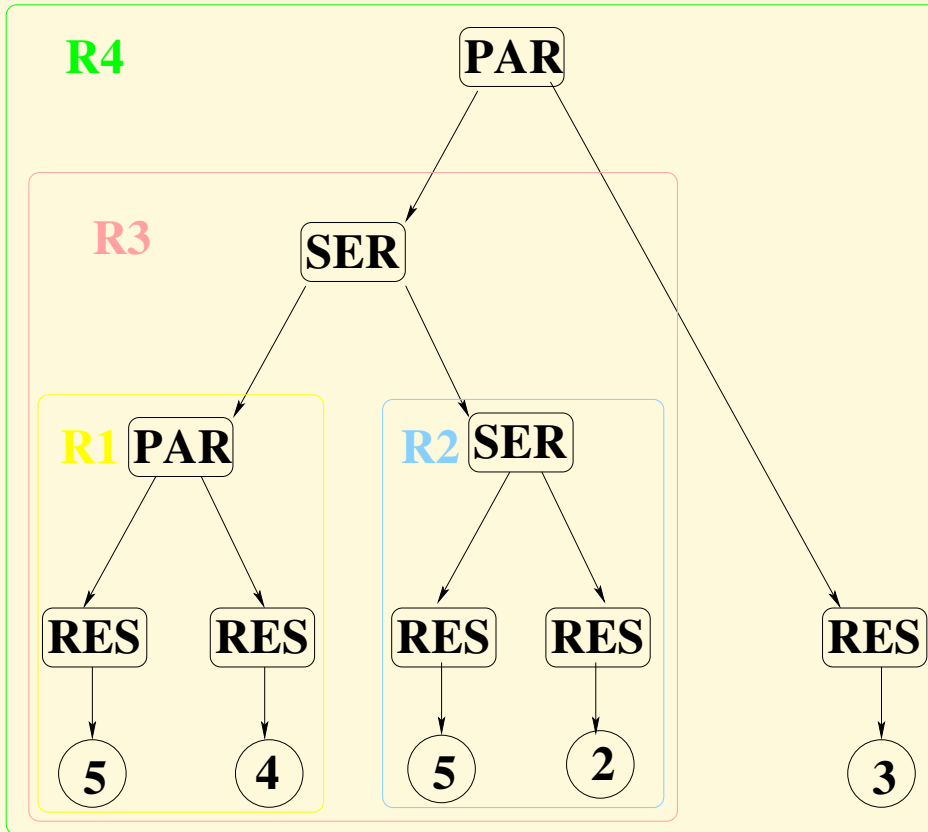
Full Screen

Close

Quit

Resistance Expressions

A resistance expression



Circuit Diagram



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 555 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 556 of 709

Go Back

Full Screen

Close

Quit

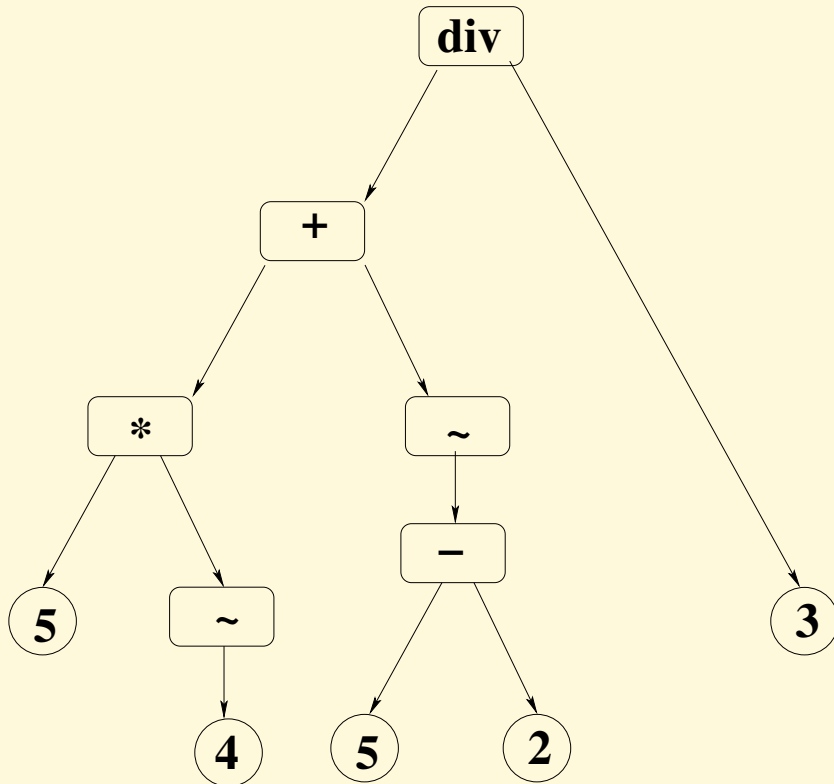
Arithmetic Expressions

ML arithmetic expressions:

$((5 * \sim 4) + \sim (5 - 2)) \text{ div } 3$

are represented as **trees**

Arithmetic Expressions: 0



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 557 of 709

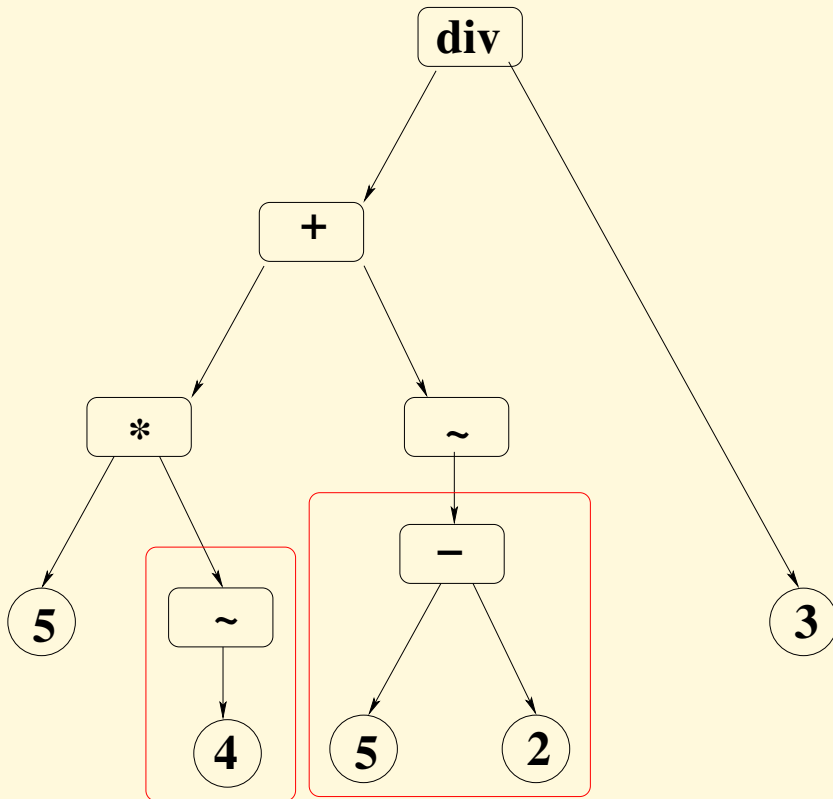
Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 1



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 558 of 709

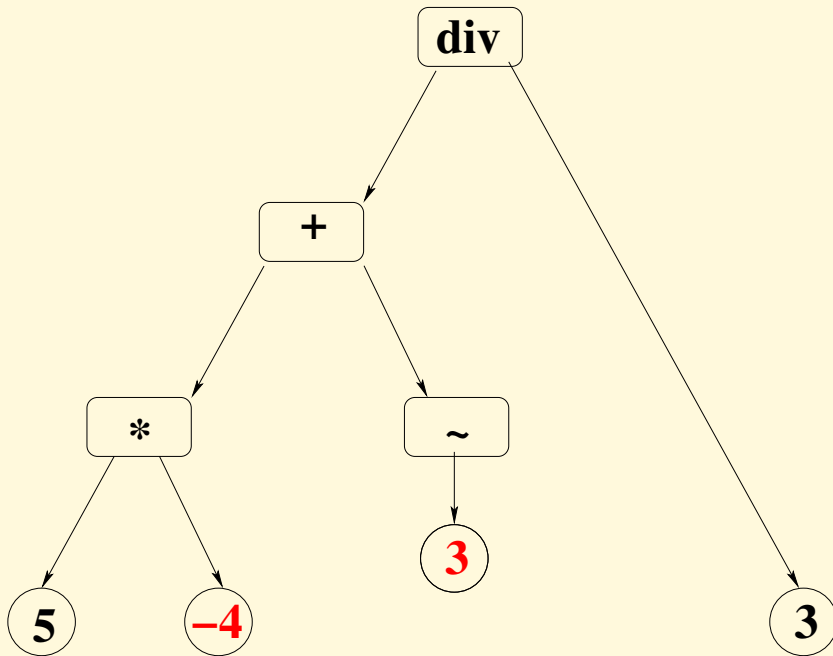
Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 2



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 559 of 709

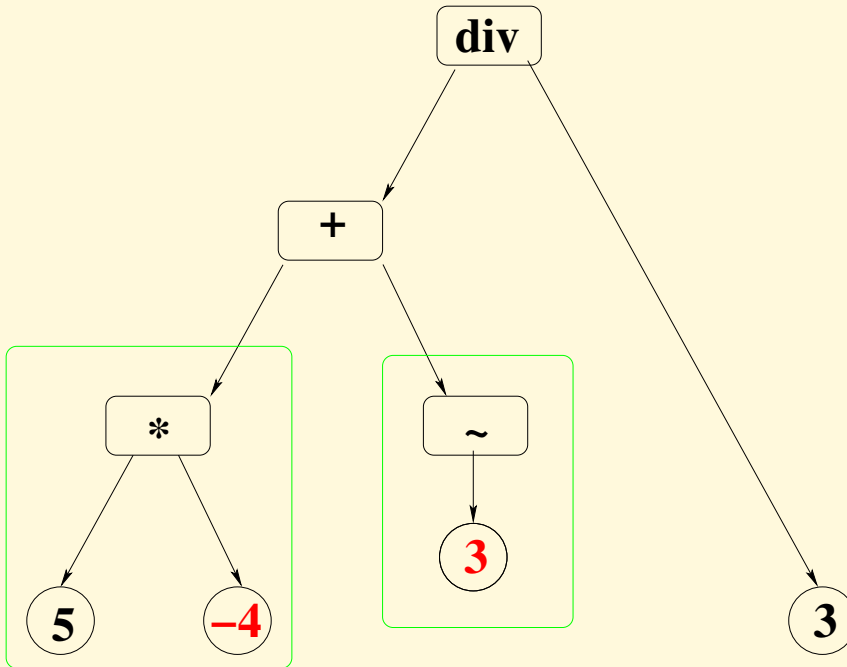
Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 3



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 560 of 709

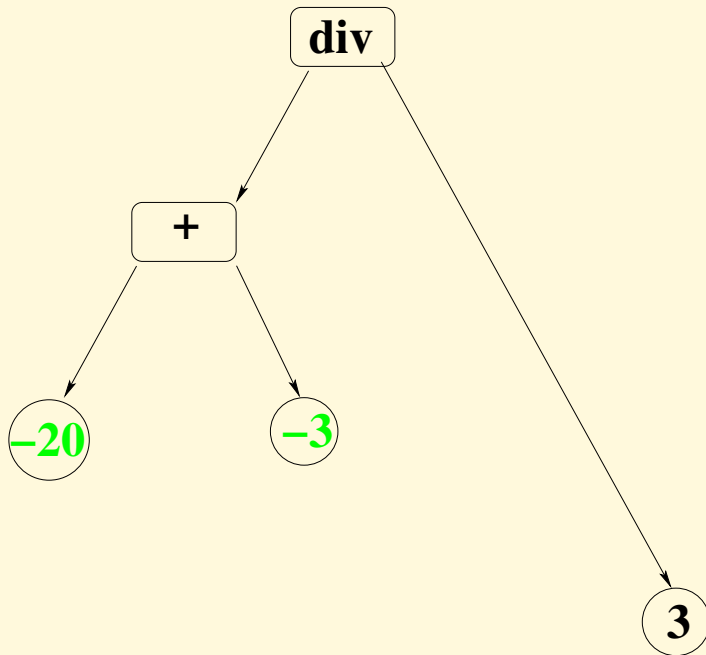
Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 4



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 561 of 709

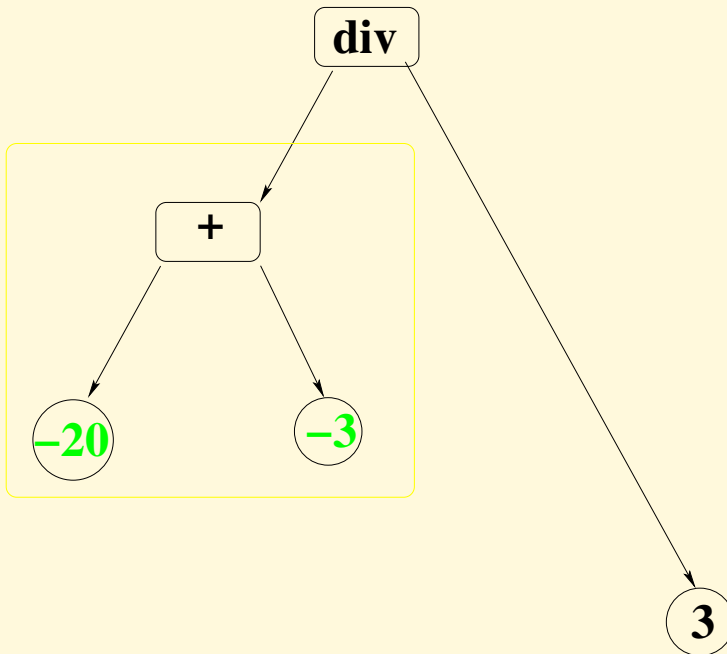
Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 5



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 562 of 709

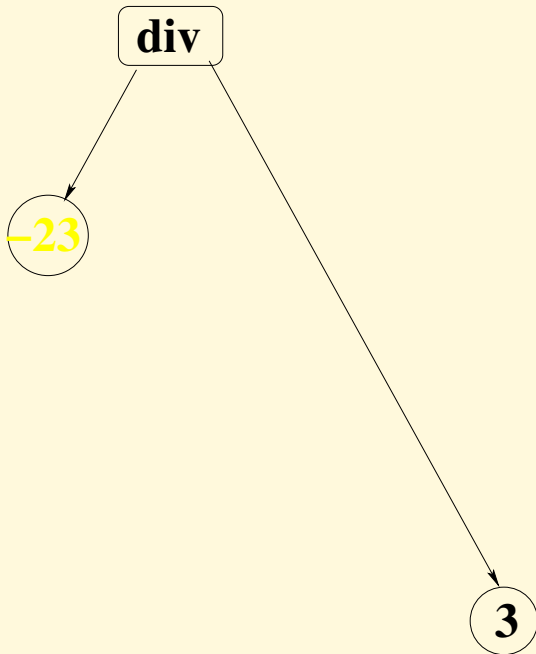
Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 6



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 563 of 709

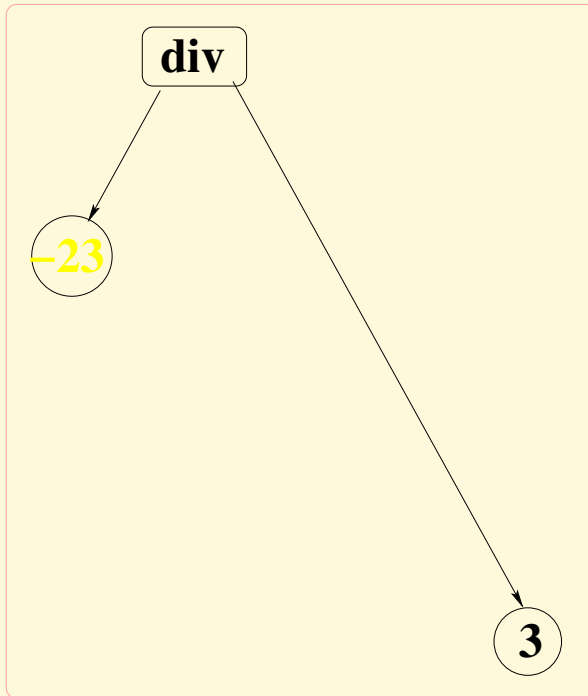
Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 7



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 564 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 565 of 709

Go Back

Full Screen

Close

Quit

Arithmetic Expressions: 8

-8



Binary Trees

```
datatype 'a bintree =  
  Empty |  
  Node of 'a *  
        'a bintree *  
        'a bintree
```

[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)

[◀](#) [▶](#)

[◀](#) [▶](#)

Page 566 of 709

[Go Back](#)

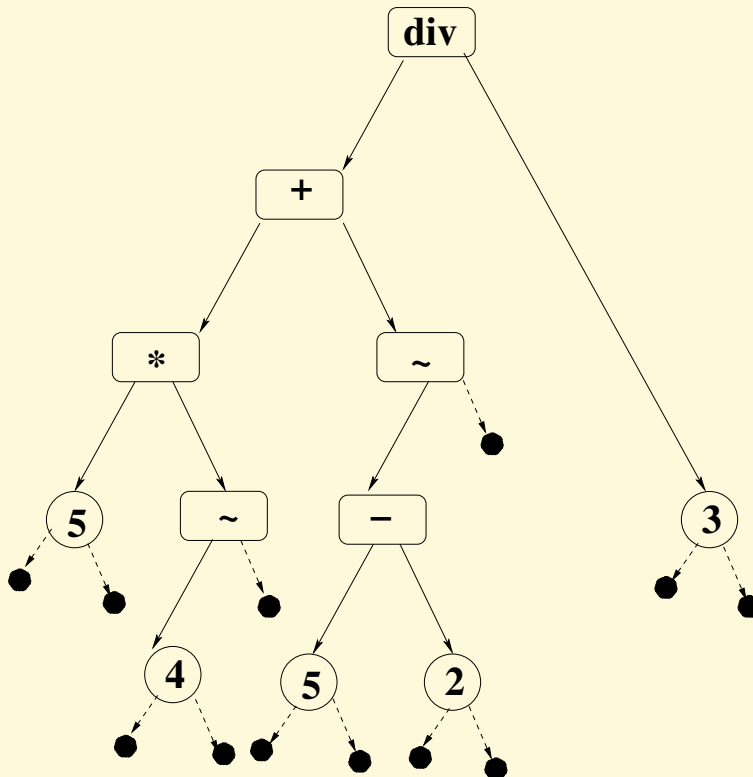
[Full Screen](#)

[Close](#)

[Quit](#)

Arithmetic Expressions: 0

Arithmetic Expressions



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 567 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 568 of 709

Go Back

Full Screen

Close

Quit

Trees: Traversals

- preorder
- inorder
- postorder



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 569 of 709

Go Back

Full Screen

Close

Quit

Recursive Data Types: Correctness

Correctness on lists by cases

P is proved by case analysis.



Data Types: Correctness

Basis Prove $P(c)$ for each non-recursive constructor c

Induction hypothesis (IH) Assume $P(T)$ for all elements of the data type less than a certain depth

Induction Step Prove $P(r(T_1, \dots, T_n))$ for each recursive constructor r

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 570 of 709

Go Back

Full Screen

Close

Quit

7. Imperative Programming: An Introduction

7.1. Introducing a Memory Model

1. Summary: Functional Model
2. CPU & Memory: Simplified
3. Resource Management
4. Shell: User Interface
5. GUI: User Interface
6. Memory Model: Simplified
7. Memory
8. The Imperative Model
9. State Changes: σ
10. State
11. State Changes
12. State Changes: σ
13. State Changes: σ_1
14. State Changes: σ_2
15. Languages
16. User Programs
17. Imperative Languages



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 571 of 709

Go Back

Full Screen

Close

Quit

18. Imperative vs Functional Variables
19. Assignment Commands
20. Assignment Commands
21. Assignment Commands
22. Assignment Commands
23. Assignment Commands
24. Assignment Commands: Swap
25. Swap
26. Swap
27. Swap



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 572 of 709

Go Back

Full Screen

Close

Quit

Summary: Functional Model

- **Stateless** (as is most mathematics)
- Notion of **value** is paramount
 - Integers, reals, booleans, strings and characters are all **values**
 - Every **function** is also a **value**
 - Every complex piece of data is also a **value**
- No concept of **storage** (except for space complexity calculations)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 573 of 709

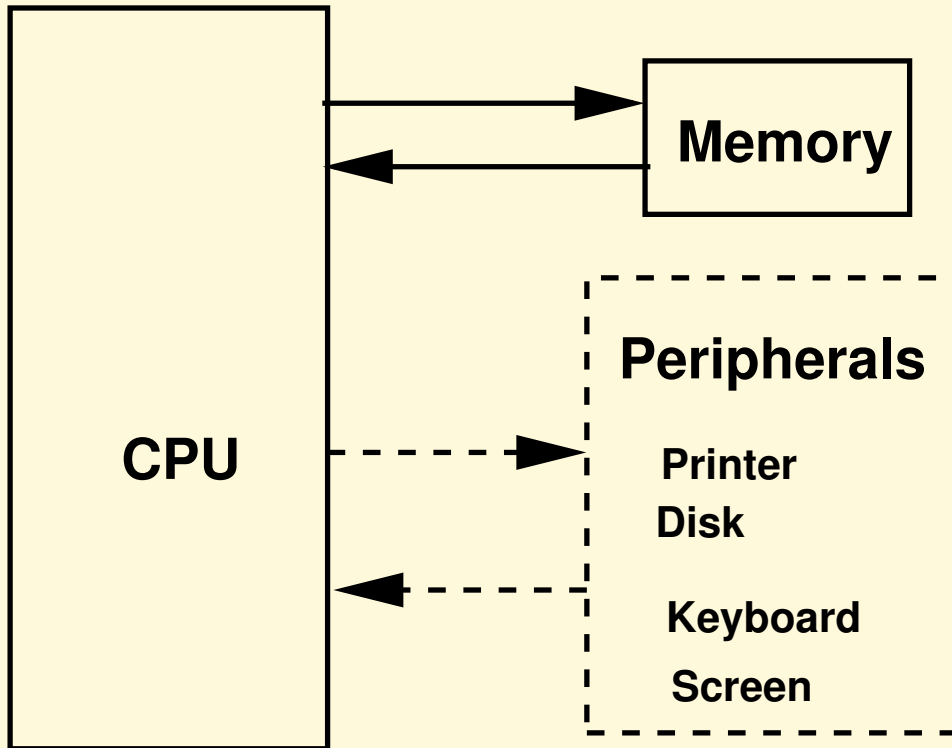
Go Back

Full Screen

Close

Quit

CPU & Memory: Simplified



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 574 of 709

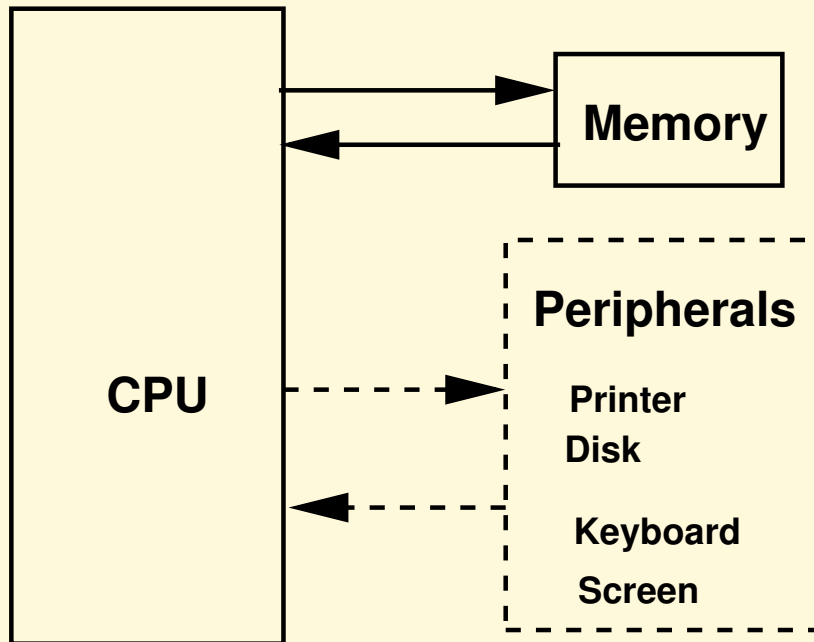
Go Back

Full Screen

Close

Quit

Resource Management



Operating System



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 575 of 709

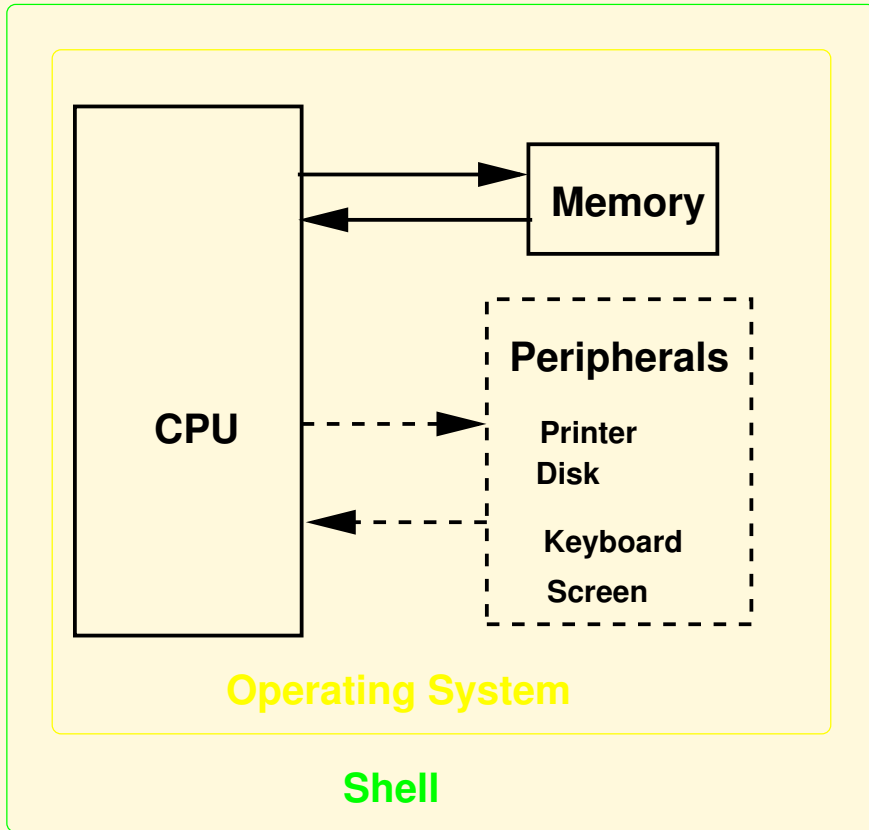
Go Back

Full Screen

Close

Quit

Shell: User Interface



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 576 of 709

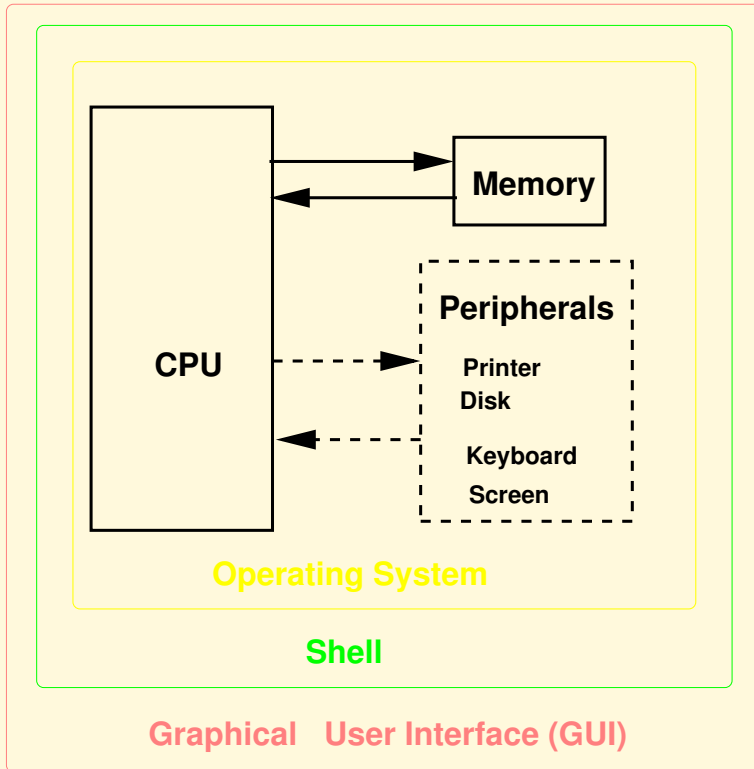
Go Back

Full Screen

Close

Quit

GUI: User Interface



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 577 of 709

Go Back

Full Screen

Close

Quit

Memory Model: Simplified

1. A sequence of storage **cells**
2. Each **cell** is a **container** of a single unit of information.
 - integer, real, boolean, character or string
3. Each cell has a unique name, called its **address**
4. The memory cell addresses range from **0** to (usually) $2^k - 1$ (for some k)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 578 of 709

Go Back

Full Screen

Close

Quit

Memory

0	1	2	3						

32K-1



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 579 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 580 of 709

Go Back

Full Screen

Close

Quit

The Imperative Model

- **Memory** or Storage made explicit
- Notion of **state** (of memory)
 - **State** is simply the value contained in each cell.
 - *state : Addresses \rightarrow Values*
- **State changes**

State Changes: σ

0	1	2	3						
		4							
						3.1			
		true							
						"#a"			

Assume all other cells are filled with null



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 581 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 582 of 709

Go Back

Full Screen

Close

Quit

State

The state σ

- $\sigma(12) = 4 : int$
- $\sigma(20) = null$
- $\sigma(43) = true : bool$
- $\sigma(66) = " \#a " : char$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 583 of 709

Go Back

Full Screen

Close

Quit

State Changes

- A **state change** takes place when the value in some cell changes
- The contents of only one cell may be changed at a time.

State Changes: σ

0	1	2	3						
		4							
						3.1			
		true							
						"#a"			

Assume all other cells are filled with null



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 584 of 709

Go Back

Full Screen

Close

Quit

State Changes: σ_1

0	1	2	3						
		5							
						3.1			
		true							
						"#a"			

Assume all other cells are filled with null



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 585 of 709

Go Back

Full Screen

Close

Quit

State Changes: σ_2

0	1	2	3						
		5							
						3.1			
		true							
12						"#a"			

Assume all other cells are filled with null



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 586 of 709

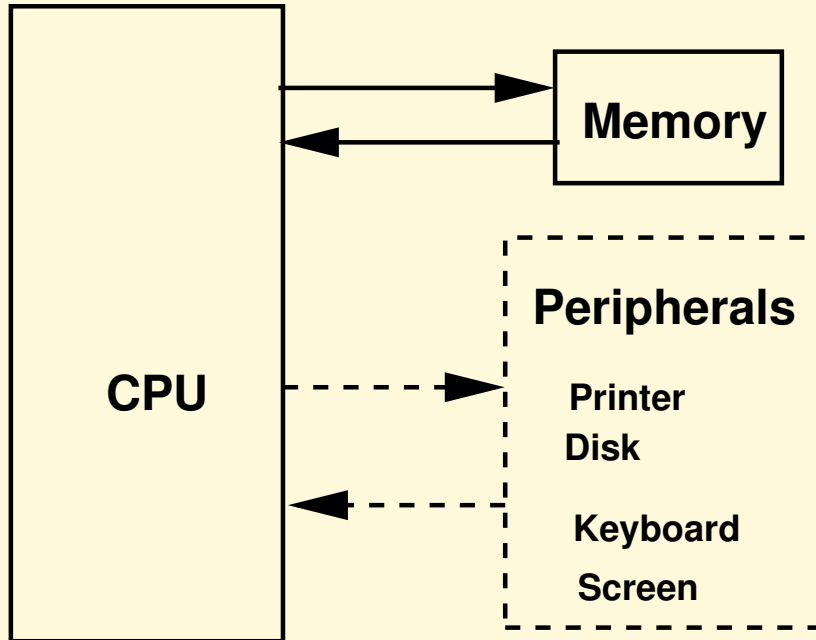
Go Back

Full Screen

Close

Quit

Languages



Operating System

Programming Language Interface



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 587 of 709

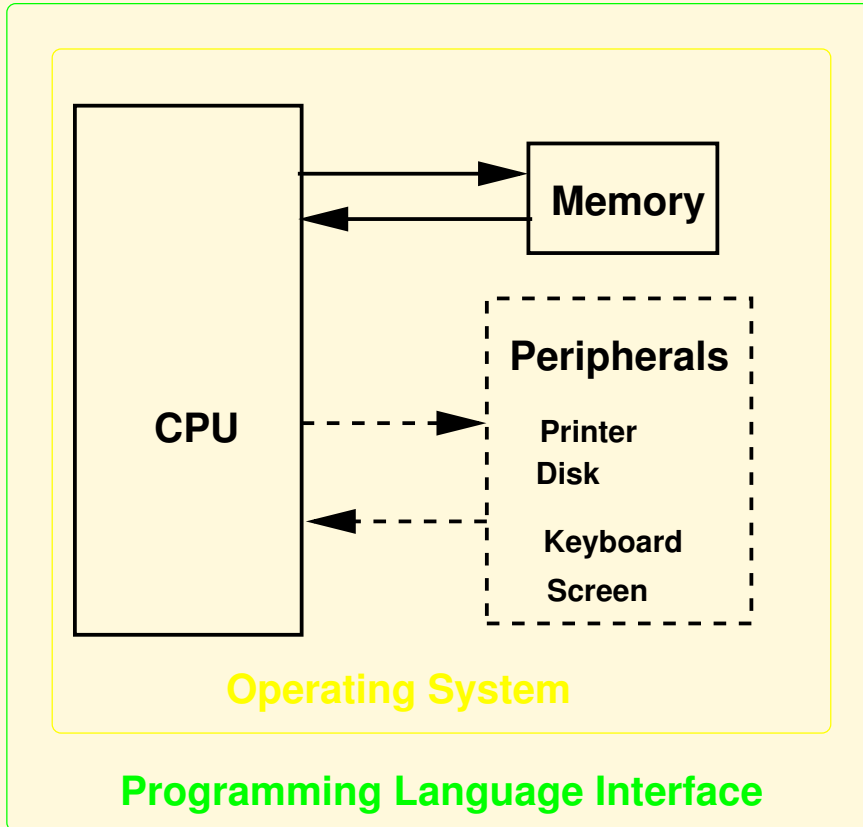
Go Back

Full Screen

Close

Quit

User Programs



User Programs



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 588 of 709

Go Back

Full Screen

Close

Quit

Imperative Languages

- How is the memory accessed?
 - Through system calls to the OS.
- How are memory cells identified?
 - Use **Imperative variables**.
 - Each such variable is a *name* mapped to an *address* .
- How are state changes accomplished?
 - By the **assignment** command.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 589 of 709

Go Back

Full Screen

Close

Quit



Imperative vs Functional Variables

Functional	Imperative
name of a value constant	name of an address could change with time

The value contained in an imperative variable x is denoted $!x$.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 590 of 709

Go Back

Full Screen

Close

Quit



Assignment Commands

Let x and y be imperative variables.
Consider the following commands.
Assuming $!x = 1$ and $!y = 2$.

x

1

y

2

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 591 of 709

Go Back

Full Screen

Close

Quit



Assignment Commands

Store the value 5 in x .

$x := 5$

x

5

y

2

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 592 of 709

Go Back

Full Screen

Close

Quit



Assignment Commands

Copy the value contained in y into x .

$x := !y$

x

2

y

2

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 593 of 709

Go Back

Full Screen

Close

Quit



Assignment Commands

Increment the value contained in x by 1.

$x := !x + 1$

.

x

3

y

2

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 594 of 709

Go Back

Full Screen

Close

Quit



Assignment Commands

*Store the product of the values in x and y
in y .*

$y := !x * !y$

x

3

y

6

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 595 of 709

Go Back

Full Screen

Close

Quit

Assignment Commands: Swap

Swap the values in x and y .

Swapping values implies trying to make two state changes simultaneously!

Requires a new memory cell t to temporarily store one of the values.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 596 of 709

Go Back

Full Screen

Close

Quit



Swap

How does one get a new memory cell?

```
val t = ref 0
```

Then the rest is easy

```
val t = ref 0;  
t := !x;  
x := !y;  
y := !t;
```

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 597 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 598 of 709

Go Back

Full Screen

Close

Quit

Swap

Could be made simpler!

```
val t = ref (!x);  
x := !y;  
y := !t;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 599 of 709

Go Back

Full Screen

Close

Quit

Swap

Could use a temporary **functional** variable *t* instead of an **imperative** variable

```
val t = !x;  
x := !y;  
y := t;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 600 of 709

Go Back

Full Screen

Close

Quit

7.2. Imperative Programming:

1. Imperative vs Functional
2. Features of the Store
3. References: Experiments
4. References: Experiments
5. References: Experiments
6. Aliases
7. References: Experiments
8. References: Aliases
9. References: Experiments
10. After Garbage Collection
11. Side Effects
12. Imperative ML
13. Imperative ML
14. Imperative ML
15. Imperative ML
16. Nasty Surprises
17. Imperative ML

18. Imperative ML
19. Common Errors
20. Aliasing & References
21. Dangling References
22. New Reference
23. Imperative Commands: Basic
24. Imperative Commands: Compound
25. Predefined Compound Commands



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 601 of 709

Go Back

Full Screen

Close

Quit



Imperative vs Functional

- Functional Model
- Memory/Store Model
- Imperative Model
- State Changes
- Accessing the store

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 602 of 709

Go Back

Full Screen

Close

Quit



Features of the Store

Memory is treated as a datatype with constructors

Allocation $ref : \alpha \rightarrow \alpha \text{ ref}$

Dereferencing $! : \alpha \text{ ref} \rightarrow \alpha$

Updation $:= : \alpha \text{ ref} * \alpha \rightarrow \text{unit}$

Deallocation of memory is **automatic!**

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 603 of 709

Go Back

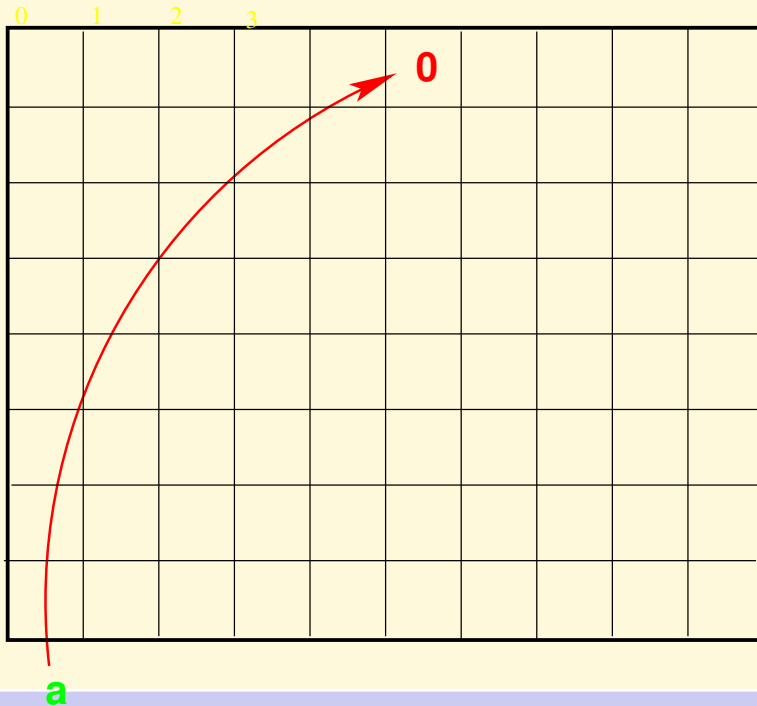
Full Screen

Close

Quit

References: Experiments

```
- val a = ref 0;  
val a = ref 0 : int ref
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 604 of 709

Go Back

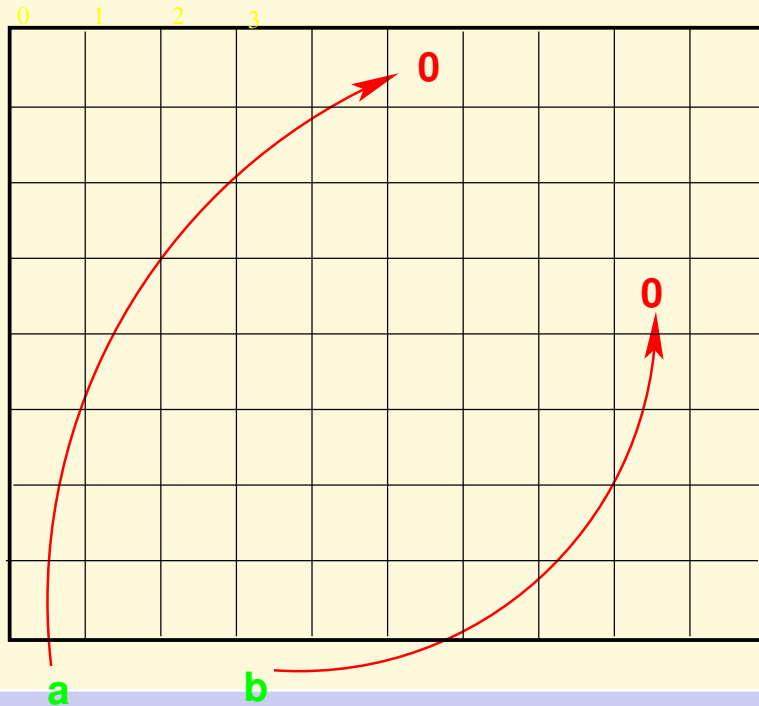
Full Screen

Close

Quit

References: Experiments

```
- val b = ref 0;  
val b = ref 0 : int ref
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 605 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 606 of 709

Go Back

Full Screen

Close

Quit

References: Experiments

```
- a = b;
```

```
val it = false : bool
```

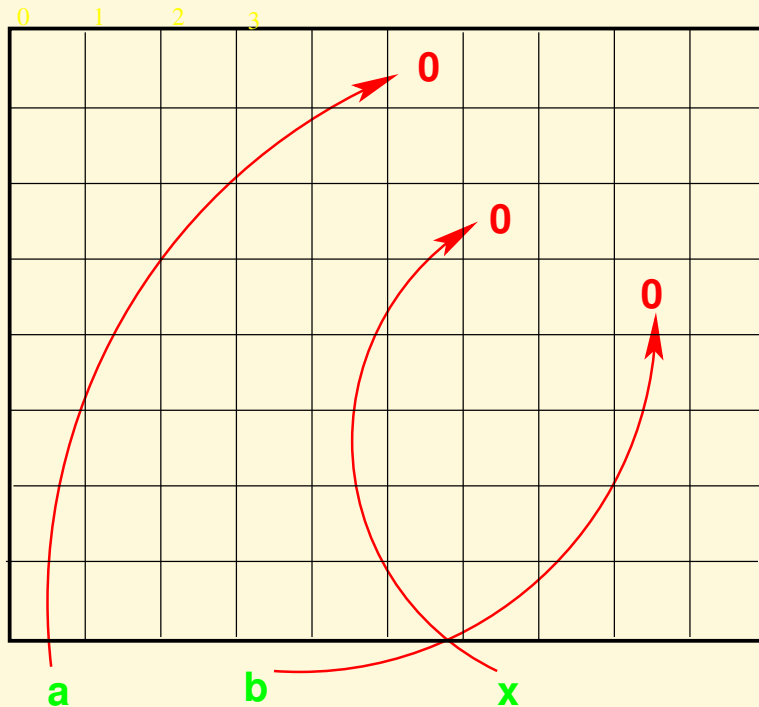
```
- !a = !b;
```

```
val it = true : bool
```

Aliases

```
- val x = ref 0;
```

```
val x = ref 0 : int ref
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 607 of 709

Go Back

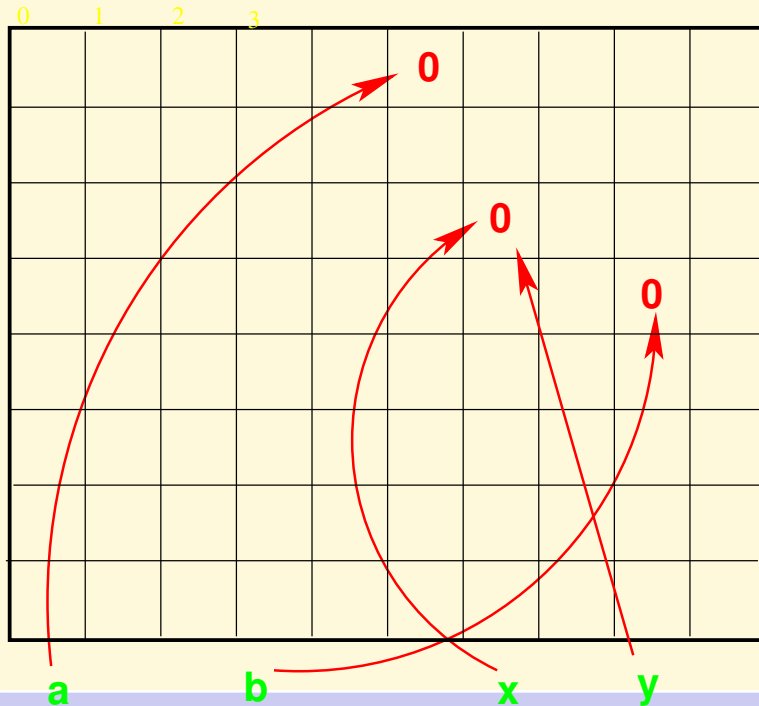
Full Screen

Close

Quit

References: Experiments

```
- val y = x;  
val y = ref 0 : int ref
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 608 of 709

Go Back

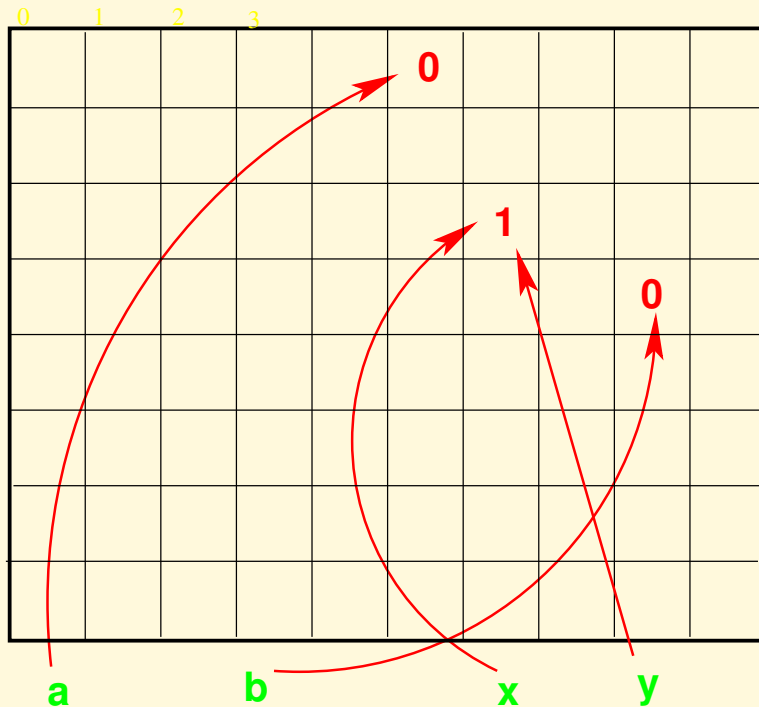
Full Screen

Close

Quit

References: Aliases

```
- x := !x + 1;  
val it = () : unit
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 609 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 610 of 709

Go Back

Full Screen

Close

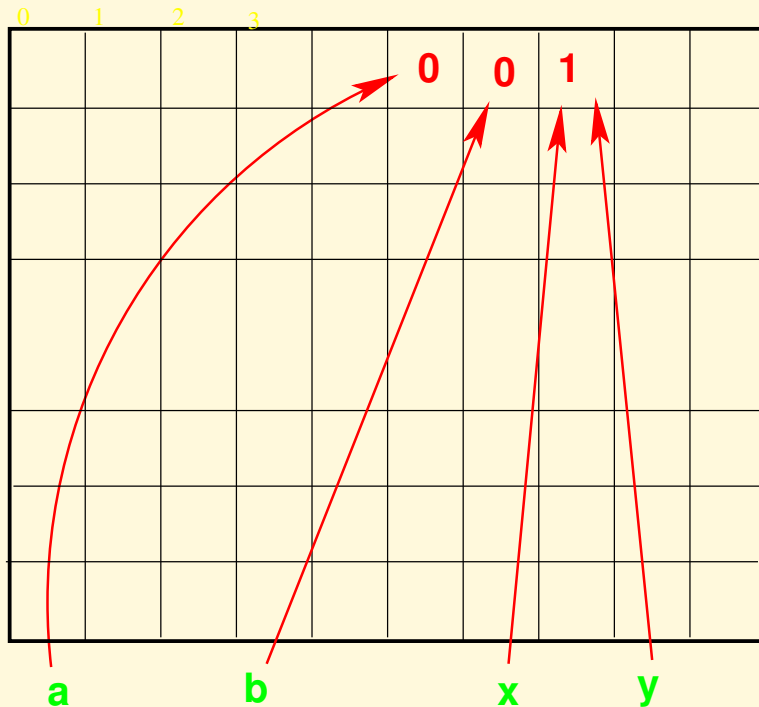
Quit

References: Experiments

```
- !y;  
val it = 1 : int  
- x = y;  
val it = true : bool
```

After Garbage Collection

GC #0.0.0.0.2.45: (0 ms)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 611 of 709

Go Back

Full Screen

Close

Quit



Side Effects

- **Assignment** does not produce a value
- It produces only a state change (side effect)
- But side-effects are compatible with functional programming since it is provided as a new data type with constructors and destructors.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 612 of 709

Go Back

Full Screen

Close

Quit



Imperative ML

- Does not provide **direct access** to memory addresses
- Does not allow for uninitialized imperative variables
- Provides a type with every memory location
- Manages the memory completely automatically

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 613 of 709

Go Back

Full Screen

Close

Quit

Imperative ML

- Does not provide direct access to memory addresses
 - Prevents the use of memory addresses as integers that can be manipulated by the user program
- Does not allow for **uninitialized** imperative variables
- Provides a type with every memory location
- Manages the memory completely automatically



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 614 of 709

Go Back

Full Screen

Close

Quit

Imperative ML

- Does not provide direct access to memory addresses
- Does not allow for uninitialized imperative variables
 - Most imperative languages keep declarations **separate** from initializations
- Provides a **type** with every memory cell
- Manages the memory completely automatically



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 615 of 709

Go Back

Full Screen

Close

Quit

Imperative ML

- Does not provide direct access to memory addresses
- Does not allow for uninitialized imperative variables
 - A frequent source of surprising results in most imperative language programs
- Provides a **type** with every memory cell
- Manages the memory completely automatically



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 616 of 709

Go Back

Full Screen

Close

Quit

Nasty Surprises

Separation of declaration from initialization

- Uninitialized variables
- Execution time errors if not detected by compiler, since every memory location contains some data
- Might use a value stored previously in that location by some imperative variable that no longer exists.
- Errors due to type violations.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 617 of 709

Go Back

Full Screen

Close

Quit



Imperative ML

- Does not provide direct access to memory addresses
- Does not allow for uninitialized imperative variables
- Provides a type with every memory cell
- **Manages** the memory completely **automatically** and securely.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 618 of 709

Go Back

Full Screen

Close

Quit

Imperative ML

- Does not provide direct access to memory addresses
- Does not allow for uninitialized imperative variables
- Provides a **type** with every memory cell
- Manages the memory completely automatically and securely
 - Memory has to be managed by the user program in most languages
 - Prone to various **errors**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 619 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 620 of 709

Go Back

Full Screen

Close

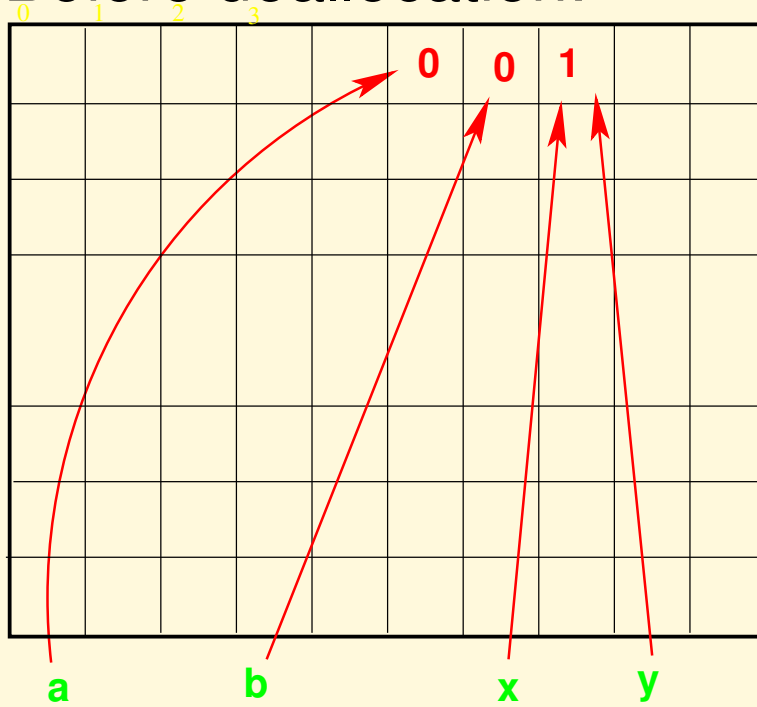
Quit

Common Errors

- Memory access errors due to integer arithmetic, especially in large structures (arrays)
- **Dangling references** on deallocation of aliased memory

Aliasing & References

Before deallocation:



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 621 of 709

Go Back

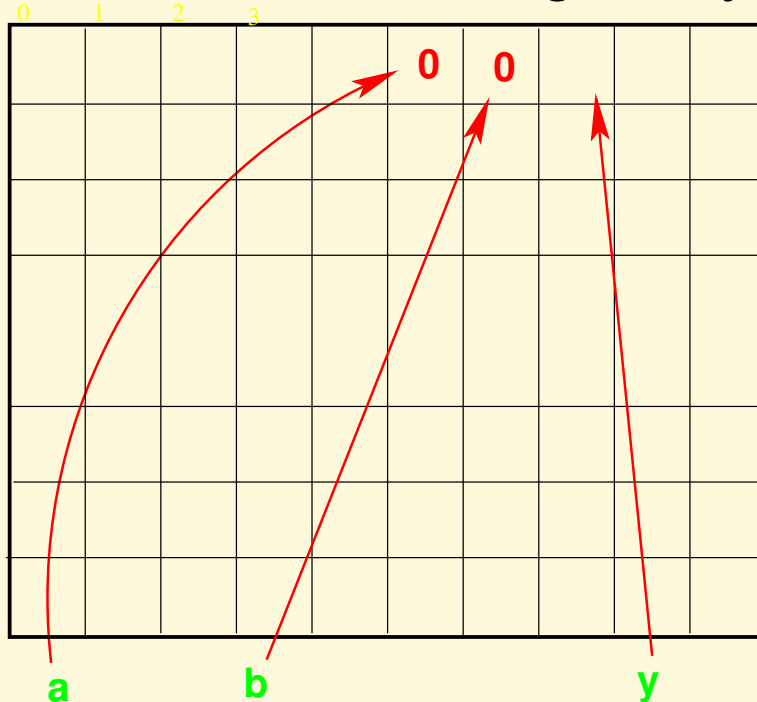
Full Screen

Close

Quit

Dangling References

Deallocate x through a system call



y is left dangling!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 622 of 709

Go Back

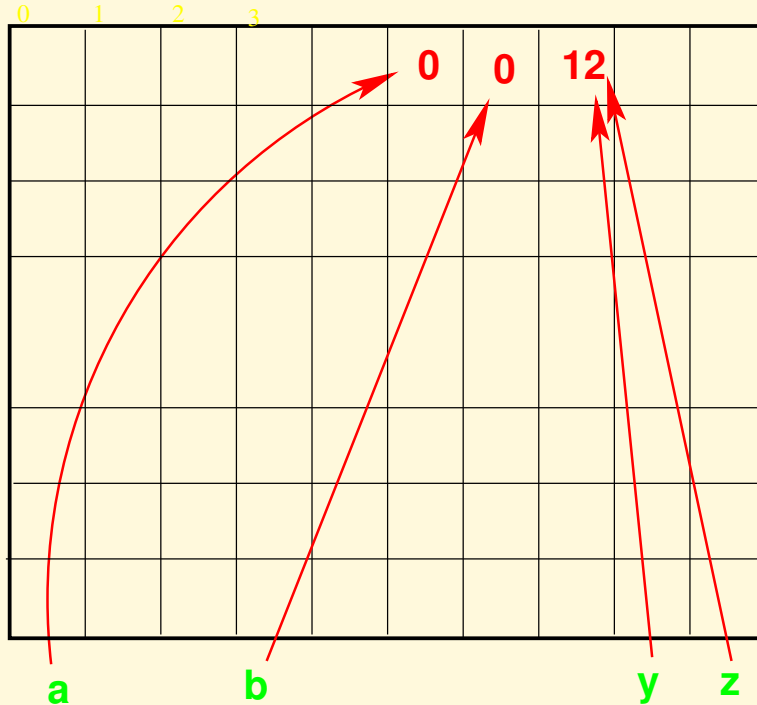
Full Screen

Close

Quit

New Reference

```
val z = ref 12;
```



By sheer coincidence $!y = 12$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 623 of 709

Go Back

Full Screen

Close

Quit



Imperative Commands: Basic

A **Command** is an ML expression that creates a **side effect** and returns an empty tuple $(()) : \textit{unit}$.

Assignment
print

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 624 of 709

Go Back

Full Screen

Close

Quit

Imperative Commands: Compound

Any complex ML expression or function definition whose type is of the form $\alpha \rightarrow \textit{unit}$ is a compound command.

- **Predefined** ML compound commands
- Could be user-defined. After all, *everything is a value!*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 625 of 709

Go Back

Full Screen

Close

Quit

Predefined Compound Commands

branching *if e then c₁ else c₀.*

cases *case e of p₁ ⇒ c₁ | ⋯ | p_n ⇒ c_n*

Sequencing $(c_1; c_2; \dots; c_n)$. Sequencing is associative

looping *while e do c₁* is defined recursively as

if e then (c₁; while e do c₁) else ()



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 626 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 627 of 709

Go Back

Full Screen

Close

Quit

7.3. Arrays

1. Why Imperative
2. Arrays
3. Indexing Arrays
4. Indexing Arrays
5. Indexing Arrays
6. Physical Addressing
7. Arrays
8. 2D Arrays
9. 2D Arrays: Indexing
10. Ordering of indices
11. Arrays vs. Lists
12. Arrays: Physical
13. Lists: Physical

Why Imperative

- Historical reasons: Early machine instruction set.
- Programming evolved from the machine architecture.
- Legacy software:
 - numerical packages
 - operating systems
- Are there any real benefits of imperative programming?



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 628 of 709

Go Back

Full Screen

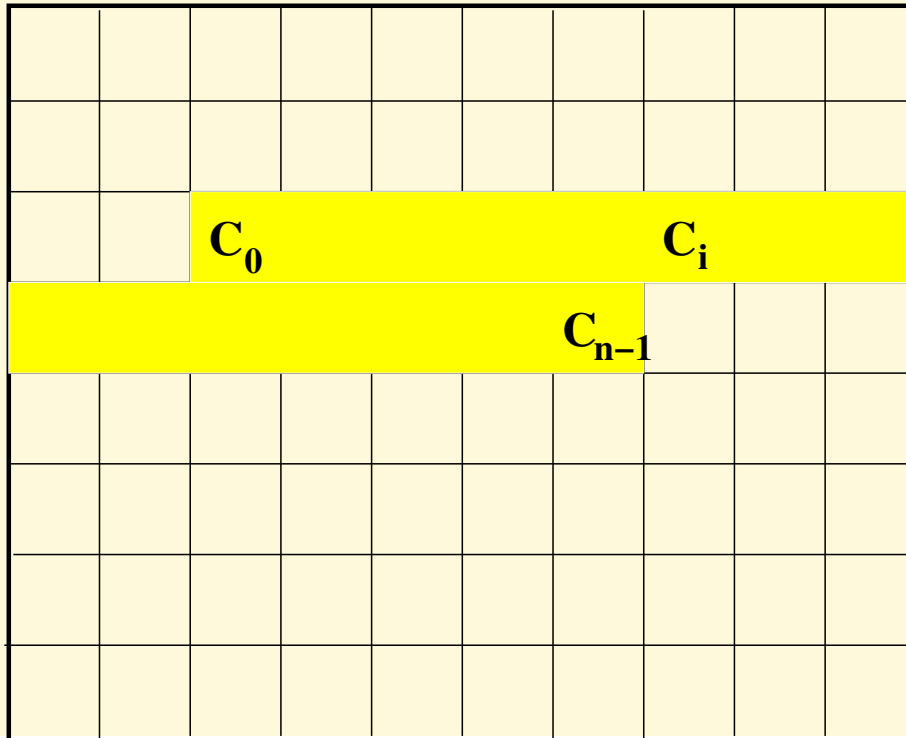
Close

Quit

Arrays

An **array** of length n is a **contiguous** sequence of n memory cells

$$C_0, C_1, \dots, C_{n-1}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 629 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 630 of 709

Go Back

Full Screen

Close

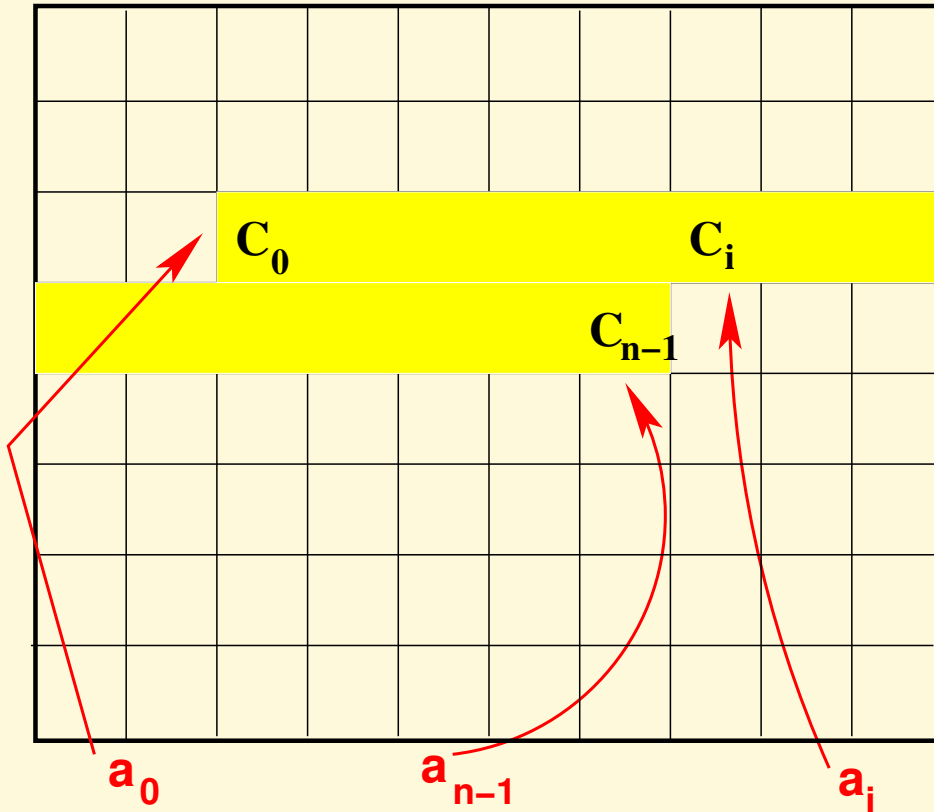
Quit

Indexing Arrays

For any array

- $i, 0 \leq i < n$ is the *index* of cell C_i .
- C_i is at a distance of i cells away from C_0

Indexing Arrays



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 631 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 632 of 709

Go Back

Full Screen

Close

Quit

Indexing Arrays

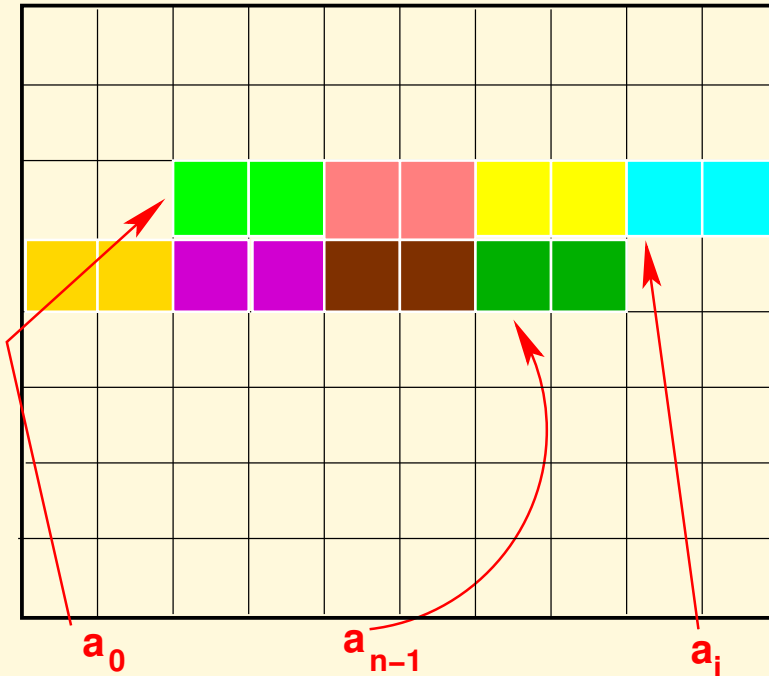
- The *start* address of the array and the *address* of C_0 are the same (say a_0)
- The address a_i of cell C_i is

$$a_i = a_0 + i$$

Physical Addressing

If each element occupies s physical memory locations, then

$$a_i = a_0 + i \times s$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 633 of 709

Go Back

Full Screen

Close

Quit

Arrays

A 2-dimensional array of

- r rows numbered 0 to $r - 1$
- each row containing c elements numbered 0 to $c - 1$

is also a **contiguous** sequence of rc memory cells

$$C_{0,0}, C_{0,1}, \dots, C_{0,c-1}, C_{1,0}, \dots, C_{r-1,c-1}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 634 of 709

Go Back

Full Screen

Close

Quit

2D Arrays

A 2 dimensional-array is represented as an array of length $r \times c$, where

- a_{00} is the start address of the array, and
- the address of the (i, j) -th cell is given by

$$a_{ij} = a_{00} + (c \times i + j)$$

- the physical address of the (i, j) -th cell is given by

$$a_{ij} = a_{00} + (c \times i + j) \times s$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 635 of 709

Go Back

Full Screen

Close

Quit



2D Arrays: Indexing

- The index (i, j) of a 2D array may be thought of as being similar to a 2-digit number in base c
- The successor of index (i, j) is the successor of a number in base c i.e.

$$\text{succ}(i, j) = \begin{cases} (i + 1, 0) & \text{if } j = n - 1 \\ (i, j + 1) & \text{else} \end{cases}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 637 of 709

Go Back

Full Screen

Close

Quit

Ordering of indices

There is a natural “ $<$ ” ordering on indices given by

$$\begin{aligned} (i, j) < (k, l) &\iff \\ (i < k) &\quad \text{or} \\ (i = k &\quad \text{and } j < l) \end{aligned}$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 638 of 709

Go Back

Full Screen

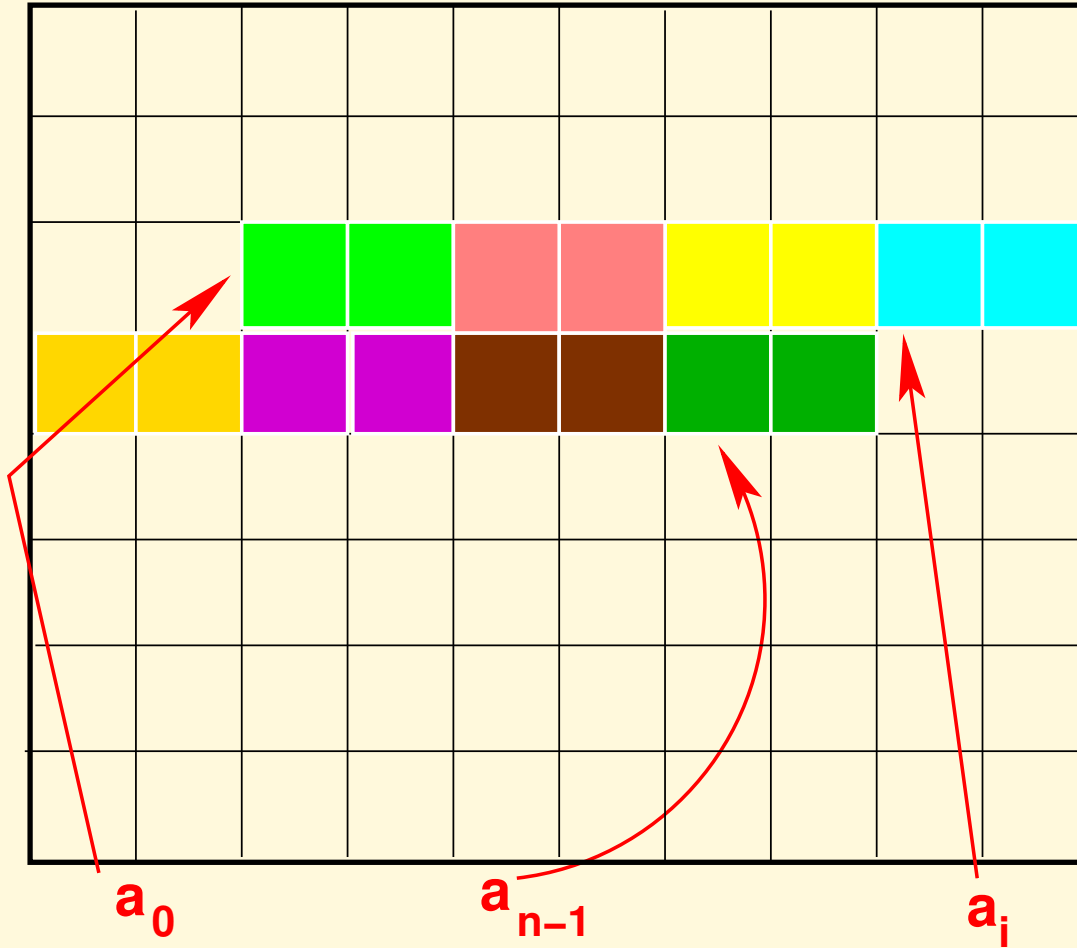
Close

Quit

Arrays vs. Lists

Lists	Arrays
Unbounded lengths	Fixed length
Insertions possible	Very complex
Indirect access	Direct access

Arrays: Physical



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 639 of 709

Go Back

Full Screen

Close

Quit

Lists: Physical



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



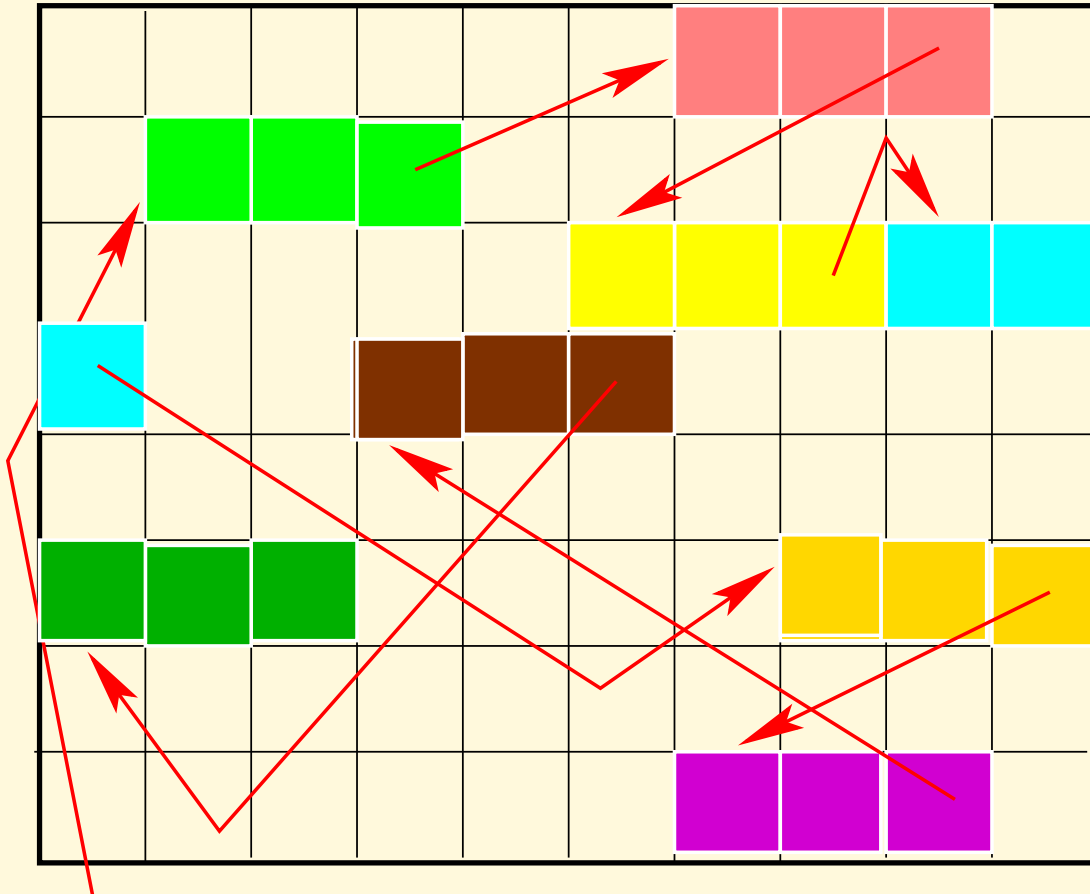
Page 640 of 709

Go Back

Full Screen

Close

Quit





8. A large Example: Tautology Checking

8.1. Large Example: Tautology Checking

1. Logical Arguments
2. Saintly and Rich
3. About Cats
4. About God
5. Russell's Argument
6. Russell's Argument
7. Russell's Argument
8. Russell's Argument
9. Propositions
10. Compound Propositions
11. Valuations
12. Valuations
13. Tautology
14. Properties
15. Negation Normal Form

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 641 of 709

Go Back

Full Screen

Close

Quit

16. Literals & Clauses
17. Conjunctive Normal Form
18. Validity
19. Logical Validity
20. Validity & Tautologies
21. Problem
22. Tautology Checking
23. Falsifying



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 642 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 643 of 709

Go Back

Full Screen

Close

Quit

Logical Arguments

Examples.

- Sainthly and Rich
- About cats
- About God
- Russell's argument



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 644 of 709

Go Back

Full Screen

Close

Quit

Saintly and Rich

hy1 *The landed are rich.*

hy2 *One cannot be both saintly and rich.*

conc The landed are not saintly



About Cats

hy1 *Tame cats are non-violent and vegetarian.*

hy2 *Non-violent cats would not kill mice.*

hy3 *Vegetarian cats are bottle-fed.*

hy4 *Cats eat meat.*

conc *Cats are not tame.*

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 645 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 646 of 709

Go Back

Full Screen

Close

Quit

About God

hy1 *God is omniscient and omnipotent.*

hy2 *An omniscient being would know there is evil.*

hy3 *An omnipotent being would prevent evil.*

hy4 *There is evil.*

conc There is no God

Russell's Argument

hy1 *If we can directly know that God exists, then we can know God exists by experience.*

hy2 *If we can indirectly know that God exists, then we can know God exists by logical inference from experience.*

hy3 *If we can know that God exists, then we can directly know that God exists, or we can indirectly know that God exists.*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 647 of 709

Go Back

Full Screen

Close

Quit

Russell's Argument

hy4 *If we cannot know God empirically, then we cannot know God by experience and we cannot know God by logical inference from experience.*

hy5 *If we can know God empirically, then “God exists” is a scientific hypothesis and is empirically justifiable.*

hy6 *“God exists” is not empirically justifiable.*

conc We cannot know that God exists.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 648 of 709

Go Back

Full Screen

Close

Quit

Russell's Argument

hy1 *If we can directly know that God exists, then we can know God exists by experience.*

hy2 *If we can indirectly know that God exists, then we can know God exists by logical inference from experience.*

hy3 *If we can know that God exists, then (we can directly know that God exists, or we can indirectly know that God exists).*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 649 of 709

Go Back

Full Screen

Close

Quit

Russell's Argument

hy4 *If we cannot know God empirically, then (we cannot know God by experience and we cannot know God by logical inference from experience.)*

hy5 *If we can know God empirically, then (“God exists” is a scientific hypothesis and is empirically justifiable.)*

hy6 *“God exists” is not empirically justifiable.*

conc *We cannot know that God exists.*



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 650 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 651 of 709

Go Back

Full Screen

Close

Quit

Propositions

A **proposition** is a sentence to which a **truth value** may be assigned.

In any real or imaginary world of facts a proposition has a truth value, **true** or **false**.

An **atom** is a simple proposition that has no propositions as components.

Compound Propositions

Compound propositions may be formed from **atoms** by using the following operators/constructors.

operator	symbol
not	\neg
and	\wedge
or	\vee
if... then...	\Rightarrow
equivalent	\Leftrightarrow



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 652 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 653 of 709

Go Back

Full Screen

Close

Quit

Valuations

Given truth values to individual atoms the truth values of compound propositions are evaluated as follows:

p	$\neg p$
true	false
false	true



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 654 of 709

Go Back

Full Screen

Close

Quit

Valuations

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \iff q$
true	true	true	true	true	true
true	false	false	true	false	false
false	true	false	true	true	false
false	false	false	false	true	true



Tautology

A (compound) proposition is a **tautology** if it is **true** regardless of what truth values are assigned to its **atoms**.

Examples.

- $p \vee \neg p$
- $(p \wedge q) \Rightarrow p$
- $(p \wedge \neg p) \Rightarrow q$

Properties

- Every proposition may be expressed in a logically equivalent form using only the operators \neg , \wedge and \vee

$$(p \iff q) = (p \implies q) \wedge (q \implies p)$$

$$(p \implies q) = (\neg p \vee q)$$

- De Morgan's laws may be applied to push \neg inward

$$\neg(p \wedge q) = \neg p \vee \neg q$$

$$\neg(p \vee q) = \neg p \wedge \neg q$$



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 656 of 709

Go Back

Full Screen

Close

Quit



Negation Normal Form

- Double negations may be removed since

$$\neg\neg p = p$$

- Every proposition may be expressed in a form containing only \wedge and \vee with \neg appearing only in front of atoms.

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 657 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 658 of 709

Go Back

Full Screen

Close

Quit

Literals & Clauses

- A **literal** is either an **atom** or \neg applied to an **atom**
- \vee is commutative and associative
- A **clause** is of the form $\bigvee_{j=1}^m l_j$, where each l_j is a **literal**.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 659 of 709

Go Back

Full Screen

Close

Quit

Conjunctive Normal Form

- \vee may be distributed over \wedge

$$p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$$

- \wedge is commutative and associative.
- Every proposition may be expressed in the form $\bigwedge_{i=1}^n q_i$, where each q_i is a clause.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 660 of 709

Go Back

Full Screen

Close

Quit

Validity

- A **logical argument** consists of a number of **hypotheses** and a single **conclusion** ($[h_1, \dots, h_n] | c$)
- A logical argument is **valid** if the conclusion **logically follows** from the hypotheses.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 661 of 709

Go Back

Full Screen

Close

Quit

Logical Validity

The **conclusion** logically follows from the given **hypotheses** if for any truth assignment to the atoms,

either some **hypothesis** h_i is **false**

or whenever every one of the hypotheses is **true** the conclusion is also **true**



Validity & Tautologies

- A **tautology** is a **valid** argument in which there is a **conclusion** without any hypothesis.
- A **logical argument** $[h_1, \dots, h_n] | c$, is **valid** if and only if

$$(h_1 \wedge \dots \wedge h_n) \Rightarrow c$$

is a **tautology**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 663 of 709

Go Back

Full Screen

Close

Quit

Problem

Given an argument $[h_1, \dots, h_n] | c$,

- determine whether $(h_1 \wedge \dots \wedge h_n) \Rightarrow c$ is a tautology, and
- If it is not a tautology, to determine what truth assignments to the atoms make it **false**.



Tautology Checking

A proposition in CNF ($\bigwedge_{i=1}^n p_i$)

- is a tautology if and only if every proposition p_i , $1 \leq i \leq m$, is a tautology.
- otherwise at least one **clause** p_i must be **false**
- **Clause** $p_i = \bigvee_{j=1}^m l_{ij}$ is **false** if and only if every literal l_{ij} , $1 \leq j \leq m$ is **false**

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 664 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 665 of 709

Go Back

Full Screen

Close

Quit

Falsifying

For a proposition in CNF ($\bigwedge_{i=1}^n p_i$) that is not a tautology

- A **clause** $p_i = \bigvee_{j=1}^m l_{ij}$ is **false**
- A truth assignment that **falsifies** the argument
 - sets the atoms that occur **negatively** in p_i to **true**,
 - sets every other atom to **false**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 666 of 709

Go Back

Full Screen

Close

Quit

8.2. Tautology Checking Contd.

1. Tautology Checking
2. Normal Forms
3. Top-down Development
4. The Signature
5. The Core subproblem
6. The datatype
7. Convert to CNF
8. Rewrite into NNF
9. \iff and \Rightarrow Elimination
10. Push \neg inward
11. Push \neg inward
12. conj_of_disj
13. Push \vee inward
14. Tautology & Falsification



Tautology Checking

- Logical arguments
- Propositional forms
- Propositions
- Compound Propositions
- Truth table
- Tautologies

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 667 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 668 of 709

Go Back

Full Screen

Close

Quit

Normal Forms

- Properties
- Negation Normal Form
- Conjunctive Normal Forms
- Valid Propositional Arguments as tautologies
- The problem

Top-down Development

- Transform the **argument** into a single proposition.
 - Transform the single **proposition** into one in **CNF**
-
- Check whether every **clause** is a tautology
 - If any **clause** is not a tautology, find the truth assignment(s) that make it **false**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 669 of 709

Go Back

Full Screen

Close

Quit

The Signature

```
signature PropLogic =  
sig datatype Prop = ??  
  type Argument =  
    Prop list * Prop  
  val falsifyArg :  
    Argument -> Prop list list  
  val Valid :  
    Argument -> bool *  
      Prop list list  
  ...  
end;
```



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 670 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 671 of 709

Go Back

Full Screen

Close

Quit

The Core subproblem

- Representing **propositions**
- Transformation of **propositions** into CNF
 - Transform into **Negation Normal Form (NNF)**
 - Transform NNF into **Conjunctive Normal Form (CNF)**

The datatype

datatype Prop =
 ATOM of string |
 NOT of Prop |
 AND of Prop * Prop |
 OR of Prop * Prop |
 IMP of Prop * Prop |
 EQL of Prop * Prop



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 672 of 709

Go Back

Full Screen

Close

Quit



Convert to CNF

Convert a given proposition into CNF

```
fun cnf (P) =  
  conj_of_disj (  
    nnf (rewrite (P)) );
```

where

- `rewrite` eliminates \iff and \implies
- `nnf` converts into NNF
- `conj_of_disj` converts into CNF

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 673 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 674 of 709

Go Back

Full Screen

Close

Quit

Rewrite into NNF

- Eliminate \iff and then \implies
- Push \neg inward using De Morgan's laws and eliminate double negations.



\iff and \Rightarrow Elimination

```
fun rewrite (ATOM a) = ATOM a
| rewrite (IMP (P, Q)) =
  OR (NOT (rewrite(P)),
      rewrite(Q))
| rewrite (EQL (P, Q)) =
  rewrite (AND (IMP (P, Q),
               IMP (Q, P)))
| ...
```

Proposition made up of only \neg , \wedge and \vee .

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents

◀ ▶

◀ ▶

Page 675 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 676 of 709

Go Back

Full Screen

Close

Quit

Push \neg inward

```
fun nnf (ATOM a) =  
  ATOM a
```

```
| nnf (NOT (ATOM a)) =  
  NOT (ATOM a)
```

```
| nnf (NOT (NOT (P))) =  
  nnf (P)
```



Push \neg inward

$$\begin{aligned} | \quad & \text{nnf (NOT (AND (P, Q)))) =} \\ & \text{nnf (OR (NOT (P),} \\ & \qquad \qquad \qquad \text{NOT (Q)))} \end{aligned}$$

$$\begin{aligned} | \quad & \text{nnf (NOT (OR (P, Q)))) =} \\ & \text{nnf (AND (NOT (P),} \\ & \qquad \qquad \qquad \text{NOT (Q)))} \end{aligned}$$

| . . .

Proposition made up of only \wedge and \vee
applied to positive or negative literals.



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 678 of 709

Go Back

Full Screen

Close

Quit

conj_of_disj

```
fun conj_of_disj (AND (P, Q)) =
  AND (conj_of_disj (P),
        conj_of_disj (Q))
| conj_of_disj (OR (P, Q)) =
  distOR (conj_of_disj (P),
           conj_of_disj (Q))
| conj_of_disj (P) = P
```

where **distOR** is



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 679 of 709

Go Back

Full Screen

Close

Quit

Push \vee inward

Use distributivity of \vee over \wedge

$$\begin{aligned} \text{fun } \text{distOR } (P, \text{AND } (Q, R)) &= \\ &\text{AND } (\text{distOR } (P, Q), \\ &\quad \text{distOR } (P, R)) \\ | \text{distOR } (\text{AND } (Q, R), P) &= \\ &\text{AND } (\text{distOR } (Q, P), \\ &\quad \text{distOR } (R, P)) \\ | \text{distOR } (P, Q) &= \text{OR } (P, Q) \end{aligned}$$

Tautology & Falsification

Falsifying a proposition

- A **proposition** Q in **CNF**, **not a tautology** if and only if at least one of the clauses can be made **false**, by a suitable truth assignment
- The list of atoms which are set **true** to **falsify** a clause is called a falsifier.
- A **proposition** is a tautology if and only if there is **no falsifier!**



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 680 of 709

Go Back

Full Screen

Close

Quit



9. Lecture-wise Index to Slides

Index

Introduction to Computing

(3-8)

1. Introduction
2. Computing tools
3. Ruler and Compass
4. Computing and Computers
5. Primitives
6. Algorithm
7. Problem: Doubling a Square
8. Solution: Doubling a Square
9. Execution: Step 1
10. Execution: Step 2
11. Doubling a Square: Justified
12. Refinement: Square Construction
13. Refinement 2: Perpendicular at a point
14. Solution: Perpendicular at a point
15. Perpendicular at a point: Justification

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 681 of 709

Go Back

Full Screen

Close

Quit

Next: [Our Computing Tool](#)

Our Computing Tool

(9-33)

Previous: [Introduction to Computing](#)

1. [The Digital Computer: Our Computing Tool](#)
2. [Algorithms](#)
3. [Programming Language](#)
4. [Programs and Languages](#)
5. [Programs](#)
6. [Programming](#)
7. [Computing Models](#)
8. [Primitives](#)
9. [Primitive expressions](#)
10. [Methods of combination](#)
11. [Methods of abstraction](#)
12. [The Functional Model](#)
13. [Mathematical Notation 1: Factorial](#)
14. [Mathematical Notation 2: Factorial](#)
15. [Mathematical Notation 3: Factorial](#)
16. [A Functional Program: Factorial](#)
17. [A Computation: Factorial](#)



[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)



[Page 682 of 709](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

18. A Computation: Factorial
19. A Computation: Factorial
20. A Computation: Factorial
21. A Computation: Factorial
22. A Computation: Factorial
23. A Computation: Factorial
24. Standard ML
25. SML: Important Features

Next: Primitives: Integer & Real

Primitives: Integer & Real

(34-60)

Previous: Primitives: Integer & Real

1. Algorithms & Programs
2. SML: Primitive Integer Operations 1
3. SML: Primitive Integer Operations 1
4. SML: Primitive Integer Operations 1
5. SML: Primitive Integer Operations 1
6. SML: Primitive Integer Operations 1
7. SML: Primitive Integer Operations 1
8. SML: Primitive Integer Operations 2
9. SML: Primitive Integer Operations 2



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 683 of 709

Go Back

Full Screen

Close

Quit

10. SML: Primitive Integer Operations 2
11. SML: Primitive Integer Operations 2
12. SML: Primitive Integer Operations 2
13. SML: Primitive Integer Operations 3
14. SML: Primitive Integer Operations 3
15. SML: Primitive Integer Operations 3
16. SML: Primitive Integer Operations 3
17. SML: Primitive Integer Operations 3
18. Quotient & Remainder
19. SML: Primitive Real Operations 1
20. SML: Primitive Real Operations 1
21. SML: Primitive Real Operations 1
22. SML: Primitive Real Operations 2
23. SML: Primitive Real Operations 3
24. SML: Primitive Real Operations 4
25. SML: Precision
26. Fibonacci Numbers
27. Euclidean Algorithm

Next: Technical Completeness & Algorithms



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 684 of 709

Go Back

Full Screen

Close

Quit

1. Recapitulation: Integers & Real
2. Recap: Integer Operations
3. Recapitulation: Real Operations
4. Recapitulation: Simple Algorithms
5. More Algorithms
6. Powering: Math
7. Powering: SML
8. Technical completeness
9. What SML says
10. Technical completeness
11. What SML says ... *contd*
12. Powering: Math 1
13. Powering: SML 1
14. Technical Completeness
15. What SML says
16. Powering: Integer Version
17. Exceptions: A new primitive
18. Integer Power: SML
19. Integer Square Root 1
20. Integer Square Root 2
21. An analysis
22. Algorithmic idea



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 685 of 709

Go Back

Full Screen

Close

Quit

23. Algorithm: isqrt
24. Algorithm: shrink
25. SML: shrink
26. SML: intsqrt
27. Run it!
28. SML: Reorganizing Code
29. Intsqrt: Reorganized
30. shrink: Another algorithm
31. Shrink2: SML
32. Shrink2: SML ... *contd*

Algorithm Refinement

(93-122)

1. Recap: More Algorithms
2. Recap: Power
3. Recap: Technical completeness
4. Recap: More Algorithms
5. Intsqrt: Reorganized
6. Intsqrt: Reorganized
7. Some More Algorithms
8. Combinations: Math
9. Combinations: Details
10. Combinations: SML



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 686 of 709

Go Back

Full Screen

Close

Quit

11. Perfect Numbers
12. Refinement
13. Perfect Numbers: SML
14. $\sum_i^u \text{ifdivisor}(k)$
15. SML: sum_divisors
16. *ifdivisor* and *ifdivisor*
17. SML: Assembly 1
18. SML: Assembly 2
19. Perfect Numbers ...*contd.*
20. Perfect Numbers ...*contd.*
21. SML: Assembly 3
22. Perfect Numbers: Run
23. Perfect Numbers: Run
24. SML: Code variations
25. SML: Code variations
26. SML: Code variations
27. Summation: Generalizations
28. Algorithmic Improvements:
29. Algorithmic Variations
30. Algorithmic Variations



1. Recap
2. Recap: Combinations
3. Combinations 1
4. Combinations 2
5. Combinations 3
6. Perfect 2
7. Power 2
8. A Faster Power
9. Power2: Complete
10. Power2: SML
11. Power2: SML
12. Computation: Issues
13. General Correctness
14. Code: Justification
15. Recall
16. Features: Definition before Use
17. Run ifdivisor
18. Diagnosis: Features of programs
19. Back to Math
20. Incorrectness
21. ifdivisor3
22. Run it!



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 688 of 709

Go Back

Full Screen

Close

Quit

23. Try it!
24. Hey! Wait a minute!
25. The n 's
26. Scope
27. Scope Rules

Names, Scopes & Recursion

(150-181)

1. Disjoint Scopes
2. Nested Scopes
3. Overlapping Scopes
4. Spanning
5. Scope & Names
6. Names & References
7. Names & References
8. Names & References
9. Names & References
10. Names & References
11. Names & References
12. Names & References
13. Names & References
14. Names & References
15. Names & References



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 689 of 709

Go Back

Full Screen

Close

Quit



16. Definition of Names
17. Use of Names
18. `local...in...end`
19. `local...in...end`
20. `local...in...end`
21. `local...in...end`
22. Scope & `local`
23. Computations: Simple
24. Simple computations
25. Computations: Composition
26. Composition: Alternative
27. Compositions: Compare
28. Compositions: Compare
29. Computations: Composition
30. Recursion
31. Recursion: Left
32. Recursion: Right

Floating Point

(182-201)

1. So Far-1: Computing
2. So Far-2: Algorithms & Programs
3. So far-3: Top-down Design

IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 690 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 691 of 709

Go Back

Full Screen

Close

Quit

4. So Far-4: Algorithms to Programs
5. So far-5: Caveats
6. So Far-6: Algorithmic Variations
7. So Far-7: Computations
8. Floating Point
9. Real Operations
10. Real Arithmetic
11. Numerical Methods
12. Errors
13. Errors
14. Infinite Series
15. Truncation Errors
16. Equation Solving
17. Root Finding-1
18. Root Finding-2
19. Root Finding-3
20. Root Finding-4

Root Finding, Composition and Recursion

(202-229)

1. Root Finding: Newton's Method
2. Root Finding: Newton's Method
3. Root Finding: Newton's Method

4. Root Finding: Newton's Method
5. Root Finding: Newton's Method
6. Root Finding: Newton's Method
7. Newton's Method: Basis
8. Newton's Method: Basis
9. Newton' Method: Algorithm
10. What can go wrong!-1
11. What can go wrong!-2
12. What can go wrong!-2
13. What can go wrong!-3
14. What can go wrong!-4
15. Real Computations & Induction: 1
16. Real Computations & Induction: 2
17. What's it good for? 1
18. What's it good for? 2
19. *newton*: Computation
20. Generalized Composition
21. Two Computations of $h(1)$
22. Two Computations of $h(-1)$
23. Recursive Computations
24. Recursion: Left
25. Recursion: Right



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 692 of 709

Go Back

Full Screen

Close

Quit

26. Recursion: Nonlinear
27. Some Practical Questions
28. Some Practical Questions

Termination and Space Complexity

(230-266)

1. Recursion Revisited
2. Linear Recursion: Waxing
3. Recursion: Waning
4. Nonlinear Recursions
5. Fibonacci: *contd*
6. Recursion: Waxing & Waning
7. Unfolding Recursion
8. Non-termination
9. Termination
10. Proofs of termination
11. Proofs of termination: Induction
12. Proof of termination: Factorial
13. Proof of termination: Factorial
14. Fibonacci: Termination
15. GCD computations
16. Well-foundedness: GCD
17. Well-foundedness



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 693 of 709

Go Back

Full Screen

Close

Quit

18. Induction is Well-founded
19. Induction is Well-founded
20. Where it doesn't work
21. Well-foundedness is inductive
22. Well-foundedness is inductive
23. GCD: Well-foundedness
24. Newton: Well-foundedness
25. Newton: Well-foundedness
26. Example: Zero
27. Questions
28. The Collatz Problem
29. Questions
30. Space Complexity
31. Newton & Euclid: Absolute
32. Newton & Euclid: Relative
33. Deriving space requirements
34. GCD: Space
35. Factorial: Space
36. Fibonacci: Space
37. Fibonacci: Space

Efficiency Measures and Speed-ups

(267-295)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 694 of 709

Go Back

Full Screen

Close

Quit

1. Recapitulation
2. Recapitulation
3. Time & Space Complexity
4. *isqrt*: Space
5. Time Complexity
6. *isqrt*: Time Complexity
7. *isqrt2*: Time
8. *shrink* vs. *shrink2*: Times
9. Factorial: Time Complexity
10. Fibonacci: Time Complexity
11. Comparative Complexity
12. Comparisons
13. Comparisons
14. Efficiency Measures: Time
15. Efficiency Measures: Space
16. Speeding Up: 1
17. Speeding Up: 2
18. Factoring out calculations
19. Tail Recursion: 1
20. Tail Recursion: 2
21. Factorial: Tail Recursion
22. Factorial: Tail Recursion



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 695 of 709

Go Back

Full Screen

Close

Quit

23. A Computation
24. Factorial: Issues
25. Fibonacci: Tail Recursion
26. Fibonacci: Tail Recursion
27. fibTR: SML
28. State in Tail Recursion
29. Invariance

Invariance & Correctness

(296-319)

1. Recap
2. Recursion Transformation
3. Tail Recursion: Examples
4. Comparisons
5. Transformation Issues
6. Correctness Issues 1
7. Correctness Issues 2
8. Correctness Theorem
9. Invariants & Correctness 1
10. Invariants & Correctness 2
11. Invariance Lemma: $factL_{tr}$
12. Invariance: Example
13. Invariance: Example



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 696 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 697 of 709

Go Back

Full Screen

Close

Quit

14. Proof
15. Invariance Lemma: *fib_iter*
16. Proof
17. Correctness: Fibonacci
18. Variants & Invariants
19. Variants & Invariants
20. More Invariants
21. Fast Powering 1
22. Fast Powering 2
23. Root Finding: Bisection
24. Advantage Bisection

Tuples, Lists & the Generation of Primes

(320-341)

1. Recap: Tail Recursion
2. Examples: Invariants
3. Tuples
4. Lists
5. New Lists
6. List Operations
7. List Operations: *cons*
8. Generating Primes upto n
9. More Properties

10. Composites
11. Odd Primes
12. *primesUpto(n)*
13. *generateFrom(P, m, n, k)*
14. *generateFrom*
15. *primeWRT(m, P)*
16. *primeWRT(m, P)*
17. *primeWRT*
18. Density of Primes
19. The Prime Number Theorem
20. The Prime Number Theorem
21. Complexity
22. Diagnosis

Compound Data & Lists

1. Compound Data
2. Recap: Tuples
3. Tuple: Formation
4. Tuples: Selection
5. Tuples: Equality
6. Tuples: Equality errors
7. Lists: Recap

(342-367)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 698 of 709

Go Back

Full Screen

Close

Quit



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 699 of 709

Go Back

Full Screen

Close

Quit

8. Lists: Append
9. *cons* vs. @
10. Lists of Functions
11. Lists of Functions
12. Arithmetic Sequences
13. Tail Recursion
14. Tail Recursion Invariant
15. Tail Recursion
16. Another Tail Recursion: *AS3*
17. Another Tail Recursion: *AS3_iter*
18. *AS3*: Complexity
19. Generating Primes: 2
20. *primes2Upto*(*n*)
21. *generate2From*(*P*, *m*, *n*, *k*)
22. *generate2From*
23. *prime2WRT*(*m*, *P*)
24. *prime2WRT*
25. *primes2*: Complexity
26. *primes2*: Diagnosis

Compound Data & List Algorithms

(368-394)

1. Compound Data: Summary

2. Records: Constructors
3. Records: Example 1
4. Records: Example 2
5. Records: Destructors
6. Records: Equality
7. Tuples & Records
8. Back to Lists
9. Lists: Correctness
10. Lists: Case Analysis
11. Lists: Correctness by Cases
12. List-functions: *length*
13. List Functions: *search*
14. List Functions: *search2*
15. List Functions: *ordered*
16. List Functions:*insert*
17. List Functions: *reverse*
18. List Functions: *reverse2*
19. List Functions:*merge*
20. List Functions:*merge*
21. List Functions:*merge contd.*
22. ML: *merge*
23. Sorting by Insertion



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 700 of 709

Go Back

Full Screen

Close

Quit

24. [Sorting by Merging](#)
25. [Sorting by Merging](#)
26. [Functions as Data](#)
27. [Higher Order Functions](#)

Higher Order Functions

(395-419)

1. [Summary: Compound Data](#)
2. [List: Examples](#)
3. [Lists: Sorting](#)
4. [Higher Order Functions](#)
5. [An Example](#)
6. [Currying](#)
7. [Currying: Contd](#)
8. [Generalization](#)
9. [Generalization: 2](#)
10. [Applying a list](#)
11. [Trying it out](#)
12. [Associativity](#)
13. [Apply to a list](#)
14. [Sequences](#)
15. [Further Generalization](#)
16. [Further Generalization](#)



[IIT Delhi](#)

[S. Arun-Kumar, CSE](#)

[Title Page](#)

[Contents](#)



[Page 701 of 709](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

17. Sequences
18. Efficient Generalization
19. Sequences: 2
20. More Generalizations
21. More Summations
22. Or Maybe . . . Products
23. Or Some Other \otimes
24. Other \otimes
25. Examples of \otimes , e

Structured Data

(420-451)

1. Transpose of a Matrix
2. Transpose: 0
3. Transpose: 10
4. Transpose: 01
5. Transpose: 20
6. Transpose: 02
7. Transpose: 30
8. Transpose: 03
9. *trans*
10. *is2DMatrix*
11. User Defined Types



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 702 of 709

Go Back

Full Screen

Close

Quit

12. Enumeration Types
13. User Defined Structural Types
14. Functions vs. data
15. Data as 0-ary Functions
16. Data vs. Functions
17. Data vs. Functions: Recursion
18. Lists
19. Constructors
20. Shapes
21. Shapes: Triangle Inequality
22. Shapes: Area
23. Shapes: Area
24. ML: Try out
25. ML: Try out (contd.)
26. Enumeration Types
27. Recursive Data Types
28. Resistors: Datatype
29. Resistors: Equivalent
30. Resistors
31. Resistors: Example
32. Resistors: ML session



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 703 of 709

Go Back

Full Screen

Close

Quit

User Defined Structured Data Types

(452-472)

1. User Defined Types
2. Resistors: Grouping
3. Resistors: In Pairs
4. Resistor: Values
5. Resistance Expressions
6. Resistance Expressions
7. Arithmetic Expressions
8. Arithmetic Expressions: 0
9. Arithmetic Expressions: 1
10. Arithmetic Expressions: 2
11. Arithmetic Expressions: 3
12. Arithmetic Expressions: 4
13. Arithmetic Expressions: 5
14. Arithmetic Expressions: 6
15. Arithmetic Expressions: 7
16. Arithmetic Expressions: 8
17. Binary Trees
18. Arithmetic Expressions: 0
19. Trees: Traversals
20. Recursive Data Types: Correctness



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 704 of 709

Go Back

Full Screen

Close

Quit

21. Data Types: Correctness

Introducing a Memory Model

(473-499)

1. Summary: Functional Model
2. CPU & Memory: Simplified
3. Resource Management
4. Shell: User Interface
5. GUI: User Interface
6. Memory Model: Simplified
7. Memory
8. The Imperative Model
9. State Changes: σ
10. State
11. State Changes
12. State Changes: σ
13. State Changes: σ_1
14. State Changes: σ_2
15. Languages
16. User Programs
17. Imperative Languages
18. Imperative vs Functional Variables
19. Assignment Commands



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 705 of 709

Go Back

Full Screen

Close

Quit

20. Assignment Commands
21. Assignment Commands
22. Assignment Commands
23. Assignment Commands
24. Assignment Commands: Swap
25. Swap
26. Swap
27. Swap

Imperative Programming:

(500-524)

1. Imperative vs Functional
2. Features of the Store
3. References: Experiments
4. References: Experiments
5. References: Experiments
6. Aliases
7. References: Experiments
8. References: Aliases
9. References: Experiments
10. After Garbage Collection
11. Side Effects
12. Imperative ML



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 706 of 709

Go Back

Full Screen

Close

Quit

13. Imperative ML
14. Imperative ML
15. Imperative ML
16. Nasty Surprises
17. Imperative ML
18. Imperative ML
19. Common Errors
20. Aliasing & References
21. Dangling References
22. New Reference
23. Imperative Commands: Basic
24. Imperative Commands: Compound
25. Predefined Compound Commands

Arrays

1. Why Imperative
2. Arrays
3. Indexing Arrays
4. Indexing Arrays
5. Indexing Arrays
6. Physical Addressing
7. Arrays

(525-537)



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 707 of 709

Go Back

Full Screen

Close

Quit

8. 2D Arrays
9. 2D Arrays: Indexing
10. Ordering of indices
11. Arrays vs. Lists
12. Arrays: Physical
13. Lists: Physical

Large Example: Tautology Checking

(538-560)

1. Logical Arguments
2. Saintly and Rich
3. About Cats
4. About God
5. Russell's Argument
6. Russell's Argument
7. Russell's Argument
8. Russell's Argument
9. Propositions
10. Compound Propositions
11. Valuations
12. Valuations
13. Tautology
14. Properties



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 708 of 709

Go Back

Full Screen

Close

Quit

15. Negation Normal Form
16. Literals & Clauses
17. Conjunctive Normal Form
18. Validity
19. Logical Validity
20. Validity & Tautologies
21. Problem
22. Tautology Checking
23. Falsifying

Tautology Checking Contd.

(561-574)

1. Tautology Checking
2. Normal Forms
3. Top-down Development
4. The Signature
5. The Core subproblem
6. The datatype
7. Convert to CNF
8. Rewrite into NNF
9. \iff and \Rightarrow Elimination
10. Push \neg inward
11. Push \neg inward



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 709 of 709

Go Back

Full Screen

Close

Quit

- 12. conj_of_disj
- 13. Push \vee inward
- 14. Tautology & Falsification



IIT Delhi

S. Arun-Kumar, CSE

Title Page

Contents



Page 710 of 709

Go Back

Full Screen

Close

Quit