CS4215: Programming Language Implementation
2018/9, Sem 2

Introduction

Lecture 1

Synopsis

1. Brief introduction to the course
2. Administrative matters
3. Introductory concepts
    a. Language processing
    b. Inductive definitions

Staff

- Razvan Voicu, adjunct instructor, staff engineer at Indeed.com
  - Course lecturer, will be grading your exams
  - razvan@comp.nus.edu.sg, rvoicu@indeed.com
- Assoc Prof Wei Ngan Chin
  - Kindly provides help with co-ordindinating the labs
  - Matters related to homework, homework submissions, homework marking
  - chinwn@comp.nus.edu.sg
- Lab TAs will be introduced later

IVLE

- Main medium for interaction and homework submission
- All materials will be there
- Lectures will have screencast recorded

What this course is about

- Implementation of major programming language concepts
- Distilled, focus on the gist, keep things as simple as possible
- Practical approach, we will be looking at code
- We will also focus on high-level abstract concepts such as semantics
    - Strive to keep a balance
    - Provide an understanding of how high-level solutions/concepts make their way to practice
    - Abstract as they may be, high level concepts are often subtle; understanding them well provides deep insights that can be crucial in finding good practical solutions.

Learn by Coding

- Nothing beats running your own interpreter or compiler
- Implement a sequence of toy language, each adding a new layer of complexity
  - Expressions
  - Statements
  - Types
  - Procedural/functional abstractions
  - Objects and Exceptions
  - Virtual machines
  - Write toy programs in your toy programming language
- Course revamp: everything will be based on Scala
  - Extensive software support provided

Incremental and Exploratory

- Incremental
  - Sequence of programming language, each adding a new layer.
  - Platform: start with interpreter and progress to virtual machines
- Exploratory
  - Code given out to experiment with
  - Miniproject on domain specific languages

Overview of Module Content

- Programming language processing tools and inductive definitions
- Scala as an implementation language
- ePL: An expression language
- simPL: A simple functional language
- polyPL: Polymorphism and exceptions
- dPL: Algebraic data types
- imPL: A simple imperative language
- oPL: A simple object-oriented language
- Domain-specific languages

Housekeeping

- Use IVLE
  - Discussions in the forum
  - Announcements
  - Assignments and assignment submission
- Notes and slides, no textbook
  - Web-based, links will be added to IVLE
- Tutorial cum lab sessions to focus on practical aspects

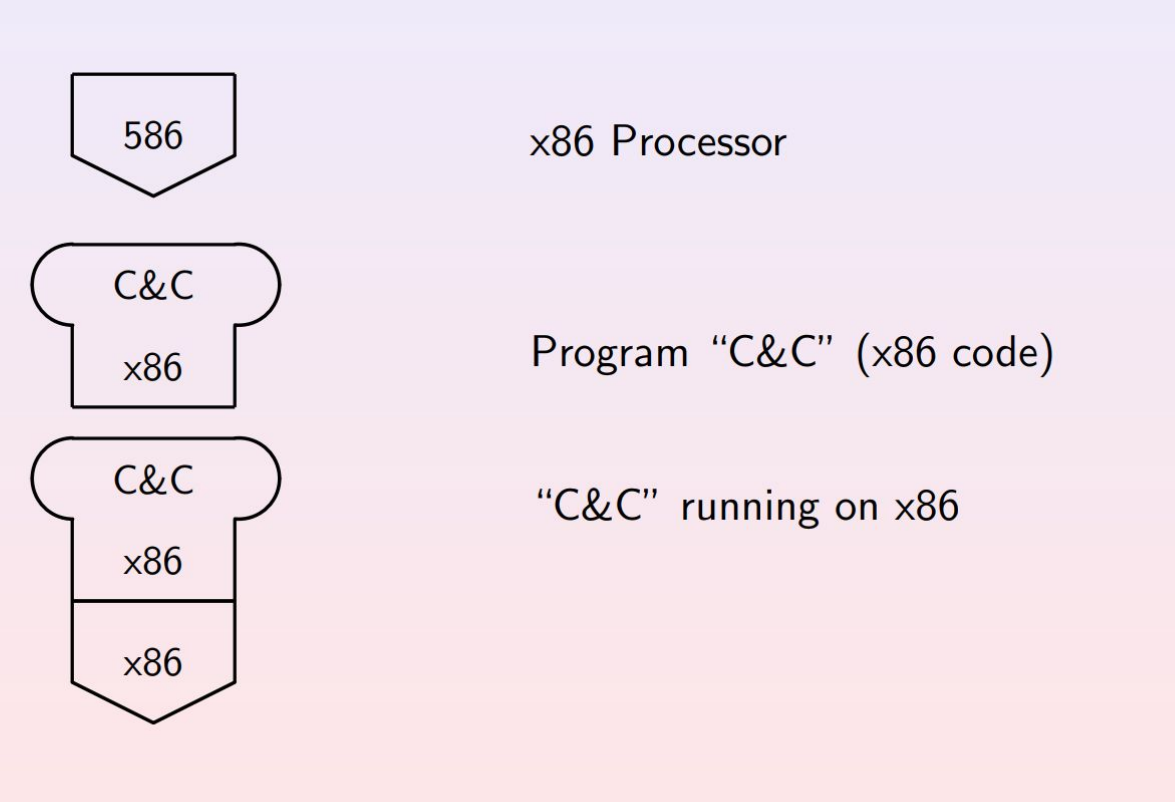
Assessment

- Labs and assignments: 30%
- Paper reading and presentations: 10%
- Mini-project on DSL: 15%
- Exam: 45%

Language Processing

- T-Diagrams
- Translators
- Interpreters
- Combinations (virtual machines)

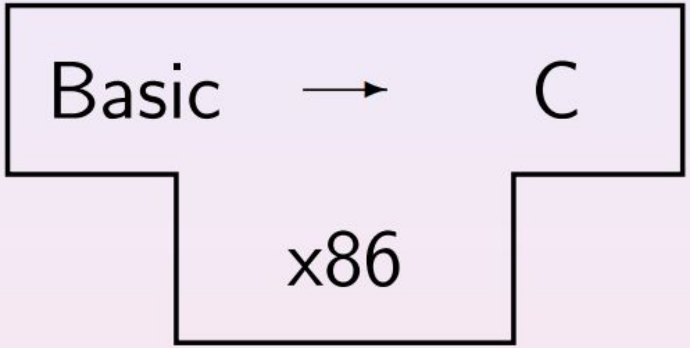
T-Diagrams


586
x86 Processor
C&C
x86
Program "C&C" (x86 code)
C&C
x86
x86
"C&C" running on x86

Translators

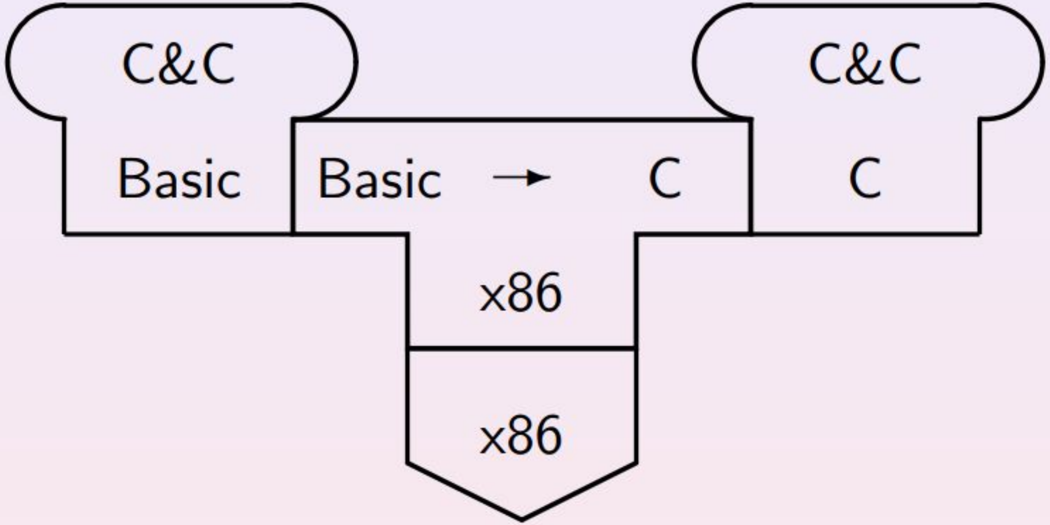
- Translate from one language (source) to another language (object)
- Compiler: translates from high(er)-level to a low(er) level language.
- De-compiler: low-level ⇒ high level

T-Diagram of Translator



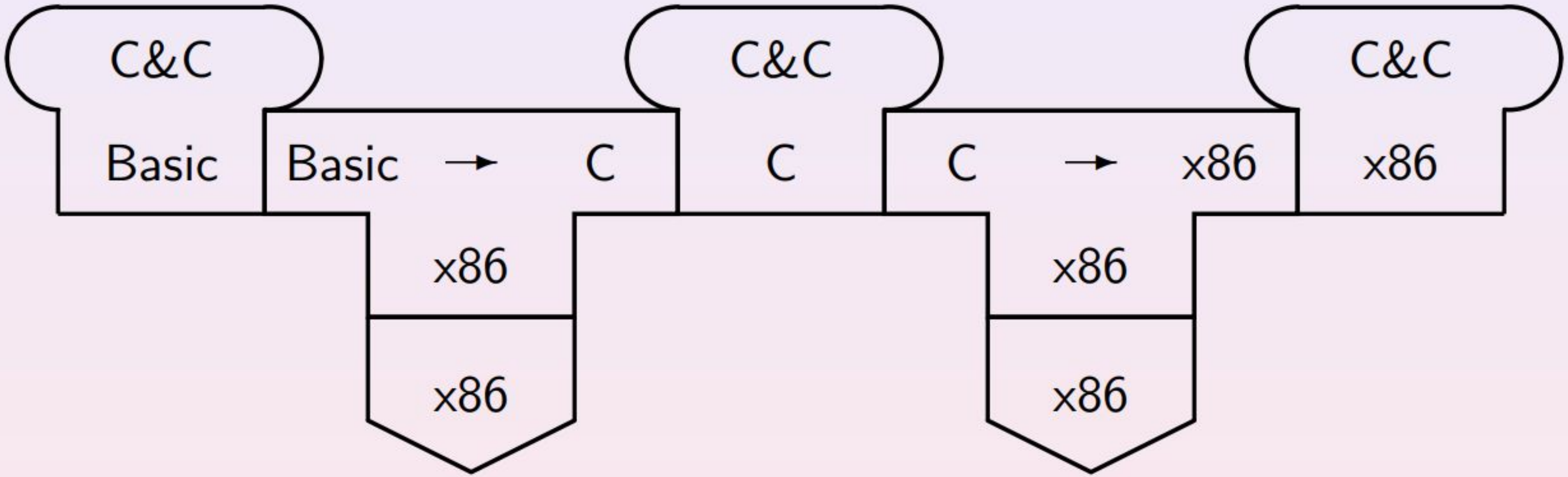
Basic-to-C compiler implemented in x86 machine code

Compilation



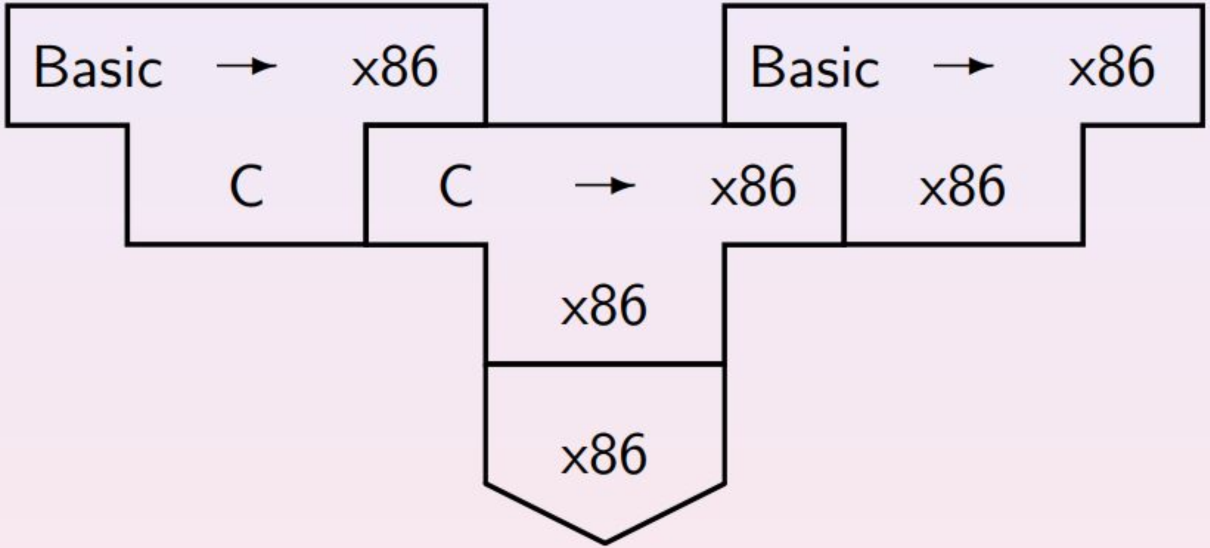
Compiling “C&C” from Basic to C

Two-Stage Compilation



Compiling "C&C" from Basic to C to x86 machine code

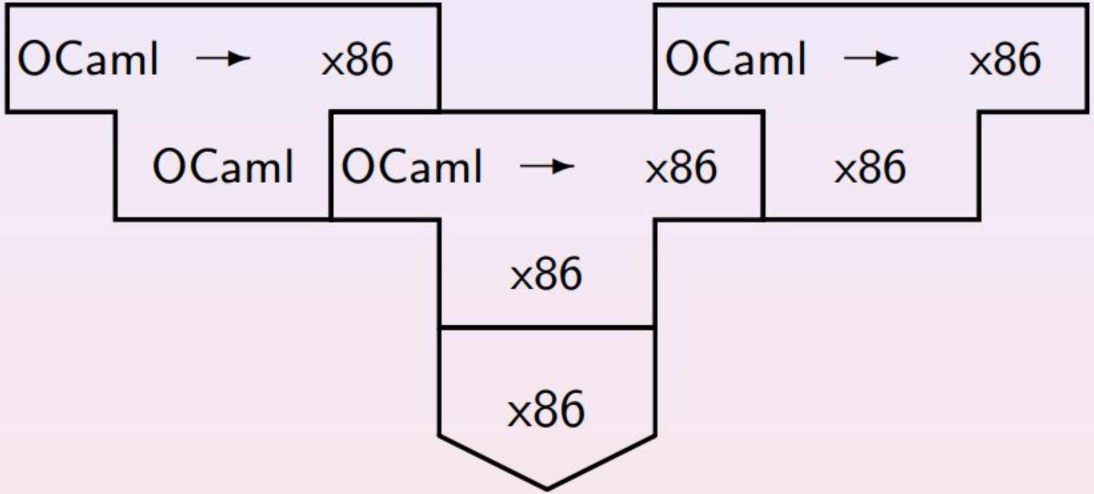
Compiling a Compiler



Compiling a Basic-to-x86 compiler from C to x86 machine code

Bootstrapping a Compiler

Chicken and Egg Problem

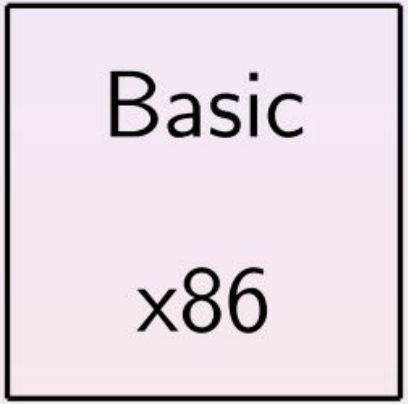


Compiling a OCaml-to-x86 compiler implemented in OCaml to run natively on x86 machine code

Interpreter

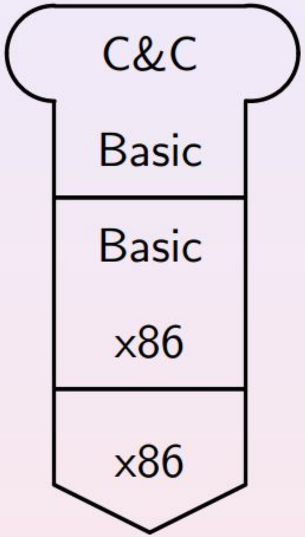
- A program that executes another program
- The interpreter runs throughout the entire execution
- The target program is interpreted, and appears as running to the user.

T-Diagram for Basic Interpreter



Interpreter for Basic, implemented in x86 machine code

Interpreting a Program

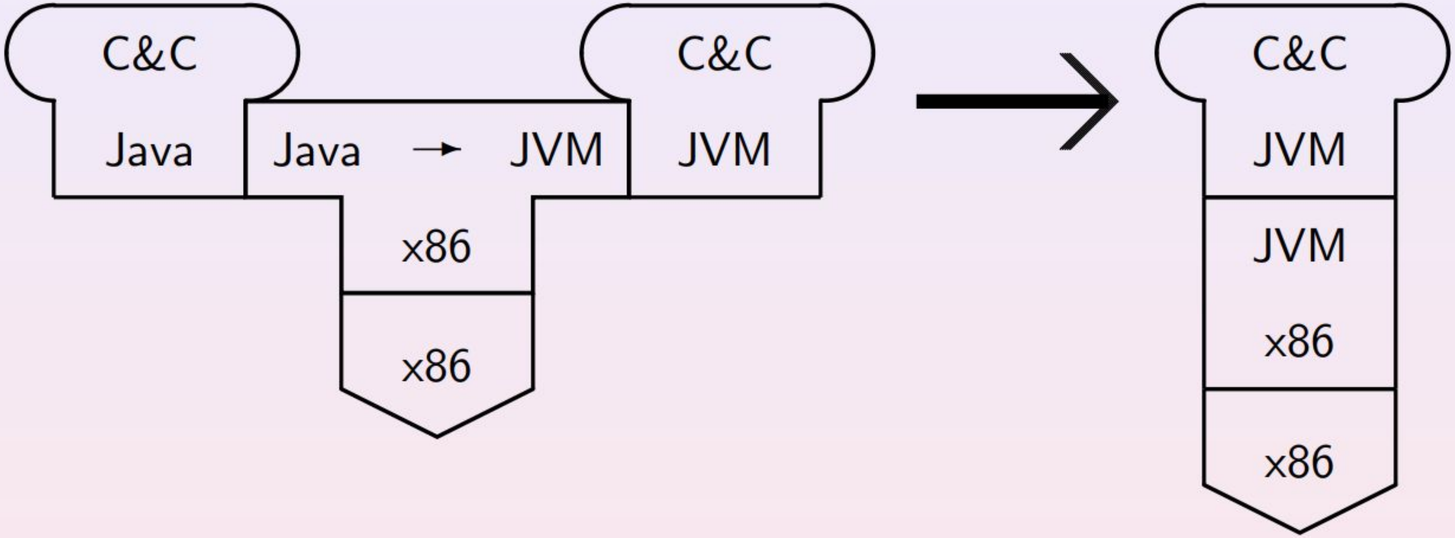


Basic program “C&C”
running on x86 using interpretation

Hardware Emulation

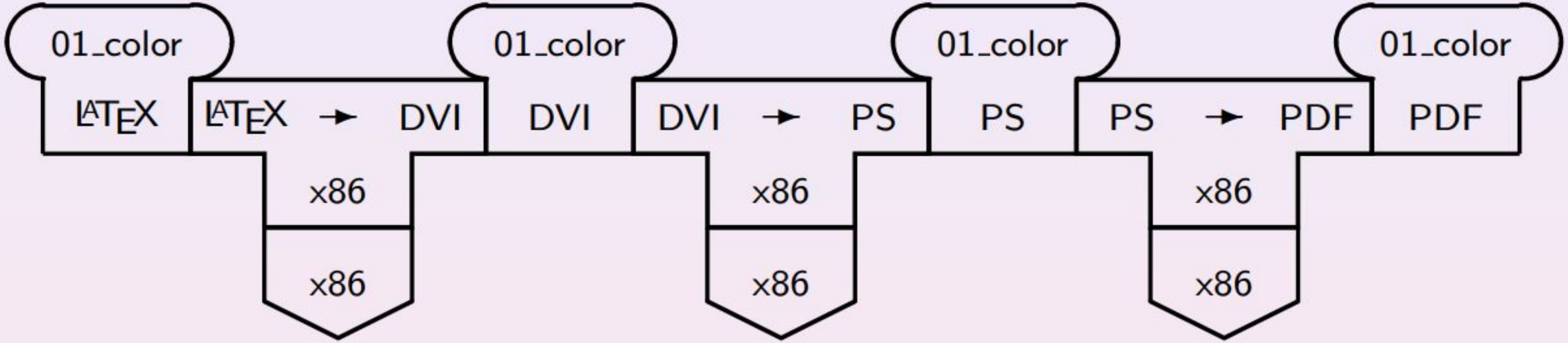
"C&C" x86 executable running on a PowerPC using hardware emulation

Typical Execution of Java Programs



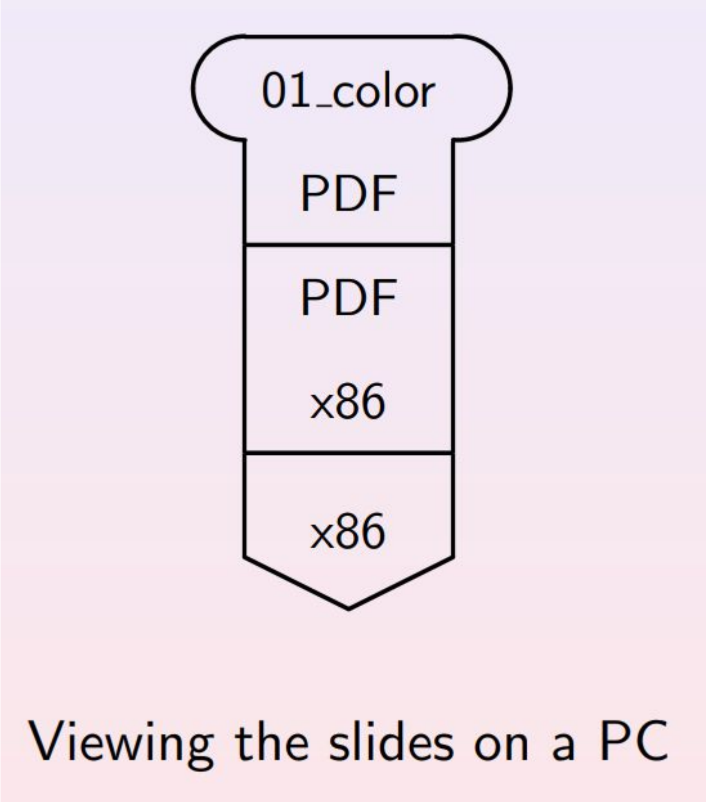
Compiling "C&C" from Java to JVM code, and running the JVM code on a JVM running on an x86

Excursion: Making Slides



Compiling these slides
from LaTeX to DVI to PostScript to PDF on x86 (PC)

Excursion: Viewing Slides



Summary: Language Processing

- Components: programs, translators, interpreters, machines
- T-diagrams
- Combination of interpretation and compilation is common
- Interpretation and compilation are ubiquitous in computing

Inductive Definitions

- Definition
- Extremal clause
  极值
- Proofs by induction

Inductive Definitions

- Set of rules that un-equivocally define mathematical objects
  - Example: the set of programs for a particular programming language

Example: Numerals

Numerals, in unary (base-1) notation

- $Zero$ is a numeral;
- if n is a numeral, then so is $Succ(n)$.

Examples

- $Zero$
- $Succ(Succ(Succ(Zero)))$

Binary Trees

Binary trees (w/o data at nodes)

- $Empty$ is a binary tree;
- if l and r are binary trees, then so is $Node(l, r)$.

Examples

- $Empty$
- $Node(Node(Empty, Empty), Node(Empty, Empty))$

More formally

- Numerals: The set Num is defined by the rules

$$\frac{}{Zero \in Num} \qquad \frac{n \in Num}{Succ(n) \in Num}$$

- Binary trees: The set $Tree$ is defined by the rules

$$\frac{}{Empty \in Tree} \qquad \frac{t_l \in Tree \qquad t_r \in Tree}{Node(t_l, t_r) \in Tree}$$

Examples

- Numerals: The set *Num* is defined by the rules

$$\frac{}{Zero} \qquad\qquad \frac{n}{Succ(n)}$$

- Binary trees: The set *Tree* is defined by the rules

$$\frac{}{Empty} \qquad\qquad \frac{t_l \quad t_r}{Node(t_l, t_r)}$$

Defining a Set By Rules

- Given a collection of rules, what set does it define?
  - What is the set of Numerals?
  - What is the set of Trees?
- Do the rules pick out a unique set

Defining a Set by Rules

- There can be many sets that satisfy a given collection of rules.
  - $Num = \{Zero, Succ(Zero), \ldots\}$
  - $StrangeNum = Num \cup \{\infty, Succ(\infty), \ldots\}$, where ∞ is an arbitrary symbol
- Both *Num* and *StrangeNum* satisfy the rules defining numerals (i.e., the rules are true for these sets). Really?

Num Satisfies the Rules

$$\frac{}{Zero \in Num} \qquad \frac{n \in Num}{Succ(n) \in Num}$$

$Num = \{Zero, Succ(Zero), Succ(Succ(Zero)), \ldots\}$

Does *Num* satisfy the rules?

- $Zero \in Num$. $\surd$
- If $n \in Num$, then $Succ(n) \in Num$. $\surd$

StrangeNum Satisfies the Rules

$$\frac{}{Zero \in Num} \qquad \frac{n \in Num}{Succ(n) \in Num}$$

$StrangeNum =$
$\{Zero, Succ(Zero), Succ(Succ(Zero)), \ldots\} \cup \{\infty, Succ(\infty), \ldots\}$
Does *StrangeNum* satisfy the rules?

- $Zero \in StrangeNum$. $\surd$
- If $n \in StrangeNum$, then $Succ(n) \in StrangeNum$. $\surd$

This is despite the fact that $\infty \notin StrangeNum$.

Defining Sets by Rules

- Both *Num* and *StrangeNum* satisfy all rules.
- It is not enough that a set satisfies all rules
- Extremal clause:
  - “And nothing else”
  - “The least set that satisfies these rules”

Inductive Definitions

- An inductively defined set is the least set that satisfies a given set of rules.
- Example: Num is the least set that satisfies these rules:
  - $\mathit{Zero} \in \mathit{Num}$
  - if $n \in \mathit{Num}$, then $\mathit{Succ}(n) \in \mathit{Num}$.

Inductive Definitions

Question: What do we mean by "least"?
Answer: The smallest with respect to the subset ordering on sets.

- Contains no "junk", only what is required by the rules.
- Since $StrangeNum \supsetneq Num$, $StrangeNum$ is ruled out by the extremal clause.
- Num is "ruled in" because it has no "junk".

What is the Big Deal?

- Inductively defined sets "come with" an induction principle.
- Suppose I is inductively defined by rules R.
- To show that every $x \in I$ has property P, it is enough to show that P satisfies the rules of R.
- Sometimes called *structural induction* or *rule induction*.

Parity of Numerals

- The numeral *Zero* has parity **0**.
- Any numeral $Succ(n)$ has parity $1 - p$ if p is the parity of n
- Let P be the following property:
  Every numeral has either parity 0 or parity 1.

奇偶性

- Does P satisfy the rules $\dfrac{}{P(Zero)}$ $\dfrac{P(n)}{P(Succ(n))}$?

Induction Principle

- To show that every $n \in Num$ has property P, it is enough to show:
    - $Zero$ has property P.
    - if n has property P, then $Succ(n)$ has property P.
- This is just ordinary mathematical induction!

Induction Principle

- To show that every tree has property P, it is enough to show that
  - $Empty$ has property P.
  - if l and r have property P, then so does $Node(l, r)$.
- We call this *structural induction on trees*.

Example: Height of a Tree

- To show: Every tree has a height, defined as follows:
  - The height of *Empty* is 0.
  - If l has height h_l and the tree r has height h_r, then the tree $Node(l, r)$ has height $1 + max(h_l, h_r)$.
- Clearly, every tree has at most one height, but does it have a height at all?

Example: height

- It may seem obvious that every tree has a height
- Justification is based on structural induction
    - An infinite tree does not have height
    - But the extremal clause rules out the infinite tree!

Example: height

- Formally, we prove that for every tree *t*, there exists a number *h* satisfying the specification of height
- Proceed by induction on the rules defining trees, showing that the property "there exists a height *h* for *t*" satisfies these rules

Example: height

- **Rule 1:** *Empty* is a tree.
  - Does there exist h such that h is the height of *Empty*?
  - **Yes**: take $h=0$
- **Rule 2:** *Node(l, r)* is a tree if l and r are trees.
  - Suppose that there exists h_l and h_r, the heights of l and r, respectively.
  - Does there exist h such that h is the height of *Node(l,r)*?
  - **Yes:** Take $h=1+max(h_l,h_r)$.

Summary

- An inductively defined set is the least set that satisfies a collection of rules.
- Rules have the form:
  "If $x_1 \in X$ and ... and $x_n \in X$, then $x \in X$."
- Notation:

$$\frac{x_1 \in X \qquad \cdots \qquad x_n \in X}{x \in X}$$

Summary

- Inductively defined sets admit proofs by rule induction.
- For each set, with rules of the form:

$$\frac{x_1 \in X \qquad \cdots \qquad x_n \in X}{x \in X}$$

  We can proof this property inductively using:

$$\frac{P(x_1) \qquad \cdots \qquad P(x_n)}{P(x)}$$

- Conclude that every element of the set satisfies P.