Programming Languages
This Course

Jython in Java

High Level Languages

Relation

Java (Object Oriented)

ASP

RDF (Horn Clause Deduction, Semantic Web)

A Snapshot of Programming Language History
Programs

This book asks students to implement language features using a combination of interpreters and little compilers. All the programming is done in Scheme, which has the added benefit of making students fairly comfortable in a language and paradigm they may not have employed before. End-of-semester surveys reveal that students are far more likely to consider using Scheme for projects in other courses after taking this course than they were before it (even when they had prior exposure to Scheme).

Though every line of code in this book has been tested and is executable, I purposely do not distribute the code associated with this book. While executable code greatly enhances the study of programming languages, it can also detract if students execute the code mindlessly. I therefore ask you, Dear Reader, to please type in this code as if you were writing it, paying close attention to every line. You may be surprised by how much many of them have to say.

Course Schedule

The course follows approximately the following schedule:

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction, Scheme tutorials, Modeling Languages</td>
</tr>
<tr>
<td>2-3</td>
<td>Substitution and Functions</td>
</tr>
<tr>
<td>3</td>
<td>Laziness</td>
</tr>
<tr>
<td>4</td>
<td>Recursion</td>
</tr>
<tr>
<td>4</td>
<td>Representation Choices</td>
</tr>
<tr>
<td>4-5</td>
<td>State</td>
</tr>
<tr>
<td>5-7</td>
<td>Continuations</td>
</tr>
<tr>
<td>7-8</td>
<td>Memory Management</td>
</tr>
<tr>
<td>8-10</td>
<td>Semantics and Types</td>
</tr>
<tr>
<td>11</td>
<td>Programming by Searching</td>
</tr>
<tr>
<td>11-12</td>
<td>Domain-Specific Languages and Metaprogramming</td>
</tr>
</tbody>
</table>

Miscellaneous “culture lecture” topics such as model checking, extensibility and future directions consume another week.

An Invitation

I think the material in these pages is some of the most beautiful in all of human knowledge, and I hope any poverty of presentation here doesn’t detract from it. Enjoy!
Concepts - Some things that computer science majors should know

Dr. Philip Cannata
"What is often not realized is that the knowledge gleaned through such interpreter implementation exercises has far reaching applicability."
6. Side Effects and Purity

Often, it is only after studying the semantics of a language with side effects that students are able to reliably distinguish the notion of variable rebind.

7. Avoiding copy-and-paste coding

Understanding the advantage of factoring out common code to keep a single point of control.

8. Continuations

The notion of a continuation, as a way to talk about what remains to be done in a computation, is a basic notion that, like decidability or algorithmic complexity, should be part of the vocabulary of anyone.

9. Term Rewriting Systems

```
root_1 = (PythonTree)adaptor.becomeRoot(new SQLInsert(INSERT, INSERT90,
    ((sqlinsert_stmt_scope)sqlinsert_stmt_stack.peek()).table,
    (java.util.List<String>)((sqlinsert_stmt_scope)sqlinsert_stmt_stack.peek()).attrs,
    (java.util.List<expr>)((sqlinsert_stmt_scope)sqlinsert_stmt_stack.peek()).exprs
    )
)
```

10. Conclusion

Many students are not aware of the nature of the material studied in the field of programming languages. A general misconception is that a course on programming languages should provide merely a comparative study of various languages, rather than underlying concepts and ways of thinking about computation.

“Many students are not aware of the nature of the material studied in the field programming languages. A general misconception is that a course on programming languages should provide merely a comparative study of various languages, rather than **underlying concepts** and ways of thinking about computation.”
# CS345 Programming Language Concepts

<table>
<thead>
<tr>
<th>Concept</th>
<th>Notes</th>
<th>Concept</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>HLL1 &amp; HLL2</td>
<td>Functions</td>
<td>HLL4</td>
</tr>
<tr>
<td>Terminal</td>
<td>HLL1 &amp; HLL2</td>
<td>Parameters and Arguments</td>
<td>HLL4</td>
</tr>
<tr>
<td>Non-terminal</td>
<td>HLL1 &amp; HLL2</td>
<td>Parameter Passing</td>
<td>HLL4</td>
</tr>
<tr>
<td>Lexical Analysis</td>
<td>HLL1 &amp; HLL2</td>
<td>Activation Records</td>
<td>HLL4</td>
</tr>
<tr>
<td>Regular Expression</td>
<td>HLL1 &amp; HLL2</td>
<td>Runtime Stack</td>
<td>HLL4</td>
</tr>
<tr>
<td>Syntactic Analysis</td>
<td>HLL1 &amp; HLL2</td>
<td>Side Effects</td>
<td>HLL4</td>
</tr>
<tr>
<td>Regular Grammar</td>
<td>HLL1 &amp; HLL2</td>
<td>First-Class Functions</td>
<td>HLL4 Chapters 5 &amp; 6</td>
</tr>
<tr>
<td>Context-free Grammar</td>
<td>HLL1 &amp; HLL2</td>
<td>Recursion</td>
<td>HLL4</td>
</tr>
<tr>
<td>BNF</td>
<td>HLL1 &amp; HLL2</td>
<td>Closure</td>
<td>HLL4</td>
</tr>
<tr>
<td>EBNF</td>
<td>HLL1 &amp; HLL2</td>
<td>Tail Recursion</td>
<td>HLL4</td>
</tr>
<tr>
<td>Parse Tree</td>
<td>HLL1 &amp; HLL2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Associativity</td>
<td>HLL1 &amp; HLL2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precedence</td>
<td>HLL1 &amp; HLL2</td>
<td>Haskell</td>
<td>Haskell1 Chapters 7 &amp; 8</td>
</tr>
<tr>
<td>Ambiguous Grammar</td>
<td>HLL1 &amp; HLL2</td>
<td>Haskell Expression</td>
<td>Haskell1</td>
</tr>
<tr>
<td>Abstract Syntax</td>
<td>HLL1 &amp; HLL2</td>
<td>Haskell Functions</td>
<td>Haskell1</td>
</tr>
<tr>
<td>Abstract Syntax Tree</td>
<td>HLL1 &amp; HLL2</td>
<td>Haskell Tuples</td>
<td>Haskell1</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Chapters 1 &amp; 2</td>
<td>Haskell Pattern Matching</td>
<td>Haskell1</td>
</tr>
<tr>
<td>Symbol Table and Dictionaries</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Lazy Evaluation</td>
<td>Haskell1 Chapters 7 &amp; 8</td>
</tr>
<tr>
<td>Scope</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Strong Type System</td>
<td>Haskell1</td>
</tr>
<tr>
<td>Nested and Dynamic Scope</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Propositional Logic</td>
<td>Haskell2</td>
</tr>
<tr>
<td>Local and Non-Local Variables</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Haskell and SQL</td>
<td>Haskell2</td>
</tr>
<tr>
<td>Static and Dynamic Scoping</td>
<td>HLL3 &amp; PLA1 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolving References</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Syllogisms</td>
<td>Syllogism and Midder Review</td>
</tr>
<tr>
<td>Overloading</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Proof by Contradiction</td>
<td>Syllogism and Midder Review</td>
</tr>
<tr>
<td>Variables</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Godel's Incompleteness Theorem</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Static and Dynamic Binding</td>
<td>HLL3 &amp; PLA1 3</td>
<td>Deductive Logic</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Types</td>
<td></td>
<td>Horn Clause</td>
<td>Prolog 1 &amp; 2 Chapters 32 - 34</td>
</tr>
<tr>
<td>Type System</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Prolog Facts and Rules</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Static and Dynamic Typing</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Prolog Lists</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Implicit Type Conversion</td>
<td>PLA1 Types &amp; HLL2</td>
<td>Proof by Contradiction</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Type System</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Unification</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Type Judgments</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Resolutions</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Type Judgment Tree</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Backtracking</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Type Soundness</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Constraint Satisfaction Programming</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Type Safety</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Prolog and SQL</td>
<td>Prolog 1 &amp; 2</td>
</tr>
<tr>
<td>Type Inferencing</td>
<td>Chapters 24 - 29 &amp; 31</td>
<td>Semantic Web Programming</td>
<td>Semantic Web &amp; Semantic DBs and OO PLs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relations</th>
<th>PLA1 1</th>
<th>Semantic Web Programming</th>
<th>Semantic Web &amp; Semantic DBs and OO PLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lambda Recursive Functions</td>
<td>Genesis</td>
<td>Compressions</td>
<td>Prolog 1, Page 9</td>
</tr>
<tr>
<td>Combinator Logic</td>
<td>Genesis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lisp</td>
<td>PLA1 1</td>
<td>Kolmogorov Complexity</td>
<td></td>
</tr>
</tbody>
</table>
Good Books to Have for a Happy Life 😊

From Frege to Gödel:

- From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931
- Lambda-Calculus and Combinators: An Introduction
- A Profile of Mathematical Logic
- Gödel's Proof
- Gödel, Escher, Bach: An Eternal Golden Braid

My Favorite

Dr. Philip Cannata
WEB DATABASE (WDB):
A JAVA SEMANTIC DATABASE
by
Bo Li
A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science In Computer Sciences: Turing Scholars Honors
University of Texas at Austin
Spring, 2006
View Thesis

CLASS Person "Persons related to the company"
(
    person-id : INTEGER, REQUIRED;
    first-name : STRING, REQUIRED;
    last-name : STRING, REQUIRED;
    home_address : STRING;
    zipcode : INTEGER;
    home-phone "Home phone number (optional)" : INTEGER;
    us-citizen "U.S. citizenship status" : BOOLEAN, REQUIRED;
    spouse "Person's spouse if married" : Person, INVERSE IS spouse;
    children "Person's children (optional)" : Person, MV (DISTINCT), INVERSE IS children;
    parents "Person's parents (optional)" : Person, MV (DISTINCT, MAX 2), INVERSE IS children;
);

Complete and Expand
This will enable SIM and SPARQL queries against the data – possibly great for Semantic Data Integration

Replace this with RDF Triple Store
Review for Final Exam

Midterm Review

Plus Homework

Plus Quizzes

Plus next pages
Haskell
**Lazy Evaluation:**

\[
v = 1/0 \\
\text{testLazy } x = 2 + 10 \\
\text{testLazy1 } x = 2 / x
\]

```haskell
*Haskell> v
Infinity
* Haskell > testLazy 22
12
* Haskell > testLazy v
12
* Haskell > testLazy1 22
9.090909090909091e-2
* Haskell > testLazy1 v
0.0
* Haskell > testLazy1 1/0
Infinity
```

**Strong Typing:**

\[
\text{logEquiv2 :: (Bool -> Bool -> Bool) -> (Bool -> Bool -> Bool) -> Bool}
\]

```haskell
\text{logEquiv2 bf1 bf2 =}
\text{and } [(bf1 r s) <=> (bf2 r s) | r <- [True,False], s <- [True,False]]
```

**List Comprehension:**

```haskell
\text{Lambda Expressions and Functions as First Class Objects:}
\text{logEquiv2 } (\ p q -> p ==> q) (\ p q -> \not p \mid q)
```
Propositional Logic

Propositions:
Statements that can be either True or False

Logical Operators:
• Negation: not
  not :: Bool -> Bool
  not True = False
  not False = True
• Conjunction: &&
  (&&) :: Bool -> Bool -> Bool
  False && x = False
  True && x = x
• Disjunction: ||
  (||) :: Bool -> Bool -> Bool
  True || x = True
  False || x = x

Logical Operators:
• Implication (if – then): ==> Antecedent ==> Consequent
  (==>) :: Bool -> Bool -> Bool
  x ==> y = (not x) || y
• Equivalence (if, and only if): <=>
  (<=>) :: Bool -> Bool -> Bool
  x <=> y = x == y
• Not Equivalent <+>
  (<+>) :: Bool -> Bool -> Bool
  x <+> y = x /= y
**Truth tables:**

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P &amp;&amp; Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

| P    | Q    | P || Q |
|------|------|-------|
| False| False| False |
| False| True | True  |
| True | False| True  |
| True | True | True  |

<table>
<thead>
<tr>
<th>P</th>
<th>(\neg P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P (\rightarrow) Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P (\leftrightarrow) Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P (&lt;\leftrightarrow&gt;) Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Reasoning with Truth Tables

Proposition (WFF): \((P \lor Q) \land \((\neg P) \rightarrow Q))\)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>((P \lor Q))</th>
<th>((\neg P))</th>
<th>((\neg P) \rightarrow Q))</th>
<th>((P \lor Q) \land ((\neg P) \rightarrow Q)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

If prop is True when all variables are True: 
\[ P, Q \models ((P \lor Q) \land (\neg P) \rightarrow Q))\]

Some True: prop is Satisfiable*

If they were all True: Valid / Tautology

All False: Contradiction
(not satisfiable*)

*A Satisfiability was the first known NP-complete problem

A Truth double turnstile

Dr. Philip Cannata
Truth Table Application

\[ \text{truthTable :: (Bool -> Bool -> Bool) -> [Bool]} \]
\[ \text{truthTable wff = [ (wff p q) | p <- [True,False], q <- [True,False]]} \]

\[ \text{tt = \( \neg (p \rightarrow q) \)} \]

Hugs> :load 10Logic.hs
LOGIC> :type tt
tt :: Bool -> Bool -> Bool
LOGIC> truthTable tt
[False,True,False,False]
LOGIC> or (truthTable tt)
True
LOGIC> and (truthTable tt)
False
Satisfiable:
Are there well formed propositional formulas that return True for some input?

`satisfiable1 :: (Bool -> Bool) -> Bool
satisfiable1 wff = (wff True) || (wff False)

`satisfiable2 :: (Bool -> Bool -> Bool) -> Bool
satisfiable2 wff = or [ (wff p q) | p <- [True,False],
                      q <- [True,False]]

`satisfiable3 :: (Bool -> Bool -> Bool -> Bool) -> Bool
satisfiable3 wff = or [ (wff p q r) | p <- [True,False],
                      q <- [True,False],
                      r <- [True,False]]

( \ p -> not p)
( \ p q -> (not p) || (not q) )
( \ p q r -> (not p) || (not q) && (not r) )

Define these first

infix 1 ==> 

(==>) :: Bool -> Bool -> Bool
x ==> y = (not x) || y

infix 1 <=>

(<=>) :: Bool -> Bool -> Bool
x <==> y = x == y

infixr 2 <+>

(<+) :: Bool -> Bool -> Bool
x <+> y = x /= y
Validity (Tautology):
Are there well formed propositional formulas that return True no matter what their input values are?

valid1 :: (Bool -> Bool) -> Bool
valid1 wff = (wff True) && (wff False)

valid2 :: (Bool -> Bool -> Bool) -> Bool
valid2 wff = (wff True True)
& & (wff True False)
& & (wff False True)
& & (wff False False)

(\ p -> p || not p ) -- Excluded Middle
(\ p -> p ==> p )
(\ p q -> p ==> (q ==> p) )
(\ p q -> (p ==> q) ==> p )
Contradiction (Not Satisfiable):
Are there well formed propositional formulas that return False no matter what their input values are?

contradiction1 :: (Bool -> Bool) -> Bool
contradiction1 wff = not (wff True) && not (wff False)

contradiction2 :: (Bool -> Bool -> Bool) -> Bool
contradiction2 wff = and [not (wff p q) | p <- [True, False], q <- [True, False]]

contradiction3 :: (Bool -> Bool -> Bool -> Bool) -> Bool
contradiction3 wff = and [not (wff p q r) | p <- [True, False], q <- [True, False], r <- [True, False]]

( \ p -> p && not p)
( \ p q -> (p && not p) || (q && not q) )
( \ p q r -> (p && not p) || (q && not q) && (r && not r) )
Truth: \[ \models \]

Are there well formed propositional formulas that return True when their input is True

\[ \text{truth1} :: (\text{Bool} \rightarrow \text{Bool}) \rightarrow \text{Bool} \]
\[ \text{truth1} \ wff = (wff \ True) \]

\[ \text{truth2} :: (\text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool}) \rightarrow \text{Bool} \]
\[ \text{truth2} \ wff = (wff \ True \ True) \]

\[ (\ \ p \rightarrow \text{not} \ p) \]
\[ (\ \ p \ q \rightarrow (p \ \&\& \ q) \ || \ \text{(not} \ p \ \Rightarrow \ q)) \]
\[ (\ \ p \ q \rightarrow \text{not} \ p \ \Rightarrow \ q) \]
\[ (\ \ p \ q \rightarrow \text{(not} \ p \ \&\& \ q) \ \&\& \ (\text{not} \ p \ \Rightarrow \ q) \ ) \]
Equivalence:

logEquiv1 :: (Bool -> Bool) -> (Bool -> Bool) -> Bool
logEquiv1 bf1 bf2 =
    (bf1 True  <=> bf2 True) && (bf1 False <=> bf2 False)

logEquiv2 :: (Bool -> Bool -> Bool) -> (Bool -> Bool -> Bool) -> Bool
logEquiv2 bf1 bf2 =
    and [(bf1 r s) <=> (bf2 r s) | r <- [True,False],
        s <- [True,False]]

logEquiv3 :: (Bool -> Bool -> Bool -> Bool) -> (Bool -> Bool -> Bool -> Bool) -> Bool
logEquiv3 bf1 bf2 =
    and [(bf1 r s t) <=> (bf2 r s t) | r <- [True,False],
        s <- [True,False],
        t <- [True,False]]

formula3 p q = p
formula4 p q = (p <+> q) <+> q
formula5 p q = p <=> ((p <+> q) <+> q)

*Haskell> logEquiv2 formula3 formula4
True
*Haskell> logEquiv2 formula4 formula5
False
### Equivalence continued:

<table>
<thead>
<tr>
<th>Logical Equivalence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>logEquiv1 id (\ p -&gt; not (not p))</code></td>
<td>-- Idempotence</td>
</tr>
<tr>
<td><code>logEquiv1 id (\ p -&gt; p &amp;&amp; p)</code></td>
<td>-- Idempotence</td>
</tr>
<tr>
<td>`logEquiv1 id (\ p -&gt; p</td>
<td></td>
</tr>
<tr>
<td>`logEquiv2 (\ p q -&gt; p ==&gt; q) (\ p q -&gt; not p</td>
<td></td>
</tr>
<tr>
<td><code>logEquiv2 (\ p q -&gt; not (p ==&gt; q)) (\ p q -&gt; p &amp; &amp; not q)</code></td>
<td>-- Contrapositive</td>
</tr>
<tr>
<td><code>logEquiv2 (\ p q -&gt; not p ==&gt; not q) (\ p q -&gt; q ==&gt; p)</code></td>
<td>-- Contrapositive</td>
</tr>
<tr>
<td><code>logEquiv2 (\ p q -&gt; p ==&gt; not q) (\ p q -&gt; q ==&gt; not p)</code></td>
<td>-- Contrapositive</td>
</tr>
<tr>
<td><code>logEquiv2 (\ p q -&gt; not p ==&gt; q) (\ p q -&gt; not q ==&gt; p)</code></td>
<td>-- Contrapositive</td>
</tr>
<tr>
<td><code>logEquiv2 (\ p q -&gt; p &lt;= &gt; q) (\ p q -&gt; (p ==&gt; q) &amp; &amp; (q ==&gt; p))</code></td>
<td>-- Commutativity</td>
</tr>
<tr>
<td>`logEquiv2 (\ p q -&gt; p &lt;= &gt; q) (\ p q -&gt; (p &amp; &amp; q)</td>
<td></td>
</tr>
<tr>
<td><code>logEquiv2 (\ p q -&gt; p &amp; &amp; q) (\ p q -&gt; q &amp; &amp; p)</code></td>
<td>-- Commutativity</td>
</tr>
<tr>
<td>`logEquiv2 (\ p q -&gt; p</td>
<td></td>
</tr>
<tr>
<td>`logEquiv2 (\ p q -&gt; not (p &amp; &amp; q)) (\ p q -&gt; not p</td>
<td></td>
</tr>
<tr>
<td>`logEquiv2 (\ p q -&gt; not (p</td>
<td></td>
</tr>
<tr>
<td><code>logEquiv3 (\ p q r -&gt; p &amp; &amp; (q &amp; &amp; r)) (\ p q r -&gt; (p &amp; &amp; q) &amp; &amp; r)</code></td>
<td>-- Associativity</td>
</tr>
<tr>
<td>`logEquiv3 (\ p q r -&gt; p</td>
<td></td>
</tr>
<tr>
<td>`logEquiv3 (\ p q r -&gt; p &amp; &amp; (q</td>
<td></td>
</tr>
<tr>
<td>`test9b logEquiv3 (\ p q r -&gt; p</td>
<td></td>
</tr>
</tbody>
</table>
Standard Oracle scott/tiger emp dept database

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>7639</td>
<td>KING</td>
<td>PRESIDENT</td>
<td>(null)</td>
<td>17-NOV-81</td>
<td>5000</td>
<td>(null)</td>
<td>10</td>
<td>10 ACCOUNTING</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>MANAGER</td>
<td>7839</td>
<td>01-MAY-81</td>
<td>2850</td>
<td>(null)</td>
<td>30</td>
<td>20 RESEARCH</td>
<td>DALLAS</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>MANAGER</td>
<td>7839</td>
<td>09-JUN-81</td>
<td>2450</td>
<td>(null)</td>
<td>10</td>
<td>30 SALES</td>
<td>CHICAGO</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
<td>7839</td>
<td>02-APR-81</td>
<td>2975</td>
<td>(null)</td>
<td>20</td>
<td>40 OPERATIONS</td>
<td>BOSTON</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>7556</td>
<td>09-DEC-81</td>
<td>3000</td>
<td>(null)</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
<td>7556</td>
<td>03-DEC-81</td>
<td>3000</td>
<td>(null)</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7366</td>
<td>SMITH</td>
<td>CLERK</td>
<td>7902</td>
<td>17-DEC-80</td>
<td>800</td>
<td>(null)</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>20-FEB-81</td>
<td>1600</td>
<td>300</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>SALESMAN</td>
<td>7698</td>
<td>22-FEB-81</td>
<td>1250</td>
<td>500</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>28-SEP-81</td>
<td>1250</td>
<td>1400</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>SALESMAN</td>
<td>7698</td>
<td>08-SEP-81</td>
<td>1500</td>
<td>0</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>CLERK</td>
<td>7798</td>
<td>12-JAN-83</td>
<td>1100</td>
<td>(null)</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>CLERK</td>
<td>7698</td>
<td>03-DEC-81</td>
<td>950</td>
<td>(null)</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>CLERK</td>
<td>7792</td>
<td>23-JAN-82</td>
<td>1300</td>
<td>(null)</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Standard Oracle scott/tiger emp dept database in Haskell

emp = [(7839, "KING", "PRESIDENT", 0, "17-NOV-81", 5000, 10),
      (7698, "BLAKE", "MANAGER", 7839, "01-MAY-81", 2850, 30),
      (7782, "CLARK", "MANAGER", 7839, "09-JUN-81", 2450, 10),
      (7566, "JONES", "MANAGER", 7839, "02-APR-81", 2975, 20),
      (7788, "SCOTT", "ANALYST", 7566, "09-DEC-82", 3000, 20),
      (7902, "FORD", "ANALYST", 7566, "03-DEC-81", 3000, 20),
      (7369, "SMITH", "CLERK", 7902, "17-DEC-80", 800, 20),
      (7499, "ALLEN", "SALESMAN", 7698, "20-FEB-81", 1600, 30),
      (7521, "WARD", "SALESMAN", 7698, "22-FEB-81", 1250, 30),
      (7654, "MARTIN", "SALESMAN", 7698, "28-SEP-81", 1250, 30),
      (7844, "TURNER", "SALESMAN", 7698, "08-SEP-81", 1500, 30),
      (7876, "ADAMS", "CLERK", 7788, "12-JAN-83", 1100, 20),
      (7900, "JAMES", "CLERK", 7698, "03-DEC-81", 950, 30),
      (7934, "MILLER", "CLERK", 7782, "23-JAN-82", 1300, 10)]

depth = [(10, "ACCOUNTING", "NEW YORK"),
         (20, "RESEARCH", "DALLAS"),
         (30, "SALES", "CHICAGO"),
         (40, "OPERATIONS", "BOSTON") ]
Main> [(empno, ename, job, sal, deptno) | (empno, ename, job, _, _, sal, deptno) \(\leftarrow\) emp]

[(7839,"KING","PRESIDENT",5000,10),
(7698,"BLAKE","MANAGER",2850,30),
(7782,"CLARK","MANAGER",2450,10),
(7566,"JONES","MANAGER",2975,20),
(7788,"SCOTT","ANALYST",3000,20),
(7902,"FORD","ANALYST",3000,20),
(7369,"SMITH","CLERK",800,20),
(7499,"ALLEN","SALESMAN",1600,30),
(7521,"WARD","SALESMAN",1250,30),
(7654,"MARTIN","SALESMAN",1250,30),
(7844,"TURNER","SALESMAN",1500,30),
(7876,"ADAMS","CLERK",1100,20),
(7900,"JAMES","CLERK",950,30),
(7934,"MILLER","CLERK",1300,10)]
Main> [(empno, ename, job, sal, deptno) | (empno, ename, job, _, _, sal, deptno) <- emp, \texttt{deptno \texttt{==} 10}]

[(7839,"KING","PRESIDENT",5000,10),
 (7782,"CLARK","MANAGER",2450,10),
 (7934,"MILLER","CLERK",1300,10)]

Main>
Main> [(empno, ename, job, sal, dname) | (empno, ename, job, _, _, sal, edeptno) <- emp, (deptno, dname, loc) <- dept, edeptno == deptno ]

Main>
Prolog
Haskell

head :: [a] -> a
head (x : _) = x

tail :: [a] -> [a]
tail (_ : xs) = xs

null :: [a] -> Bool
null [] = True
null (_ : _) = False

lastelem :: [a] -> a
lastelem [x] = x
lastelem (_ : xs) = lastelem xs

initelem :: [a] -> [a]
nul ( _ : _ ) = []
initelem (x : xs) = x : initelem xs

listlength :: [a] -> Int
listlength [] = 0
listlength (_ : l) = 1 + listlength l

sumList :: (Num a) => [a] -> a
sumList [] = 0
sumList (x : xs) = x + sumList xs

append :: [a] -> [a] -> [a]
append [] ys = ys
append (x : xs) ys = x : append xs ys

Prolog

head( [ X | _ ], X).
tail( [ _ | Xs ], Xs).
nul( [ ] ).
lastelem( [ X ], X).
lastelem( [ _ | Xs ], Y) :- lastelem(Xs, Y).

initelem( [ _ ], [ ]).
initelem( [ X | Xs ], [ X | Ys ]) :- initelem(Xs, Ys).

listlength( [ ] , 0).
listlength( [ _ | L ], N) :- listlength(L, N0), N is 1+N0.

sumList( [ ] , 0).
sumList( [ X | Xs], N):- sumList(Xs, N0), N is X+N0.

appendList( [ ] , Ys, Ys).
appendList( [X | Xs], Ys, [X | Zs]) :- appendList(Xs, Ys, Zs).

These are in 11Prolog2.p

Tail Recursion

Recursion is the root of computation since it trades description for time”
- Alan Perlis

The benefits of an elegant recursive description, but equivalent in space and time, to an iteration
- requires O(1) stack space

Dr. Philip Cannata
Horn Clause

parent(hank,ben).
parent(hank,denise).
pARENT(irene,ben).
pARENT(irene,denise).
pARENT(alice,carl).
pARENT(ben,carl).
pARENT(denise,frank).
pARENT(denise,gary).
pARENT(earl,frank).
pARENT(earl,gary).

grandparent(X,Z) :- parent(X,Y) , parent(Y,Z).

\[ \text{logEquiv2 (\ p q \rightarrow p \implies q) (\ p q \rightarrow \ not p \ || q)} \]

\[ \text{not( parent(X,Y), parent(Y,Z)) \ || \ grandparent(X,Z)} \]

\[ \text{logEquiv2 (\ p q \rightarrow \ not (p \& \& q)) (\ p q \rightarrow \ not p \ || \ not q)} \]

\[ \text{not(parent(X,Y)) \ || \ not(parent(Y,Z)) \ || \ grandparent(X,Z)} \]

A Horn clause is a disjunction of Predicates in which at most one of the Predicates is not negative.
Horn Clause?

reads(X) || writes(X) :- literate(X).

logEquiv2 (\ p q -> p ==> q) (\ p q -> not p || q)

not(literate(X)) || reads(X) || writes(X)

Prolog only deals with Horn Clauses
**Syllogisms**

P implies Q (i.e., Q is True if P is True or If P is True then Q is True)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>Q → P</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Valid

Invalid - One of Aristotle's 13 Fallacies

**Modus Ponens**

- 1). P is True; 2). P → Q is True, therefore Q is True.

**Modus Tollens**

- If the result is just True, then the syllogism is Valid.
- If the results are True and False, then the syllogism is Invalid.

**Denying the Antecedent**

- 3). If P is True then Q is True. Therefore if P is False, then Q is False.

**Mixed Hypothetical Syllogisms**

**Conjunctive Syllogisms**

**Disjunctive Syllogisms**

**Modus ponendo tollens**

**Modus tollendo ponens**

Syllogism – e.g., Modus Ponens – 1). P is True; 2). P → Q is True, therefore Q is True.

Green – assume 2 things; the implied Result is in Red; if the result is just True, then the syllogism is Valid, if the results are True and False, then the syllogism is Invalid.
Proof by Contradiction

1. Let $P =$ It’s raining, I’m outside (comma means “&&”)
2. $P_1.$ (P1 is True, i.e., it’s raining)
3. $P_2.$ (P2 is True, i.e., I’m outside)
4. $Q :- P =$ I’m wet :- It’s raining, I’m outside. (if it’s raining and I’m outside then I’m wet)
5. $\neg Q$ (To answer the Query “Am I wet” against the Database, assume I’m not wet)
6. $\neg (It’s raining, I’m outside)$ (From 4 and 5 and Pattern 1)
7. $\neg I’m outside$ (From 2 and 6 and Pattern 2)
8. Contradiction – Therefore I’m wet (From 3 and 7 and Pattern 3)
Proof by Contradiction, Unification, Resolution and Backtracking

Pattern 1:
Q :- (P1, P2).
-Q
⇒ -(P1, P2)

Pattern 2:
P1.
-(P1, P2)
⇒ -P2

Pattern 3:
P2.
-P2
⇒ Contradiction

1). parent(hank, ben).
2). parent(ben, carl).
3). parent(ben, sue).
4). grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
5). –grandparent(A, B)
   (Unify A to X) (Unify B to Z) then Resolve 5 & 4
6). –(parent(A, Y), parent(Y, B)).
   (Unify A to hank) (Unify Y to ben)
   (Unify B to carl) then Resolve 6 & 1
7). –parent(ben, carl)
Contradiction  ⇒ grandparent(hank, carl)
Backtrack to 6 and
   (Unify B to sue) then Resolve 6 & 1
9). –parent(ben, sue)
Contradiction  ⇒ grandparent(hank, sue)
1). parent(hank, ben).
2). parent(ben, carl).
3). parent(ben, sue).
4). grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
5). –grandparent(A, B)
   (Unify A to X) (Unify B to Z) then Resolve 5 & 4
6). –(parent(A, Y), parent(Y, B)).
   (Unify A to hank) (Unify Y to ben)
   (Unify B to carl) then Resolve 6 & 1
7). –parent(ben, carl)
Contradiction  ⇒ grandparent(hank, carl)
Backtrack
   (Unify B to sue) then Resolve 6 & 1
9). –parent(ben, sue)
Contradiction  ⇒ grandparent(hank, sue)
Proof by Contradiction, Unification, Resolution and Backtracking

Pattern 1:
Q :- (P1, P2).
-Q
→ - (P1, P2)

Pattern 2:
P1.
-(P1, P2)
→ - P2

Pattern 3:
P2.
-P2
→ Contradiction

1). factorial(0, 1).
2). factorial(N, Result) :- N > 0, M is N -1, factorial(M, S), Result is N * S.
3). –factorial(2, X)
   (Unify 2 to N) (Unify X to Result) then Resolve 3 & 2
6). –(2 > 0, M is 1, factorial(1, S), X is 2 * S.
   (Unify 1 to N) (Unify S to Result) then Resolve 6 & 2
7). –(1 > 0, M is 0, factorial(0, S1), S is 1 * S1.
   (Unify 0 to N) (Unify S1 to Result) then Resolve 7 & 2
9). –factorial(0, 1)

Contradiction \(\rightarrow\) There is a factorial 2, now return from the proof with the answer.
Proof by Contradiction, Unification, Resolution and Backtracking

1). factorial(0, 1).
2). factorial(N, Result) :- N > 0, M is N -1, factorial(M, S), Result is N * S.
3). –factorial(2, _16)
   (Unify 2 to N) (Unify _16 to Result) then Resolve 3 & 2
6). –(2 > 0, _113 is 1, factorial(1, _138), _16 is 2 * _138.
   (Unify 1 to N) (Unify _138 to Result) then Resolve 6 & 2
7). –(1 > 0, _190 is 0, factorial(0, S_215), _138 is 1 * _215.
   (Unify 0 to N) (Unify S_215 to Result) then Resolve 7 & 2
9). –factorial(0, 1)
Contradiction  → There is a factorial of 2, now return from the proof with the answer.

Note: the trace shows _243 is 1*1 but then that value gets moved into _138

GNU Prolog console
File Edit Terminal Help
c:Documents and Settings\Nicolette Cannata

yes
| ?- trace.
uncaught exception: error(existence_error
| ?- trace.
The debugger will first creep -- showing

(l16 ms) yes
{trace}
| ?- factorial(2,X).
   1   1 Call: factorial(2,_16) ?
   2   2 Call: 2>0 ?
   2   2 Exit: 2>0 ?
   3   2 Call: _113 is 2-1 ?
   3   2 Exit: 1 is 2-1 ?
   4   2 Call: factorial(1,_138) ?
   5   3 Call: 1>0 ?
   5   3 Exit: 1>0 ?
   6   3 Call: _190 is 1-1 ?
   6   3 Exit: 0 is 1-1 ?
   7   3 Call: factorial(0,_215) ?
   7   3 Exit: factorial(0,1) ?
   8   3 Call: _243 is 1*1 ?
   8   3 Exit: 1 is 1*1 ?
   4   2 Exit: factorial(1,1) ?
   9   2 Call: _16 is 2*1 ?
   9   2 Exit: 2 is 2*1 ?
  1   1 Exit: factorial(2,2) ?

X = 2 ?

yes
{trace}
| ?-
7839's manager is 0

mgr(0, 7839).
mgr(7839, 7698).
mgr(7839, 7782).
mgr(7839, 7566).
mgr(7566, 7788).
mgr(7566, 7902).
mgr(7902, 7369).
mgr(7698, 7499).
mgr(7698, 7521).
mgr(7698, 7654).
mgr(7698, 7844).
mgr(7788, 7876).
mgr(7698, 7900).
mgr(7782, 7934).
mgrmgr(X, Y) :- mgr(X, Y).
mgrmgr(X, Y) :- mgr(Z, Y), mgrmgr(X, Z).

X = 7839 ? ;
1 1 Redo: mgrmgr(7839, 7698) ?
2 2 Redo: mgr(7839, 7698) ?
2 2 Fail: mgr(_16, 7698) ?
2 2 Call: mgr(_65, 7698) ?
2 2 Exit: mgr(7839, 7698) ?
3 2 Call: mgrmgr(_16, 7839) ?
4 3 Call: mgr(_16, 7839) ?
4 3 Exit: mgr(0, 7839) ?
3 2 Exit: mgrmgr(0, 7839) ?
1 1 Exit: mgr(0, 7698) ?

X = 0 ? ;
1 1 Redo: mgrmgr(0, 7698) ?
3 2 Redo: mgrmgr(0, 7839) ?
4 3 Redo: mgr(0, 7839) ?
4 3 Fail: mgr(_16, 7839) ?
4 3 Call: mgr(_134, 7839) ?
4 3 Exit: mgr(0, 7839) ?
5 3 Call: mgrmgr(_16, _0) ?
6 4 Call: mgr(_16, _0) ?
6 4 Fail: mgr(_16, _0) ?
6 4 Fail: mgr(_183, _0) ?
6 4 Fail: mgr(_171, _0) ?
5 3 Fail: mgrmgr(_16, _0) ?
4 3 Redo: mgr(0, 7839) ?
4 3 Fail: mgr(_122, 7839) ?
3 2 Fail: mgrmgr(_16, 7839) ?
2 2 Redo: mgr(7839, 7698) ?
2 2 Fail: mgr(_73, 7698) ?
1 1 Fail: mgrmgr(_16, 7698) ?

(15 ms) no
Building Problem part 1

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.

Pattern 1:  
Q :- (P1, P2).
-Q  
⇒ -(P1, P2)

Pattern 2:  
P1.
-(P1, P2)  
⇒ -P2

Pattern 3:  
P2.  
-P2  
⇒ Contradiction

floors([floor(_,5), floor(_,4), floor(_,3), floor(_,2), floor(_,1)]).
building(Floors) :- floors(Floors),
bmember(floor(baker, B), Floors),
bmember(floor(cooper, C), Floors),
bmember(floor(fletcher, F), Floors),
bmember(floor(miller, M), Floors),
bmember(floor(smith, S), Floors).
bmember(X, [X | _]).
bmember(X, [_ | Y]) :- bmember(X, Y).
?-building(X)
Building Problem part 2

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building. Baker doesn’t live on the 5th floor and Cooper doesn’t live on the 1st floor. Fletcher doesn’t live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper. Who lies on what floors?

```
floors([floor(_,5),floor(_,4),floor(_,3),floor(_,2),floor(_,1)]).
building(Floors) :- floors(Floors),
    bmember(floor(baker, B), Floors), B \= 5,
    bmember(floor(cooper, C), Floors), C \= 1,
    bmember(floor(fletcher, F), Floors), F \= 1, F \= 5,
    bmember(floor(miller, M), Floors), M > C,
    bmember(floor(smith, S), Floors), not(adjacent(S, F)),

print_floors(Floors).

print_floors(
    [A | B]) :- write(A), nl, print_floors(B).
print_floors([|]).

bmember(X, [X | _]).

Pattern 1:
Q :- (P1, P2).
-P1, P2

Pattern 2:
P1.
print_floors(Floors).
-P1, P2

Pattern 3:
bmember(X, [_ | Y]) :- bmember(X, Y).
P2.
adjacent(X, Y) :- X =:= Y+1.
-P2
adjacent(X, Y) :- X =:= Y-1.

?-building(X)
```

Dr. Philip Cannata
40
Standard Oracle emp / dept Database
emp/dep Database in Prolog

emp(7839, king, president, 0, 17-nov-81, 5000, 0, 10).
emp(7698, blake, manager, 7839, 01-may-81, 2850, 0, 30).
emp(7782, clark, manager, 7839, 09-jun-81, 2450, 0, 10).
emp(7566, jones, manager, 7839, 02-apr-81, 2975, 0, 20).
emp(7788, scott, analyst, 7566, 09-dec-82, 3000, 0, 20).
emp(7902, ford, analyst, 7566, 03-dec-81, 3000, 0, 20).
emp(7369, smith, clerk, 7902, 17-dec-80, 800, 0, 20).
emp(7499, allen, salesman, 7698, 20-feb-81, 1600, 300, 30).
emp(7521, ward, salesman, 7698, 22-feb-81, 1250, 500, 30).
emp(7654, martin, salesman, 7698, 28-sep-81, 1250, 1400, 30).
emp(7844, turner, salesman, 7698, 08-sep-81, 1500, 0, 30).
emp(7900, james, clerk, 7698, 03-dec-81, 950, 0, 30).
emp(7934, miller, clerk, 7782, 23-jan-82, 1300, 0, 10).

dep(10, accounting, new_york).
dep(20, research, dallas).
dep(30, sales, chicago).
dep(40, operations, boston).

select 'emp(' || empno || ', ' || lower(ename) || ', ' || lower(job) || ', ' || nvl(mgr, 0) || ', ' || lower(hiredate) || ', ' || sal || ', ' || nvl(comm, 0) || ', ' || deptno || ').' from emp

select 'dept(' || deptno || ', ' || lower(dname) || ', ' || lower(loc) || ').' from dept
emp / dept Database in Prolog

?- emp(Empno, Ename, Job, _, _, Sal, _, Deptno).

Deptno = 10
Empno = 7839
Ename = king
Job = president
Sal = 5000 ? ;

Deptno = 30
Empno = 7698
Ename = blake
Job = manager
Sal = 2850 ?

...
Dr. Philip Cannata

emp / dept Database in Prolog

| ?- emp(Empno, Ename, Job, _, _, Sal, _, 10).

Empno = 7839
Ename = king
Job = president
Sal = 5000 ? ;

Empno = 7782
Ename = clark
Job = manager
Sal = 2450 ? ;

Empno = 7934
Ename = miller
Job = clerk
Sal = 1300

yes
| ?- Haskell> [(empno, ename, job, sal, deptno) | (empno, ename, job, _, _, sal, deptno) <- emp, deptno == 10]
emp / dept Database in Prolog

?- emp(Empno, Ename, Job, _, _, Sal, _, D), dept(D, Dname, _).

D = 10
Dname = accounting
Empno = 7839
Ename = king
Job = president
Sal = 5000 ? ;

D = 30
Dname = sales
Empno = 7698
Ename = blake
Job = manager
Sal = 2850 ? ;

D = 10
Dname = accounting
Empno = 7782
Ename = clark
Job = manager
Sal = 2450 ?

Main> [(empno, ename, job, sal, dname) | (empno, ename, job, _, _, sal, edeptno) <- emp, (deptno, dname, loc) <- dept, edeptno == deptno ]
Types
# Type Judgments

<table>
<thead>
<tr>
<th>Type Judgment</th>
<th>Pseudo Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma</td>
<td>- L : \text{number} \quad \Gamma</td>
</tr>
<tr>
<td>$\quad \quad \quad \quad \quad \quad \quad \quad \Gamma</td>
<td>- (+ L R) : \text{number}$</td>
</tr>
<tr>
<td>$\Gamma</td>
<td>- f : (\tau_1 \rightarrow \tau_2) \quad \Gamma</td>
</tr>
<tr>
<td>$\quad \quad \quad \quad \quad \quad \quad \quad \Gamma</td>
<td>- (f a) : \tau_2$</td>
</tr>
<tr>
<td>$\Gamma</td>
<td>- c : \text{boolean} \quad \Gamma</td>
</tr>
</tbody>
</table>

**True in Hmm?**
Type Soundness

*Type soundness:* For all programs $p$, if the type checker assigns $p$ the type $\tau$, and if the semantics cause $p$ to evaluate to a value $v$ of type $t$, then the type checker will also have assigned $v$ the type $\tau$.

(first (list))

Shoud this

- Return a value such as $-1$.
- Diverge, i.e., go into an infinite loop.
- Raise an exception.
Type safety is the property that no primitive operation ever applies to values of the wrong type. By primitive operation we mean not only addition and so forth, but also procedure application. A safe language honors the abstraction boundaries it erects.

So what is “Strong Typing”? This appears to be a meaningless phrase, and people often use it in a nonsensical fashion. To some it seems to mean “The language has a type checker”. To others it means “The language is sound” (that is, the type checker and run-time system are related). To most, it seems to just mean, “A language like Pascal, C or Java, related in a way I can’t quite make precise”. If someone uses this phrase, be sure to ask them to define it for you. (For amusement, watch them squirm.)
Type Safety

<table>
<thead>
<tr>
<th>staticky checked</th>
<th>not statically checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>type safe</td>
<td>ML, Java</td>
</tr>
<tr>
<td>type unsafe</td>
<td>C, C++ (mainly because it allows embedded C)</td>
</tr>
<tr>
<td></td>
<td>Scheme</td>
</tr>
<tr>
<td></td>
<td>assembly</td>
</tr>
</tbody>
</table>

The important thing to remember is, due to the Halting Problem, some checks simply can never be performed statically; something must always be deferred to execution time. The trade-off in type design is to maximize the number of these decisions statically without overly restricting the power of the programmer.

The designers of different languages have divergent views on the powers a programmer should have.
CS345 Fall 2011, Pre-Final Exam Grades
51 As, 38 Bs, 11 Cs
Final Grade = \( \max(\text{Current Grade}, \text{Current Grade} + \text{Final Exam}) \)

(i.e., the Final is \textit{optional} and it can’t hurt you, it can only help – I thank you for your effort this semester especially on the projects.)