COMP 4400
Principles of Programming Languages

- [x] Class Participation (15%)
  - [x] C1
  - [ ] C2
  - [ ] C3
  - [ ] C4
  - [ ] C5
  - [ ] C6
  - [ ] C7
  - [ ] C8
  - [ ] C9
  - [ ] C10
  - [ ] C11
  - [ ] C12
  - [ ] C13
  - [ ] C14
  - [ ] C15
- [ ] Assignments (8% x5)
  - [ ] A1 8%
  - [ ] A2
  - [ ] A3
  - [ ] A4
  - [ ] A5
- [ ] Quizzes (5%)
  - [ ] ~~Quiz 1~~
  - [ ] Quiz 2
  - [ ] Quiz 3
  - [ ] Quiz 4
  - [ ] Quiz 5
- [ ] Midterm 1st Nov 15%

September 13, 2021

COMP 4400
Principles of Programming Langage

She is not Recording the class.

Office Hours: 1:30 - 3:30pm on Monday's.

Basic concepts.

Programming Paradigms: imperative, functional, and OOP, logic program.

Feature	Semantics
Prolog & logic Programming (includes the semantic)	
Scheme (functional programming)	λ - calculy lamda calculus.
Imperative (most of the programm we learned)	Polymorphism, Operational semantic type system

If time -> concurrent Programmly.

Course slides will be posted on BB

5 Assignments for this class. so DEAD

→ True/False questions, MCQ.

Polling

1.) Bonus Marks for correct Answers.

2.) Recording, class stuff.

3.) unannounced class quizes. (5%.)

↳ Multiple Quizzes. Assessment

↳ Answers should be correct.

Bonus Questions.

If you receive 5 bonus Marks, the weight of the final will be reduced to 20% instead of 25%.

Start studying ... You got this.

5 unannounced Quizzes

1.1. each.

Sept 15th, 2021

How many Programming languages do you know?
↳ ambiguity Questions.

What is a programming languages...
companies can train your language.

Core language ... Concepts & Principle.

Design → Specify → Compiler to run the programming languages.

Use a language ..... define & Implement
↳ API serves as the definition of the language.
↳ be precisely

Programming Language: Use, Read, Define the language.
→ Precise specification of a language. like Software Documentation.

How do you choose a language.

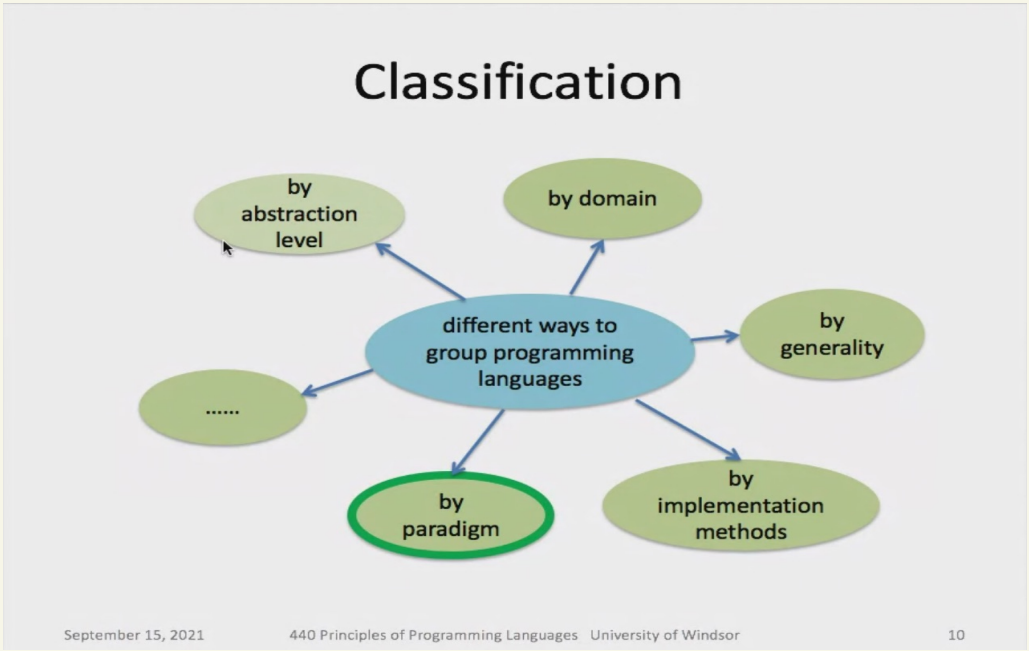
Classification of Programming languages.

Strongly typed
low level
Compiled
Domain
memory

specific
manageral

Imperative (OOP) Functional
user-friendliness
Server / client-based
platform independence
implementation

Concurrency (feature)
Runtime Environment.



Classification by Paradigm

Imperative, Logic-based, Functional, others (Event-driven)

"How" "What"

Imperative Programming languages

- Program state
- Program Execution

Assignment
Choice
Loop

These statements would change these states.

$x=0$, $y=1$ —$x=1$→ $x=1$, $y=1$

state → state 2

Summarization: Program is to move from one state to another.

A program is a sequence of commands that change states.

Turing Machine

register states? defining states is abstraction already.

lower level ⇒ more details in state

Imperative Programming Languages

C BASIC JavaScript Modula ML

Java Fortran Perl Smalltalk PL/1

Lisp Pascal Scala Haskell Ruby

Prolog PHP Python Algol Ada

Imperatives are are in Blue highlights.

Functional Programming Languages

C BASIC JavaScript Modula ML

Java Fortran Perl Smalltalk PL/1

Lisp Pascal Scala Haskell Ruby

Prolog PHP Python Algol Ada

Scheme → we will be learning in Clojure.

Functional Programming languages → Map/Reduce.

Map/Reduce - Applications

What google uses for search Engine.

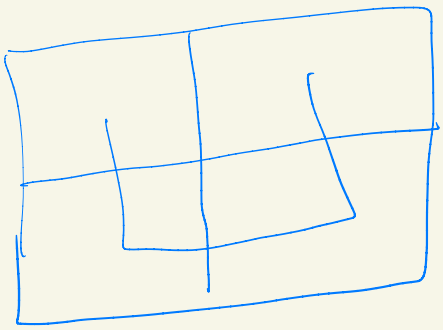
at Google:
- *index construction* for Google Search
- *article clustering* for Google News
- *statistical machine translation*

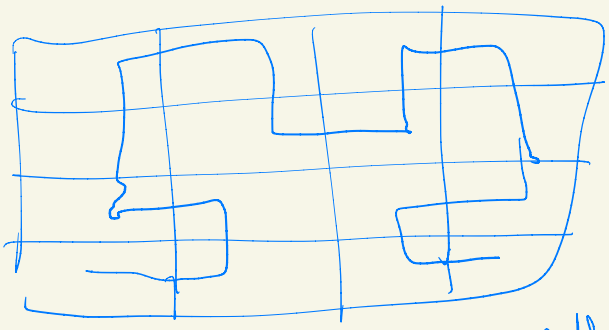
at Yahoo!:
- *web map* powering Yahoo! Search
- *spam detection* for Yahoo! Mail

at Facebook:
- data mining
- ad optimization
- spam detection

numerous startup companies and research institutes

Hilbert Curve: a space filling curve that visits every point in a square grid





visit each & every cell exactly once.

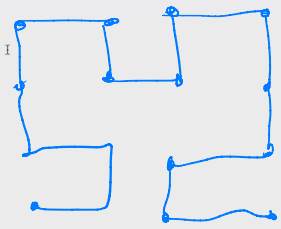
DrRacket

Functional Programming

run → (hilbert 2 'up)

Ans:

```
#lang r5rs
Welcome to DrRacket, version 6.2.1 [3m].
Language: r5rs; memory limit: 128 MB.
> (hilbert 2 'up)
(mcons
 'right
 (mcons
  'up
  (mcons
   'left
   (mcons
    'up
    (mcons
     'up
     (mcons
      'right
      (mcons
       'down
       (mcons
        'right
        (mcons
         'up
         (mcons
          'right
          (mcons 'down (mcons 'down (mcons 'left (mcons 'down (mcons 'right '())))))))))))))))
>
```



Functional Programming has no variable / assignment / constant.

Sept 20, 2021

Imperative — state — state change

Functional — no assignment — no war

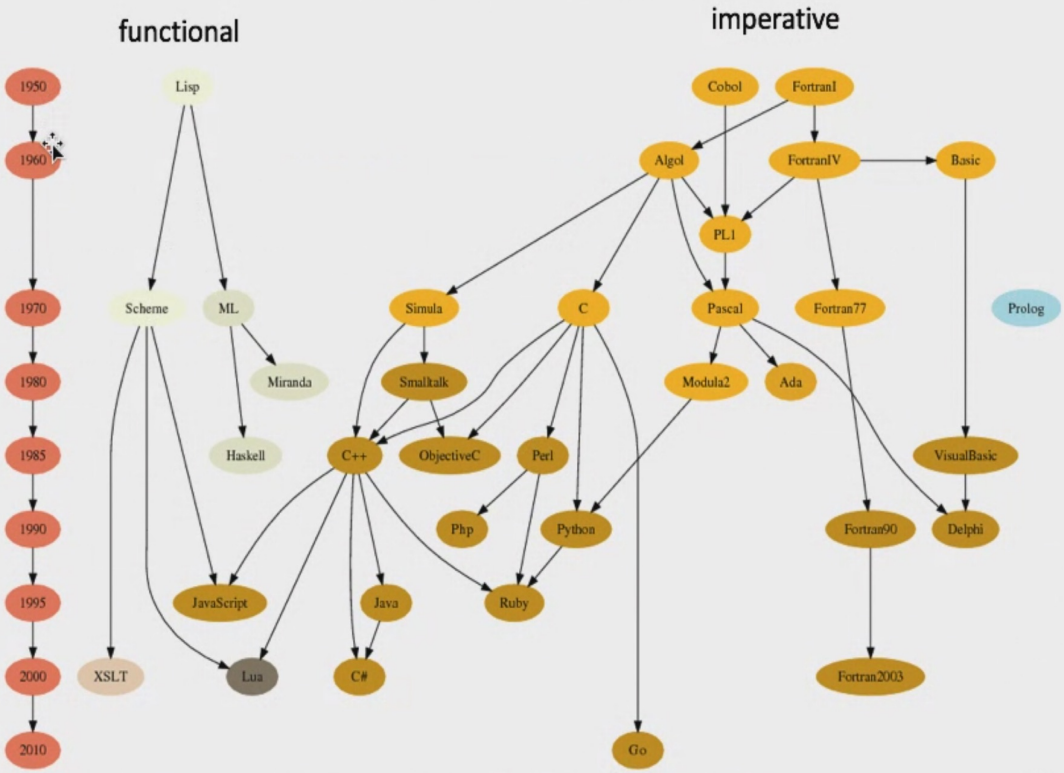
component of function.

logic programming. — logical theory.

Comparison

programming paradigm	understanding of computation
functional programming	the evaluation (simplification) of expressions to values
imperative programming	state transformation
logic programming	proof search

Evolution of Programming Languages



each function can have a state, in C programming.

What is logic programming?

Statement, you don't have if-then-else loops

but here is statements.

statements are based on First Order Logic

Theorem proving.

PROLOG

↳ programming in logic. Based on

First order Logic

↳ It's mostly used for Artificial Intelligence

FOL → we limit to Horn clause

Declarative Programming Language →

It's very high level.

- declarative → functional, logic, prolog.
- procedural

you only specify

what you want,

not how to do it.

what you want, not how you want?

logical inferences of the compiler will

deduce what we want.

Sample.pl

statement — Symboliz logic.

model(cascada, buick). ← Computer recognizes this has a fact

model(encore, buick). ↙

running Prolog.

?- [sample]. period ends the statement.

?- model(cascada, buick), query sending can you construct a proof of this. The computer recognizes, yes

?- model(X, buick).

X = cascada ? it checks if there are any values that has X

X = cascada ? ; when you put a semi-colon it will look for other values of X

to end..

?- halt.

Program

transport(windsor, toronto).

Query

1.) transport(windsor, toronto)
true

2.) transport(windsor, ottawa)
False

3.) transport(windsor, X)
Toronto

4.) transport(windsor, where)
Toronto

5.) transport(windsor, ottawa)
Toronto.

Prolog variables are capitalized. So query recognizes X capital letter as

Facts
Rules
Query { what true / false

Fact : transport (windsor, toronto)
name

transport (windsor)
↳ two different predicates.

transport /1 different predicate
/2 different predicate

want computer to learn if

implicit Rule

transport(windsor, ottawa) :-
trasport (toronto, ottawa)

transport(windsor, X) :-

transport(toronto, X)

all the cities reachable from toronto is reachable from windsor.

Rules

, and

:- if

transport(windsor, ottawa) :- transport

head

body

Rules → ∀ all

∃ some

sibling(A, B) :- parent(C, A),

parent(C, B).

its like relation (X, Y) so X has a specific relation to Y. C is a parent to A and C is a parent to B hence making A and B sibling

Sept 22, 2021

transport(windsor, where)

So where is a variable that remembers all the statements.

semantics behind the statement.

Predicate → transport, can have any number of arguments

Prolog → FOL, we can't have predicate inside a predicate.

Learn, FOL, propositional logic or high order logic

↳ argument can only be terms.

sibling(A, B) :- parent(C, A), parent(C, B)

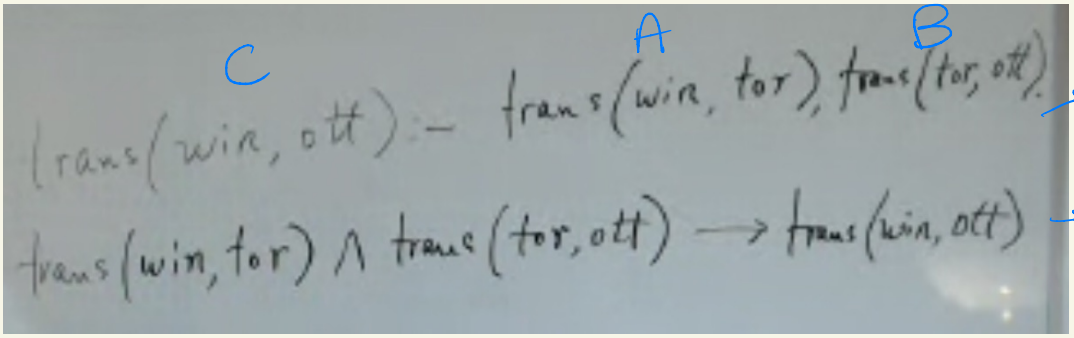
↳ A, B can be bound to have same value

?- sibling(X, X) what will be the answer??

parent(C, X), parent(C, X)

Symbolic Computation. → internally there is a algorithm. The computer will run the algorithm.

Example: transport(Toronto, Where), same as transport(X, Y)



$(A \wedge B) \rightarrow C$

$\sim C \; (A \wedge B) \vee C$

$\sim A \vee \sim B \vee C$ (Disjunctive clause)

↳ we can have at most one positive literal (Horn clause)

So first order logic can be made into disjunctive

→ Prolog only works on Horn Clause, it works on a sub set of First Order Logic.

→ relationship between all statements → AND the comma

Unification

i) transport (toronto, ottawa) — we do mapping.
transport (windsor, ottawa)

?- trasport(windsor, Where) ; answer would be ottawa.

↳ not variable since first two argument then unification is possible.

Unification

2) transport (toronto, X) two-way mapping.

transport (Y, windsor)

Successful, two way binding.

Parent (X, Y)

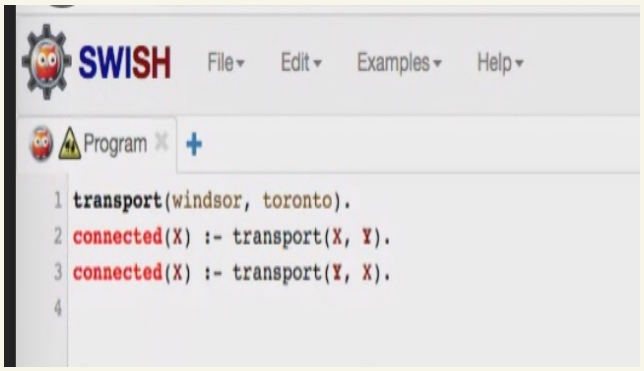
course (4400, Fall 2021)] this won't succeed.

↳ because predicate won't succeed

For unification to be successful

→ Predicate names should Match

→ Number of arguments should be same.



?- connected(A)

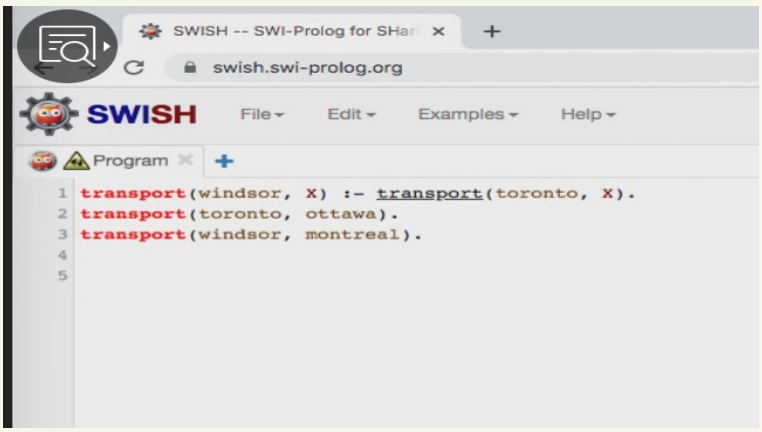
it will give Singleton Variable replace with underscore

↳ Answer would be

A = windsor

A = toronto

underscore represents two different variables



?- transport(windsor, X).

we can do trace to see what going on.

?- trace, transport(windsor, X)

internally same memory code

L1. transport(windsor, 4328)
transport(toronto, 4328)
Exit transport(toronto, ottawa)

answer to transport(windsor, X).
would be ottawa

X = ottawa

Redo transport(windsor, 4328)
Exit transport(windsor, montreal)
X = Montreal

So answer would be
X = ottawa
X = Montreal

Continue from where we are from previous search.

Internally Tree is a search tree to remember stuff.

Recursive Rule in Prolog what are we allowed to do

ansestor outline

ascestor(X,Y) :- Parent(X,Y)

ansestor(X,Y) :- parent(X,U), parent(U,Y)

ansestor(X,Y) :- parent(X,U),
parent(U,V),
Parent(V,Y),

How to write this recursively.

recursion ansestor(X,Y) :- parent(X,U), ansestor(U,Y)

recursion can be done in Prolog.

Sept 29th 2021

Damn she liked my question :)

Can the word symbolic be in the question

I got bonus for it

gm (cascada, buick)
model (buick, cacasada)

within your own interpretation is fin.

>>> grep

pattern fname

symbolic (buick, cascada) → Test Question.

or grep grep grep
cat grep

No loops or if then else statements.

1) Parent (Edward, Emma)
2) Parent (Edward, Logan)
3) Parent (Alice, Emma)
4) Parent (Alice, Logan)
5) Sibling (X,Y) :- parent (Z,X), parent (Z,Y)

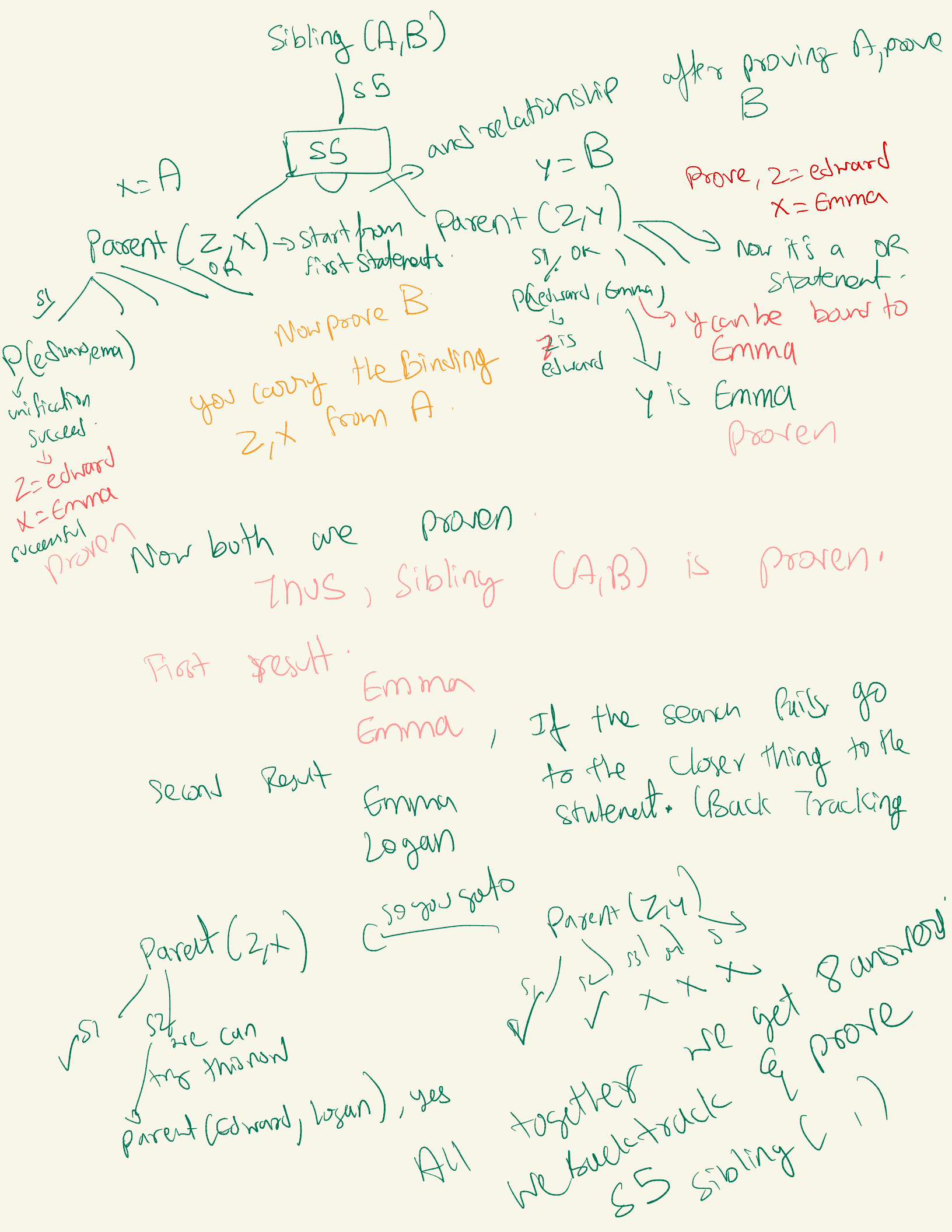
Sibling (A,B) you can do whatever binding.

Sibling (A,B)

S1 X S2 X S3 X S4 X S5 ✓ use S5

you use statement 1 to prove it or the others.

S1 fails, so you move onto S2


Sibling (A,B)
S5
SS
and relationship
after proving A, prove B
x = A
y = B
Parent (Z,X)
OR
start from first statement
Parent (Z,Y)
S1, OR
Prove, Z = edward
X = Emma
Now it's a OR statement.
P(edward, Emma)
Z is edward
y can be bound to Emma
Y is Emma
Proven
S1
P(edward, Emma)
unification succeed
Z = edward
X = Emma
successful
Proven
you carry the Binding Z, X from A.
Now prove B
Now both are proven
Thus, Sibling (A,B) is proven.
First result.
Emma
Emma
Second Result
Emma
Logan
, If the search fails go to the closer thing to the statement. (Back Tracking
so you goto
Parent (Z,X)
S1
S2
we can try this now
Parent (edward, logan), yes
Parent (Z,Y)
S1
S2
S
All together we get 8 answers
we backtrack & prove
S5 Sibling (,)

parent(Z,X)

S1 / S2 S3 S4 S5

↓

parent(edward, logan)

parent(Z,Y)

S1

Z = edward

Y = logan

so now Y is bound to Logan

Z = edward

X = logan

so now sibling(A,B) is proven.

1.) Emma Emerson

2.) Emma logan

3.) Logan Emma

This Backward chaining is so fucked up.

parent(Z,Y)

S2

parent(Edward, logan)

S3

At the Enemy, NO

S4.) NO

parent(Z,X)

/ / / / / S5

verification fail, so we finished.

Remove duplicate... in statement.
In prolog no because it's artificial intelligence.

=(X,Y) we will talk about this later on.

In the sibling problem all you do is binding.

Infinite Loops :- assignments on graph too graphs