Love Languages

Re-imagining English Syntax as a Turing Complete Programming Language

Cassidy Diamond

Love Languages

An esoteric programming language inspired by early concepts in **linguistics**, **computer science**, and **computer art**

Cassidy Diamond

Linguistics

Introduction: X-Bar Theory

- Constructed by Noam Chomsky in 1957
- Represents the syntax of a language in binary trees and rules
  - “Syntactically correct”
  - *Colorless green ideas sleep furiously*
- X-Bar rules determine how pieces of language can “correctly” fit together

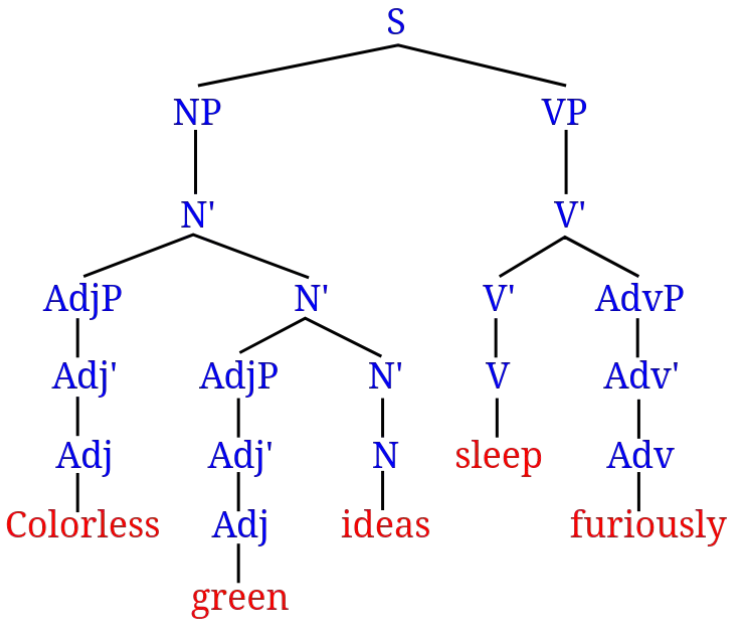


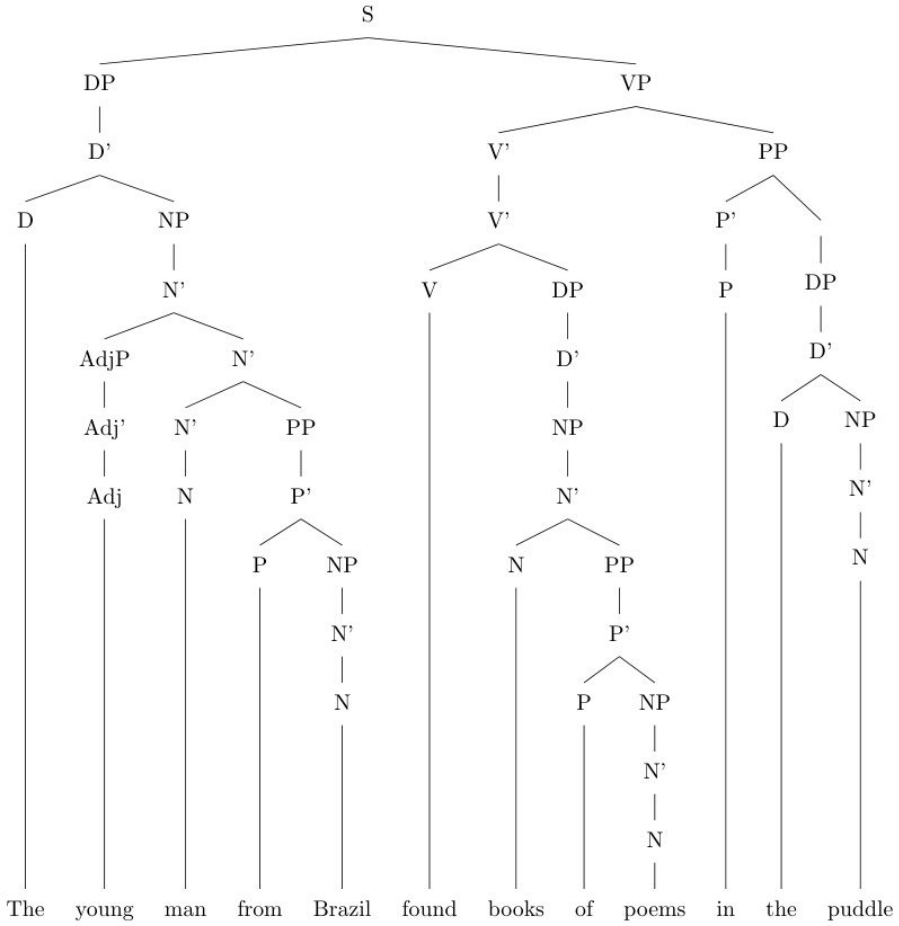
Image Source:
Wikimedia (Creative Commons License)

X-Bar Theory Rules (Example)

Sentence rules	SP	→ DP	VP	
Determiner rules	DP	→ DP	Conj	DP
	DP	→ Pronoun		
	DP	→ D’		
	D’	→ D	NP	
	D’	→ NP		
	D’	→ NP		
	DTVDP	→ DP	DP	
Noun rules	NP	→ N’		
	NP	→ NP	Conj	NP
	N’	→ AP	N’	
	N’	→ N’	PP	
	N’	→ N		

Verb rules	VP	→ V’		
	VP	→ VP	Conj	VP
	V’	→ V’	PP	
	V’	→ V’	AdvP	
	V’	→ AdvP	V’	
	V’	→ TV	DP	
	V’	→ DTV	DTVDP	
	V’	→ V		

Fig. 3: Example X-Bar rules



Observations:

- Every sentence is composed of a determiner phrase and a verb phrase
- All adjectives are followed by nouns
- Adverbs can modify adjs or verbs
- Prepositional phrases can modify nouns or verbs

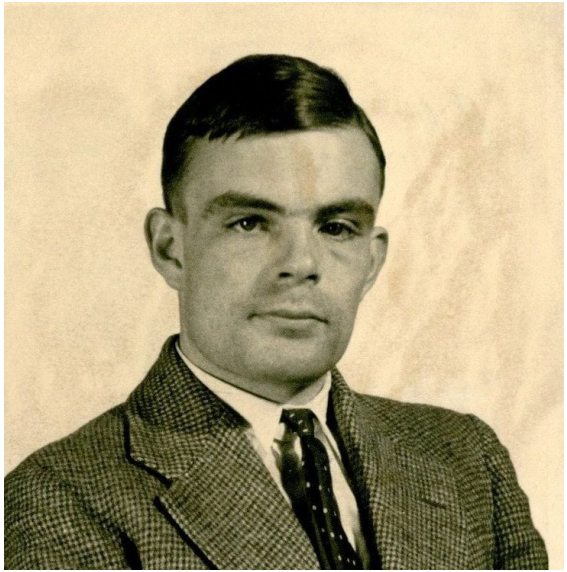
- **Highly structured output based on starting rules**
- **Rules can exhibit recursive structures**

Example Source:
Syntax: A Generative Introduction by Andrew Carnie

Computer Science

Well known computer-scientist: Alan Turing

- Concept of Turing Completeness originating in 1950
    - ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM
- **Turing Machine** - conceptual model of a computer
- Any programming language / schema that can represent a Turing Machine is **Turing Complete**
- **Church-Turing Thesis** - Any problem that a computer can possibly solve, can be solved by a Turing Complete programming language



Brainf*ck (bf)

This is a programming language!

```
++++++++++[>+>+++>+++++++>
++++++++++<<<<-]>>>++.>+.+
++++++..+++.<<++.>++++++++
+++++++.>.+++.------.-----
---.<<+.<.
```

`Output:` “`Hello World!`”

- Incredibly simple Turing Complete programming language
- 8 basic operations with frequent repetition: `+-[]<>.,`
- **Highly structured output based on its operations**
- **Operations can exhibit recursion, or loops** – like X-Bar rules!

Idea:

A mapping from syntactically correct **programs** (in the computer science sense) to syntactically correct **sentences** (in the linguistics sense).

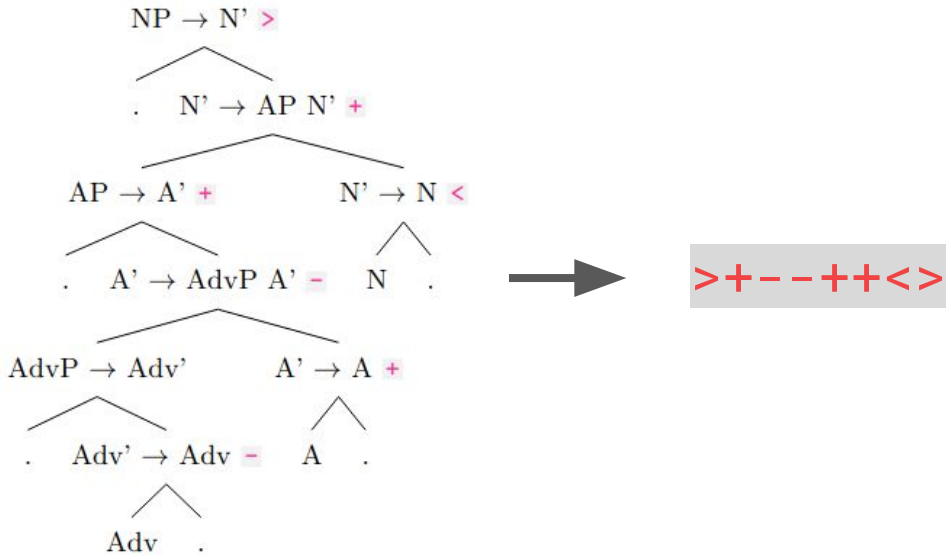
A programming language where each program is a collection of X-Bar syntax trees, which can be represented by sentences.

Computer Science
Implementation

1. Design a mapping between syntax trees and computer programs
2. Create a way to easily encode arbitrary programs into syntax trees
3. Assign words to syntax trees programs to represent them as sentences

1. Map syntax trees to computer programs

- Goals
  - use structure of syntax tree
  - programs → varied sentences
  - short programs → short sentences
- Convert X-Bar syntax trees into `bf` programs
  - computer science concept: **In-Order traversal**
- Each node in a syntax tree gets mapped to a set of `bf` operations



"incredibly green ideas"
or "*very confusing concept*"

Fig. 5: Converting a syntax tree into a `bf` program

1. Assign bf ops to X-Bar Rules

- Each node in a syntax tree gets mapped to a set of `bf` operations
- These rules are pretty much the entire programming language specification

Sentence rules	SP	→ DP VP	.	(1)
	SP	→ EXC	.	(2)
	SP	→ QUERY	,	(3)
Determiner rules	DP	→ DP Conj DP	>>	(4)
	DP	→ Pronoun	<<<<	(5)
	DP	→ D’	.	(6)
	D’	→ D NP	.	(7)
	D’	→ NP	>>>	(8)
	DTVDP	→ DP DP	.	(9)
Noun rules	NP	→ N’	>	(10)
	NP	→ NP Conj NP	]	(11)
	N’	→ AP N’	+	(12)
	N’	→ N’ PP	.	(13)
	N’	→ N	<	(14)
Verb rules	VP	→ V’	>	(15)
	VP	→ VP Conj VP	<<<<	(16)
	V’	→ V’ PP	.	(17)
	V’	→ V’ AdvP	.	(18)
	V’	→ AdvP V’	.	(19)
	V’	→ TV DP	>	(20)
	V’	→ DTV DTVDP	>	(21)
	V’	→ V	.	(22)
Adverb rules	AdvP	→ Adv’	.	(23)
	AdvP	→ AdvP Conj AdvP	.	(24)
	Adv’	→ Adv’ Conj Adv’	++	(25)
	Adv’	→ AdvP Adv’	[	(26)
	Adv’	→ Adv	-	(27)
Adjective rules	AP	→ A’	+	(28)
	AP	→ AP Conj AP	------	(29)
	A’	→ A’ Conj A’	--	(30)
	A’	→ AdvP A’	-	(31)
	A’	→ A	+	(32)
Preposition rules	PP	→ P’	>>	(33)
	PP	→ PP Conj PP	>>>>	(34)
	P’	→ P DP	.	(35)

Table 2: X-Bar Rules used, and their assigned operations

2. Encode arbitrary programs into syntax trees

- Design and implement `find_bf` function
  - **Input:** Any `bf` program
  - **Output:** a collection of syntax trees that encode a functionally equivalent program
- Two approaches: *Graph Search* and *Tree Search*

2. Combine Rules into Programs: `Graph Search`

Roughly:

- Created an algorithm capable of generating all possible syntax trees
  (given my input X-Bar rules)
- Use computer science concept **A* search algorithm** to look for sentences that encode desired program

First problem: my `Graph Search` algorithm is too slow

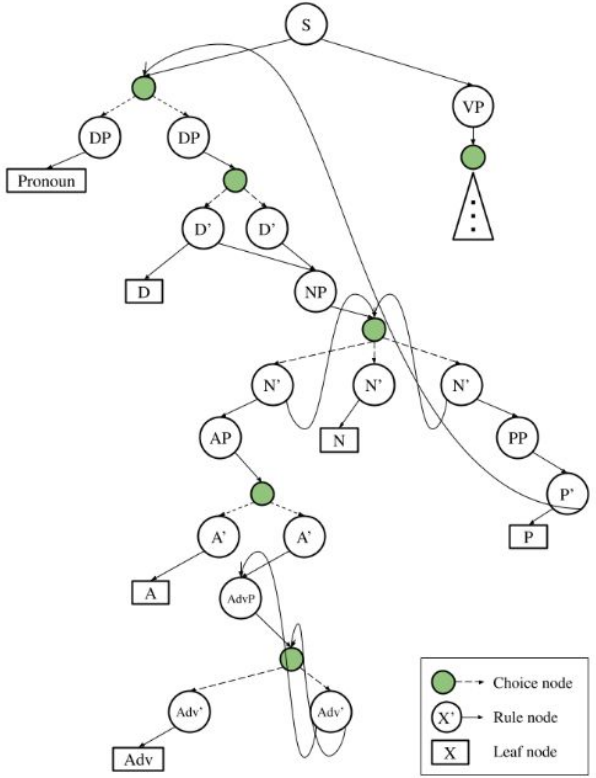


Fig. 6: X-Bar rules laid out in Recursive Tree format (VP subtree and conjugation rules omitted)

2. Graph Search: Memoization

Linguistics concept: **constituency** and **substitution**

- We can tell which words are part of the same phrase by constituency tests
- "The undergraduate student gave her presentation"
- "She gave her presentation"
- ⇒ "She" and "the undergraduate student" are both determiner phrases

Computer science concept: **memoization**

- Speed up my search function by remembering syntax phrases we've explored before
- Use constituency to substitute like phrases

2. `Graph Search`: Results

didn’t work! 😔

or, it’s still too slow

:(

2. New Approach: `Tree Search`

`find_bf`: takes in a `bf` program as input, outputs syntax trees

Graph Search

- both use A* search algorithm in some capacity
- Constructs sentences **left to right**, in speaking order
- Memoization
- too slow to work effectively

Tree Search

- both use A* search algorithm in some capacity
- Constructs sentences **starting in middle, extending outwards**
- Memoization/Dynamic programming
- fast! it works!

- less “human” model of language

3. Assign Words to Syntax Trees

- Convert syntax trees into sentences, like *Mad Libs*
- Replace “syntactic categories” (e.g *noun*, *adjective*, *verb*, etc) with words from a word bank, respecting grammar
  - Pronoun agreement
  - Verb conjugation
  - Noun pluralization
- Important concept: The words in a sentence do not affect the meaning of the encoded program. Only the syntax of the sentence
  - “*Colorless green ideas*” == “*Very important concept*”
  - Semantics of a sentence vs semantics of a program

3. Assign Words to Syntax Trees

If the words in a sentence don’t affect the meaning of the program, what do I choose to say?

Computer Art

Lesser known computer-scientist: Christopher Strachey

- Christopher Strachey: fellow student and friend, of Alan Turing. Foundational to computer science in his own right. Both queer men.
- Strachey Love Letter Algorithm
  - In 1952, letters started appearing on the bulletin boards of the Manchester Computer Lab
  - First piece of computer-generated literature

YOU ARE MY [Adjective] [Noun]

MY [Adjective] [Noun] [Adverbs] [Verbs]
YOUR [Adjective] [Noun]

Strachey Love Letter Algorithm

In 1952, letters started appearing on the bulletin boards of the Manchester Computer Lab

```
HONEY SWEETHEART

MY THIRST PINES FOR YOUR FOND LONGING. YOU ARE MY
WISTFUL AFFECTION: MY PRECIOUS INFATUATION. MY
AFFECTION COVETOUSLY LOVES YOUR TENDERNESS. MY
PASSIONATE AFFECTION CURIOUSLY HOPES FOR YOUR
LOVEABLE HEART.

YOURS DEVOTEDLY,
M.U.C.
```

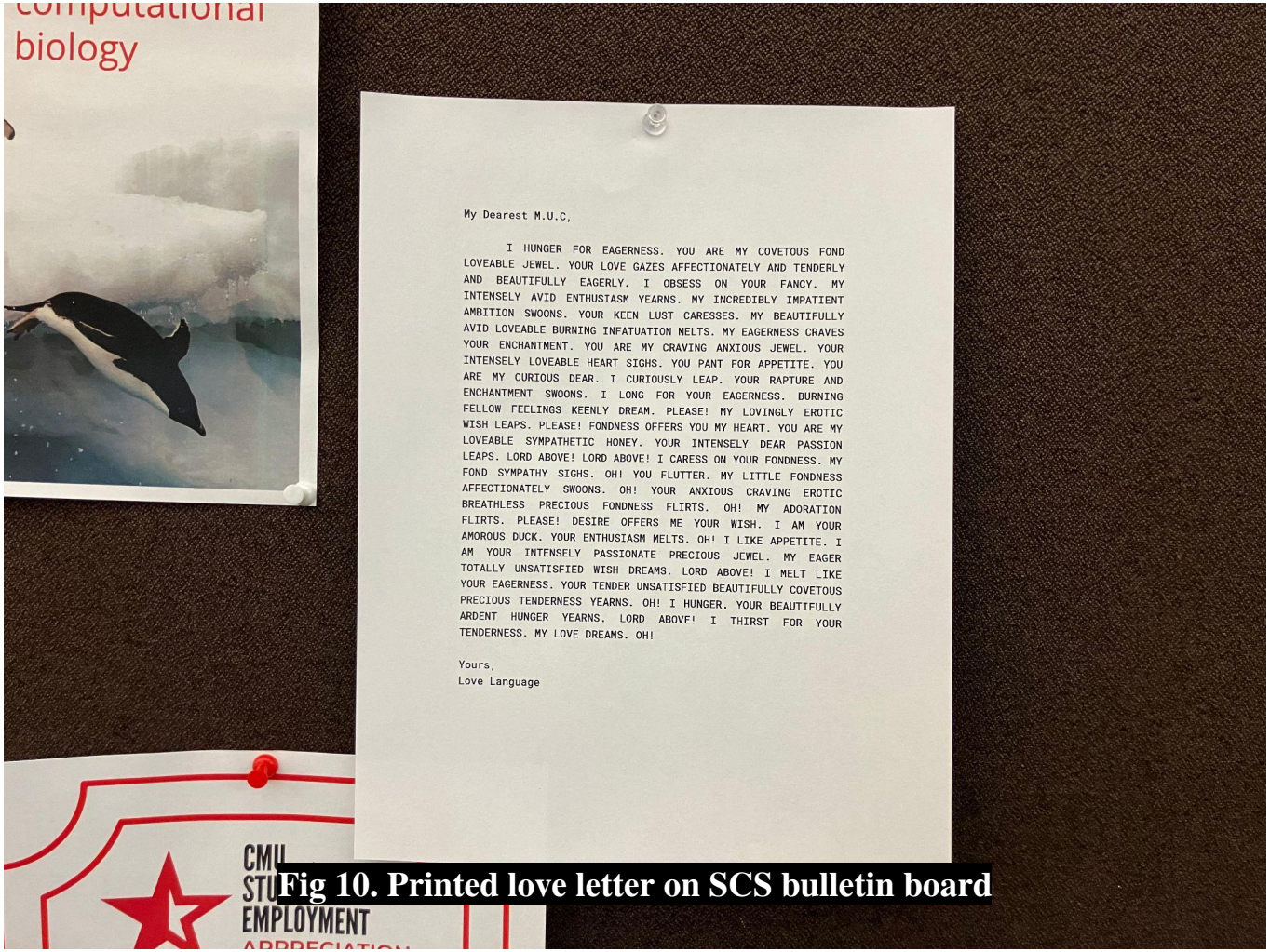


Fig 10. Printed love letter on SCS bulletin board

Love Language: *My Dearest M.U.C,*

I HUNGER FOR EAGERNESS. YOU ARE MY COVETOUS FOND LOVEABLE JEWEL. YOUR LOVE GAZES AFFECTIONATELY AND TENDERLY AND BEAUTIFULLY EAGERLY. I OBSESS ON YOUR FANCY. MY INTENSELY AVID ENTHUSIASM YEARNS. MY INCREDIBLY IMPATIENT AMBITION SWOONS. YOUR KEEN LUST CARESSES. MY BEAUTIFULLY AVID LOVEABLE BURNING INFATUATION MELTS…

```
++++++++++[>+>+++>++++++
+>++++++++++<<<<-]>>>++.
>+.+++++++..+++.<<++.>++
++++++++++++.>.+++.----
--.--------.<<+.<.
```

```
Output: “Hello World!”
```

Love Languages vs Strachey Love Letter Algorithm

- Strachey's Love Letter algorithm as a queer critique of heteronormative displays of affection
- Turning **algorithmic love letters** into **literal algorithms**
- Turing Completeness: Limited semantics[1] of each sentence vs unlimited semantics[2] of program
    - 1: In the linguistics sense
    - 2: In the computer science sense

Love Languages vs Strachey Love Letter Algorithm

“In addition to readings based on **queer critique of heterosexual phatic writing**, I also often suggest that **these love letters might threaten human jobs**. The most literate soldier in the platoon during World War II, for instance, would often write love letters for others in exchange for extra rations. Perhaps I’m joking, but I find these sorts of overreadings productive.”

- ChatGPT, programmers today, computer-generated-literature…
- A more optimistic reading: expressing love

Nick Montfort, professor of digital media at MIT

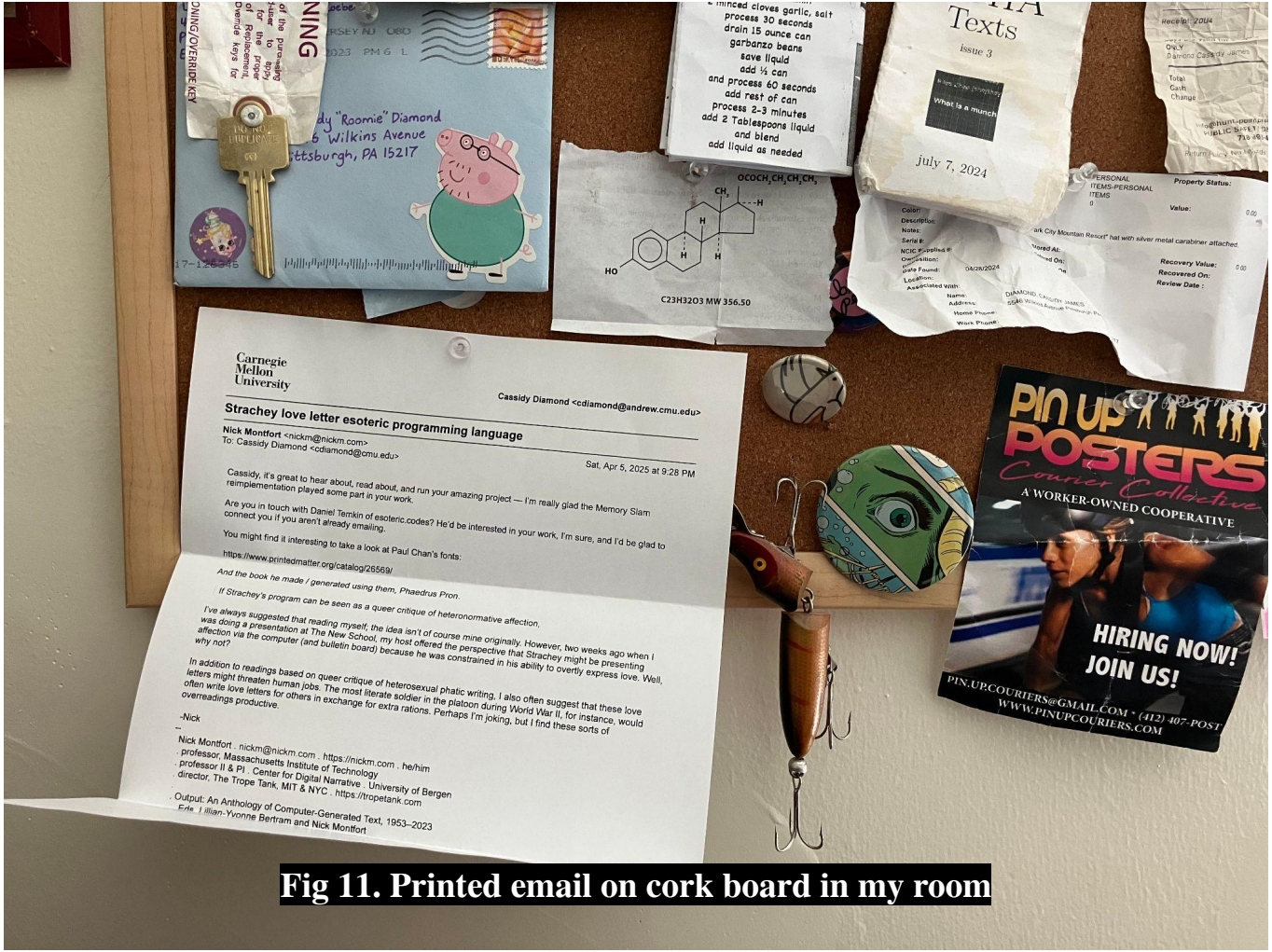


Fig 11. Printed email on cork board in my room

Thank you!

Read the paper/code

https://cassidydiamond.me/love-languages/