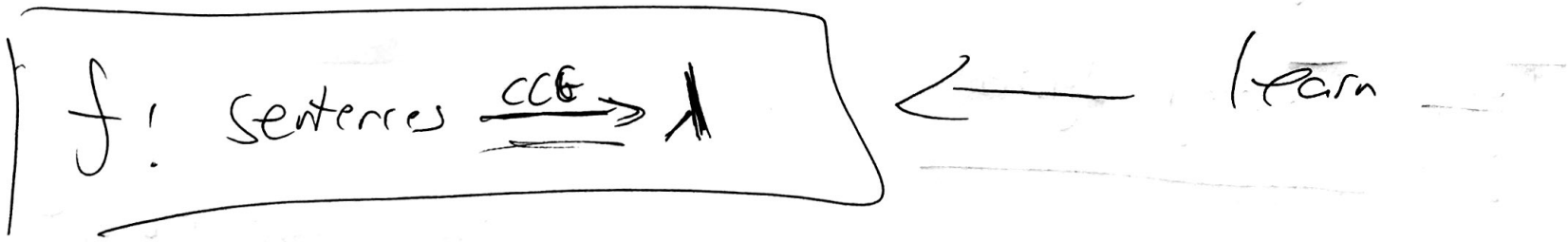
Today: Semantic parsing with CCG

Combinatory Categorial Grammar



- Lambda calculus
- CCG]
- Learning

(2)

Lambda Calc.

- Formal rep. lang.
- higher order functions ——> you can take func. as arguments
- typed

~~Type~~ Typing System:

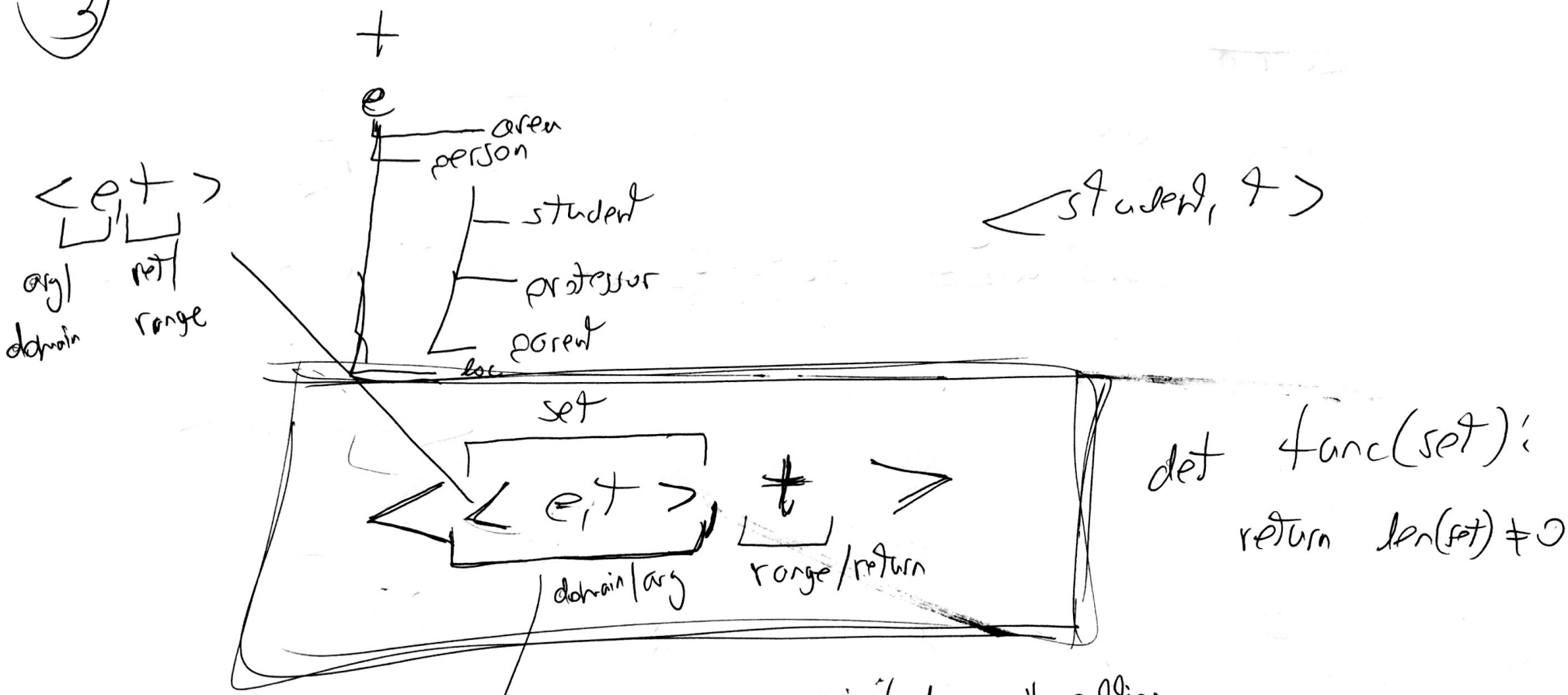
- entity, truth value, event, city, airport, student
- complex type: <domain, range>

<<city, student>, t>

arg1: city
arg2: student
return: t

<city, <student, t>>

(3)




```
def func(set):
    return len(set) != 0
```

set = include all entities for which func. is true

map(lambda x: len(x), lists) → (...)

(4)

~~Lambd~~

λ-calculus!

1) constant – ALANE$_{student}$, MAX$_{student}$, studies_in

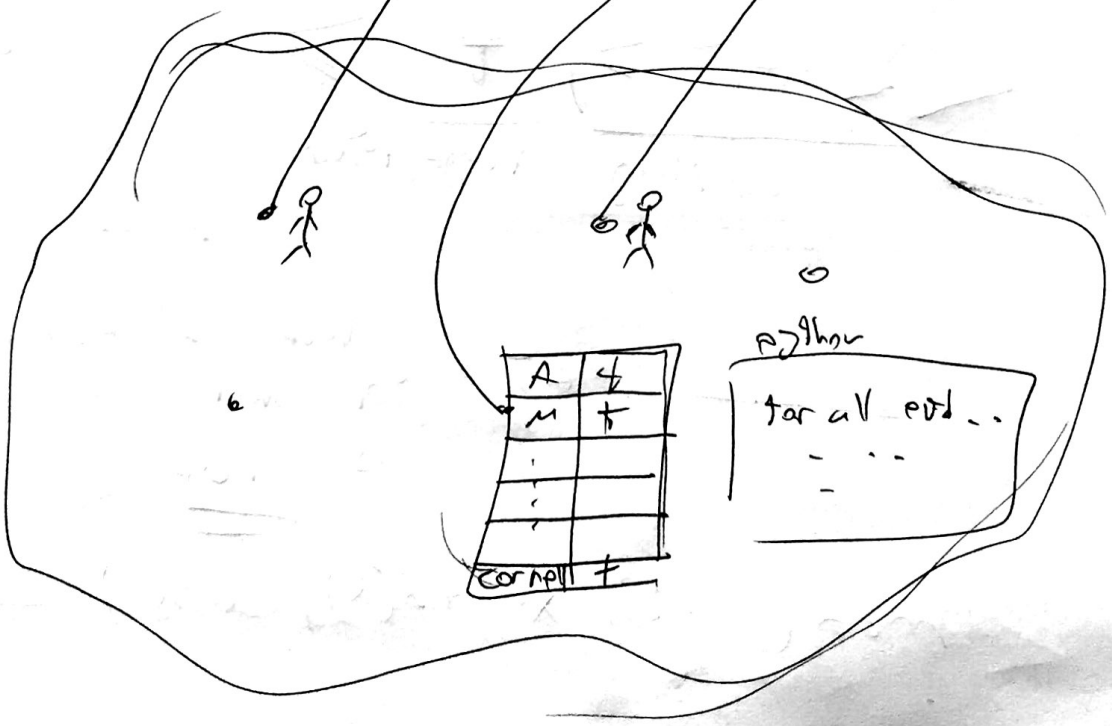
student$_{<entity, +>}$

<student, <loc, +>>

2)

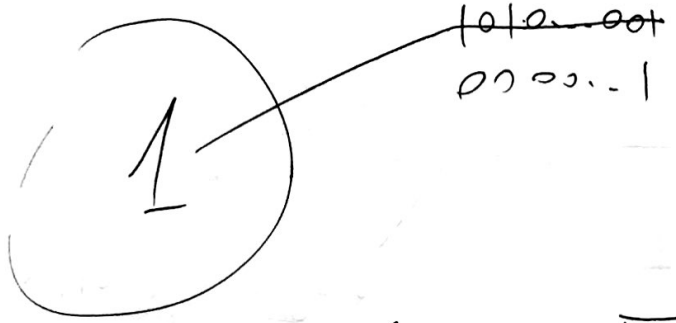
3)

4)



(5)

2) Variable x_e y_t $z_{\langle student, t \rangle}$



3) Literal predicate(arg, ..., arg)

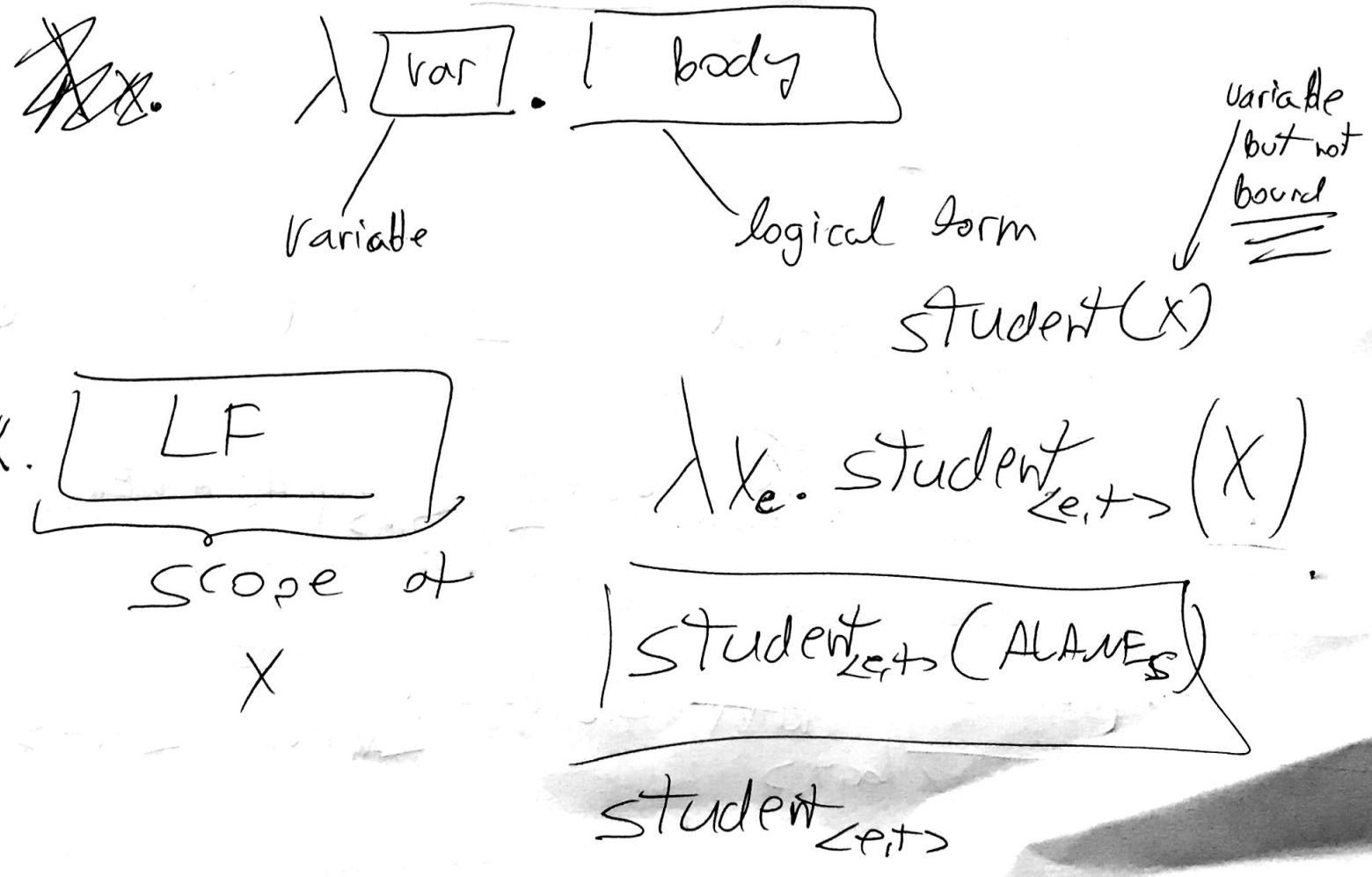
student$_{\langle entity, t \rangle}$ (ALANE$_{student}$)

studies-in (ALANE)$_{\langle loc, t \rangle}$

studies-in$_{\langle student, \langle loc, t \rangle \rangle}$ (ALANE$_{student}$, Cornell$_{area}$)

4) Lambda term

— anonymous function



(5) set $\{e \mid f(e) = true\}$

$f = \lambda x.\ student(x)$

$f(5, 6)$.

$5 + 6$

function: arg ⟶ t/f
if the arg is a student

literal

conj $\wedge(arg1, arg2)$ $\langle t, \langle t, t \rangle \rangle$ type of ∧

disj ~~$\wedge(\wedge(arg1, arg2), arg3) \Longrightarrow arg1 \wedge arg2 \wedge arg3$~~

~~$arg \vee$~~ $arg1 \vee arg2 \vee arg3 \vee arg4 \ldots$

(6)

- application
- composition

Application

$(\lambda x.\ student(x))(ESIN)$

ESIN

$student(x) \xrightarrow{x \to ESIN} student(ESIN)$

~~$(\lambda x.\ student(x))(\lambda x.\ student(x))$~~ bad x going

7 Composition

$$g \circ f = g(f(x))$$ ← func. composition

$g = \lambda f \lambda x . f(x) \,|\, red(x)$

↓ func

$f = \lambda x . f(x) \,|\, square(x)$

↑ func

~~$\lambda f \lambda x . f(x)$~~

$\lambda f \lambda x . f(x) \,|\, red(x) \,|\, square(x)$

8

Alone is a student.

⇒ CCG

- Categorial formalism

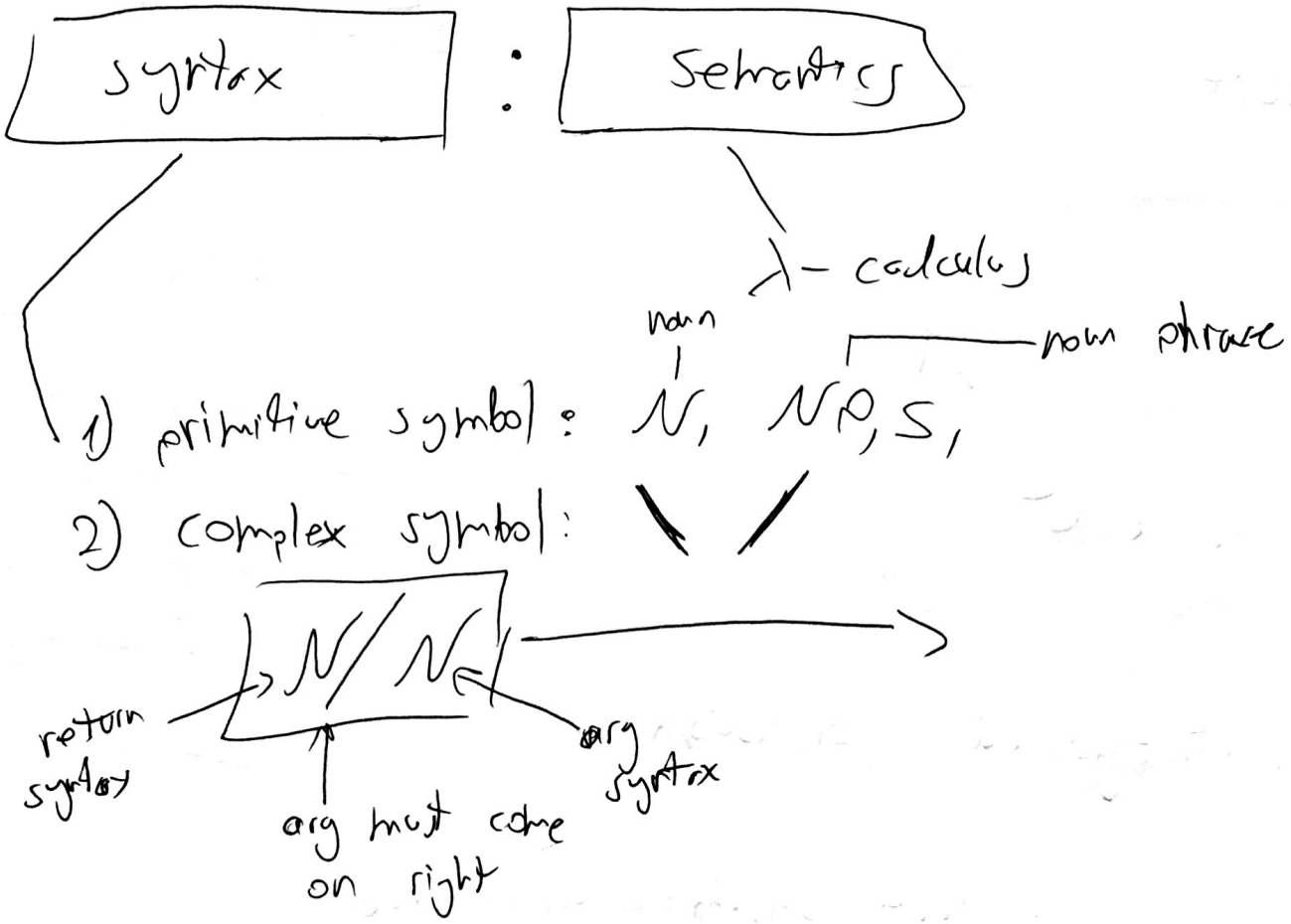
CFG

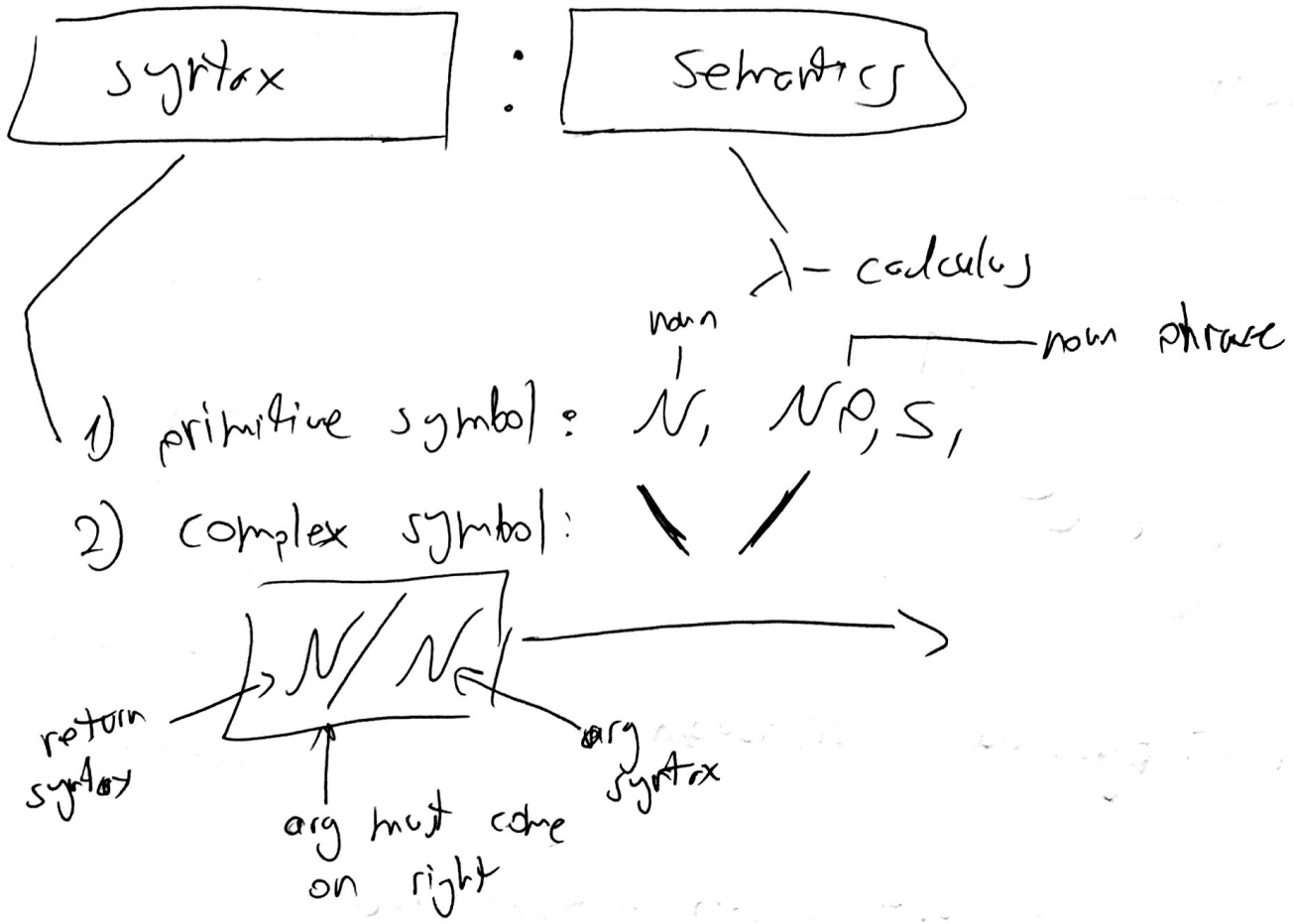
- set of symbols A, B, C, ...
- set of rules A → B C D

- many symbols → unbounded
- very few rules

categories

- interface for syntax - semantics


syntax
:
semantics
λ - calculus
noun
noun phrase
1) primitive symbol: N, NP, S,
2) complex symbol:
N/N
return syntax
arg must come on right
arg syntax


syntax
:
semantics
λ- calculus
noun
noun phrase
1) primitive symbol: N, NP, S,
2) complex symbol:
N/N
return syntax
arg must come on right
arg syntax

Function Application

$$(\lambda f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}.f(\lambda x_e.\text{book}_{\langle e,t\rangle}(x)))(\lambda g_{\langle e,t\rangle}.\lambda y_e.g(y) \wedge \text{red}_{\langle e,t\rangle}(y))$$
$$\mapsto (\lambda g_{\langle e,t\rangle}.\lambda y_e.g(y) \wedge \text{red}_{\langle e,t\rangle}(y))(\lambda x_e.\text{book}_{\langle e,t\rangle}(x))$$
$$\mapsto \lambda y_e.(\lambda x_e.\text{book}_{\langle e,t\rangle}(x))(y) \wedge \text{red}_{\langle e,t\rangle}(y)$$
$$\mapsto \lambda y_e.\text{book}_{\langle e,t\rangle}(y) \wedge \text{red}_{\langle e,t\rangle}(y)$$

Function Composition

$g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda h_{\langle e,t\rangle}.\lambda x_e.h(x) \wedge \text{yellow}(x)$

$f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda k_{\langle e,t\rangle}.\lambda y_e.k(y) \wedge \text{square}(y)$

Let $A_{\langle e,t\rangle}$ be a $\langle e,t\rangle$-typed logical expression

$g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(A_{\langle e,t\rangle}) = (\lambda h_{\langle e,t\rangle}.\lambda x_e.h(x) \wedge \text{yellow}(x))(A_{\langle e,t\rangle}) =$

$(\lambda x_e.h(x) \wedge \text{yellow}(x))[h \mapsto A_{\langle e,t\rangle}] =$

$\lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x)$

$f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(h_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(A_{\langle e,t\rangle})) =$

$(\lambda k_{\langle e,t\rangle}.\lambda y_e.k(y) \wedge \text{square}(y))(\lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x)) =$

$(\lambda y_e.k(y) \wedge \text{square}(y))[k \mapsto \lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x)] =$

$\lambda y_e.(\lambda x_e.A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x))(y) \wedge \text{square}(y)) =$

$\lambda y_e.(A_{\langle e,t\rangle}(x) \wedge \text{yellow}(x))[x \mapsto y] \wedge \text{square}(y)) =$

$\lambda y_e.A_{\langle e,t\rangle}(y) \wedge \text{yellow}(y) \wedge \text{square}(y))$

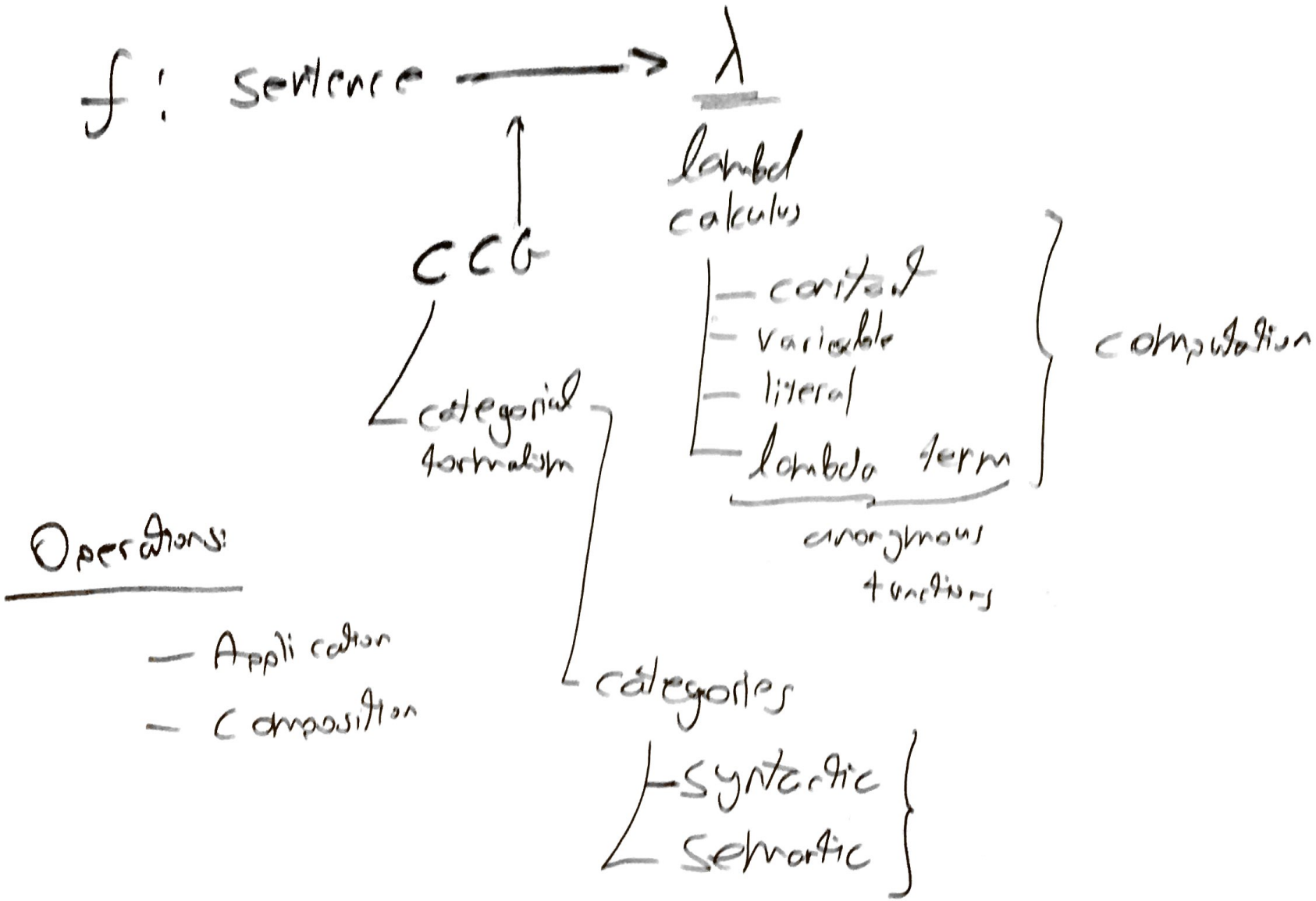
Add a new variable and mapping: $\lambda z_{\langle e,t\rangle}.f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(A_{\langle e,t\rangle}))[A \mapsto z] =$

$(\lambda z_{\langle e,t\rangle}.\lambda y_e.A_{\langle e,t\rangle}(y) \wedge \text{yellow}(y) \wedge \text{square}(y)))[A \mapsto z] =$

$\lambda z_{\langle e,t\rangle}.\lambda y_e.z(y) \wedge \text{yellow}(y) \wedge \text{square}(y)) = (f_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} \cdot g_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle})_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}$

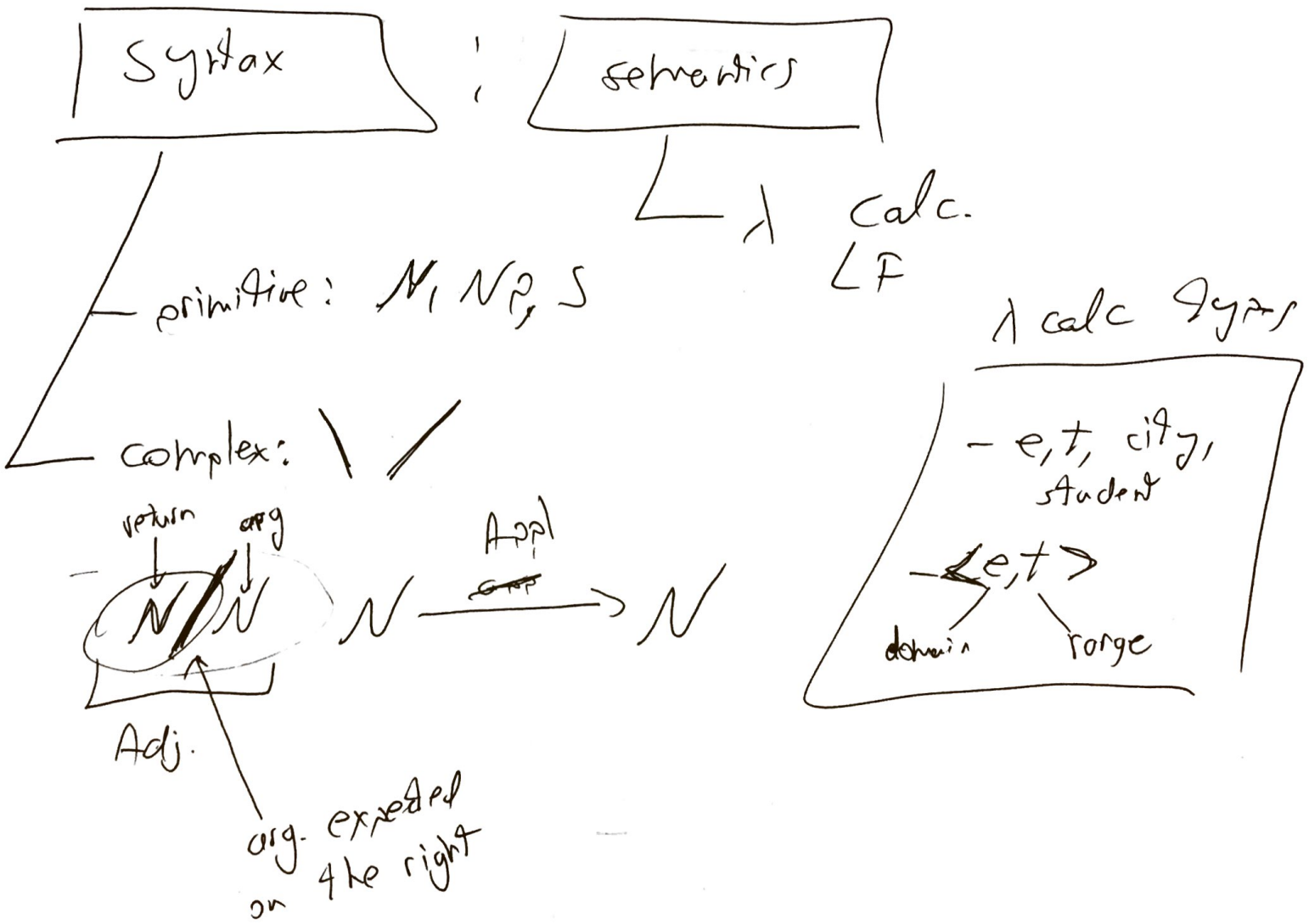
(1)

Goal: recover formal meaning representation

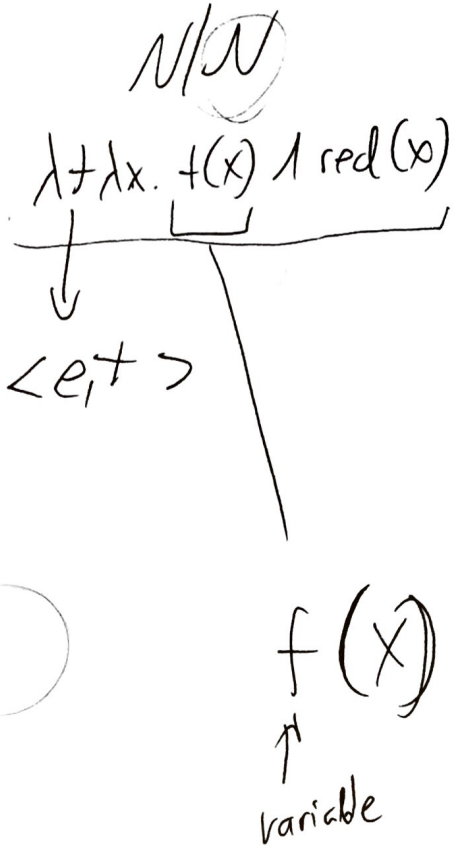


(2)

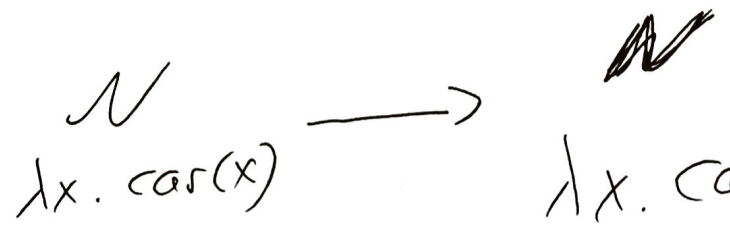
Categories



(3)



N
λx. car(x)

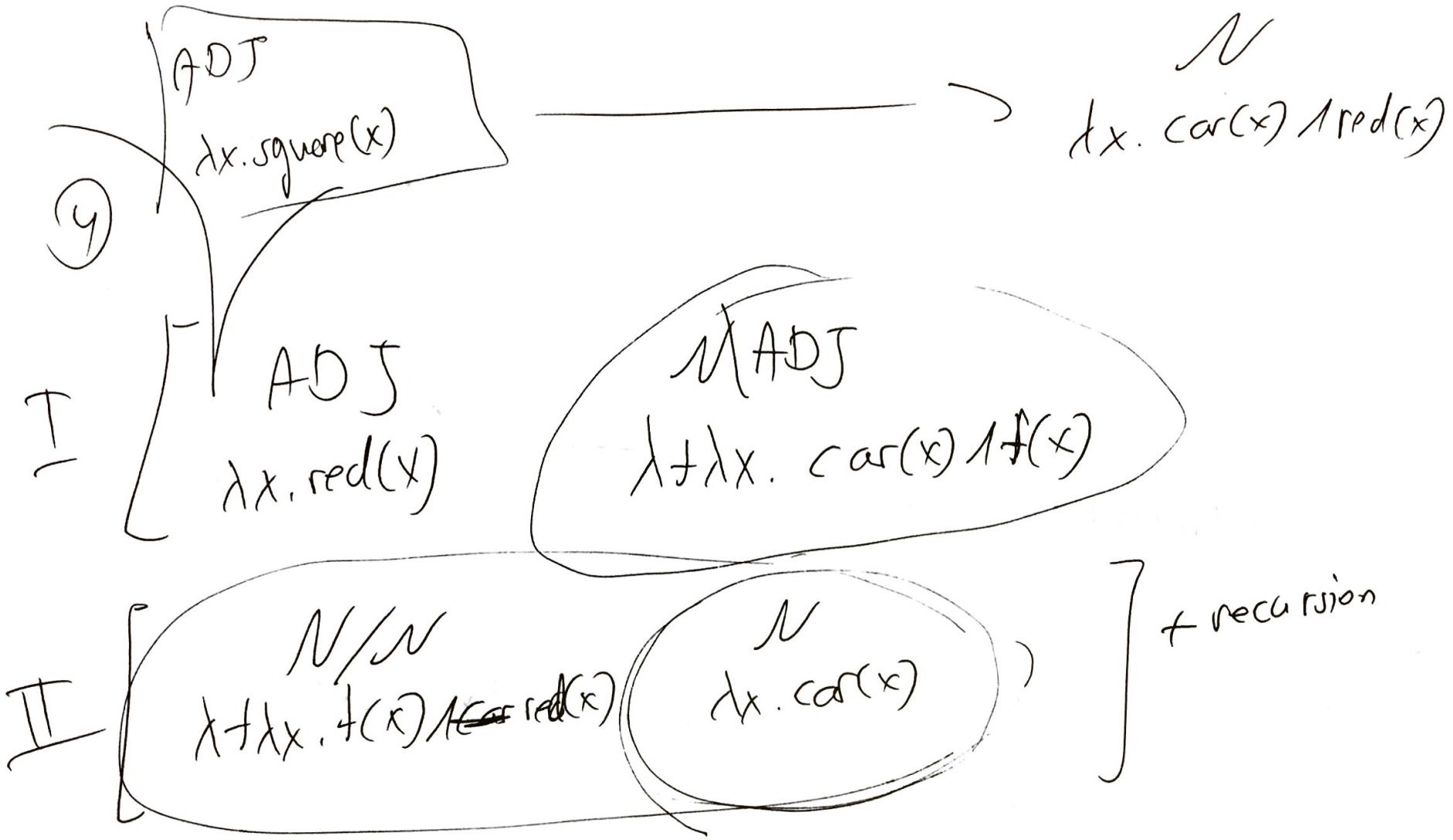


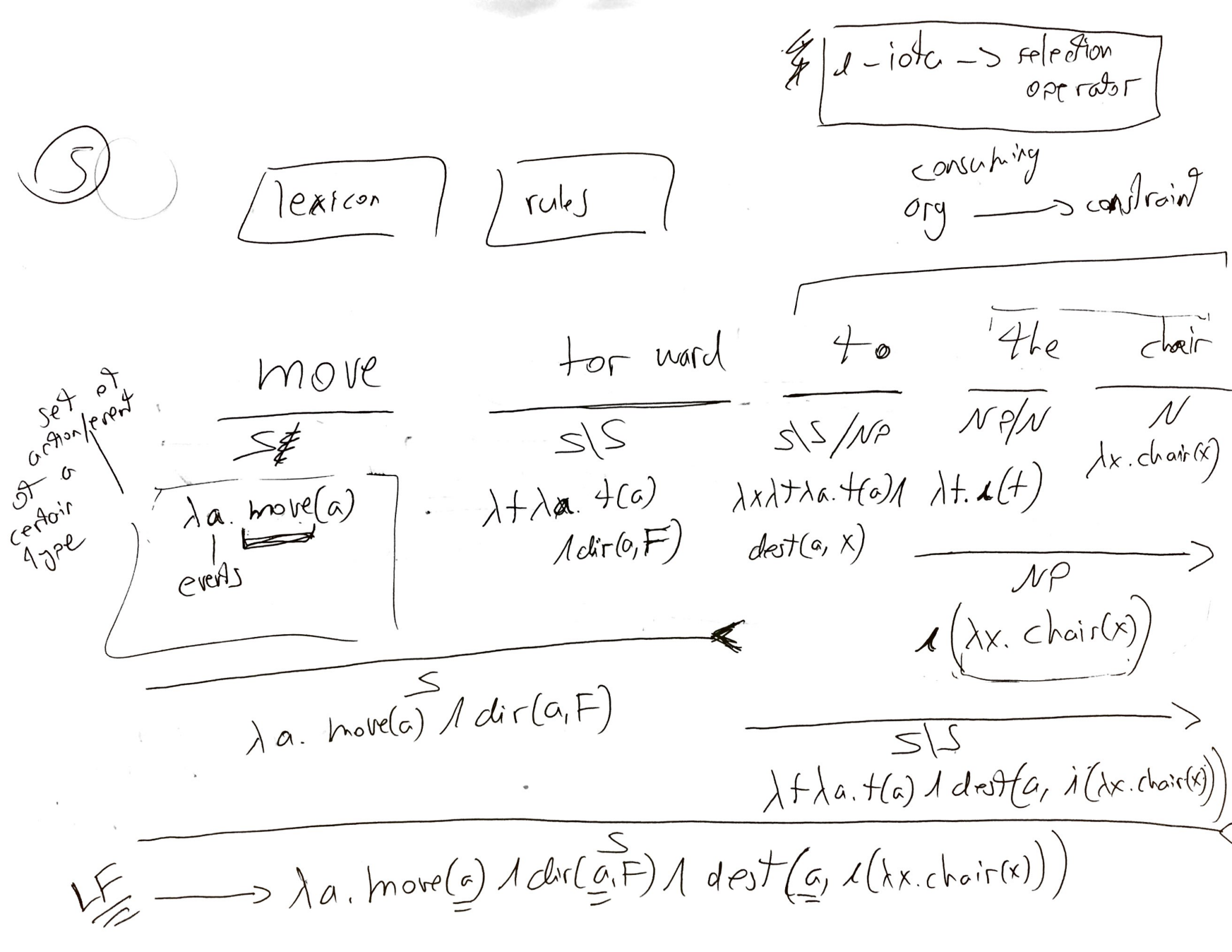
N
λx. car(x) ∧ red(x)

$(\lambda f \lambda x.\ f(x) \wedge red(x))\ (\lambda x.\ car(x))$

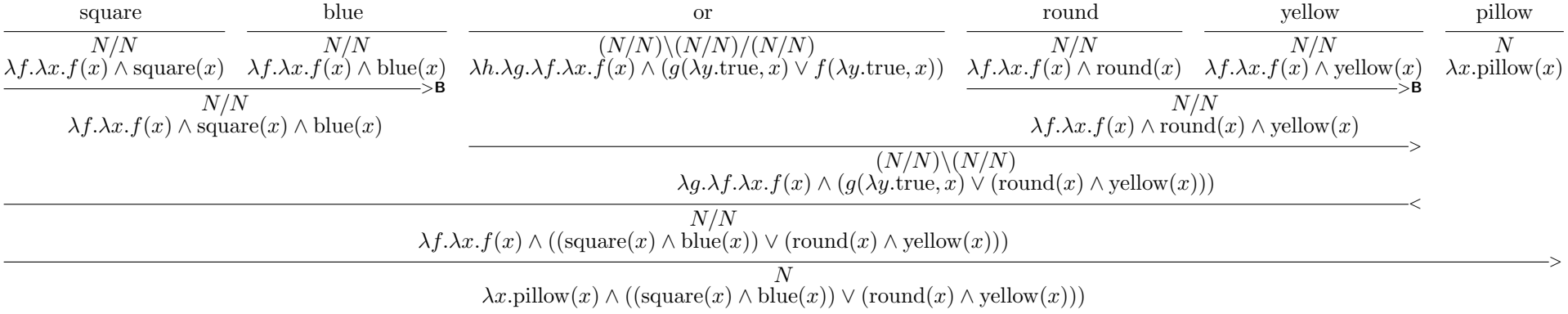
$\lambda x.\ (\lambda x.\ car(x))(x) \wedge red(x)$

$\lambda x.\ car(x) \wedge red(x)$


ADJ
λx.square(x)
N
λx. car(x) ∧ red(x)
9
I
ADJ
λx. red(x)
N/ADJ
λfλx. car(x) ∧ f(x)
II
N/N
λfλx. f(x) ∧ red(x)
N
λx. car(x)
+ recursion


ι - iota -> selection operator
S
lexicon
rules
consuming org → constraint
move
forward
to
the
chair
S
S\S
S\S/NP
NP/N
N
λx.chair(x)
λa. move(a)
events
set of action/event of a certain type
λf λa. f(a) ∧ dir(a, F)
λx λf λa. f(a) ∧ dest(a, x)
λf. ι(f)
NP
ι(λx. chair(x))
S
λa. move(a) ∧ dir(a, F)
S\S
λf λa. f(a) ∧ dest(a, ι(λx.chair(x)))
S
LF
λa. move(a) ∧ dir(a, F) ∧ dest(a, ι(λx.chair(x)))

Complex Conjunction with Composition



(6)

lexicon

the ↦ N/N : λf.ι(f)

Templates

ADJ → N/N : λfλx.f(x) ∧ owned(x, ι(λy.house(y)))

ι(λx.dog(x) ∧ owned(x, ι(λy.house(y))))

the house dog

the dog of the house

Lexeme

N : λy.house(y)

ADJ → N/N : λfλx.f(x) ∧ owned(x, ι(λy.garden(y)))

the garden dog

ι(λx.dog(x) ∧ owned(x, ι(λy.garden(y))))

Lexeme – atomic meaning w/o compositionality

Template – compositionality of category

Factored Lexicon

Original Lexicon

flight $\vdash S|NP : \lambda x.flight(x)$

flight $\vdash S|NP/(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$

flight $\vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$

ground transport $\vdash S|NP : \lambda x.trans(x)$

ground transport $\vdash S|NP/(S|NP) : \lambda f.\lambda x.trans(x) \wedge f(x)$

ground transport $\vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.trans(x) \wedge f(x)$

Factored Lexicon

(flight, $\{flight\}$)

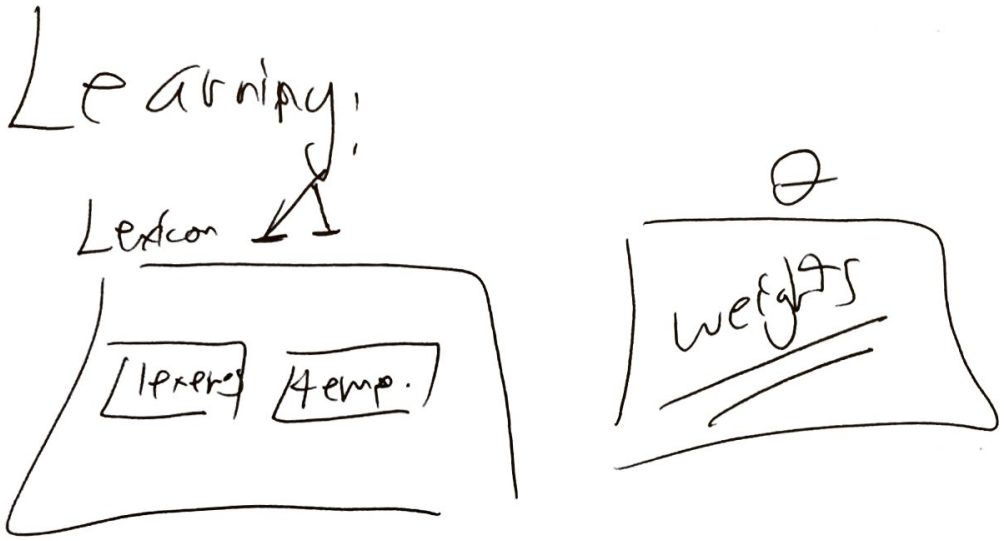
(ground transport, $\{trans\}$)

$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP : \lambda x.v_1(x)]$

$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP/(S|NP) : \lambda f.\lambda x.v_1(x) \wedge f(x)]$

$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.v_1(x) \wedge f(x)]$

Learning:



Helper functions:

- validation func: $V(\lambda) \rightarrow +/-$ (parse)
- genlex – lexical generation: $genlex(x, V; \Lambda, \theta)$
  - x: sentence
  - Λ: lexicon
  - θ: weights
  - genlex: (over) generate potential lexical entries

Structured Perceptron

- Simple additive updates
  - Only requires efficient decoding (argmax)
  - Closely related to MaxEnt and other feature rich models
  - Provably finds linear separator in finite updates, if one exists
- Challenge: learning with hidden variables

Structured Perceptron

Data: $\{(x_i, y_i) : i = 1 \dots n\}$

For $t = 1 \dots T$: [iterate epochs]

For $i = 1 \dots n$: [iterate examples]

$y^* \leftarrow \arg\max_y \langle \theta, \Phi(x_i, y) \rangle$ [predict]

If $y^* \neq y_i$: [check]

$\theta \leftarrow \theta + \Phi(x_i, y_i) - \Phi(x_i, y^*)$ [update]

weight/parameters

feature function

parse

gold label

argmax

[Collins 2002]

One Derivation of the Perceptron

Log-linear model: $p(y|x) = \frac{e^{w \cdot f(x,y)}}{\sum_{y'} e^{w \cdot f(x,y')}}$

Step 1: Differentiate, to maximize data log-likelihood

$$update = \sum_i f(x_i, y_i) - E_{p(y|x_i)} f(x_i, y)$$

Step 2: Use online, stochastic gradient updates, for example i:

$$update_i = f(x_i, y_i) - E_{p(y|x_i)} f(x_i, y)$$

Step 3: Replace expectations with maxes (Viterbi approx.)

$$update_i = f(x_i, y_i) - f(x_i, y^*) \text{ where } y^* = \arg\max_y w \cdot f(x_i, y)$$

Hidden Variable Perceptron

Data: $\{(x_i, y_i) : i = 1 \ldots n\}$

For $t = 1 \ldots T$: [iterate epochs]

hidden variables

For $i = 1 \ldots n$: [iterate examples]

$y^*, h^* \leftarrow \arg\max_{y,h} \langle \theta, \Phi(x_i, h, y) \rangle$ [predict]

If $y^* \neq y_i$: [check]

$h' \leftarrow \arg\max_h \langle \theta, \Phi(x_i, h, y_i)$ [predict hidden]

$\theta \leftarrow \theta + \Phi(x_i, h', y_i) - \Phi(x_i, h^*, y^*)$ [update]

[Liang et al. 2006; Zettlemoyer and Collins 2007]

The Perceptron with Hidden Variables

Log-linear model: $p(y|x) = \sum_h p(y, h|x)$

hidden variables

$$p(y, h|x) = \frac{e^{w \cdot f(x,h,y)}}{\sum_{y',h'} e^{w \cdot f(x,h',y')}}$$

Step 1: Differentiate marginal, to maximize data log-likelihood

$$update = \sum_i E_{p(h|y_i,x_i)}[f(x_i, h, y_i)] - E_{p(y,h|x_i)}[f(x_i, h, y)]$$

Step 2: Use online, stochastic gradient updates, for example i:

$$update_i = E_{p(y_i,h|x_i)}[f(x_i, h, y_i)] - E_{p(y,h|x_i)}[f(x_i, h, y)]$$

Step 3: Replace expectations with maxes (Viterbi approx.)

$$update_i = f(x_i, h', y_i) - f(x_i, h^*, y^*) \text{ where}$$

$$y^*, h^* = \arg\max_{y,h} w \cdot f(x_i, h, y) \quad \text{and} \quad h' = \arg\max_h w \cdot f(x_i, h, y_i)$$

Hidden Variable Perceptron

- No known convergence guarantees
  - Log-linear version is non-convex
- Simple and easy to implement
  - Works well with careful initialization
- Modifications for semantic parsing
  - Lots of different hidden information
  - Can add a margin constraint, do probabilistic version, etc.

Learning Algorithm

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

- Online
- Input:

$$\{(x_i, \mathcal{V}_i) : i = 1 \ldots n\}$$

- 2 steps:
  - Lexical generation
  - Parameter update

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from
$GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

weights

initial lexicon

lexicon of our grammar

iteration

GENLEX overgenerates

n - number of training samples

potential entries

risk!

temp. lexicon

Λ - current lexicon

all parses that are valid $\mathcal{V}_i$ and are max-scoring

Lexical entries

CCG lexicon

operator

determined

GEN → solves a search problem

all parses

all parses that are highest scoring

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$ and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:

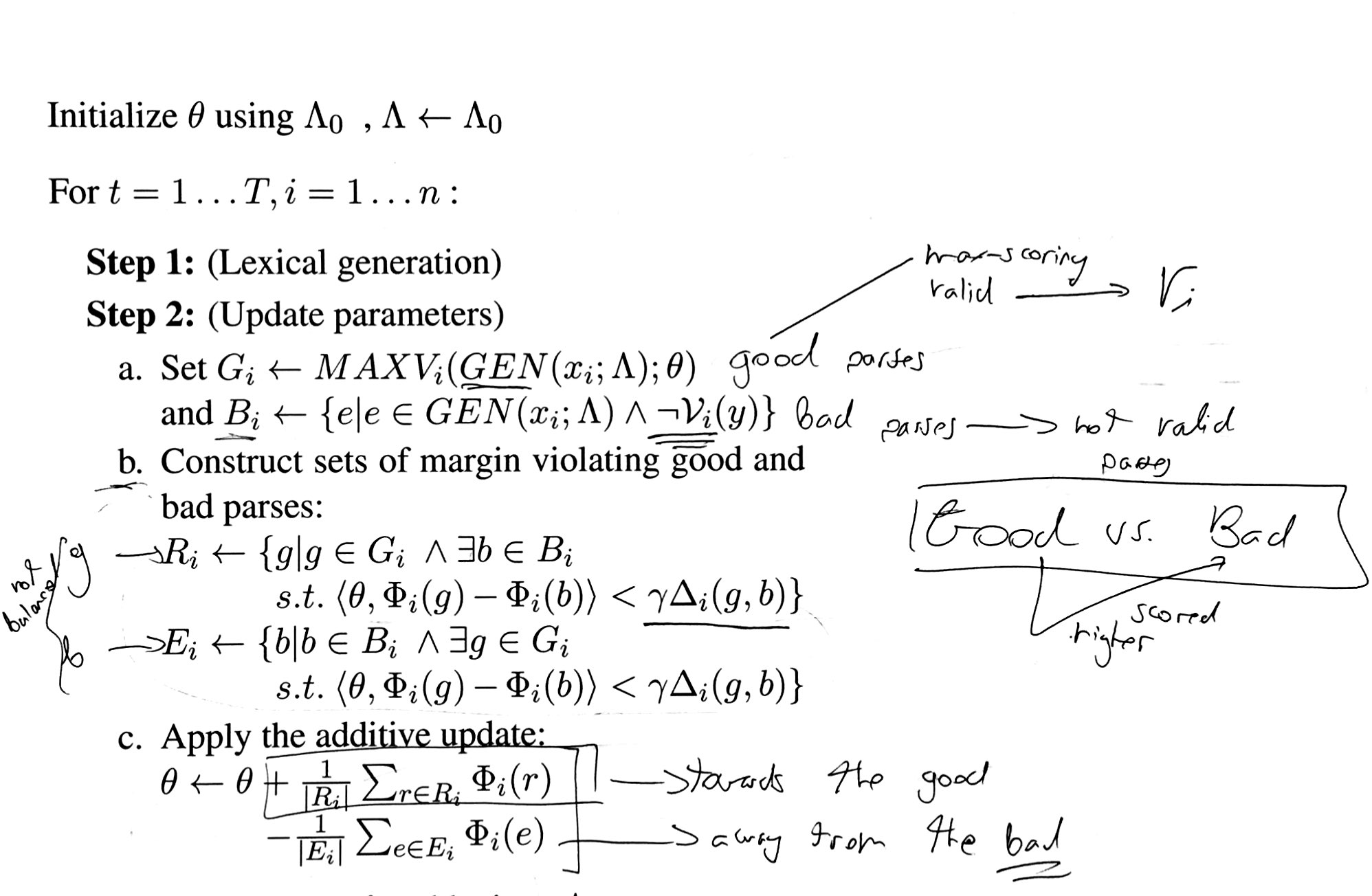
$$R_i \leftarrow \{g | g \in G_i \wedge \exists b \in B_i \; s.t. \; \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$$

$$E_i \leftarrow \{b | b \in B_i \wedge \exists g \in G_i \; s.t. \; \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$$

c. Apply the additive update:

$$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r) - \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$$

Output: Parameters θ and lexicon Λ



1) Validation → is the LF at the root of a tree ~~is~~ equivalent to z_i

2) Genlex

$love(Lucy, I) \xrightarrow{V_i} X$

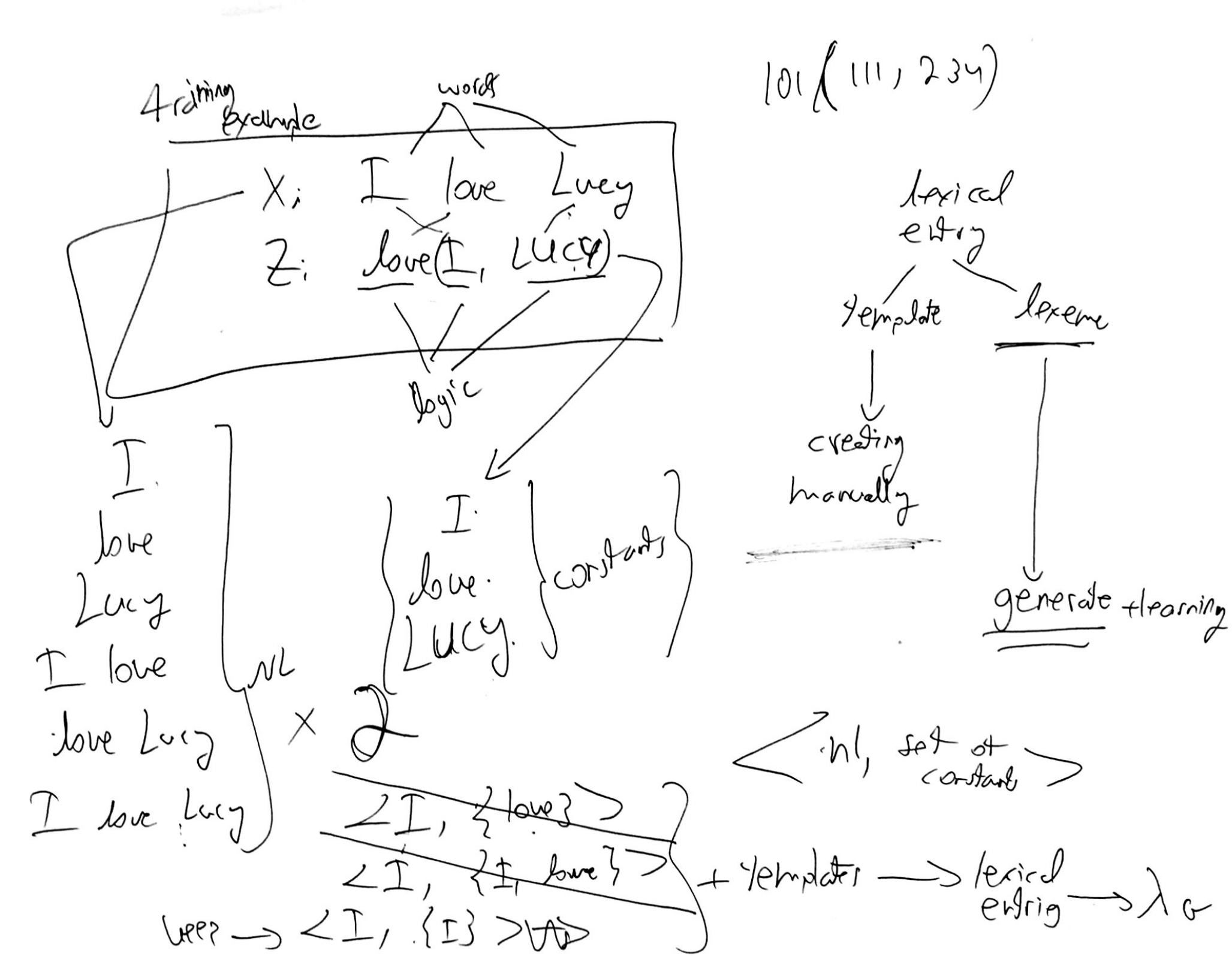
$D = \{(x_i, z_i)\}_{i=1}^{N}$

x_i: I love Lucy

z_i: love(I, Lucy)

sentence LF

y_i — tree latent


101 (111, 234)
training example
words
X: I love Lucy
Z: love(I, LUCY)
logic
lexical entry
template
lexeme
creating manually
generate + learning
I
love
Lucy
I love
love Lucy
I love Lucy
NL
I
love.
Lucy.
constants
×
< nl, set of constant >
< I, {love} >
< I, {I, love} >
+ templates → lexical entry → λ c
keep → < I, {I} > ∀

Learning Algorithm: Step-by-step

Unified Learning Algorithm

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

- Online
- Input:

  $\{(x_i, \mathcal{V}_i) : i = 1 \ldots n\}$
- 2 steps:
  - Lexical generation
  - Parameter update

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Initialize parameters and lexicon

θ weights

Λ_0 initial lexicon

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Iterate over data

T # iterations

n # samples

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Generate a large set of potential lexical entries

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

$GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$
lexical generation function

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Generate a large set of potential lexical entries

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

$GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$
lexical generation function

Procedure to propose potential new lexical entries for a sentence

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Generate a large set of potential lexical entries

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

$GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$
lexical generation function

$$\mathcal{V} : \mathcal{Y} \rightarrow \{t, f\}$$

$\mathcal{Y}$ all parses

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Get top parses

x_i sentence

k beam size

$GEN(x_i; \lambda)$ set of all parses

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Get lexical entries from highest scoring valid parses

θ weights

$\mathcal{V}$ validation function

$LEX(y)$ set of lexical entries

$\phi_i(y) = \phi(x_i, y)$

$MAXV_i(Y; \theta) = \{y | y \in Y \wedge \mathcal{V}_i(y) \wedge$
$\forall y' \in Y . \mathcal{V}_i(y) \implies$
$\langle \theta, \Phi_i(y') \rangle \leq \langle \theta, \Phi_i(y) \rangle\}$

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
$\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let Y be the k highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
$\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Update model's lexicon

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:
$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i$
$s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i$
$s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:
$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$
$- \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

Output: Parameters θ and lexicon Λ

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$ and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:

$$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i \ s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$$

$$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i \ s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$$

c. Apply the additive update:

$$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r) - \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$$

Output: Parameters θ and lexicon Λ

Re-parse and group all parses into 'good' and 'bad' sets

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

$GEN(x_i; \lambda)$ set of all parses

$\phi_i(y) = \phi(x_i, y)$

$MAXV_i(Y; \theta) = \{y | y \in Y \wedge \mathcal{V}_i(y) \wedge \forall y' \in Y . \mathcal{V}_i(y) \implies \langle \theta, \Phi_i(y') \rangle \leq \langle \theta, \Phi_i(y) \rangle\}$

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$ and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:

$$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i \ s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$$

$$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i \ s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$$

c. Apply the additive update:

$$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r) - \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$$

Output: Parameters θ and lexicon Λ

For all pairs of 'good' and 'bad' parses, if their scores violate the margin, add each to 'right' and 'error' sets respectively

θ weights

γ margin

$\phi_i(y) = \phi(x_i, y)$

$\Delta_i(y, y') = |\Phi_i(y) - \Phi_i(y')|_1$

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

b. Construct sets of margin violating good and bad parses:
$R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i$
$s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
$E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i$
$s.t. \ \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:
$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$
$- \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

Output: Parameters θ and lexicon Λ

Update towards violating 'good' parses and against violating 'bad' parses

θ weights

$\phi_i(y) = \phi(x_i, y)$

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Return grammar

θ weights

Λ lexicon

Unified Learning Algorithm

$\mathcal{V}$ validation function

$GENLEX(x, \mathcal{V}; \lambda, \theta)$

lexical generation function

- Two parts of the algorithm we still need to define
- Depend on the task and supervision signal