Interactive language learning from two extremes

Sida I. Wang, Percy Liang, Christopher D. Manning

+ Sam Ginn, Nadav Lidor

Stanford University
Natural language interfaces

1. Alexa
   (Amazon Echo)

2. Cortana
   (Windows 10 Phone)

3. Siri
   (iPhone)

4. Google Now
   (Android)
Natural language interfaces

Stephen Colbert: write the show

SIRI: what would you like to search for?

...

Stephen Colbert: For the love of God, the cameras are on, give me something!

SIRI: What kind of place are you looking for, camera stores or churches?
Engineering goals

we are stuck when these systems misunderstand us
receive feedback from users, and improve through use
Engineering goals

we are stuck when these systems misunderstand us
receive feedback from users, and improve through use

• Adapt to users

  *regular weekday alarm, cancel the friday meeting*
Engineering goals

we are stuck when these systems misunderstand us
receive feedback from users, and improve through use

- Adapt to users
  - *regular weekday alarm, cancel the friday meeting*

- Handle special domains and low resource languages
  - familiar words take on new meaning
  - *revert to commit 25ad3*
  - *order buy red t5 2*
Engineering goals

we are stuck when these systems misunderstand us
receive feedback from users, and improve through use

- **Adapt to users**
  
  *regular weekday alarm, cancel the friday meeting*

- **Handle special domains and low resource languages**
  
  familiar words take on new meaning
  
  *revert to commit 25ad3*  
  
  *order buy red t5 2*

- **Perform complex actions**
  
  *move my meeting with Percy to the same time as my meeting with Chris*  
  
  *call Bob every hour until he picks up, stop after 8 tries*
Research questions

- How to learn from scratch quickly?
- How to learn to perform complex, custom actions?
Main outline

• **Extreme 1: learning language games from scratch**
• **Extreme 2: naturalizing a programming language**
Learning language games

Wittgenstein. 1953. Philosophical Investigations:

*Language derives its meaning from use.*

'block' 'pillar' 'slab' 'beam'.
Interactive language game

• Iterated, cooperative game between human and computer

• The human player
  • has a goal, cannot perform actions
  • can use language and provide feedback

• The computer player
  • does not know goal, can perform the actions
  • does not understand language
Interactive language game

- Iterated, cooperative game between human and computer

- The human player
  - has a goal, cannot perform actions
  - can use language and provide feedback
  - must teach the computer a suitable language, and adapt

- The computer player
  - does not know goal, can perform the actions
  - does not understand language
  - must learn language quickly through interaction
SHRDLURN

start

✓
SHRDLURN

start

goal

✓
has a goal
has language

perform actions
does not talk

remove red

SHRDLURN

start

goal
has a goal

performs actions

has language
does not talk

SHRDLURN
remove red
add(leftmost(hascolor(red)),red)
add(red, hascolor(cyan))
remove(hascolor(red))
remove(leftmost(hascolor(red)))
SHRDLURN

remove red
add(leftmost(hascolor(red)),red)
add(red, hascolor(cyan))
remove(hascolor(red))
remove(leftmost(hascolor(red)))

has a goal performs actions
has language does not talk
SHRDLURN

remove red
add(leftmost(hascolor(red)),red)
add(red, hascolor(cyan))
remove(hascolor(red))
remove(leftmost(hascolor(red)))

has a goal performs actions
has language does not talk

has a goal

perform actions

has language
does not talk

start

goal
SHRDLURN

has a goal
has language

performs actions
does not talk

start

goal

把 红的 拿走
add(leftmost(hascolor(red)),red)
add(red, hascolor(cyan))
remove(hascolor(red))
remove(leftmost(hascolor(red)))
has a goal
has language

performs actions
does not talk

emoveray edray

\begin{align*}
\text{add(} & \text{leftmost(} \text{hascolor(red)} \text{),red)} \\
\text{add(} & \text{red, hascolor(cyan)} \\
\text{remove(} & \text{hascolor(red)} \\
\text{remove(} & \text{leftmost(} \text{hascolor(red)} \text{))} 
\end{align*}
enter a command, you did it! solve this puzzle 6 more times to advance.

remove right red
Outline

- **Computer:** semantic parsing
- **Human:** 100 Turkers
- **Pragmatics**
- **Updates**
Semantic parsing

Actions as logical forms:

```
add(hascolor(red), cyan)
```
Semantic parsing

Actions as logical forms:

- add(hascolor(red), cyan)
- remove(rightmost(all()))
- remove(rightmost(hascolor(orange))))
"Parsing" freely

- Generate logical forms
  - start from the smallest size
  - score them with a model
  - use beam search to find longer high-scoring logical forms
  - like the floating parser [Pasupat and Liang 2015]

```prolog
brown
hascolor(brown)
leftmost(hascolor(brown))
diff(all(),leftmost(hascolor(brown))
remove(diff(all(),leftmost(hascolor(brown))))
```
Model

log-linear model with features $\phi(x, z)$:

$$p_\theta(z \mid x) \propto \exp(\phi(x, z) \cdot \theta)$$

$x$: add a cyan block to red blocks
$z$: add(hascolor(red), cyan)
Learning from denotations

\[ p_\theta(z \mid x) \propto \exp(\phi(x, z) \cdot \theta) \]

\( x : \text{add a cyan block to red blocks} \)
\( z : \text{add(hascolor(red), cyan)} \)

\( y : \)

\[
\begin{array}{c}
\text{start}
\end{array}
\]
Learning from denotations

\[ p_\theta(z \mid x) \propto \exp(\phi(x, z) \cdot \theta) \]

\[ p_\theta(y \mid x) = \sum_{z: \text{Exec}(z) = y} p_\theta(z \mid x) \]

\[ x : \text{add a cyan block to red blocks} \]
\[ z : \text{add(hascolor(red), cyan)} \]

\[ y : \]

\[ \text{start} \]
Learning from denotations

\[ p_{\theta}(z \mid x) \propto \exp(\phi(x, z) \cdot \theta) \]

\[ p_{\theta}(y \mid x) = \sum_{z: \text{Exec}(z) = y} p_{\theta}(z \mid x) \]

\[ x: \text{add a cyan block to red blocks} \]
\[ z: \text{add(hascolor(red), cyan)} \]

\[ y: \]

\[ \text{start} \]

L1 penalty and update with AdaGrad
Background on features/model

Features $\phi(x, z)$: arbitrary mapping from $x, z$ to strings

- **feature**: size($x$), size($z$)
- **example**: ”sizes: 10,5”
- **weight**: -2.5

- **feature**: $x \neq z$
- **example**: ”remove red $i=\dot{i}$ remove(red)”
- **weight**: 3.1

Parameters $\theta \cdot \phi(x, z)$: scores a mapping based on its features

$p_\theta(z \mid x) \propto \exp(\phi(x, z) \cdot \theta)$: assigns probabilities to possible mappings
Features

\[ \text{add} \]

\[ 1 \quad 2 \]

\text{leftmost} \quad \text{orange}

\text{hascolor} \quad \text{red}

\[ \uparrow \]

*put orange on the very left red block*
Features

uni-, bi-, skip- grams
put, orange, on, the
put orange, orange on, ..., put * on, orange * the, ..., 

put orange on the very left red block
Features

uni-, bi-, skip- grams
put, orange, on, the
put orange, orange on, ...
p * on, orange * the, ...
tree-grams
add(leftmost(*), orange)
leftmost(hascolor(*))
λc.(hascolor(c))

put orange on the very left red block
Features

uni-, bi-, skip- grams
- put, orange, on, the
- put orange, orange on, ..., put * on, orange * the, ...

tree-grams
- add(leftmost(*), orange)
- leftmost(hascolor(*))
- λc.(hascolor(c))

cross product features
- (put,add(*,*))
- (put orange,add(*,orange))
- (put,orange)

put orange on the very left red block
Outline

- Computer: semantic parsing
- **Human: 100 Turkers**
- Pragmatics
- Updates
Experiments

• 100 Turkers played SHRDLURN
  • Got 10223 utterances in total (6 hrs to complete)
Experiments

- 100 Turkers played SHRDLURN
  - Got 10223 utterances in total (6 hrs to complete)

- Minimal instructions
  - No examples provided to avoid bias
  - Instructed to use any language
Experiments

- 100 Turkers played SHRDLURN
  - Got 10223 utterances in total (6 hrs to complete)

- Minimal instructions
  - no examples provided to avoid bias
  - instructed to use any language

- Some players liked the game
  - "That was probably the most fun thing I have ever done on mTurk."
  - "This is SO SO cool. I wish there were a way I could better contribute because this research seems to be just insanely interesting and worthwhile."
Experiments

• 100 Turkers played SHRDLURN
  • Got 10223 utterances in total (6 hrs to complete)

• Minimal instructions
  • no examples provided to avoid bias
  • instructed to use any language

• Some players liked the game
  • “That was probably the most fun thing I have ever done on mTurk.”
  • “This is SO SO cool. I wish there were a way I could better contribute because this research seems to be just insanely interesting and worthwhile.”

• performance is measured by the amount of scrolling needed
precise and consistent:

(3.01)
rem cy pos 1
stack or blk pos 4
rem blk pos 2 thru 5
rem blk pos 2 thru 4
stack bn blk pos 1 thru 2
fill bn blk
stack or blk pos 2 thru 6
rem cy blk pos 2 fill rd blk

(2.72)
Remove the center block
Remove the red block
Remove all red blocks
Remove the first orange block
Put a brown block on the first brown block
Add blue block on first blue block

(2.78)
remove the brown block
remove all orange blocks
put brown block on orange blocks
put orange blocks on all blocks
put blue block on leftmost blue block in top row
Results: average players (rank 21-50)

inconsistent or mismatches computer capability:

(9.17)
- reinsert pink
- take brown
- put in pink
- remove two pink from second layer
- Add two red to second layer in odd intervals
- Add five pink to second layer
- Remove one blue and one brown from bottom layer

(8.37)
- remove red
- remove 1 red
- remove 2 4 orange
- add 2 red
- add 2 3 4 5 blue
- remove 1 3 5 orange
- add 2 4 orange
- add 2 orange
- remove 2 3 brown
- add 1 2 3 4 5 red
- remove 2 3 4 5 6
- remove 2
- add 1 2 3 4 6 red
Results: worst players (rank 51-100)

spammy, vague, did not tokenize:

(12.6)

‘add red cubes on center left
center right
far left and far right’
‘remove blue blocks on row two column two
row two column four’
remove red blocks in center left and center right on second row

(14.15)

holdleftmost
holdbrown
holdleftmost
blueonblue
brownonblue1
blueonorange
holdblue
holdorange2
blueonred2
holdends1
holdrightend
hold2
orangeonorangerrightmost
Results: interesting players

(Polish)
usuń brązowe klocki
usuń niebieski klock
usuń pomarańczowe klocki
usuń czerwony klock
postaw brązowy klock na pierwszym klocku
postaw czerwony klock na pierwszym klocku
postaw pomarańczowe klocki na brązowych
postaw czerwone klocki
usuń ostatni brązowy klock
usuń wszystkie klocki oprócz ostatniego
postaw niebieski klock na czerwonym
postaw brązowy klock na pierwszym klocku

(Polish notation)
rm scat + 1 c
+ 1 c
rm sh
+ 1 2 4 sh
+ 1 c
- 4 o
rm 1 r
+ 1 3 o
full fill c
rm o
full fill sh
- 1 3
full fill sh
rm sh
rm r
+ 2 3 r
rm o
+ 3 sh
+ 2 3 sh
Players adapt

- More consistent
  - *remove, delete* $\rightarrow$ *remove*

- More concise
  - *Remove the red ones* $\rightarrow$ *Remove red*
  - *add brown on top of red* $\rightarrow$ *add orange on red*
  - *the, a* $\rightarrow$ $\epsilon$
Learning works fairly well, especially for top players
Outline

• Computer: semantic parsing
• Human: 100 Turkers
• Pragmatics
• Updates
Pragmatics: motivation

delete cardinal

remove(hascolor(red))
Pragmatics: motivation

delete cardinal

remove(hascolor(red))

delete cyan
Pragmatics: motivation

\textit{delete cardinal}

\texttt{remove(hascolor(red))}

\textit{delete cyan}

\texttt{remove(hascolor(red))}
\texttt{remove(hascolor(cyan))}
\texttt{remove(hascolor(brown))}
Pragmatics: motivation

delete cardinal

remove(hascolor(red))

delete cyan

remove(hascolor(red))
remove(hascolor(cyan))
remove(hascolor(brown))

Intuition: cooperative communication
Pragmatics: model

[Golland et al. 2010; Frank/Goodman, 2012]
Pragmatics: example

Listener (computer):

\[ p_\theta(z \mid x) : \text{semantic parsing model} \]

<table>
<thead>
<tr>
<th>Action</th>
<th>Remove (red)</th>
<th>Remove (cyan)</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete cardinal</td>
<td>0.8</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>delete cyan</td>
<td><strong>0.6</strong></td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Pragmatics: example

Speaker (human):

\[ S(x \mid z) \propto p_{\theta}(z \mid x)p(x) \]

(assume \( p(x) \) uniform)

remove(red) remove(cyan) others

\[
\begin{array}{ccc}
\text{delete cardinal} & 0.57 & 0.33 & 0.33 \\
\text{delete cyan} & 0.43 & 0.67 & 0.67 \\
\end{array}
\]
Pragmatics: example

Listener (computer):

\[ L(z \mid x) \propto S(x \mid z)p(z) \]

(assume \( p(z) \) uniform)

- remove(red)  
- remove(cyan)  
- others

<table>
<thead>
<tr>
<th>delete cardinal</th>
<th>0.46</th>
<th>0.27</th>
<th>0.27</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete cyan</td>
<td>0.24</td>
<td>0.38</td>
<td>0.38</td>
</tr>
</tbody>
</table>
Pragmatics: results

<table>
<thead>
<tr>
<th></th>
<th>Online accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>No pragmatics</td>
<td>33.3</td>
</tr>
<tr>
<td>Pragmatics</td>
<td>33.8</td>
</tr>
</tbody>
</table>
Pragmatics helps top (cooperative, rational) players
Outline

- Computer: semantic parsing
- Human: 100 Turkers
- Pragmatics
- Updates
The real data

- Data from June 2016 - Feb 2017
  - 19k+ examples, 1.2k+ sessions

(NLPers?)

add brown on the top unless the rightmost
not(red)
pick up blue blocks
+ 1 2 3 4 5 r
Not the brown block!
The orange block!
捨てて 桃の 0 1
待ち 桃の 東
捨てない 1 4
add blo 1 bro
rem ora blo
add blo 6 pin
add blo 134 bl
去掉最后一个块
在蓝色块上面加一层橙色块
smaz 1 a 3 jedenou
retire les blocs bleus

(NLPers?)

move all blocks but middle
- 1 br - 4 br - 6 br
一番奥にオレンジを置く
一番右の赤を消す
add red one on the first
lift 1 3 5
add one orange block on top of each orange
去掉 蓝色 方块
smaz 1 a 2 a 3 a 5
quita el bloque marrón
quita el primer bloque por la derecha
drop orange not left not right
add brown on all blue in line 2 in line 3
Add x x o x o x red block
只保留桔黄色的方块
quitar cubo rojo
quitar ultimo cubo rojo
Diverse language in blocks world

**English-like**

- add brown on the top unless the rightmost
- add a brown block on top of the right-most red block
- move all blocks but middle
- add red on top of first brown,
- add blue blocks on top of left 3 blocks
- drop orange 1

**Code**

- add blo 1 bro
- 1 br - 4 6 br
- lift 1 3 5
- + 1 2 3 4 5 r
- Add x x o x o x red block

**Foreign**

- 一番奥にオレンジを置く
- 只保留桔黄色的方块
- quita el primer bloque por la derecha
- ḡe³ ḡar³ 0 1
- retire les blocs bleus
- quitar ultimo cubo rojo
- postav na kazhdiy goluboy blok vo vtorom ryadu po korichnevomu bloku
Learning language games findings

- our system learns from scratch, quickly

- modelling pragmatics is helpful

- people adapts to the computer
  - given the chance, people use very diverse language
Drawbacks

selection as supervision signal cannot scale very well

• number of logical forms is exponential in length
  (:blk (:loop 4 (:s (:blk (:loop 2 (:s (:blk (:loop 3(:s (: add red here) (:for (call adj top) (: select))))(:for (call adj left) (: select))))))) (:for (call adj back) (: select)))))

each user has a private language – and no sharing

• the system does not continue to improve with more users

action space unclear, not communicated to users

• Add x x o x o x red block – remove 2 4 6 8 – lift 1 3 5
Main outline

• Extreme 1: learning language games from scratch
• Extreme 2: naturalizing a programming language
Goal

• handle more complex actions / programs
  • *put cols B and D in a scatter plot against col A*
  • *lowercase the first letter of all my bullets*
  • *move all my future meetings with Bob ahead by 1 hour*
  • *street with palm trees 5 spaces apart*

• evolve the language through use in a community
  • system continues to improve through use

• define and accommodate the action space
Motivation

- formal language
  - unambiguous, compose tractably
- learning through definitions
  - 3 by 4 red square := 3 red columns of height 4
  - no need to infer from many examples
  - build up complex concepts hierarchically

"There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians"
→ language derives its meaning through definition
Naturalization

• seed the system with a core programming language
  • expressive and defines action space, but tedious to use
• user teach the system by defining new things
  • ”X” means ”Y”
• evolve the language to be more natural to people while accommodating the system action space

learn from how people try to program
Shared community learning

- all users teach one system
  - initial users need to know some of the core language
  - later users can use what initial users taught
- better for new users
  - after enough usage, most simple variations are covered
- easier to use for power users
  - allowing them to customize and share
Voxelurn

- world is a set of objects with relations
  - Voxels: \((x, y, z, \text{color})\)
  - domain specific relation: [direction]: left, top, front, etc.
- domain specific actions: add, move
Core language

• programming language designed to interpolate with NL
• controls: if, foreach, repeat, while
• lambda DCS for variable-free joins, set ops, etc.
  • has color yellow or color of has row 1
• selection to avoid variables
  • select left of this
• block-structured scoping
  • , [], isolate
# Core language (domain general)

<table>
<thead>
<tr>
<th>Rule(s)</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow A; A$</td>
<td>select left; add red</td>
</tr>
<tr>
<td>$A \rightarrow \text{repeat } N \ A$</td>
<td>repeat 3-1 add red top</td>
</tr>
<tr>
<td>$A \rightarrow \text{if } S \ A$</td>
<td>if has color red [select origin]</td>
</tr>
<tr>
<td>$A \rightarrow \text{while } S \ A$</td>
<td>while not has color red [select left of this]</td>
</tr>
<tr>
<td>$A \rightarrow \text{foreach } S \ A$</td>
<td>foreach this [remove has row row of this]</td>
</tr>
<tr>
<td>$A \rightarrow [A ]$</td>
<td>[select left or right; add red; add red top]</td>
</tr>
<tr>
<td>$A \rightarrow { A }$</td>
<td>{select left; add red}</td>
</tr>
<tr>
<td>$A \rightarrow \text{isolate } A$</td>
<td>isolate [add red top; select has color red]</td>
</tr>
<tr>
<td>$A \rightarrow \text{select } S$</td>
<td>select all and not origin</td>
</tr>
<tr>
<td>$A \rightarrow \text{remove } S$</td>
<td>remove has color red</td>
</tr>
<tr>
<td>$A \rightarrow \text{update } R S$</td>
<td>update color [color of left of this]</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
</tr>
<tr>
<td>$R$ of $S$</td>
<td>has $R S$</td>
</tr>
<tr>
<td>not $S$</td>
<td>$S$ and $S$</td>
</tr>
<tr>
<td>$N$</td>
<td>$N+N$</td>
</tr>
<tr>
<td>argmax $R S$</td>
<td>argmin $R S$</td>
</tr>
</tbody>
</table>
Demo

- explain the definition process
- do palm tree, and cube, add green monster

```
begin execute x:
    if x does not parse then define x;
    if user rejects all parses then define x;
    execute user choice

begin define x:
    repeat starting with X ← []
        user enters x';
        if x' does not parse then define x';
        if user rejects all x' then define x';
        X ← [X; x'];
    until user accepts X as the def'n of x;
```
Palm tree example

- define new things in terms of what’s already defined
- everything trace back to the core language

add palm tree:

  add brown trunk height 3:

  go to top:

  add leaves here:
Palm tree example

- define new things in terms of what’s already defined
- everything trace back to the core language

add palm tree:

  add brown trunk height 3:
    add brown top 3 times:

  go to top:

  add leaves here:
Palm tree example

- define new things in terms of what’s already defined
- everything trace back to the core language

add palm tree:

add brown trunk height 3:

add brown top 3 times:

repeat 3 [add brown top]

go to top:

add leaves here:
Palm tree example

- define new things in terms of what’s already defined
- everything trace back to the core language

add palm tree:

  add brown trunk height 3:

    add brown top 3 times:

      repeat 3 [add brown top]

  go to top:

    select very top of all

add leaves here:
Palm tree example

• define new things in terms of what’s already defined
• everything trace back to the core language

add palm tree:

add brown trunk height 3:

add brown top 3 times:

repeat 3 [add brown top]

go to top:

select very top of all

add leaves here:

select left or right or front or back; add green
Model (now over derivations)

log-linear model with features $\phi(d, x, u)$:

$$p_\theta(d \mid x, u) \propto \exp(\phi(d, x, u) \cdot \theta)$$

$x$: *add two chairs 5 spaces apart*

$z$: (:blk (:loop ...))

$y$:
Learning from denotations

mainly for handling scoping automatically

\[ p_\theta(d | x, u) \propto \exp(\phi(d, x, u) \cdot \theta) \]

\( x \) : add two chairs 5 spaces apart
\( z \) : (:blk (:loop ...))
\( y \) :
Learning from denotations

mainly for handling scoping automatically

\[ p_\theta(d \mid x, u) \propto \exp(\phi(d, x, u) \cdot \theta) \]

\[ p_\theta(y \mid x, u) = \sum_{d : \text{Exec}(d) = y} p_\theta(d \mid x, y) \]

\( x \): add two chairs 5 spaces apart

\( z \): (: blk (: loop...))
Learning from denotations

mainly for handling scoping automatically

$$p_\theta(d \mid x, u) \propto \exp(\phi(d, x, u) \cdot \theta)$$

$$p_\theta(y \mid x, u) = \sum_{d: \text{Exec}(d)=y} p_\theta(d \mid x, y)$$

\(x: \text{add two chairs 5 spaces apart}\)

\(z: (: \text{blk} (: \text{loop}...))\)

\(y:\)

L1 penalty and update with AdaGrad
Derivation: process of deriving the formula from the utterance

- which rules are used
- where each thing comes from
- categories, types, etc.
## Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule.ID</td>
<td>ID of the rule</td>
</tr>
<tr>
<td>Rule.Type</td>
<td>core?, used?, used by others?</td>
</tr>
<tr>
<td>Social.Author</td>
<td>ID of author</td>
</tr>
<tr>
<td>Social.Friends</td>
<td>(ID of author, ID of user)</td>
</tr>
<tr>
<td>Social.Self</td>
<td>rule is authored by user?</td>
</tr>
<tr>
<td>Span</td>
<td>(left/right token(s), category)</td>
</tr>
<tr>
<td>Scope</td>
<td>type of scoping for each user</td>
</tr>
</tbody>
</table>
Definition

head:

(body X: (loop 3 (add red left)))

body X:

(loop 3 (add red left))

N → A → A

N

A

A
Grammar induction

- Want high precision rules
  - low precision: all users see more junk candidates
  - low recall: need more definitions
- Use the tree structure of derivation
  - instead of just the program
- Use both the derivation AND the utterance of the body
Grammar induction

Inputs: $x, X, d, \text{chart}(x)$

- $x$: add red top times 3
- $X$: repeat 3 [add red top] (often a sequence)
- $d$: (loop 3 (add red top)), and how it is derived
- $\text{chart}(x)$: 3, (add red top) and their derivations

Outputs:

- $A \rightarrow add \ C \ D \ times \ N : \lambda C D N. \text{repeat } N \ add \ C \ D$
- $A \rightarrow A \ times \ N : \lambda A N. \text{repeat } N \ [A]$
Grammar induction

Inputs: \( x, X, d, \text{chart}(x) \)

- \( x \): add red top times 3
- \( X \): repeat 3 [add red top] (often a sequence)
- \( d \): (loop 3 (add red top)), and how it is derived
- \( \text{chart}(x) \): 3, (add red top) and their derivations

Outputs:

- \( A \rightarrow \text{add } C \; D \; \text{times } N \; : \; \lambda C D N.\text{repeat} \; N \; \text{add} \; C \; D \)
- \( A \rightarrow A \; \text{times } N \; : \; \lambda A N.\text{repeat} \; N \; [A] \)
  - can be wrong: add red to row 2 times 2
Grammar induction

substitute matching derivations by their categories:

\[ \lambda AN. \text{repeat } N \ [A] \]

body:

\[(\text{loop } 3 \ (\text{add red left}))\]
Considerations

Simple heuristic would not always work:

• A1: highest coverage of 4 tokens
• A2: largest match

we extract the best scoring matches instead, inspired by GENLEX (Zettlemoyer and Collins, 2005)
Derivation scoping

\textit{put a chair leg}
\[ := \textit{brown column of height 3} \]

\textit{put 4 chair legs 3 spaces apart}
\[ := \textit{put a chair leg; move back 3 spaces; put a chair leg; move right 3 spaces; put a chair leg; move front 3 spaces; put a chair leg} \]
Highest scoring packing

- a span is a set of consecutive tokens
  - matching if the chart element is in definition

- a packing is a set of non-overlapping matching spans
  - maximal packing – no span to be added

- abstract away the highest scoring maximal packing

\[
P_l^* = \arg\max_{P \in \text{packing}(M); d \in P} \sum \text{score}(d).
\]

- solve with a dynamic program
Can people do this?

- *chair legs of height 3*

\[
\begin{align*}
\text{for (call adj top this) (: select)())} \\
\text{for (call adj bot this) (: select)())} \\
\text{for (call adj left this) (: select)())} \\
\text{for (call adj back this) (: select)())} \\
\text{for (call adj right this) (: select)())} \\
\end{align*}
\]
Experiments

- users built great structures?
Experiments

- users built great structures! (show leaderboard)
Setup

• qualifier: build a fixed structure
• post-qual: over 3 days build whatever they want
• prizes for best structures
  • day 1: bridge, house, animal
  • day 2: tower, monster(s), flower(s)
  • day 3: ship(s), dancer(s), and castle
• prize for top h-index
  • a rule (and its author) gets a citation whenever it is used
Basic statistics

- 70 workers qualified, 42 participated, 230 structures

- 64075 utterances, 36589 accepts
  - each accept leads to a datapoint labeled by derivation(s)

- 2495 definitions, 2817 induced rules (¡100 core)
Is naturalization happening

- 58% of all at the end (up from 0 in the beginning)
- 64.3% of all accepted, and 77.9% of the last 10k accepted
- Top users naturalized to different extents, but all increasing
Expressive power

- cumulative average of string.length in program / # tokens in utterance
- len(z)/len(z) is very stable at 10 for core language
- varies greatly by user
Modes of naturalization

short forms:

left, l, mov left, go left, \( i \), sel left

br, blk, blu, brn, orangeright, left3

add row brn left 5

:= add row brown left 5
Modes of naturalization

syntactic:

\[
\begin{align*}
go \text{ down and right} & : = \text{ go down; go right} \\
\text{select orange} & : = \text{ select has color orange} \\
\text{add red top 4 times} & : = \text{ repeat 4 [add red top]} \\
l \text{ white} & : = \text{ go left and add white} \\
\text{mov up 2} & : = \text{ repeat 2 [select up]} \\
\text{go up 3} & : = \text{ go up 2; go up}
\end{align*}
\]
Modes of naturalization

higher level:

\[\text{add black block width 2 length 2 height 3} := \{\text{repeat 3 [add black platform width 2...}\]

\[\text{flower petals} := \text{flower petal; back; flower petals}\]

\[\text{cube size 5, get into position start, 5 x 5 open green square, brownbase}\]
Citations

**basic statistics**: 1113 cited rules, median 3, mean 46

- left 3 : 5820
- select up : 4591
- right, ... : 2888
- go left : 1438
- select right 2 : 1268
- add b : 975
- add red top 4 times : 309
- go back and right : 272
- select orange : 256
- add white plate 6 x 7 : 232
- add brown row 3 : 203
- mov right 3 : 178
Bridge the gap in power

naturalizing a programming language:

• handle complex actions
• shared community learning
to cover more variations
• better for beginners and experts alike?

sidawmain:~ sidaw$ replace SF by Seattle in all html files modified within the last 3 days
-bash: replace: command not found
sidawmain:~ sidaw$

sidawmain:~ sidaw$ find . -mtime -3 -name '*.html'
-exec sed -i.bak 's/SF/Seattle/g' {} \

The two extremes

**LLG**: start from scratch, understands nothing, anything goes

**NPL**: start with a programming language and its power
The two extremes

**LLG**: start from scratch, understands nothing, anything goes

**NPL**: start with a programming language and its power

**LLG**: each user has a private language

**NPL**: user community has one shared language
  - with some user modelling
The two extremes

**LLG**: start from scratch, understands nothing, anything goes

**NPL**: start with a programming language and its power

**LLG**: each user has a private language

**NPL**: user community has one shared language
  - with some user modelling

**LLG**: selection is the supervision

**NPL**: definition is the supervision
  - possible to build up complex actions/concepts
The two extremes

**LLG**: start from scratch, understands nothing, anything goes

**NPL**: start with a programming language and its power

**LLG**: each user has a private language

**NPL**: user community has one shared language
  - with some user modelling

**LLG**: selection is the supervision

**NPL**: definition is the supervision
  - possible to build up complex actions/concepts

**LLG**: features, learning from denotations do the heavy lifting
  - guess any action, language agnostic

**NPL**: grammar induction do the heavy lifting
  - no parse unless well-supported
Calendar (with Nadav Lidor)

http://nlp.stanford.edu/blog/interactive-language-learning/
We use the same logical language

- **delete Thursday’s events**
  
  (:foreach (start date (date 2015 11 12)) (: remove))

- **change my 3pm meeting to be 30 minutes after my 10:15am meeting**
  
  (:s foreach (start_time (time 15 00)) (: move start_datetime (call addtime ((reverse end_datetime) (start_time (time 10 15))) (number 30 minutes))))

- **rename next meeting "Boring Family Dinner"**
  
  (:foreach (call pick_first start_datetime (call after start_datetime (call now))) (: update title (string "boring family dinner")))
Better communication with computers

#1. Alexa  
(Amazon Echo)  

#2. Cortana  
(Windows 10 Phone) 

#3. Siri  
(iPhone)  

#4. Google Now  
(Android)
Extremes of the solution space

- LLG: we can build a system that learn from scratch quickly through interaction
- NPL: a community of untrained users can use definitions to naturalize a PL
Learn from users interactively

Wittgenstein: language derives its meaning through use
Montague: language derives its meaning through definition?
Code, experiments, demo of LLG: shrdlurn.sidaw.xyz

Hmm, wait for us to release the NPL stuff