Program Synthesis with Pragmatic Communication

Yewen Pu, Kevin Ellis, Marta Kryven, Josh Tenenbaum, Armando Solar-Lezama

abstract

Program synthesis techniques construct or infer programs from user-provided specifications, such as input-output examples. Yet most specifications, especially those given by end-users, leave the synthesis problem radically ill-posed, because many programs may simultaneously satisfy the specification. Prior work resolves this ambiguity by using various inductive biases, such as a preference for simpler programs. This work introduces a new inductive bias derived by modeling the program synthesis task as rational communication, drawing insights from recursive reasoning models of pragmatics. Given a specification, we score a candidate program both on its consistency with the specification, and also whether a rational speaker would choose this particular specification to communicate that program. We develop efficient algorithms for such an approach when learning from input-output examples, and build a pragmatic program synthesizer over a simple grid-like layout domain. A user study finds that end-user participants communicate more effectively with the pragmatic program synthesizer over a non-pragmatic one.

play first!

https://evanthebouncy.github.io/projects/grids/

pragmatics (rsa)

operationalize RSA pragmatics

literary listener L0:

\[ P_{L0}(w|u) \propto \delta(u, w)P(w) \]

pragmatic speaker S1:

\[ P_{S1}(u|w) \propto P_{L0}(w|u) \]

pragmatic listener L1:

\[ P_{L1}(w|u) \propto P_{S1}(u|w) \]

can pragmatics help synthesis?

program synthesis

A <- (cuboid 16 0 4 28 16 20)  
B <- (cylinder 4 4 12 4 4 16 4)  
C <- (+ A B)  
M <- (+ K L)

3D voxel spec

CAD program

4-4-1981 -> '84/4-1981'  
22-9-2025 -> '92/2/2025'  
'13-6-1990' -> '6/13/1990'

input-output spec

program synthesis

Replace1()  
Replace2()  
GetToken1Number()  
GetProm(-)

string-editing program

relying synthesis to pragmatics

hypothesis / programs

utterances / examples

meaning matrix

pragmatic set of examples

Incremental Pragmatically Speaker S1. We now build a pragmatic speaker S1 recursively from L0. Here, rather than treating D as an unordered set, we view it as an ordered sequence of examples, and models the speaker’s generation of D incrementally, similar to autoregressive sequence generation in language modeling [26]. Let \( D = u^1 \ldots u^d \), then:

\[ P_{S1}(D|h) = P_{S1}(u_1, \ldots, u_k | h) = P_{S1}(u_1|h)P_{S1}(u_2|h, u_1) \ldots P(u_k|h, u_1 \ldots u_{k-1}) \]

where the incremental probability \( P_{S1}(u_1|h, u_1, \ldots, u_{i-1}) \) is defined recursively with L0:

\[ P_{S1}(u_1|h, u_1, \ldots, u_{i-1}) \propto P_{L0}(u_1|h, u_1, \ldots, u_{i-1}), \]

\[ P_{S1}(u_1|h, u_1, \ldots, u_{i-1}) = \sum_{u_i} P_{L0}(u_1, \ldots, u_i) \]

DSL

P -> if (x,y) in box(8,8,8,8)  
then symbol(S,C)  
else pebble  
B -> 0 | 1 | 2 | 3 | 4 | 5 | 6  
S -> ring(0,1,0,0,3)  
O -> chicken | pig  
I -> chicken | pig | pebble  
R -> 1 | 2 | 3  
G -> [red, green, blue][A2(A1)]  
A1 -> x | y | x+y  
A2 -> lambda z:0 | lambda z:1 | lambda z:2 | lambda z:z*2+1 | lambda z:2*(z%2)

results

Figure 5: (a) the density of the mean number of symbols used (N=48). (b) the mean number of
symbols used by subjects during the course of the experiment (error bars show 95% confidence
intervals), communicating with the both robots improvement over time. (c) which robot was easier to
communicate with.