Conditional Program Generation for Bimodal Program Synthesis

Swarat Chaudhuri

Rice University

www.cs.rice.edu/~swarat

(Joint work with Chris Jermaine, Vijay Murali, and Letao Qi)

CAPER

RICE Unconventional Wisdom

Program synthesis

[Simon 1963, Summers 1977, Manna-Waldinger 1977, Pnueli-Rosner 1989]

Specification

↓

Synthesizer

↓

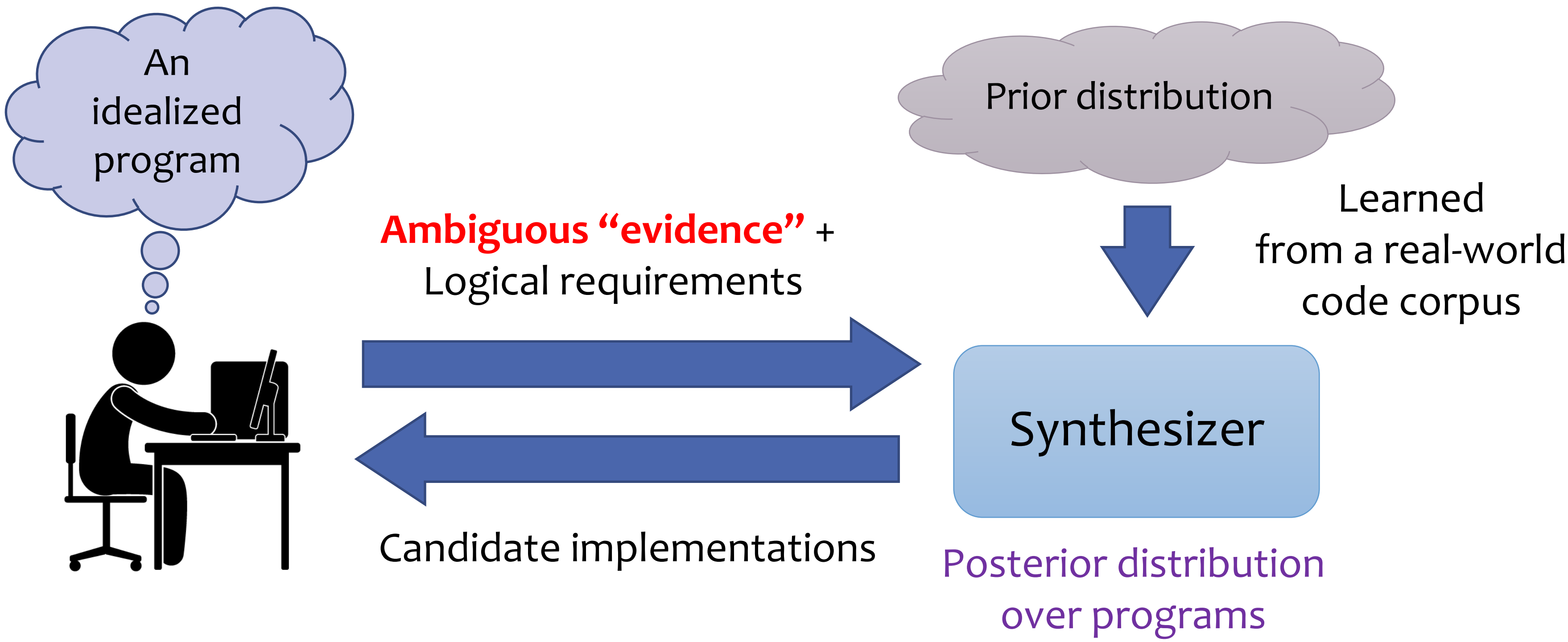
Program
+
Correctness Certificate

Specification: Logical constraint that must be satisfied exactly

Algorithm: Search for a program that satisfies the specification.

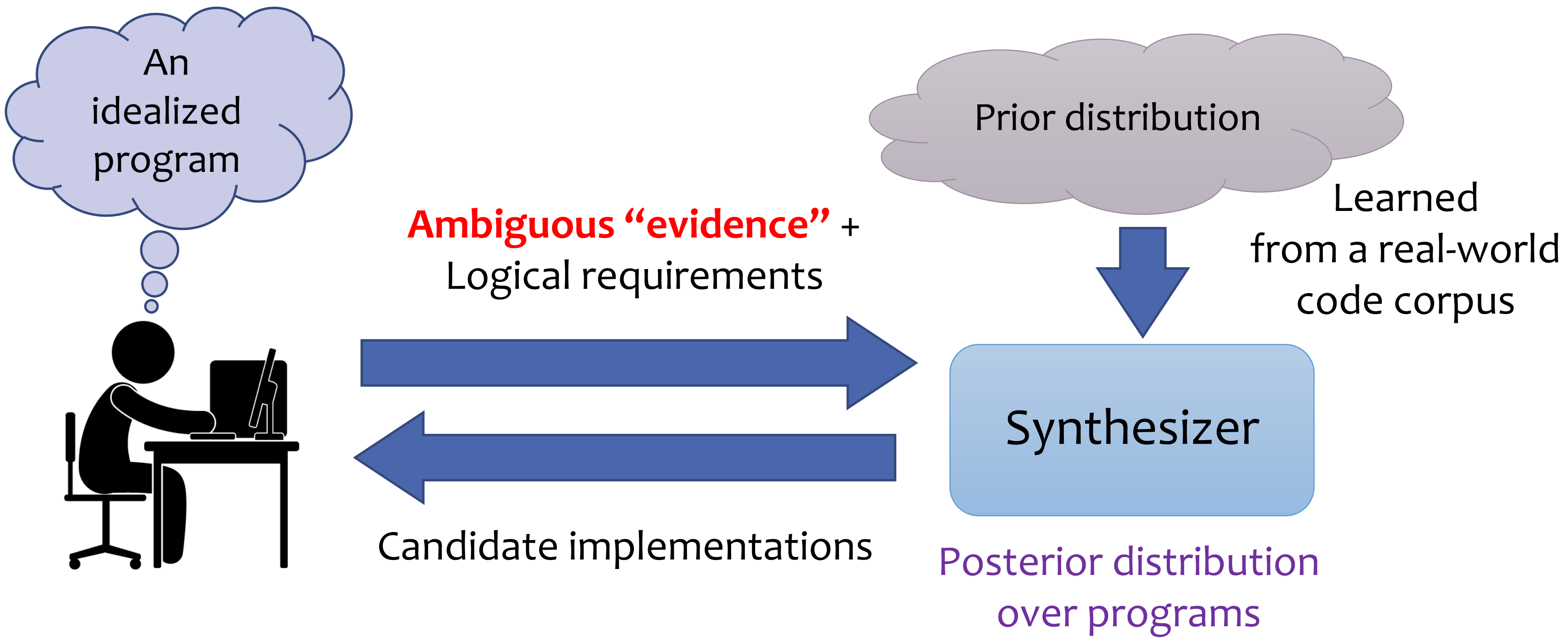
CAPER

“Bimodal” program synthesis



Neural Sketch Learning for Conditional Program Generation. Murali, Qi, Chaudhuri, and Jermaine. Arxiv 2017.

"Bimodal" program synthesis



- API calls or types that the program uses
- "Soft" I/O examples or constraints
- Natural language description of what the program does
- …

The Bayou synthesizer: A demo

http://bit.ly/2zgP5fj

Conditional program generation

Assume random variables X and $Prog$, over **labels** and **programs** respectively, following a joint distribution $Q(X, Prog)$.

Offline:

- You are given a set $\{(X_i, Prog_i)\}$ of samples from $Q(X, Prog)$. From this, learn a function g that maps evidence to programs.
- Learning goal: maximize $E_{(X,Prog)\sim Q}[I]$, where

$$I = \begin{cases} 1 & \text{if } g(X) \equiv Prog \\ 0 & \text{otherwise.} \end{cases}$$

Online: Given X, produce $g(X)$.

CAPER

In what we actually do

The map g is probabilistic.

Learning is maximum conditional likelihood estimation:

- Given $\{(X_i, Prog_i)\}$, solve $\arg\max_\theta \sum_i \log P(Prog_i | X_i, \theta)$.

Programs

Language capturing the essence of API usage in Java.

$$
\begin{array}{lcl}
\mathsf{Prog} & ::= & \mathbf{skip} \mid \mathsf{Prog}_1 ; \mathsf{Prog}_2 \mid \mathbf{call}\ \mathsf{Call} \mid \\
 & & \mathbf{let}\ x = \mathsf{Call} \mid \\
 & & \mathbf{if}\ \mathsf{Exp}\ \mathbf{then}\ \mathsf{Prog}_1\ \mathbf{else}\ \mathsf{Prog}_2 \mid \\
 & & \mathbf{while}\ \mathsf{Exp}\ \mathbf{do}\ \mathsf{Prog}_1 \mid \mathbf{try}\ \mathsf{Prog}_1\ \mathsf{Catch} \\
\mathsf{Exp} & ::= & \mathsf{Sexp} \mid \mathsf{Call} \mid \mathbf{let}\ x = \mathsf{Call} : \mathsf{Exp}_1 \\
\mathsf{Sexp} & ::= & c \mid x \\
\mathsf{Call} & ::= & \mathsf{Sexp}_0.a(\mathsf{Sexp}_1, \ldots, \mathsf{Sexp}_k) \\
\mathsf{Catch} & ::= & \mathbf{catch}(x_1)\ \mathsf{Prog}_1\ \ldots\ \mathbf{catch}(x_k)\ \mathsf{Prog}_k
\end{array}
$$

API call

API method name

CAPER

Labels

Set of API calls

- readline, write,…

Set of API datatypes

- BufferedReader, FileReader,…

Set of keywords that may appear while describing program actions in English

- read, file, write,…
- Obtained from API calls and datatypes through a camel case split

CAPER

Challenges

Directly learning over source code simply doesn't work

- Source code is full of low-level, program-specific names and operations.
- Programs need to satisfy structural and semantic constraints such as type safety. Learning to satisfy these constraints is hard.

CAPER

Language abstractions to the rescue!

Learn not over programs, but typed, syntactic models of programs.

Sketches

The sketch of a program is obtained by applying an **abstraction function** α.

From sketch Y to program $Prog$: a fixed **concretization distribution** $P(Prog \mid Y)$.

Learning goal changes to

- Given $\{(X_i, Y_i)\}$, solve $\arg\max_{\theta} \sum_i \log P(Y_i | X_i, \theta)$.

CAPER

Sketches

Sketch

$$
\begin{array}{lcl}
\mathsf{Y} & ::= & \mathsf{Call} \mid \textbf{skip} \mid \textbf{while}\ \mathsf{Cond}\ \textbf{do}\ \mathsf{Y}_1 \mid \mathsf{Y}_1 ; \mathsf{Y}_2 \mid \\
 & & \textbf{try}\ \mathsf{Y}_1\ \mathsf{Catch} \mid \textbf{if}\ \mathsf{Cond}\ \textbf{then}\ \mathsf{Y}_1\ \textbf{else}\ \mathsf{Y}_2 \\
\mathsf{Catch} & ::= & \textbf{catch}(\tau_1)\ \mathsf{Y}_1\ \ldots\ \textbf{catch}(\tau_k)\ \mathsf{Y}_k \\
\mathsf{Cond} & ::= & \{\mathsf{Call}_1, \ldots, \mathsf{Call}_k\} \\
\mathsf{Call} & ::= & a(\tau_1, \ldots, \tau_k)
\end{array}
$$

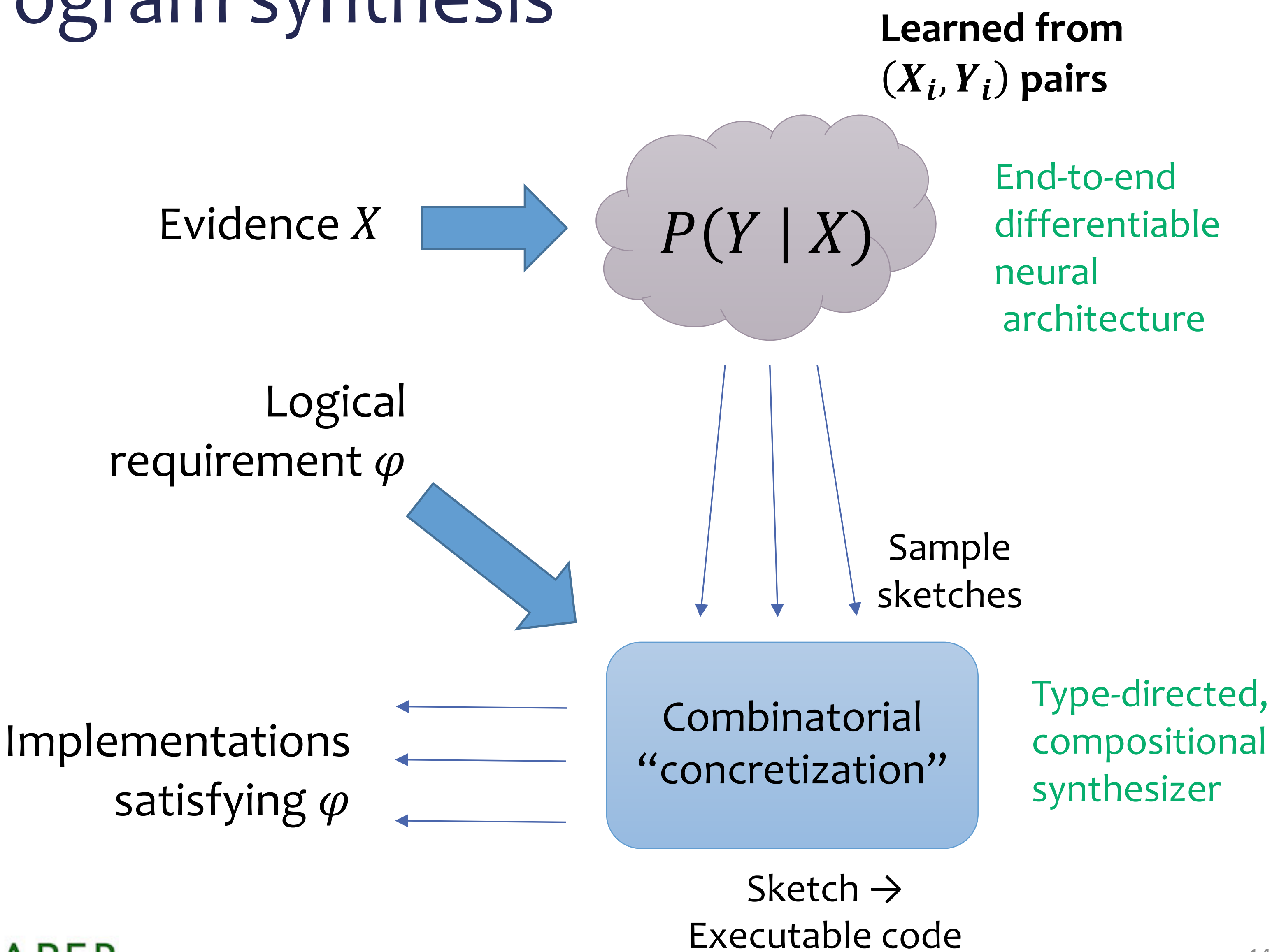
Conditions replaced by sets of abstract API calls

Abstract API call

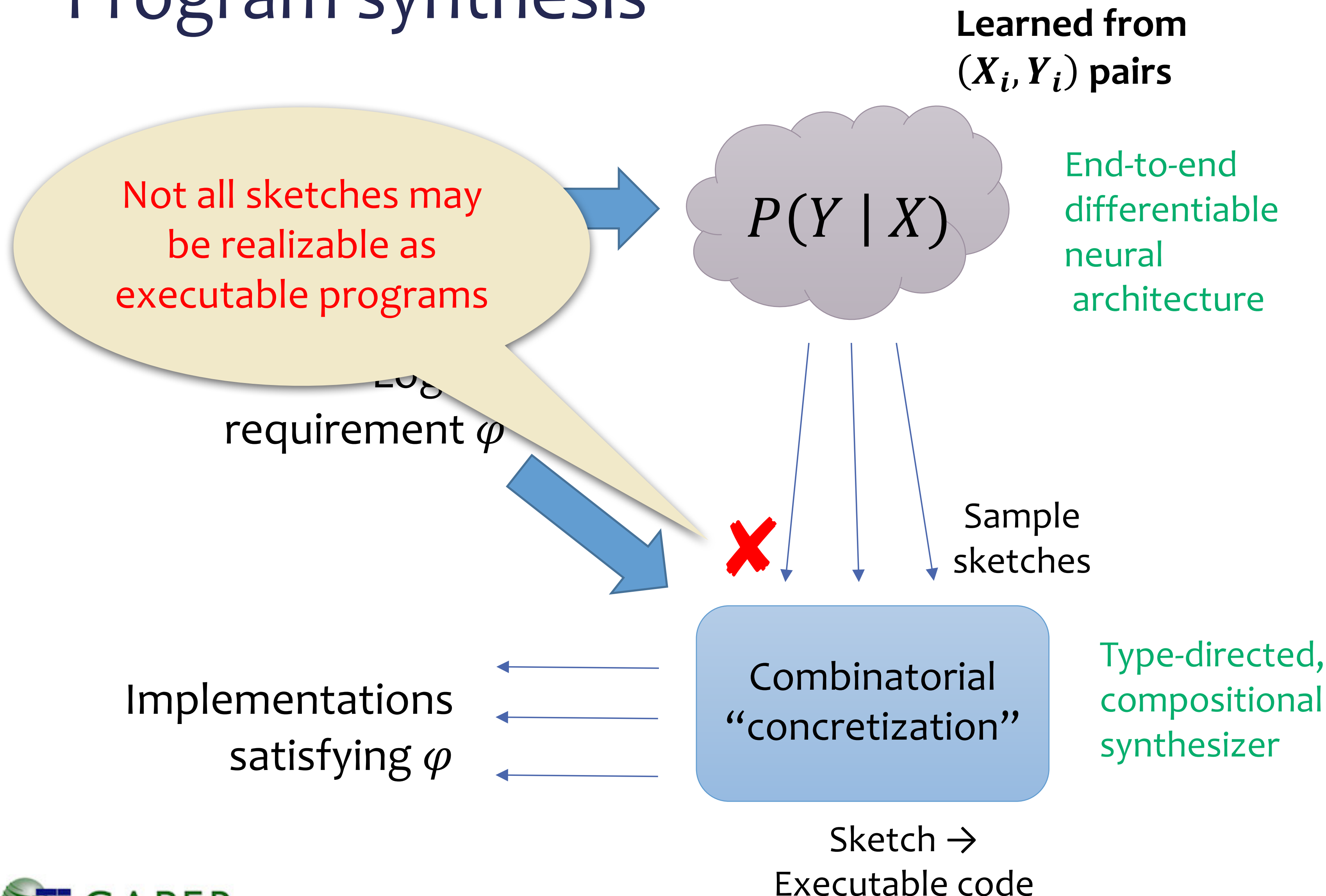
API method name

Types

Program synthesis



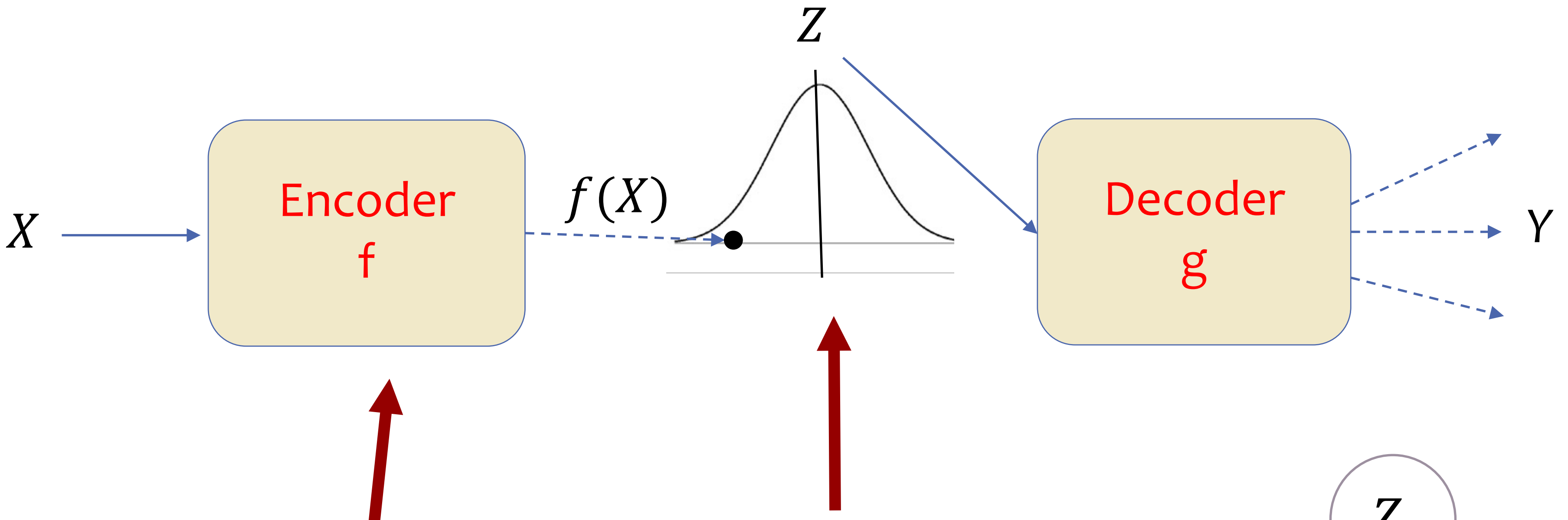
Program synthesis



CAPER

Learning using a probabilistic encoder-decoder

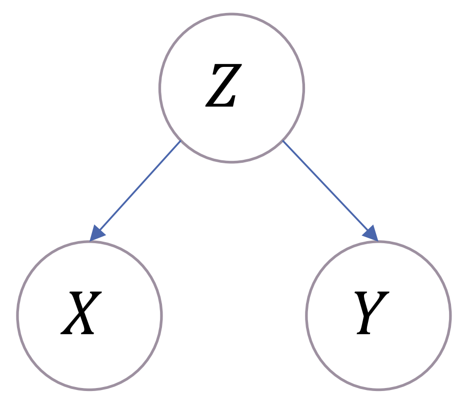
X: Evidence
Y: Sketches
Z: Latent "intent"



Feedforward neural net

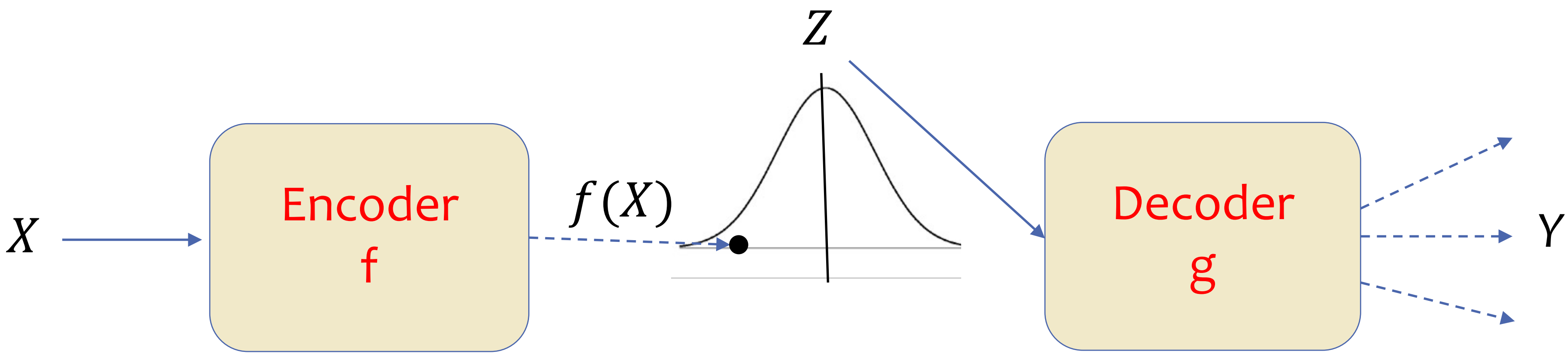
Representation of hidden intent

Prior for regularization



Learning using a probabilistic encoder-decoder

X: Evidence
Y: Sketches
Z: Latent "intent"



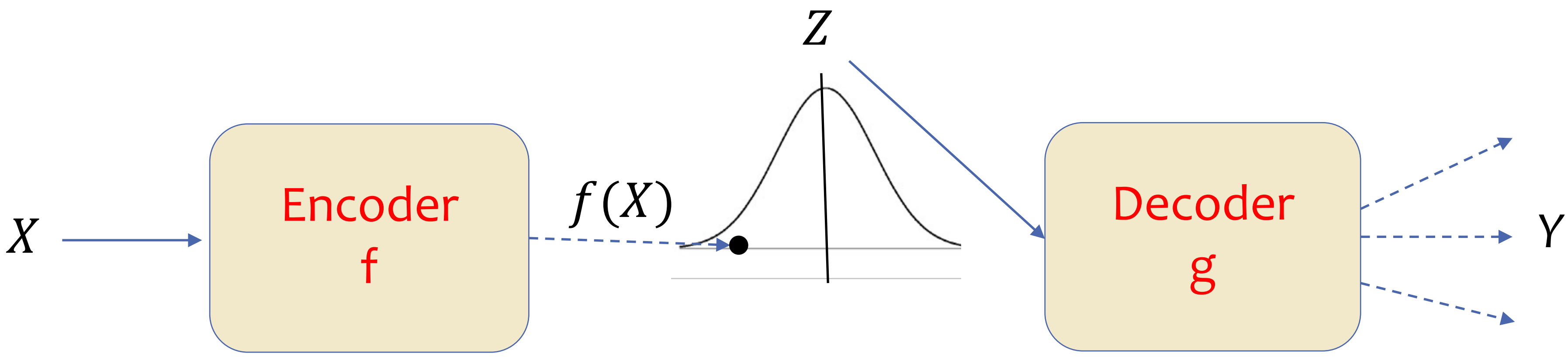
$$P(Z) = \ Normal(0, I)$$

$$P(f(X) \mid Z) = \ Normal(Z, \sigma^2 I)$$

During learning, use Jensen's inequality to get smooth loss function

Learning using a probabilistic encoder-decoder

X: Evidence
Y: Sketches
Z: Latent “intent”



$$P(Z) = Normal(0, I)$$

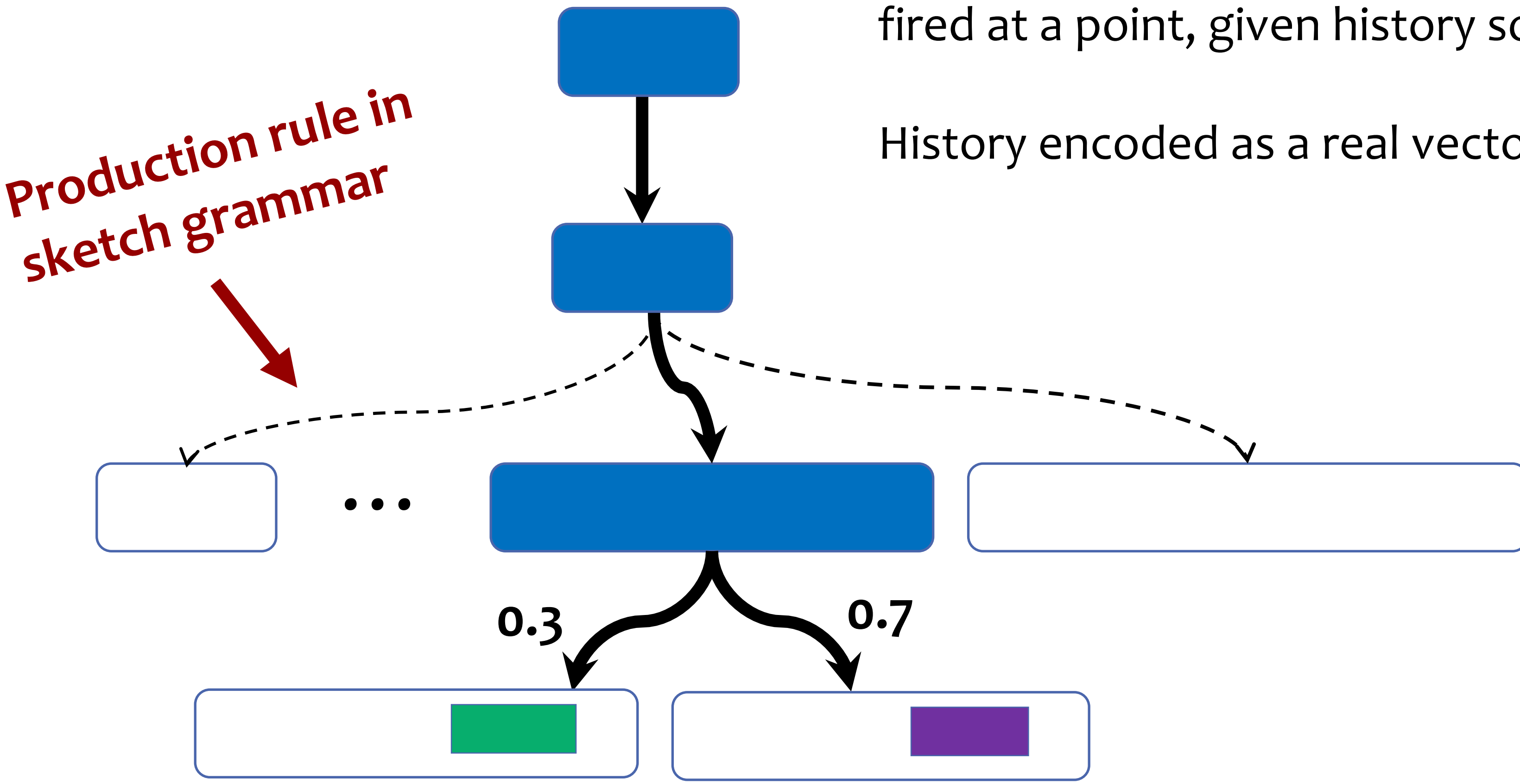
$$P(f(X) \mid Z) = Normal(Z, \sigma^2 I)$$

During inference, get P(Z | X) using normal-normal conjugacy

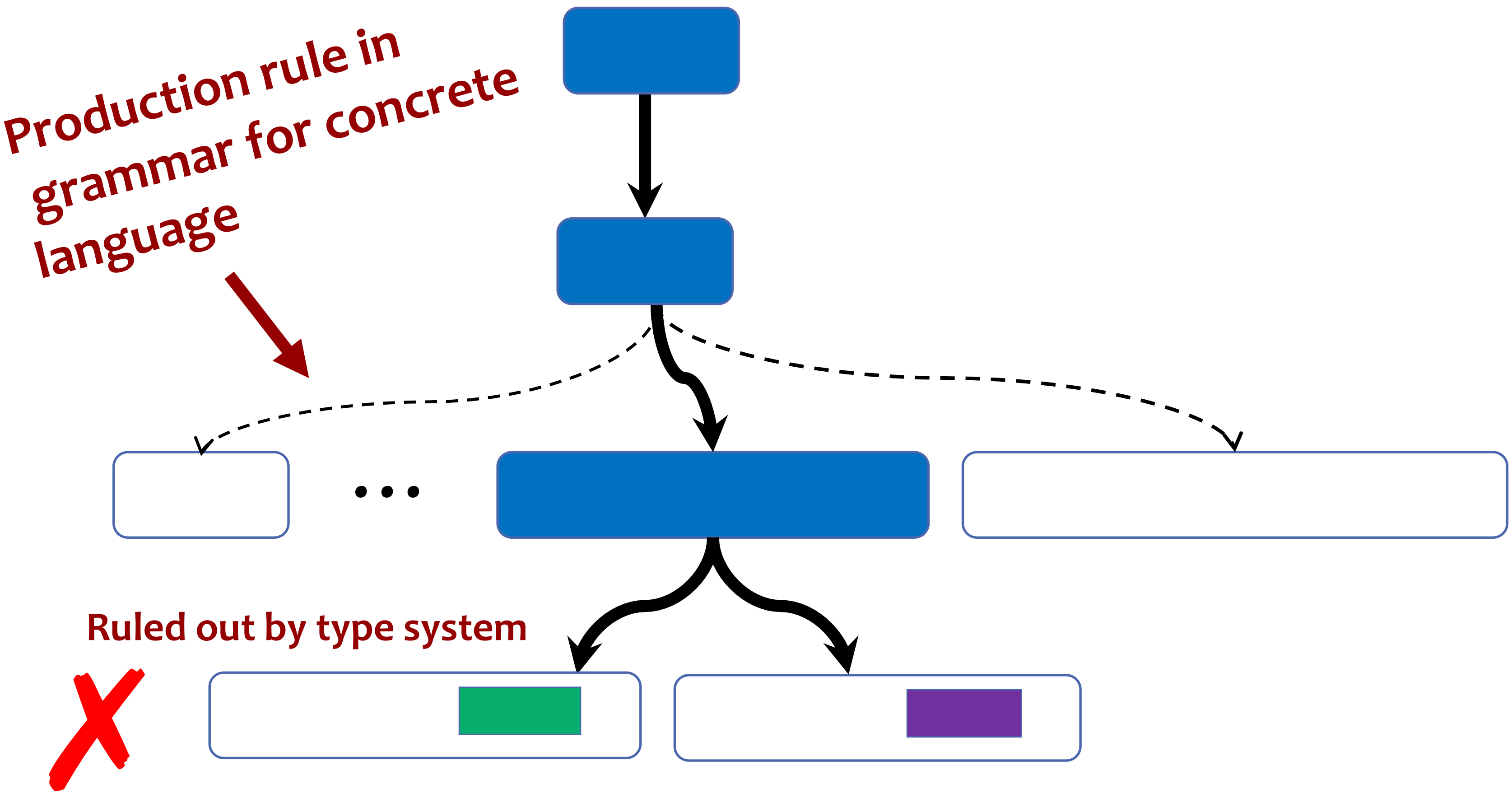
Neural decoder

Distribution on rules that can be fired at a point, given history so far.

History encoded as a real vector.



Concretization



CAPER

Results

- Trained method on 100 million lines of Java/Android code. ~2500 API methods, ~1500 types.
- Synthesis of method bodies from scratch, given 2-3 API calls and types.
- Sketch learning critical to accuracy.
- Good performance compared to GSNNs (state of the art conditional generative model).
- Good results on label-sketch pairs not encountered in training set.

CAPER

Thank you!

Questions?

swarat@rice.edu
http://www.cs.rice.edu/~swarat

(Research funded by the DARPA MUSE award #FA8750-14-2-0270)

CAPER