Lecture 33
Synthesizing under a distribution

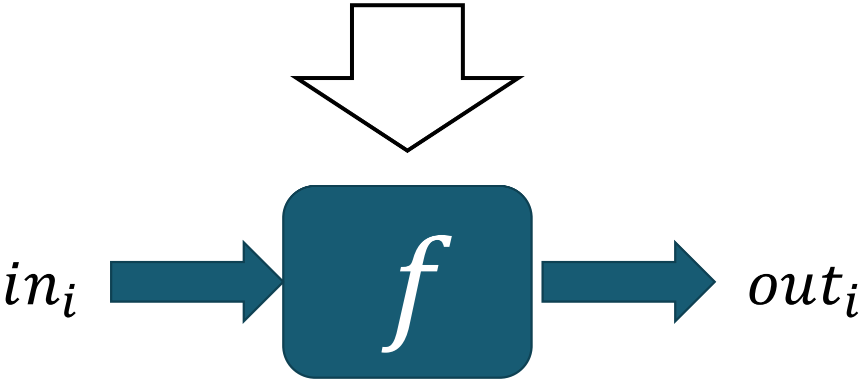
Sankha Narayan Guria

with slides from

Armando Solar-Lezama

Programming by Example

$$[(in_0|out_0),(in_1|out_1),\ldots(in_k|out_k)]$$



Problem is hopelessly underspecified

- Many semantically distinct programs can satisfy the examples

$$P(f\ |[(in_i|out_i)]_i) \approx P_f(f) * P_{io}([(in_i\ /\ out_i)]_i\ |\ f\ )$$

Key elements

Any program that does not match the I/O examples has P=0

All programs that match the I/O examples have the same $P_{io}([(in_i \,/\, out_i)]_i \mid f\ )$

Length minimization

Shortest programs are better than longer programs

- $P(f) = \begin{cases} \frac{1}{Z} * e^{-len(f)} & if\ f\ belongs\ to\ the\ space\ of\ programs \\ 0 & otherwise \end{cases}$

Bottom Up Explicit Search

When discarding observationally equivalent programs, keep shortest one.

- Naturally finds shortest programs first

Key property:

- All subprograms of the shortest program will themselves be the shortest programs

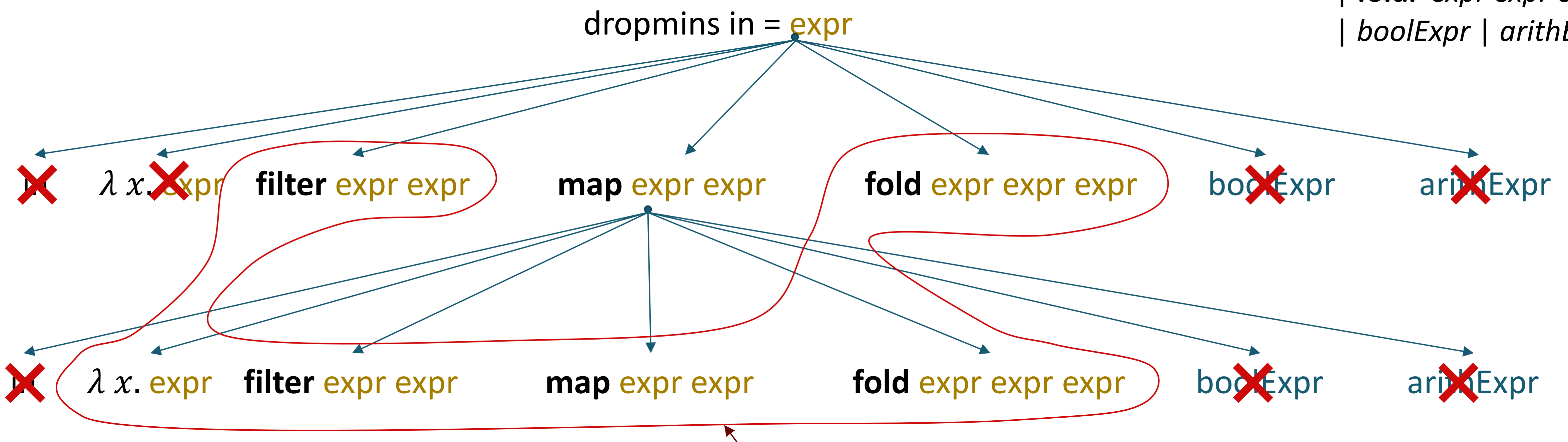
Top Down Explicit Search

Achieving minimality is trickier and more expensive

- Algorithm keeps a wavefront of partially completed programs
- Need to expand that wavefront in best-first manner

Top Down Explicit Search

expr = var
| $\lambda x.\, expr$
| **filter** *expr expr*
| **map** *expr expr*
| **foldl** *expr expr expr*
| *boolExpr* | *arithExpr*



Wavefront of partially expanded programs that have not been rejected.
At each step there is a choice of which program to try to expand.

Symbolic Search

$$\exists\phi.\,\forall in.\,Q(\phi, in)$$

$$\cancel{\exists\phi.\,\left(\forall in.\,Q(\phi, in) \land \forall\phi'\; P(\phi') \leq P(\phi)\right)}$$

$$\exists\phi.\,\left(\forall in.\,Q(\phi, in) \land \forall\phi'\;\left(\forall in.\,Q(\phi', in)\right) \Rightarrow P(\phi') \leq P(\phi)\right)$$

Symbolic Search

$$\exists\phi.\forall in.\, Q(\phi, in)$$

$$\Downarrow$$

$$\exists\phi.\forall in \in E.\, Q(\phi, in)$$

More permissive, allows all correct ϕ, but also some incorrect ones.

$$\forall\phi' \ \left(\forall in.\, Q(\phi', in)\right) \Rightarrow P(\phi') \leq P(\phi)$$

$$\Downarrow$$

$$\forall\phi' \in G \ \left(\forall in.\, Q(\phi', in)\right) \Rightarrow P(\phi') \leq P(\phi)$$

$$\Downarrow$$

$$\forall\phi' \in G \ \ P(\phi') \leq P(\phi)$$

Works if we can ensure G only contains ϕ' that satisfy $\forall in.\, Q(\phi', in)$

Symbolic Search

$$\exists\phi.\forall in.\,Q(\phi, in)$$

$$\exists\phi.\forall in \in E.\,Q(\phi, in)$$

More permissive, allows all correct ϕ, but also some incorrect ones.

$$\forall\phi' \;\left(\forall in.\,Q(\phi', in)\right) \Rightarrow P(\phi') \leq P(\phi)$$

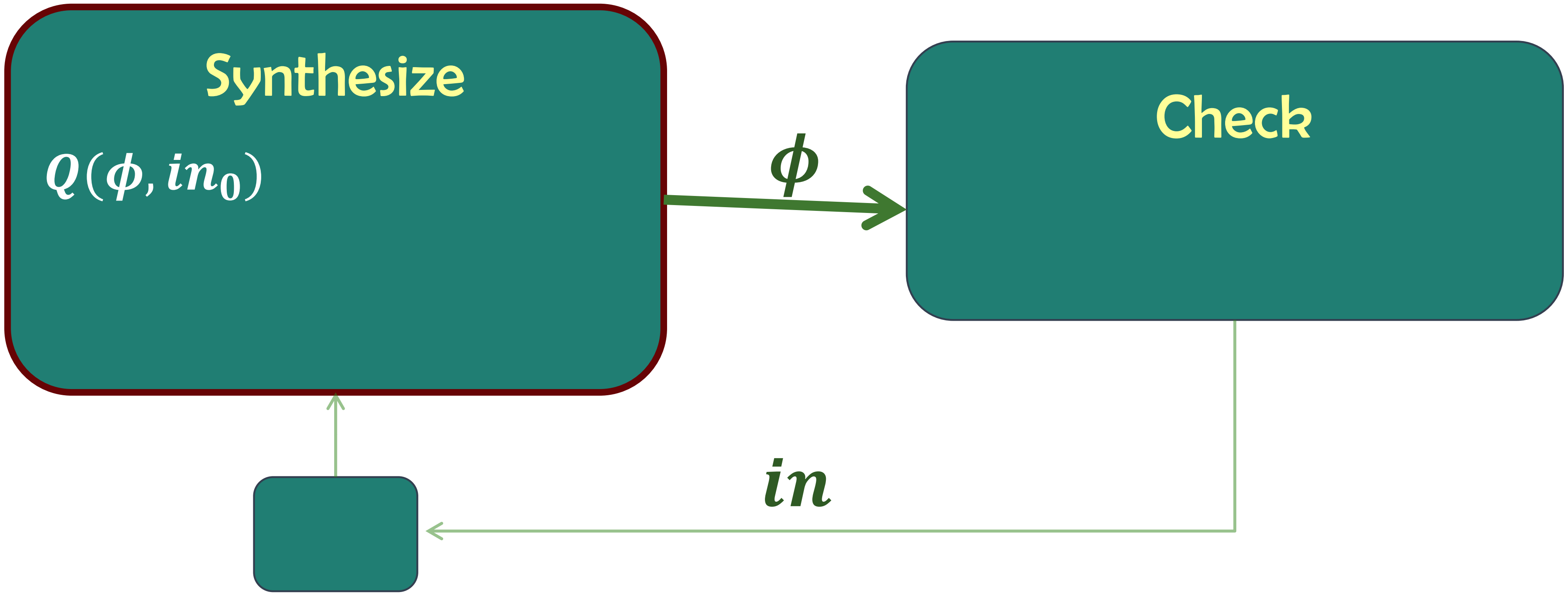
$$\forall\phi' \in G \;\left(\forall in.\,Q(\phi', in)\right) \Rightarrow P(\phi') \leq P(\phi)$$

Works if we can ensure G only contains ϕ' that satisfy $\forall in.\,Q(\phi', in)$

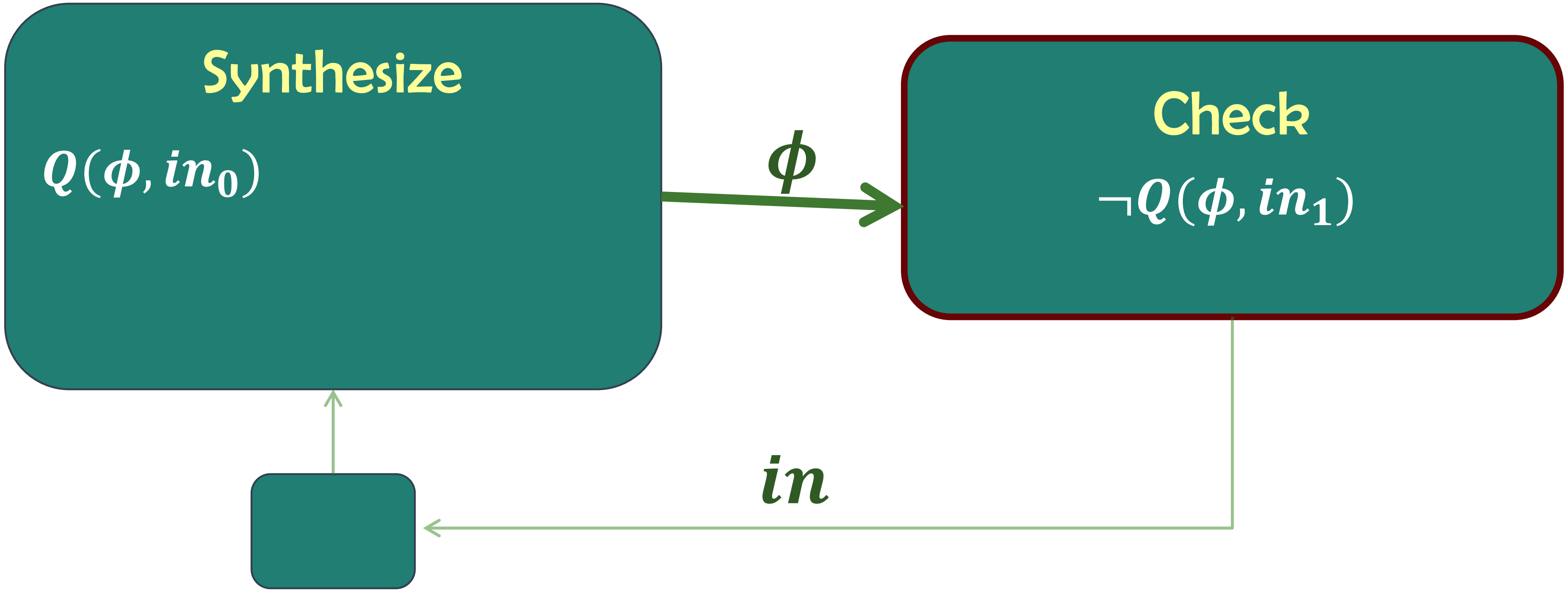
$$\forall\phi' \in G \;\; P(\phi') \leq P(\phi)$$

$$\left(\max_{\phi' \in G} P(\phi')\right) \leq P(\phi)$$

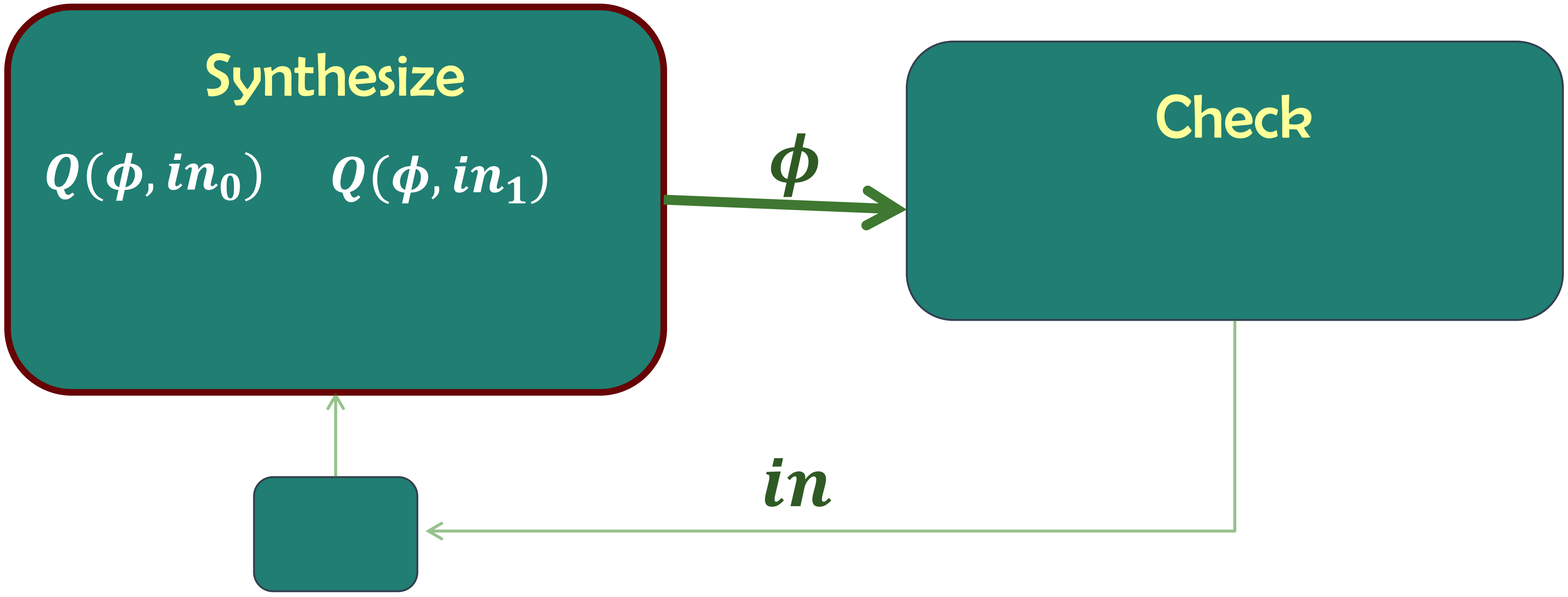
Extended CEGIS



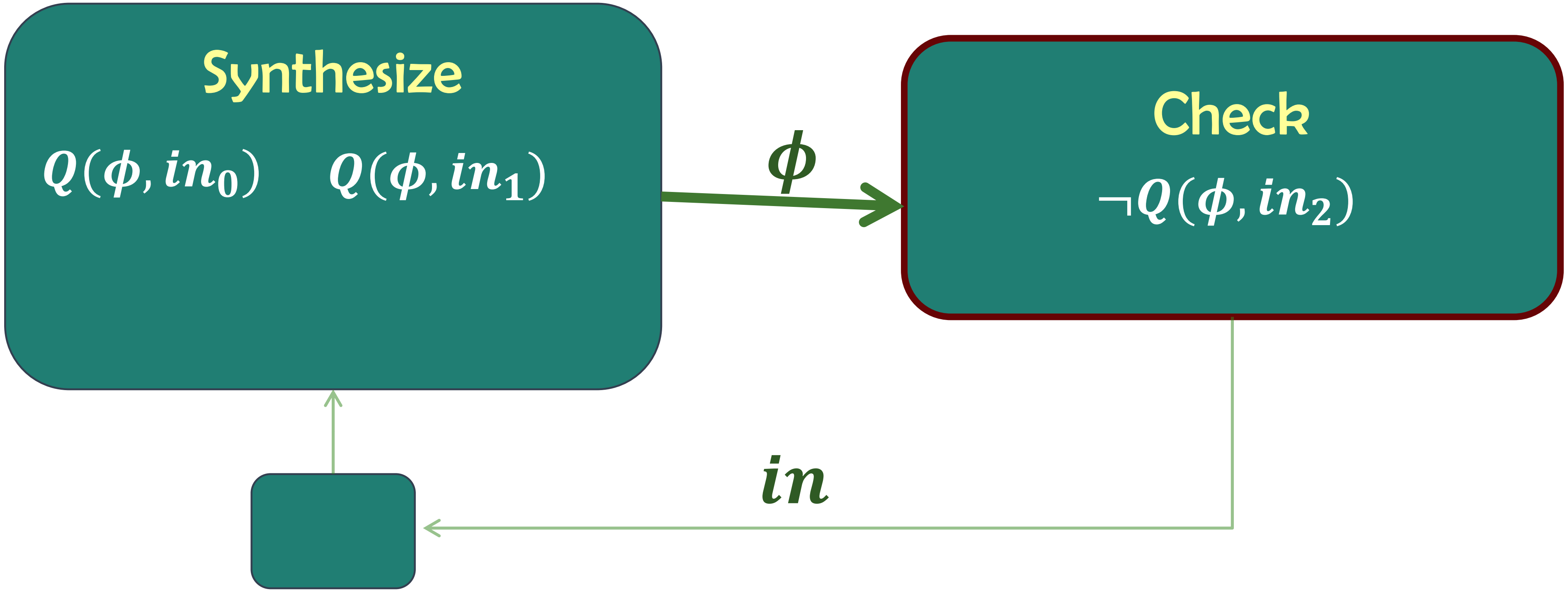
Extended CEGIS



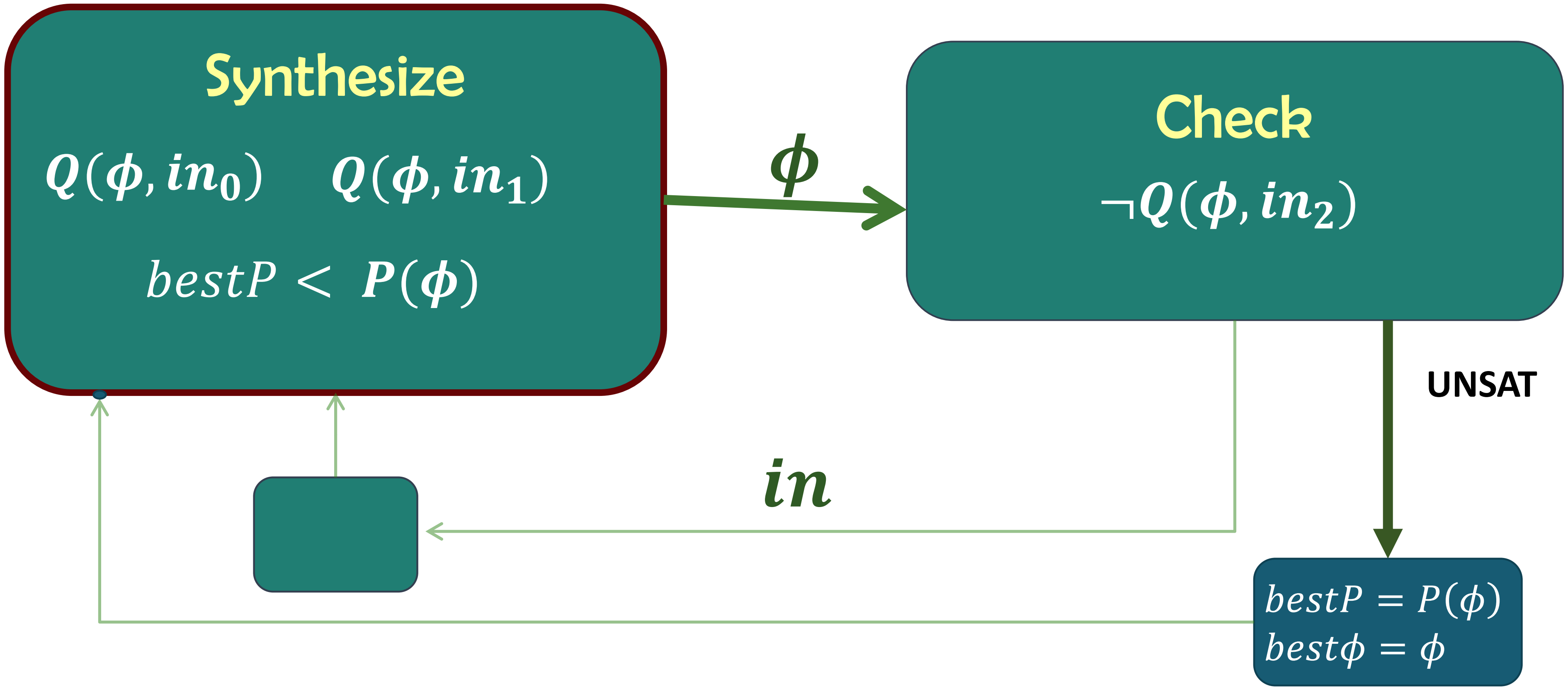
Extended CEGIS



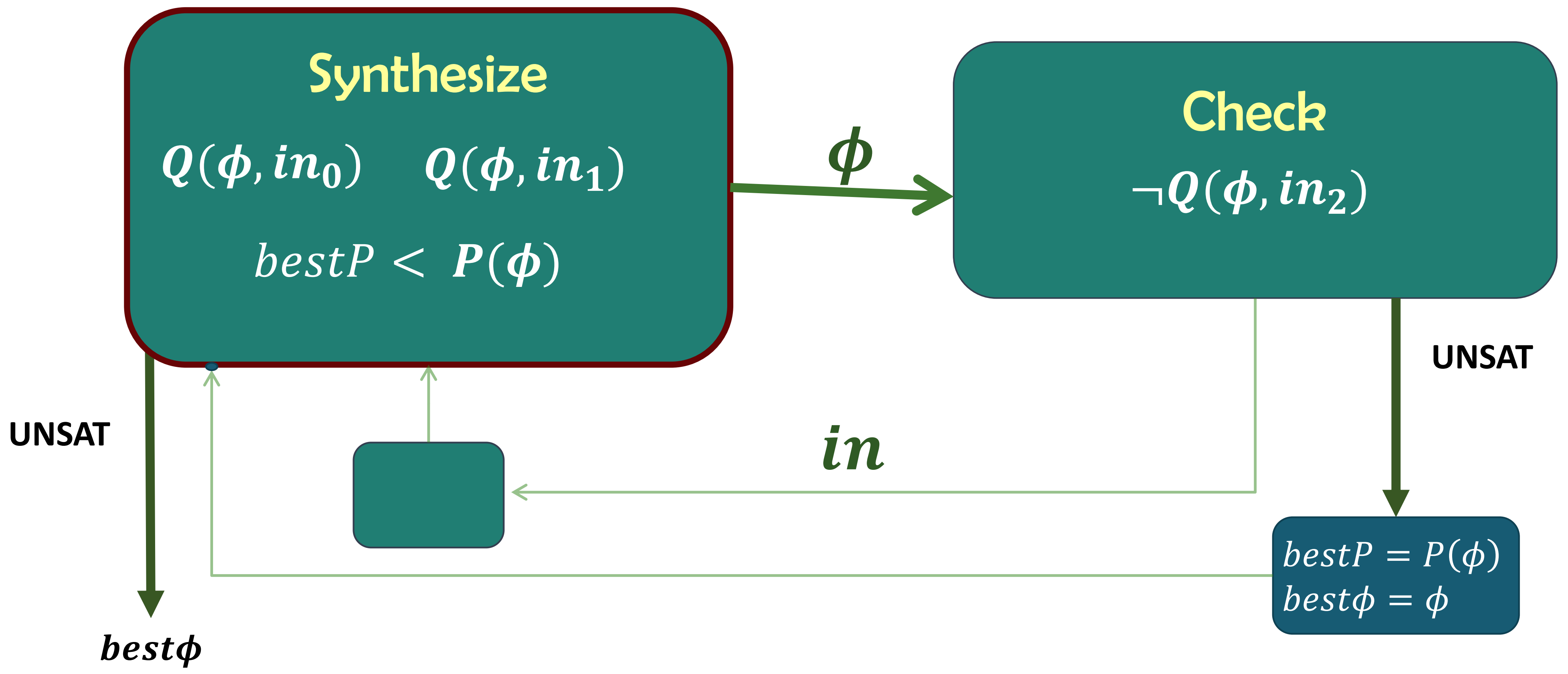
Extended CEGIS



Extended CEGIS



Extended CEGIS



Probabilistic Grammars

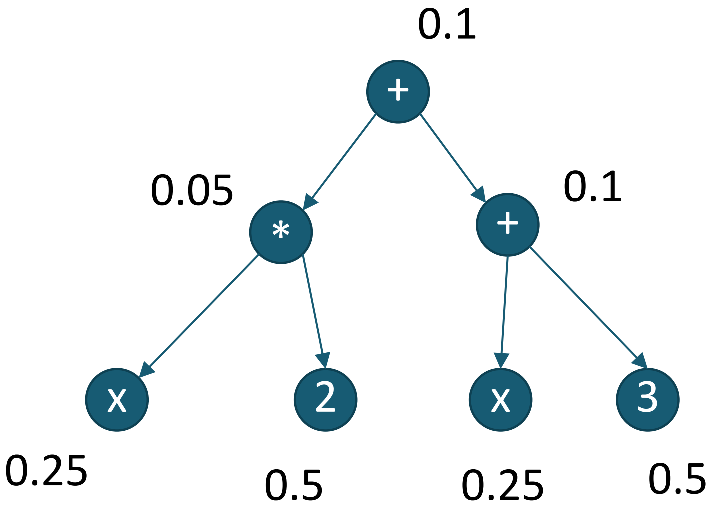
$$
\begin{aligned}
\text{expr} = \;& \text{expr} + \text{expr} \\
\mid\; & \text{expr} - \text{expr} \\
\mid\; & \text{expr} * \text{expr} \\
\mid\; & var \\
\mid\; & num
\end{aligned}
$$

Probabilistic Grammars

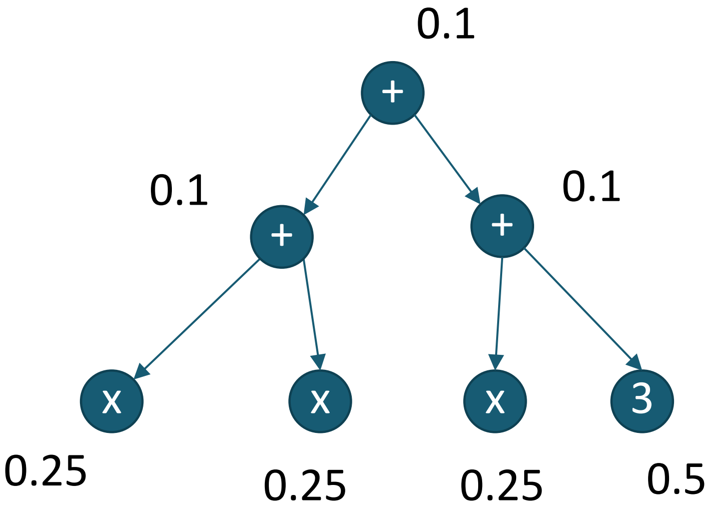
	P(expr)
expr = expr + expr	0.1
| expr – expr	0.1
| expr * expr	0.05
| var	0.25
| num	0.5

Probabilistic Grammars

		P(expr)
expr =	expr + expr	0.1
	| expr – expr	0.1
	| expr * expr	0.05
	| var	0.25
	| num	0.5



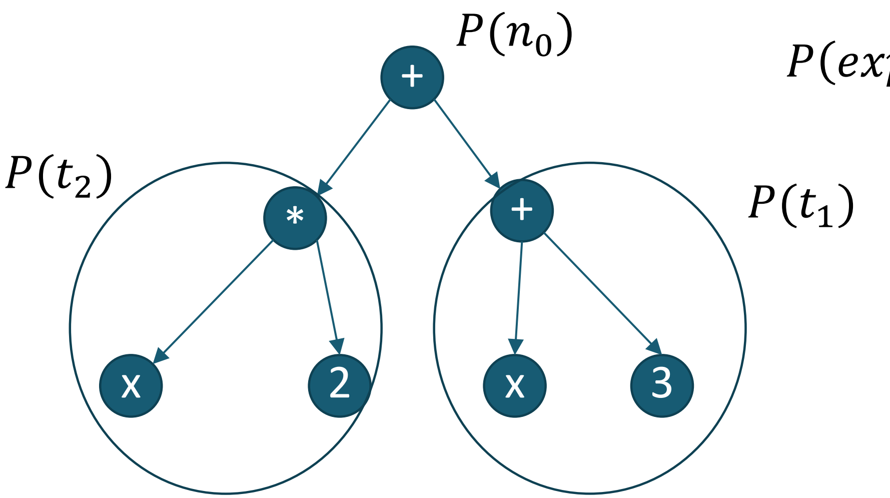
P(exp) = 7.81×10^{-6}



P(exp) = 7.81×10^{-6}

Search with P G

Probability is compositional



$$P(expr) = P(n_0) * P(t_1) * P(t_2)$$

Same strategies as length-based metric work

Context sensitive Probabilities

$$
\begin{aligned}
expr = \; & expr + expr \\
| \; & expr - expr \\
| \; & expr * expr \\
| \; & var \\
| \; & num
\end{aligned}
$$

Context sensitive Probabilities

$$
\begin{aligned}
expr = \ & expr_1 + expr_2 && \text{A} \\
| \ & expr_3 - expr_4 && \text{B} \\
| \ & expr_5 * expr_6 && \text{C} \\
| \ & var && \text{D} \\
| \ & num && \text{E}
\end{aligned}
$$

	A	B	C	D	E
Root	0.2	0.2	0.2	0.2	0.2
1	0.2	0.2	0.2	0.2	0.2
2	0	0	1/3	1/3	1/3
3	0.2	0.2	0.2	0.2	0.2
4	0	0	1/3	1/3	1/3
5	0.2	0.2	0.2	0.2	0.2
6	0.25	0.25	0	0.25	0.25

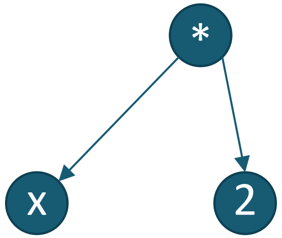
Search

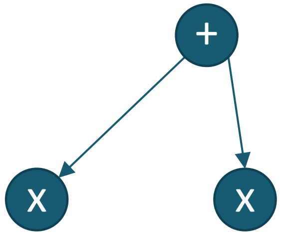
Top-down and Constraint-based search are similar

Bottom up search is significantly more challenging

Bottom-up search

x=5 => out=7





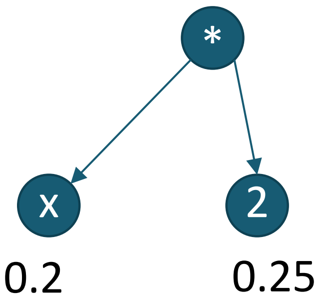
We know they are equivalent
Which one do we keep?

Bottom-up search

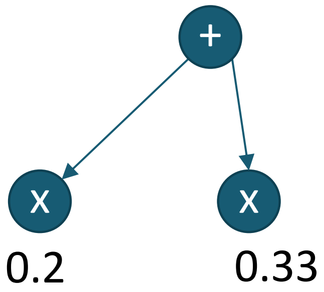
$expr =$	$expr_1 + expr_2$	A
	$\mid expr_3 - expr_4$	B
	$\mid expr_5 * expr_6$	C
	$\mid var$	D
	$\mid num$	E

	A	B	C	D	E
Root	0.2	0.2	0.2	0.2	0.2
1	0.2	0.2	0.2	0.2	0.2
2	0	0	1/3	1/3	1/3
3	0.2	0.2	0.2	0.2	0.2
4	0	0	1/3	1/3	1/3
5	0.2	0.2	0.2	0.2	0.2
6	0.25	0.25	0	0.25	0.25

x=5 => out=7



Root	0.01
1	0.01
2	**0.017**
3	0.01
4	**0.017**
5	0.01
6	0



Root	**0.013**
1	**0.013**
2	0
3	**0.013**
4	0
5	**0.013**
6	**0.0165**

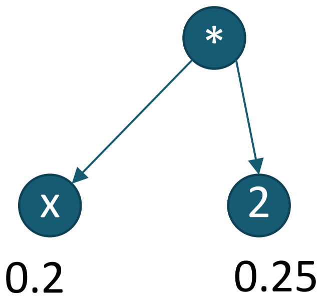
Is this exponentially slower?

Bottom-up search

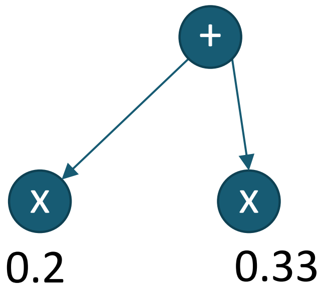
$$
\begin{aligned}
expr = \ & expr_1 + expr_2 && \text{A} \\
| \ & expr_3 - expr_4 && \text{B} \\
| \ & expr_5 * expr_6 && \text{C} \\
| \ & var && \text{D} \\
| \ & num && \text{E}
\end{aligned}
$$

	A	B	C	D	E
Root	0.2	0.2	0.2	0.2	0.2
1	0.2	0.2	0.2	0.2	0.2
2	0	0	1/3	1/3	1/3
3	0.2	0.2	0.2	0.2	0.2
4	0	0	1/3	1/3	1/3
5	0.2	0.2	0.2	0.2	0.2
6	0.25	0.25	0	0.25	0.25

x=5 => out=7



Root	0.01
1	0.01
2	**0.017**
3	0.01
4	**0.017**
5	0.01
6	0



Root	**0.013**
1	**0.013**
2	0
3	**0.013**
4	0
5	**0.013**
6	**0.0165**

Is this exponentially slower?

No!

Given K contexts, we need at most k representatives for each equivalence class

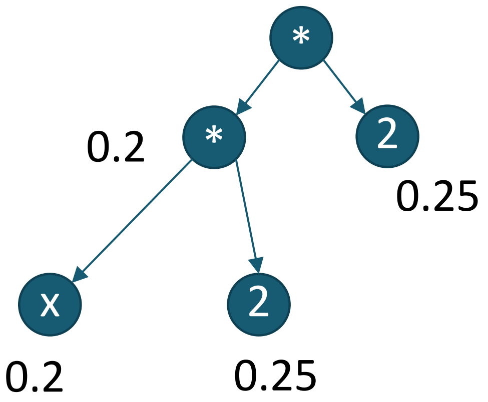
Bottom-up search

$$
\begin{aligned}
expr = \; & expr_1 + expr_2 && \text{A} \\
\mid \; & expr_3 - expr_4 && \text{B} \\
\mid \; & expr_5 * expr_6 && \text{C} \\
\mid \; & var && \text{D} \\
\mid \; & num && \text{E}
\end{aligned}
$$

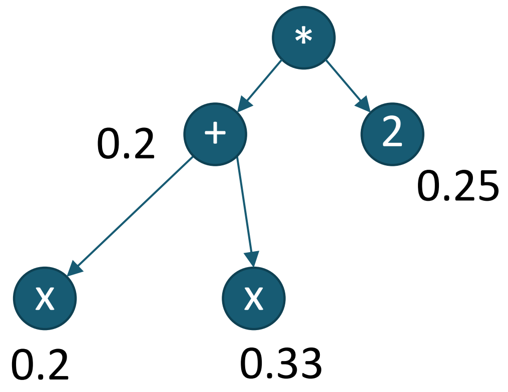
	A	B	C	D	E
Root	0.2	0.2	0.2	0.2	0.2
1	0.2	0.2	0.2	0.2	0.2
2	0	0	1/3	1/3	1/3
3	0.2	0.2	0.2	0.2	0.2
4	0	0	1/3	1/3	1/3
5	0.2	0.2	0.2	0.2	0.2
6	0.25	0.25	0	0.25	0.25

x=5 => out=7

Keeping x*2 and x+x does not force us to keep all programs generated from both



Root	0.0005
1	0.0005
2	0.0008
3	0.0005
4	0.0008
5	0.0005
6	0



Root	**0.0006**
1	**0.0006**
2	**0.001**
3	**0.0006**
4	**0.001**
5	**0.0006**
6	**0**

Is this exponentially slower?

No!

Given K contexts, we need at most k representatives for each equivalence class

Learning a pCFG

$$P_{tree}(\, t \mid p_{nodes})$$

Probability of a tree/string given the probabilities of individual tokens/chars

Usually a simple function that traverses the tree/string and multiplies probabilities (adds their logs)

Learning a pCFG

$$
\begin{array}{rll}
expr = & expr_1 + expr_2 & \text{A} \\
| & expr_3 - expr_4 & \text{B} \\
| & expr_5 * expr_6 & \text{C} \\
| & var & \text{D} \\
| & num & \text{E}
\end{array}
$$

$$p_{nodes}^{\theta} =$$

	A	B	C	D	E
Root	$\theta_{0,A}$	$\theta_{0,B}$	$\theta_{0,C}$	$\theta_{0,D}$	$\theta_{0,E}$
1	$\theta_{1,A}$	$\theta_{1,B}$	$\theta_{1,C}$	$\theta_{1,D}$	$\theta_{1,E}$
2	$\theta_{2,A}$	$\theta_{2,B}$	$\theta_{2,C}$	$\theta_{2,D}$	$\theta_{2,E}$
3	$\theta_{3,A}$	$\theta_{3,B}$	$\theta_{3,C}$	$\theta_{3,D}$	$\theta_{3,E}$
4	$\theta_{4,A}$	$\theta_{4,B}$	$\theta_{4,C}$	$\theta_{4,D}$	$\theta_{4,E}$
5	$\theta_{5,A}$	$\theta_{5,B}$	$\theta_{5,C}$	$\theta_{5,D}$	$\theta_{5,E}$
6	$\theta_{6,A}$	$\theta_{6,B}$	$\theta_{6,C}$	$\theta_{6,D}$	$\theta_{6,E}$

Learning a pCFG

Given a set of sample trees $\mathrm{T} = \{\mathrm{t_i}\}_{\mathrm{i<N}}$
and a parametric distribution p_{nodes}^{θ}
the goal is to find parameters that maximize the probability of the samples.

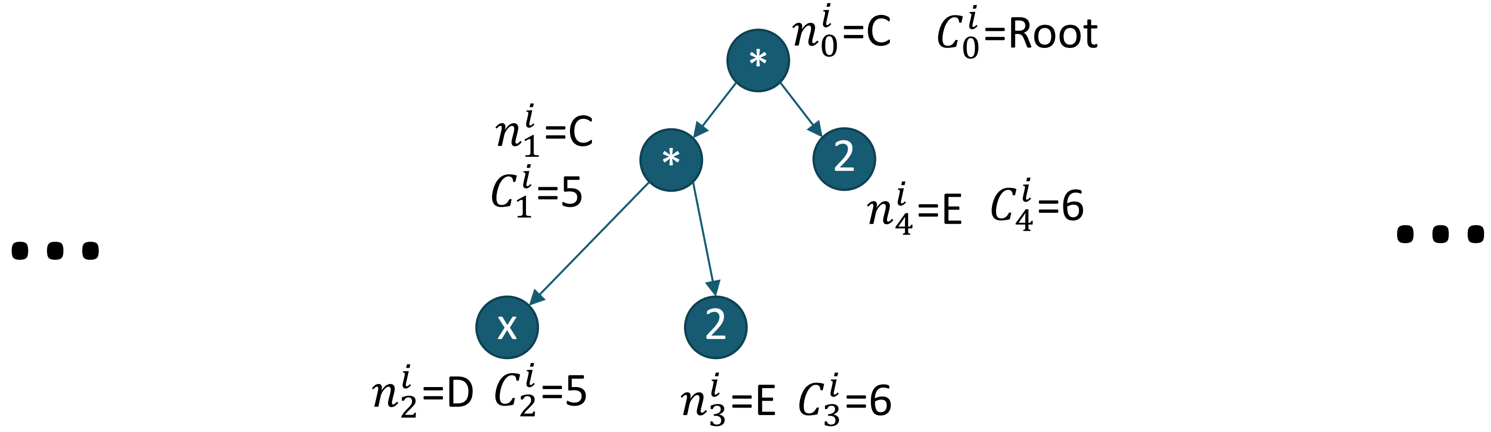
$$\operatorname*{argmax}_{\theta} \Pi_i\ P_{tree}\left(t\ | p_{nodes}^{\theta}\right) = \operatorname*{argmax}_{\theta} \Sigma_i \log P_{tree}\left(t_i\ | p_{nodes}^{\theta}\right)$$

$$= \operatorname*{argmax}_{\theta} \Sigma_i \Sigma_j \log P\left(n_j^i\ | C_j^i, p_{nodes}^{\theta}\right)$$

where n_j^i is node j of tree i in the dataset and C_j^i is the context where node j of tree j appears

Learning a pCFG

Dataset



Tree i

$$p^{\theta}_{nodes} =$$

	A	B	C	D	E
Root	$\theta_{0,A}$	$\theta_{0,B}$	$\theta_{0,C}$	$\theta_{0,D}$	$\theta_{0,E}$
1	$\theta_{1,A}$	$\theta_{1,B}$	$\theta_{1,C}$	$\theta_{1,D}$	$\theta_{1,E}$
2	$\theta_{2,A}$	$\theta_{2,B}$	$\theta_{2,C}$	$\theta_{2,D}$	$\theta_{2,E}$
3	$\theta_{3,A}$	$\theta_{3,B}$	$\theta_{3,C}$	$\theta_{3,D}$	$\theta_{3,E}$
4	$\theta_{4,A}$	$\theta_{4,B}$	$\theta_{4,C}$	$\theta_{4,D}$	$\theta_{4,E}$
5	$\theta_{5,A}$	$\theta_{5,B}$	$\theta_{5,C}$	$\theta_{5,D}$	$\theta_{5,E}$
6	$\theta_{6,A}$	$\theta_{6,B}$	$\theta_{6,C}$	$\theta_{6,D}$	$\theta_{6,E}$

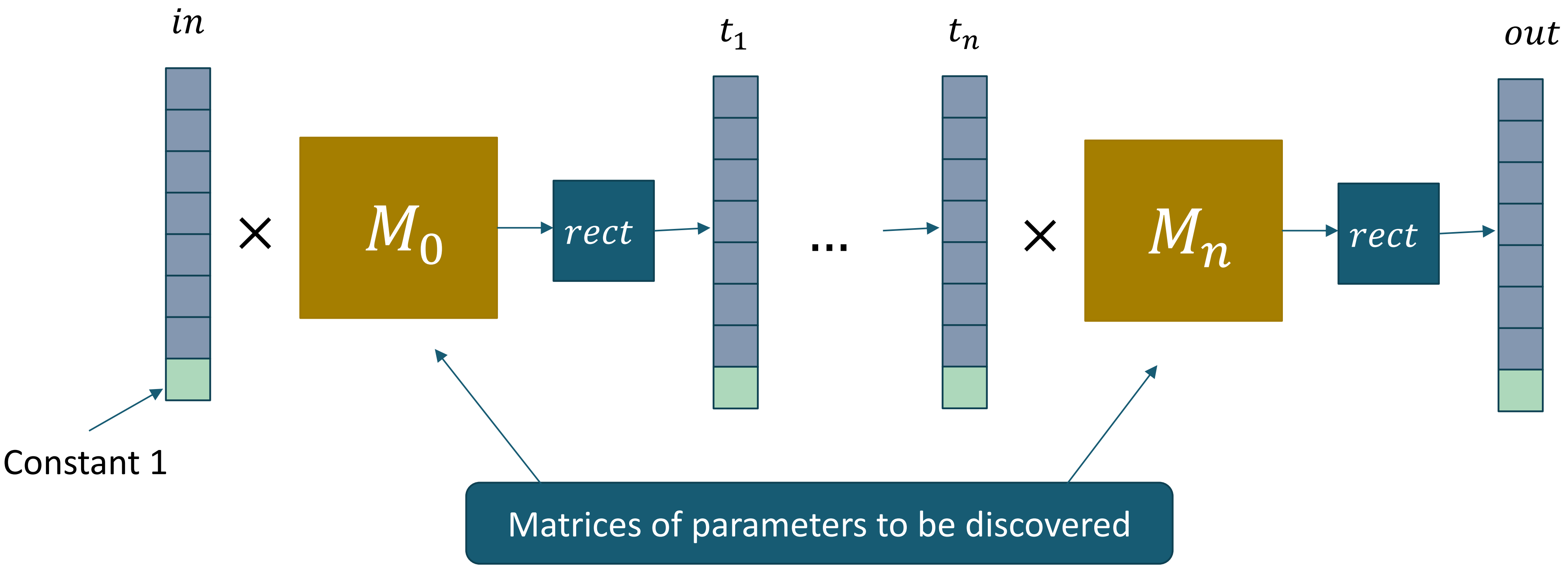
$expr = expr_1 + expr_2$ A
$\mid expr_3 - expr_4$ B
$\mid expr_5 * expr_6$ C
$\mid var$ D
$\mid num$ E

$$\log P_{tree}(t_i \mid p^{\theta}_{nodes}) = \log\theta_{0,C} + \log\theta_{5,C} + \log\theta_{5,D} + \log\theta_{6,E} + \log\theta_{6,E}$$

Neural Networks

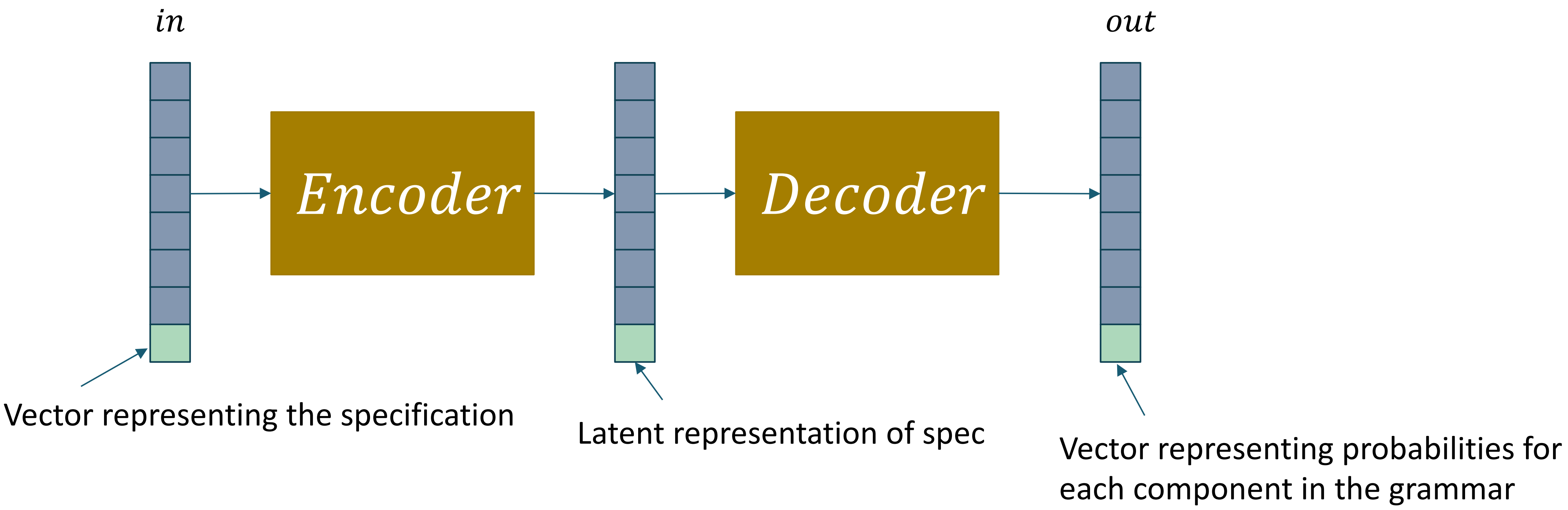
Standard neural network model:

- very general parametric function



DeepCoder

Learn a conditional distribution for a simple probabilistic grammar

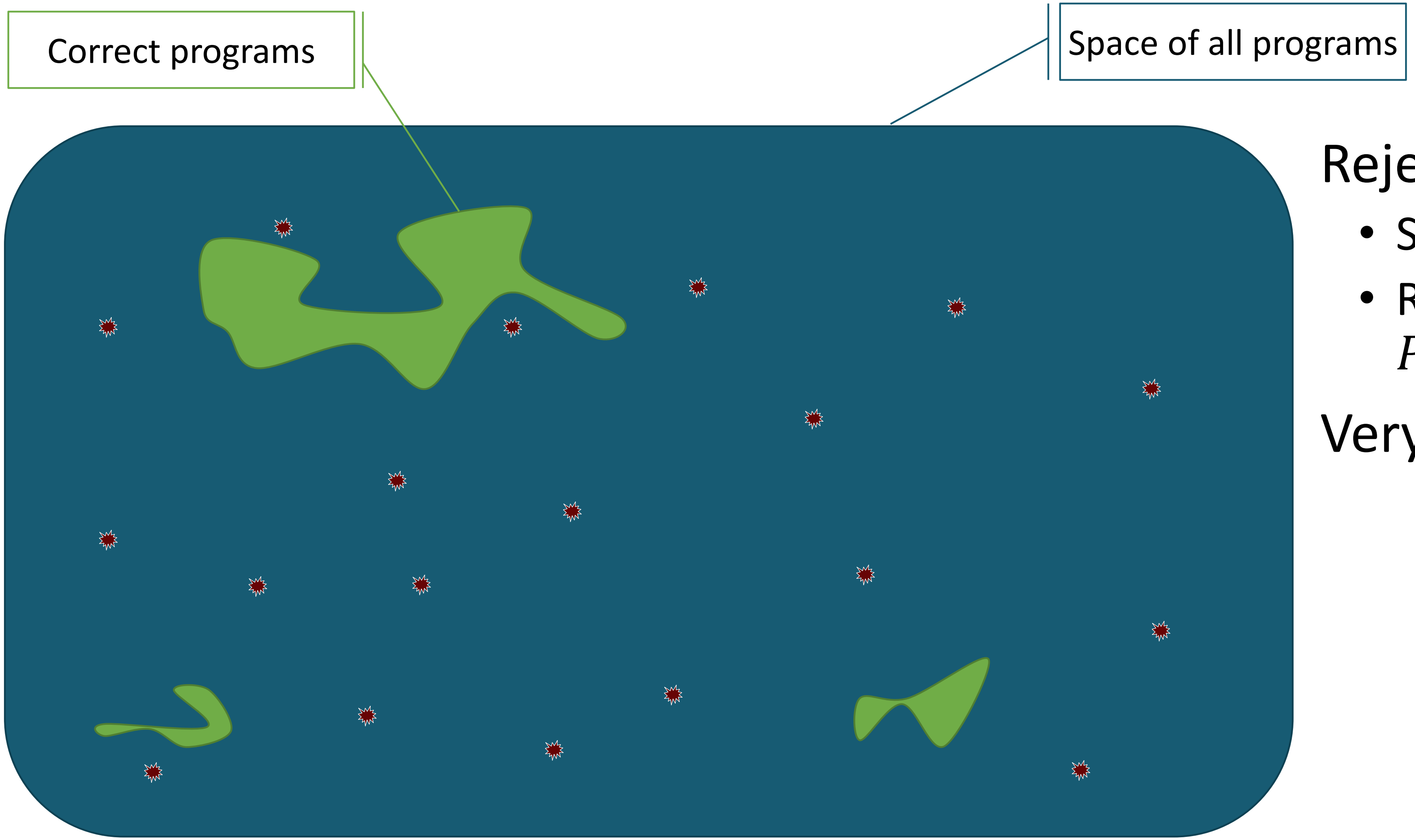


Sampling

What if we do not want the best?

- Instead we want to sample from the distribution

Sampling with Constraints

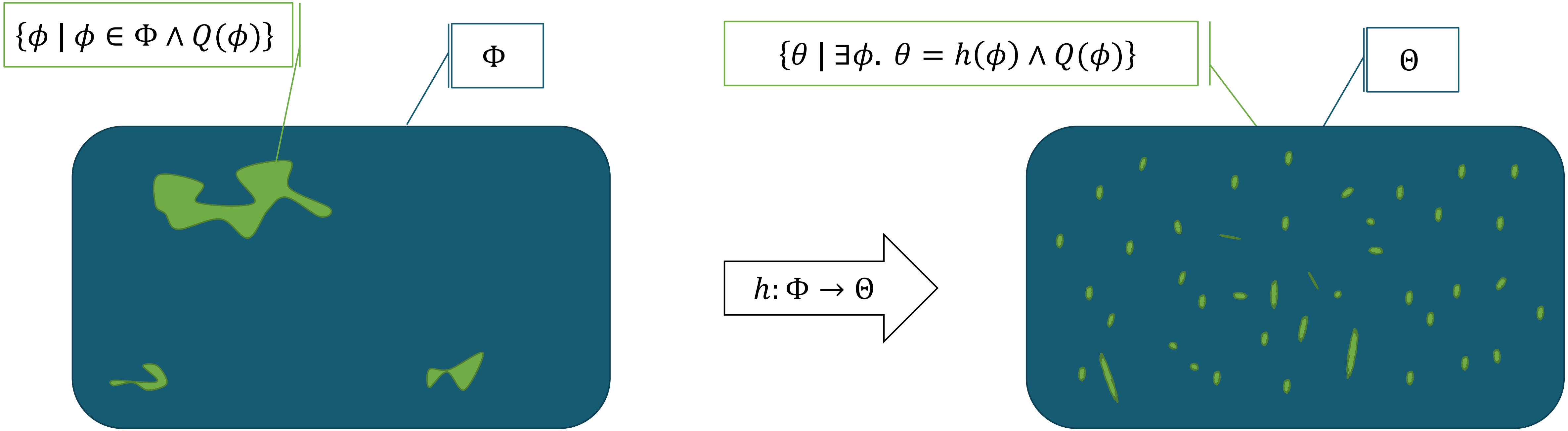


Rejection sampling

- Sample according to $P(f)$
- Reject anything where $P(evidence|f) = 0$

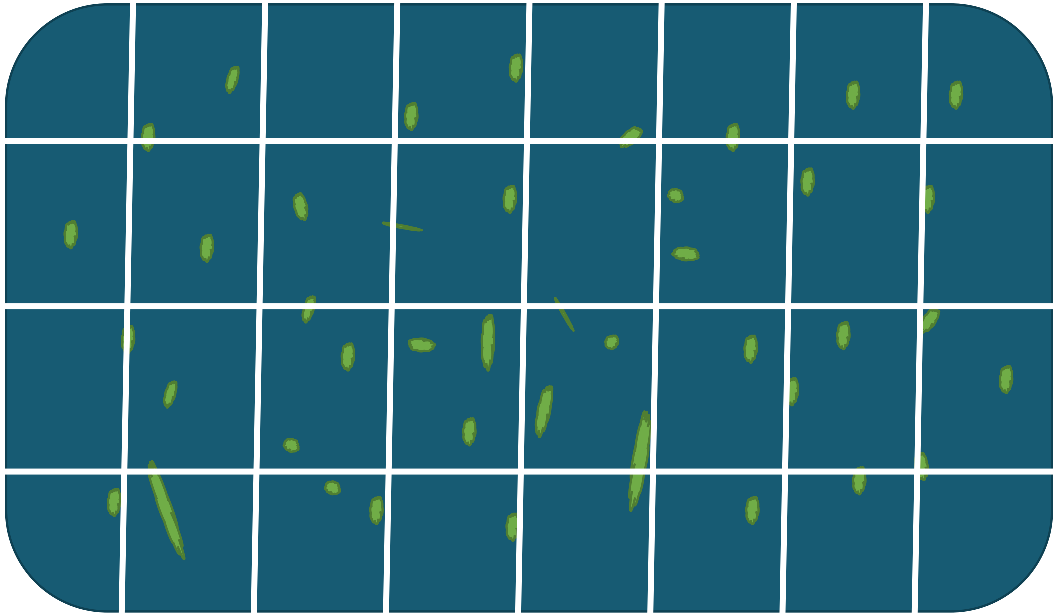
Very inefficient

Sampling with Constraints



If h is measure preserving, sampling uniformly from Φ is equivalent to sampling uniformly from θ and then solving for ϕ s.t. $\theta = h(\phi)$

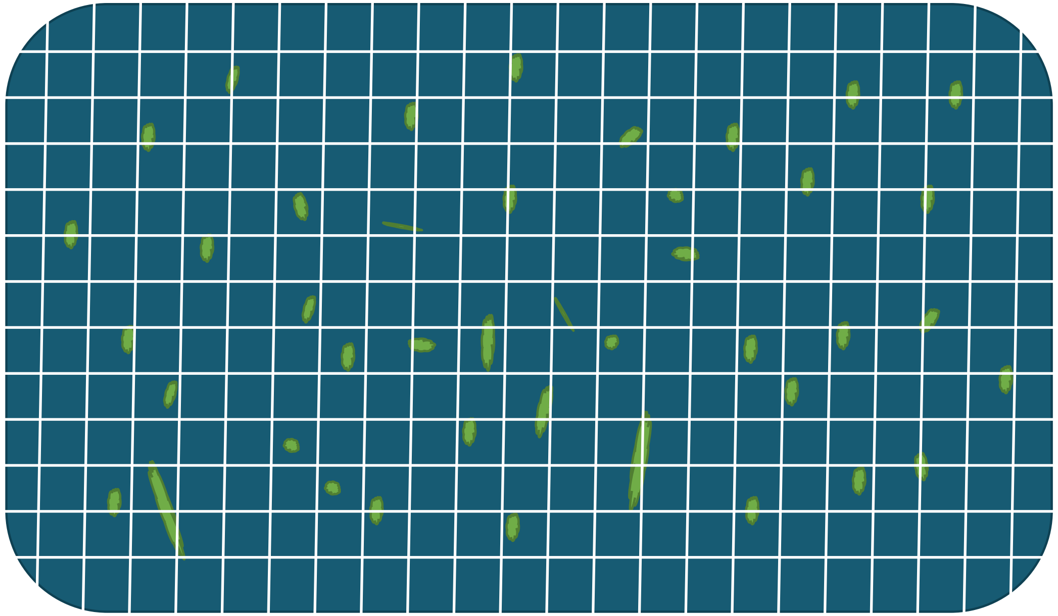
Sampling with Constraints



Approach:

- Partition Θ
- Sample uniformly from the partitions to pick $\overline{\theta}_i \subset \Theta$
- Solve for ϕ s.t.
  $$Q(\phi) \wedge h(\phi) \in \overline{\theta}_i$$

Sampling with Constraints



Approach:

- Partition Θ
- Sample uniformly from the partitions to pick $\overline{\theta_i} \subset \Theta$
- Solve for ϕ s.t.
  $$Q(\phi) \wedge h(\phi) \in \overline{\theta_i}$$

Tradeoff

- Coarse partition efficient but poor quality sampling
- Fine partition expensive but high quality sampling
- Approximates rejection sampling in the limit of very fine partition