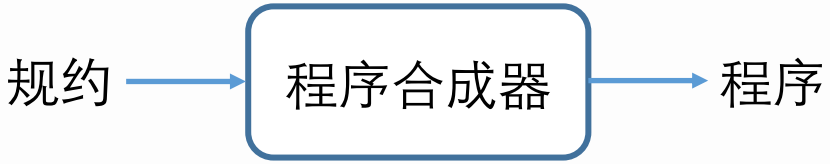
可泛化的基于合一化的程序合成

北京大学　吉如一

程序合成简介

程序合成

- 根据用户给定的规约，自动合成满足规约的程序



$x \leq f(x,y) \wedge y \leq f(x,y) \wedge (f(x,y) = x \vee f(x,y) = y)$

$f(x,y) = \max(x,y)$

- Church's Problem [1957]: 为数学定义的变换合成对应的计算电路
- "程序语言理论中最核心的问题之一" —— Amir Pnueli (1996 年图灵奖得主)
- 挑战：
  - (使用角度) 在实践中如何获取目标程序的规约？
  - (计算角度) 程序空间很大，如何找到满足条件的程序？

样例编程 (Programming by Example)

- 从领域特定语言中寻找满足输入输出样例的程序

$\langle x = 0, y = 1 \rangle \mapsto 1$ 输入输出样例

$S \to x \mid y \mid \max(S, S)$ 领域特定语言

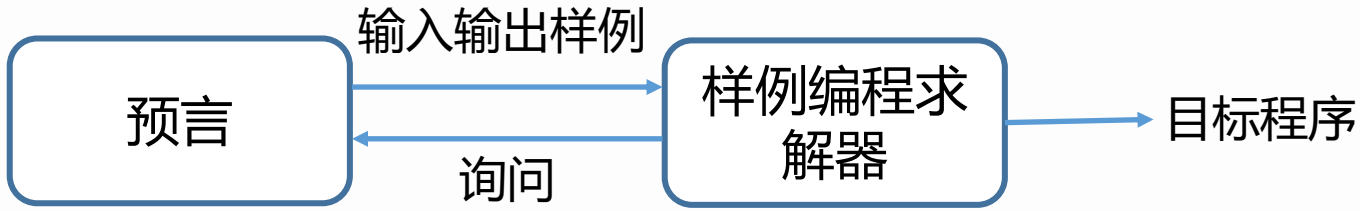
- 经典工作：FlashFill，用于 Excel 自动补全

	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani

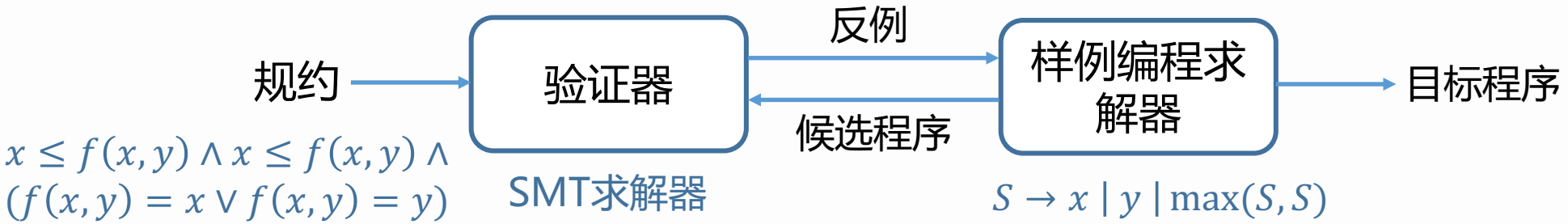
$$
\begin{array}{rcl}
\text{String expr } P & := & \texttt{Switch}((\mathrm{b}_1, \mathrm{e}_1), \cdot\cdot, (\mathrm{b}_n, \mathrm{e}_n)) \\
\text{Bool b} & := & \mathrm{d}_1 \vee \cdot\cdot \vee \mathrm{d}_n \\
\text{Conjunct d} & := & \pi_1 \wedge \cdot\cdot \wedge \pi_n \\
\text{Predicate } \pi & := & \texttt{Match}(v_i, \mathrm{r}, k) \mid \neg\, \texttt{Match}(v_i, \mathrm{r}, k) \\
\text{Trace expr e} & := & \texttt{Concatenate}(\mathrm{f}_1, \cdot\cdot, \mathrm{f}_n) \\
\text{Atomic expr f} & := & \texttt{SubStr}(v_i, \mathrm{p}_1, \mathrm{p}_2) \\
 & \mid & \texttt{ConstStr}(s) \\
 & \mid & \texttt{Loop}(\lambda w : \mathrm{e}) \\
\text{Position p} & := & \texttt{CPos}(k) \mid \texttt{Pos}(\mathrm{r}_1, \mathrm{r}_2, \mathrm{c}) \\
\text{Integer expr c} & := & k \mid k_1 w + k_2 \\
\text{Regular Expression r} & := & \texttt{TokenSeq}(\mathrm{T}_1, \cdot\cdot, \mathrm{T}_m) \\
\text{Token T} & := & C+ \mid [\neg C]+ \\
 & \mid & \text{SpecialToken}
\end{array}
$$

预言制导的归纳程序合成 (Oracle-Guided Inductive Synthesis)

- 将样例编程用于更一般的程序合成问题



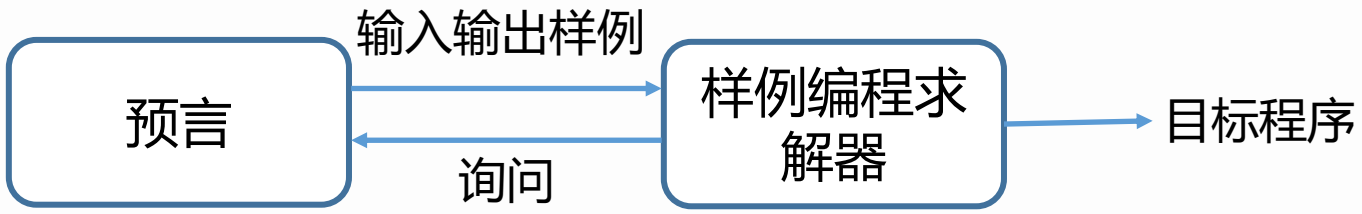
- 例子：反例制导的程序合成



轮数	候选程序	反例
1	x	$\langle x = 0, y = 1 \rangle \mapsto 1$
2	y	$\langle x = 1, y = 0 \rangle \mapsto 1$
3	$\max(x, y)$	√

程序合成与机器学习的联系

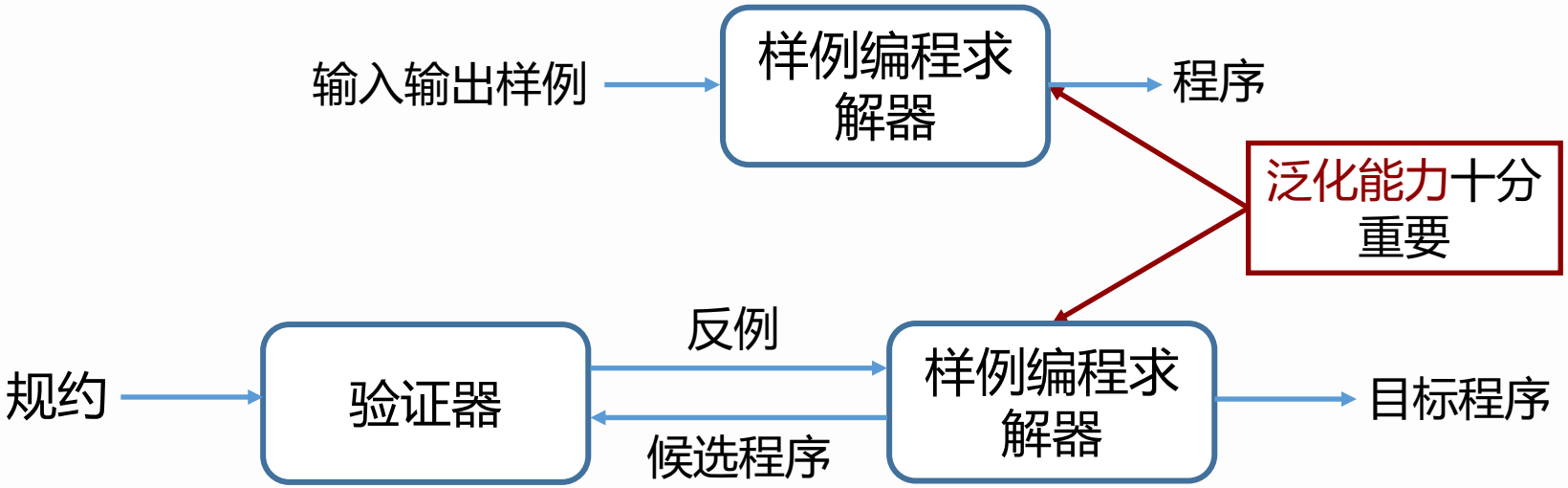
	目标：找到完全正确的程序	离散	语法+语义	
样例编程任务	样例输入	样例输出	领域特定语言	程序	可解释性
机器学习任务	训练数据	标签	假设空间	模型	求解能力
	目标：拟合某个复杂的函数	连续	语义		



主动学习

可泛化的基于合一化的程序合成

样例编程求解器的泛化能力



- 泛化能力：样例编程求解器从少量输入输出样例生成正确程序的能力
- 从样例中合成：增加合成结果的可靠性，减少用户负担
- 从规约中合成：减少程序合成所需要的轮数，从而减少程序合成的时间开销

研究现状

程序合成领域关于泛化能力的研究十分有限

- **理论层面**：没有关于程序合成泛化能力的理论结果。
- **应用层面**：利用语法信息转成最优化问题

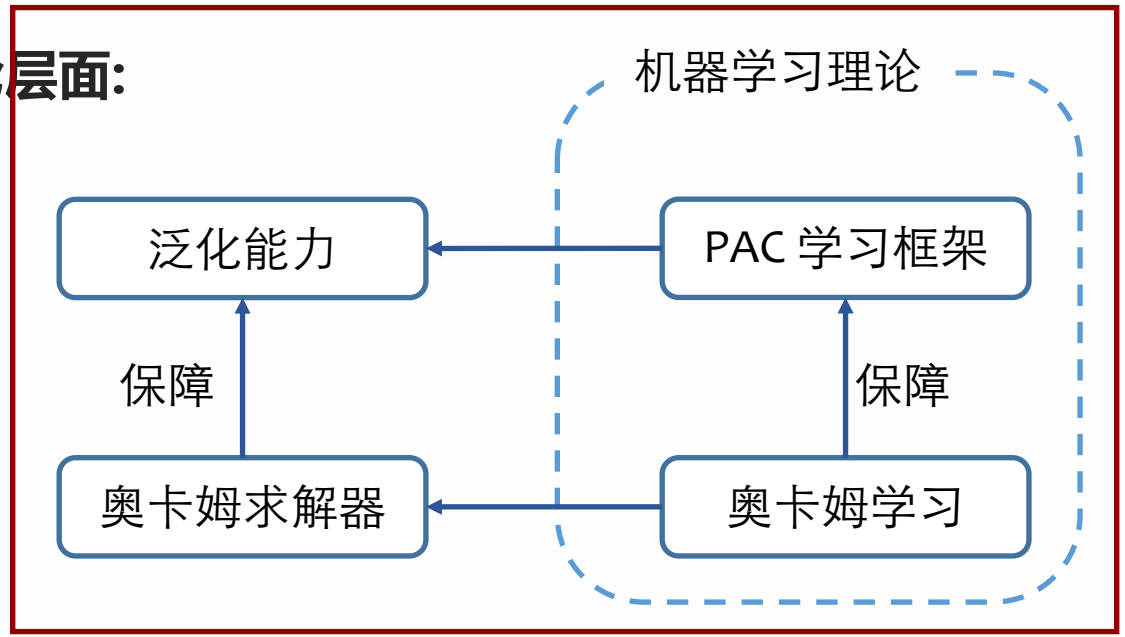
Ranking Method	Acc@1	
	$m=1$	$m=2$
(A) SHORTEST PROGRAM	0.37	0.69
(B) FEWER CONSTANTS	0.38	0.60
(A) and (B)	0.44	0.72
ML-based Ranker	0.65	0.81

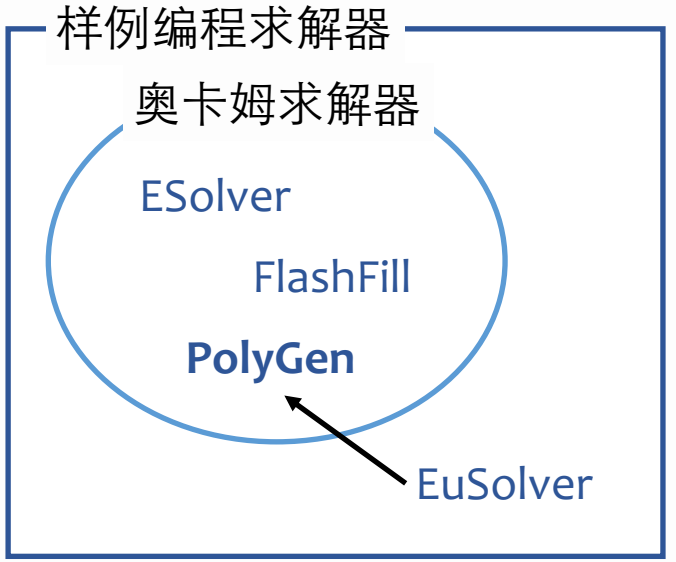
- 问题：无法处理最优化问题难以求解的情况
  - 假设要合成等价于 $\max(x_1, x_2, x_4 + x_5)$ 的条件线性程序
  - Eusolver 需要 393 个输入输出样例。

if-then-else，线性运算符，大小比较，逻辑运算

我们的工作

- **理论层面:**



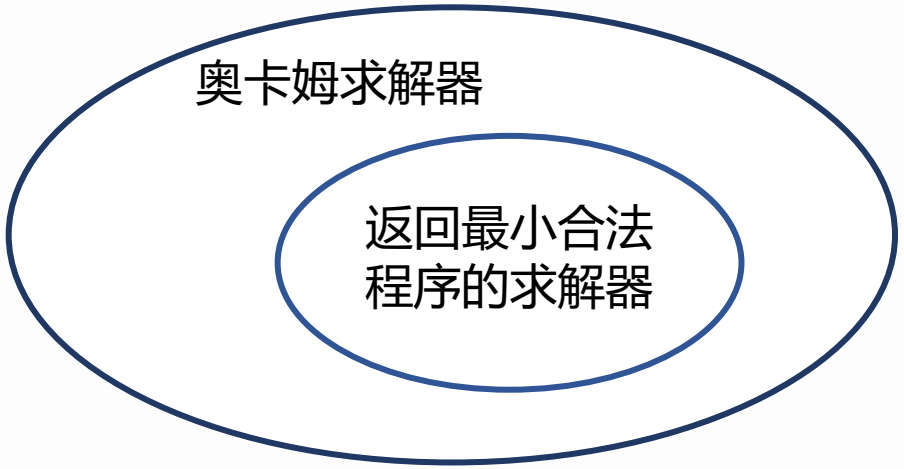


- **应用层面:** 提出了奥卡姆求解器 PolyGen
  - 假设要合成等价于 $\max(x_1, x_2, x_4 + x_5)$ 的条件线性表达式
  - Eusolver 需要 393 个输入输出样例
  - PolyGen 仅需要 10 个输入输出样例

样例编程中的奥卡姆学习

- **奥卡姆求解器**：保证合成结果的大小是有界的
  - 关于最小合法程序的大小是多项式的
  - 关于输入输出样例的数量是亚线性的
- **泛化能力的理论保障**
  - Blumer et al. [1987]: “合成结果的泛化误差大于某个阈值” 的概率是有界的

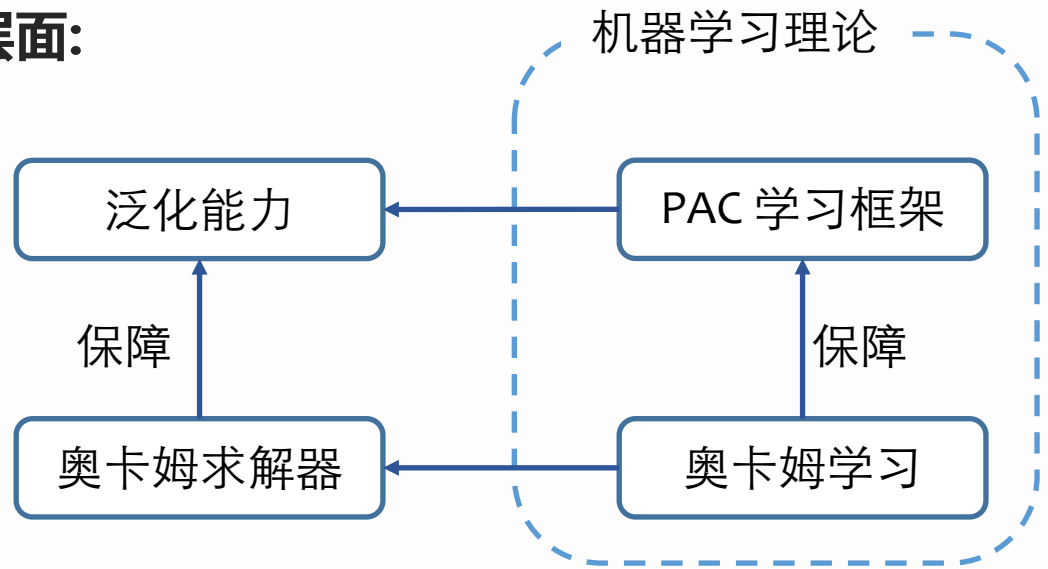
PAC 学习框架

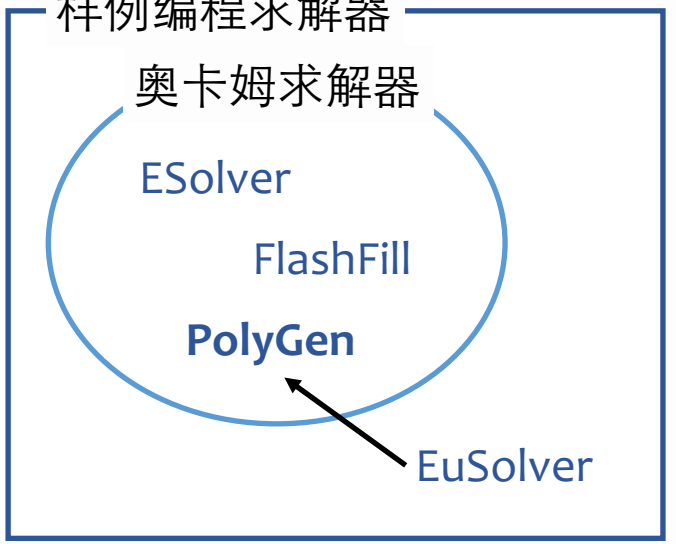


- 将返回最小合法程序的要求放松为了返回一个较小的合法程序。
- 提供了在合成问题的求解难度与泛化能力之间更加细粒度的取舍方法

我们的工作

- **理论层面:**





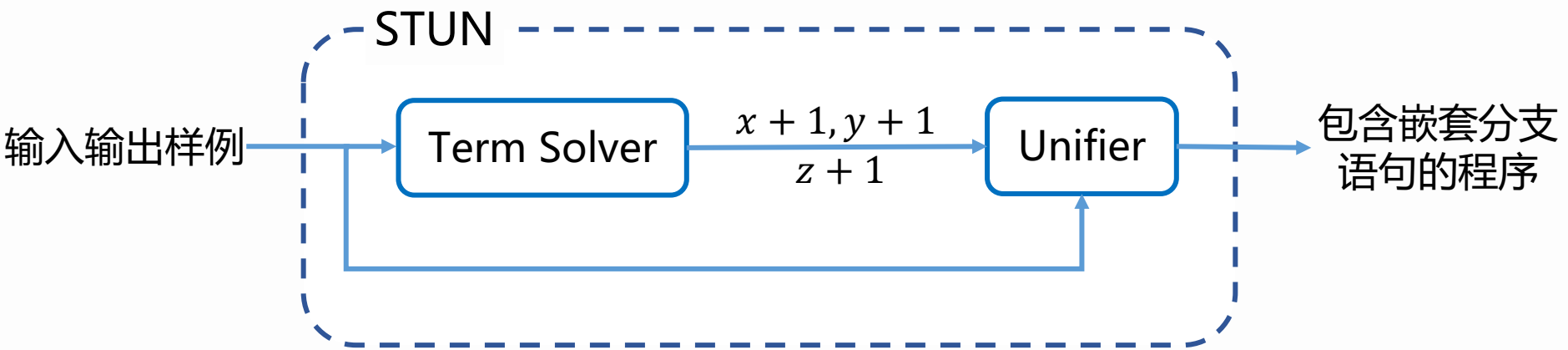
- **应用层面:** 提出了奥卡姆求解器 PolyGen
  - 假设要合成等价于 $\max(x_1, x_2, x_4 + x_5)$ 的条件线性程序
  - Eusolver 需要 393 个输入输出样例
  - PolyGen 仅需要 10 个输入输出样例

基于合一化的程序合成框架 (Synthesis through Unification, STUN)

- 一个用于处理嵌套分支语句(if-then-else)的合成框架

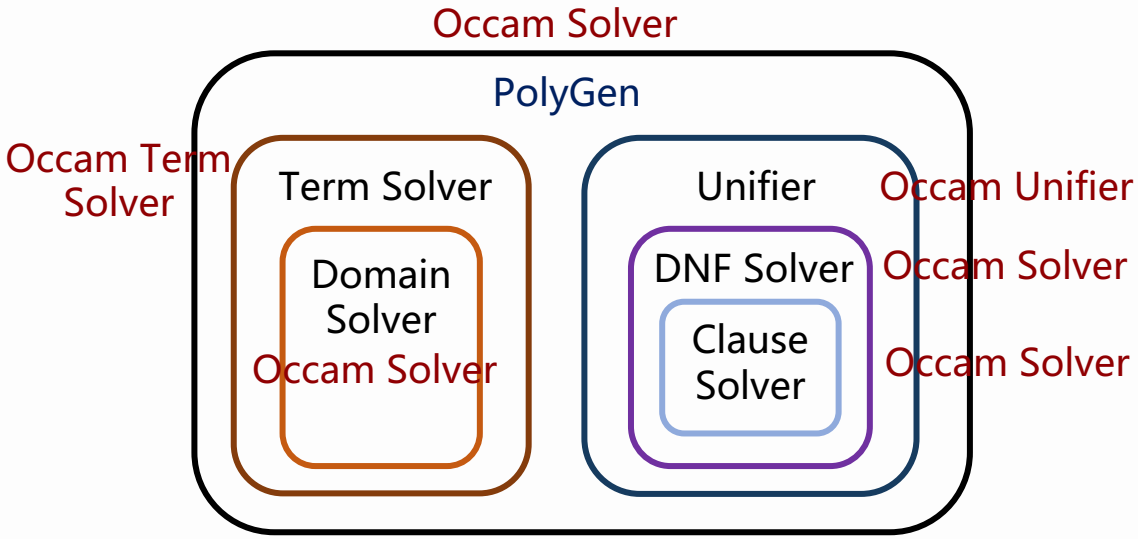
$$\begin{array}{rll} \texttt{if} & ((x+y\geq 1)\ \texttt{and}\ (x+z\geq 1)) & \texttt{then}\ \{x+1\} \\ \texttt{else if} & ((x+y\geq 1)\ \texttt{or}\ (y+z<1)) & \texttt{then}\ \{y+1\} \\ & & \texttt{else}\ \{z+1\} \end{array}$$

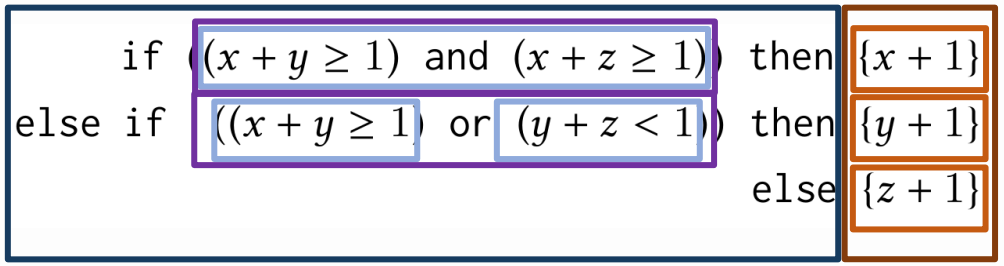
- 首先合成需要的 if-terms，再找到合适的 if-conditions 将其合并



PolyGen

- **主要思路:**
  - 在基于合一化的程序合成的基础上，进一步地将合成问题拆分成子问题
  - 将奥卡姆求解器的条件分解为每一个子问题上独立的条件
  - 根据对应的条件对每一个子问题设计高效的求解器





泛化能力方面的实验结果

Model	$\mathbb{O}_V$			$\mathbb{O}_R$			
Solver	#Solved	#Example	Time Cost	#Solved	#Example	#Example≥	Time Cost
PolyGen	97	×1.000	×1.000	93	×1.000		×1.000
Esolver	9	×0.969	×3.668	9	×1.065		×52.271
Esolver$^+$	26	×0.912	×8.233	15	×0.709		×43.261
Eusolver	65	×2.332	×6.140	65	×1.639	×3.320	×12.825
Euphony	51	×2.271	×7.417	53	×1.115	×3.302	×15.067

- 与保证返回最小合法程序的求解器相比 PolyGen 达到了几乎相同的泛化能力.
- 与已有的 STUN 求解器相比 PolyGen 的泛化能力显著更好

求解能力方面的实验结果

Model	$\mathbb{O}_V$			$\mathbb{O}_R$			
Solver	#Solved	#Example	Time Cost	#Solved	#Example	#Example≥	Time Cost
PolyGen	97	×1.000	×1.000	93	×1.000		×1.000
Esolver	9	×0.969	×3.668	9	×1.065		×52.271
Esolver$^+$	26	×0.912	×8.233	15	×0.709		×43.261
Eusolver	65	×2.332	×6.140	65	×1.639	×3.320	×12.825
Euphony	51	×2.271	×7.417	53	×1.115	×3.302	×15.067

- 与所有的 baseline 相比，PolyGen 都用更短的时间求解了更多的任务
  - PolyGen 求解了多 43.08%-90.20% 的任务，并达到了 ×6.14-×15.07 的平均加速比.

Summary

- **理论层面:**
  - 根据奥卡姆学习理论提出了奥卡姆求解器的概念
    - 给出了关于泛化能力的理论保障.
- **应用层面:**
  - 提出了奥卡姆求解器 PolyGen.
    - 将合成问题分解成子问题，并同时将奥卡姆求解器的要求分解到每一个子问题上.
- 我们的实验结果证明了 PolyGen 的有效性.

谢谢！