# Intent Formalization

# PBE is not always enough

**Pros**

- Ease of use
- Broader base of users
- Error detection

**Cons**

- Ambiguity in intent
- Lack of Formal Guarantee

One needs to formally specify the desired behavior of the target program!

# Example

```
int f(int x, int y)
{
        z = 0;
        i = x;
        while (i) {
                z = z + y;
                i = i – 1;
        }
        return z;
}
```

PBE:

| $x$ | $y$ | $out$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 3 | 6 |
| 2 | -1 | -2 |
| 4 | 0 | 0 |
| 3 | 4 | 12 |

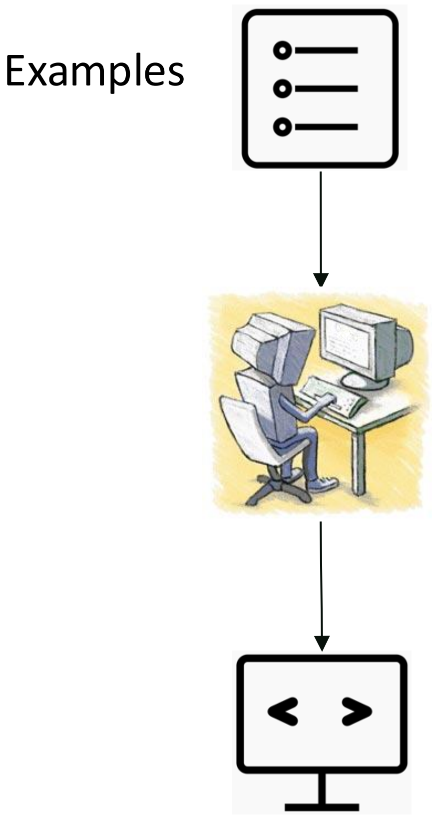
Formal spec: $out = x \cdot y$

# Who is going to write the formal specification?

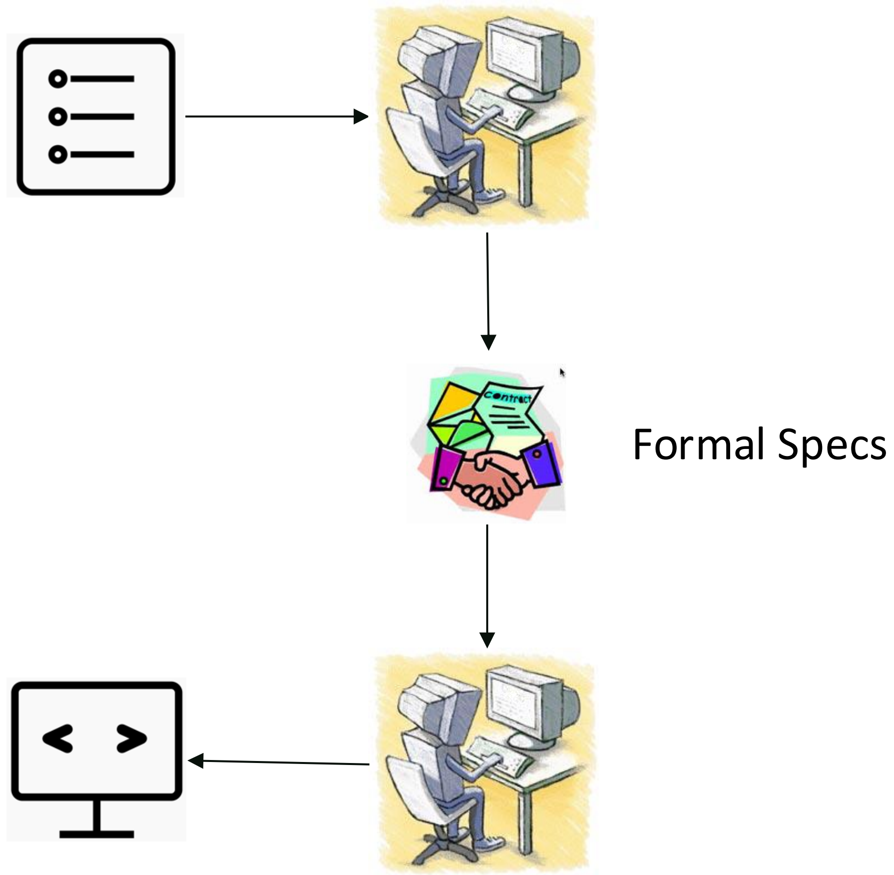
- Most users are not trained to write specifications using logics or other rigorous mathematical languages
- The synthesizer is sneaky and always finds a workaround (e.g., an infinite loop) for an incomplete specification

# Examples to the rescue!

**Traditional PBE**




**Synthesize formal specs from examples**



# From Examples to RegEx

# Regular Expressions

Regular expressions are a *syntactic tool* for defining *regular languages*

- Common feature in many languages; but the basics of regular expressions are much simpler than what you see in languages like Perl or Python
- String literals combined by **choice** and **star**
- **"Regular" languages:** Regular expressions can be represented as deterministic finite automata (DFA), and vice versa

alice@example.com
bob.smith@company.org
charlie123@mail.co.uk
david@my-email.com

→ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$

(123) 456-7890
987-654-3210
(555) 123-4567
111-222-3333

→ ^(\(\d{3}\)\s?|\d{3}[-.\s])\d{3}[-.\s]?\d{4}$

# Angluin's L* Algorithm (1987)

DFA

Query

Answer/Example

Student (Synthesizer)

Minimally Adequate Teacher (User)

**Membership Query**

- Is the string *s* accepted by the target language?
- Answer: yes/no

**Equivalence Query**

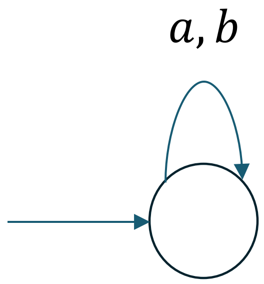
- Does the candidate DFA match the target language?
- Answer: yes/counterexample
- Done if the answer is yes!

# Idea: maintain a candidate DFA

0. Represent the DFA as an Observation Table (ignored)
1. Start with a simple hypothesis DFA (usually an empty state machine).
2. Use equivalence queries to check if the current hypothesis matches the target DFA.
3. If not match, use membership queries to refine the DFA with respect to the counterexample.
4. Repeat from step 2 until it correctly recognizes the language.

# Example

Alphabet: $\{a, b\}$

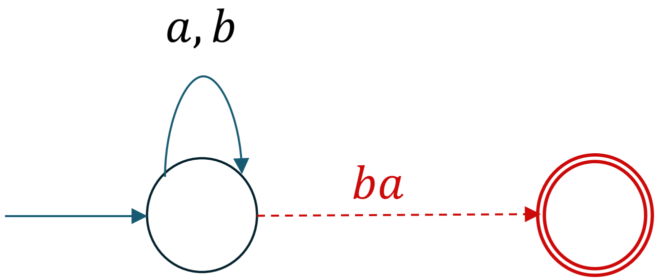



Equivalence?

Cex: $ba$

# Example

Alphabet: $\{a, b\}$



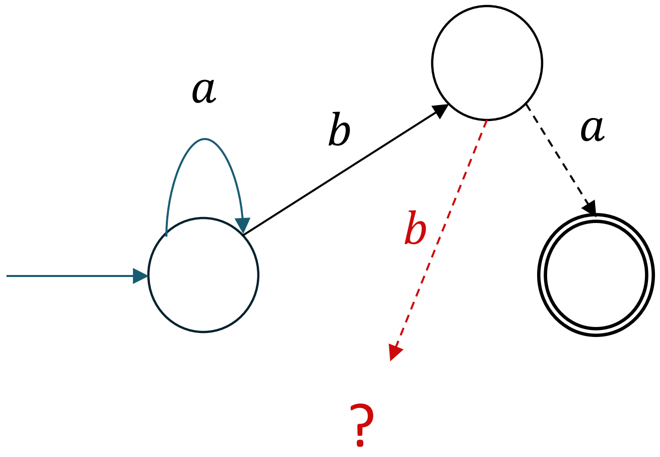

But how?

Membership($b$)?

Yes

# Example

Alphabet: $\{a, b\}$

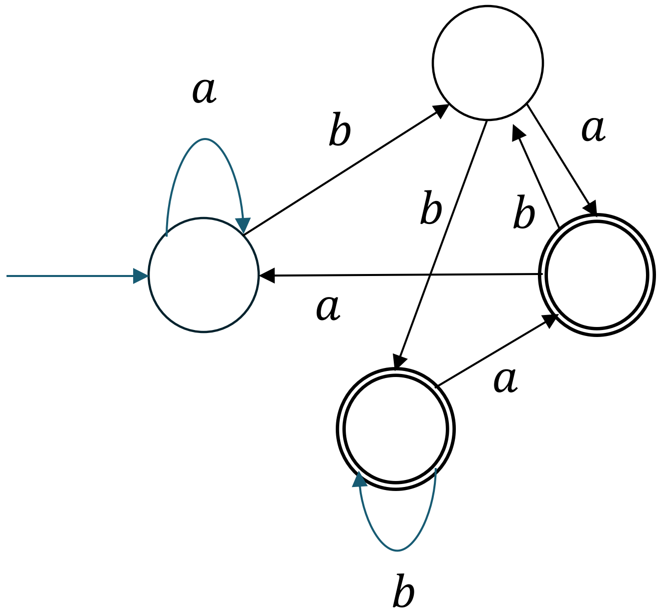



Membership($bb$)?

No

# Example

After several rounds of queries…



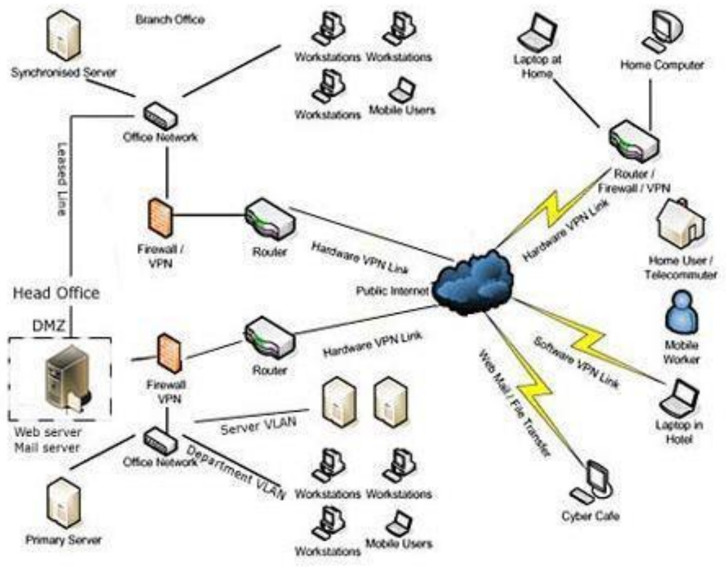

Equivalence?

# Model Learning

*"Even though faster algorithms have been proposed since then, the most efficient learning algorithms that are being used today all follow Angluin's approach of a minimally adequate teacher (MAT)." – Frits Vaandrager*

# Comparative Synthesis

# Motivation: Network Design




- Plenty of source-destination pairs
- Multiple paths per s-d pair
- Multiple traffic classes
- Capacity/security restriction
- Given demand, figure out allocations

**It is not too hard to find a mediocre network design.**

# But what is the optimal solution?

How to trade-off multiple objectives (throughput and latency)?

How to enforce fairness requirements for flows across different traffic classes?

Software-driven WAN [1]

- Maximize throughput
- Minimize latency

**How to decide the knobs?**

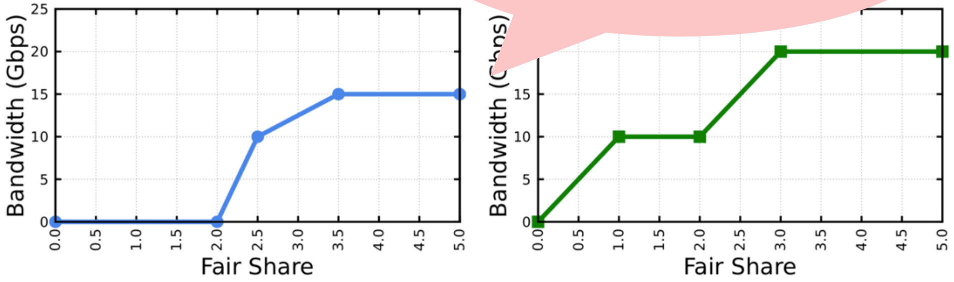
Maximize

$$
\begin{aligned}
&O(\text{throughput}, \text{latency}) \\
&= \ 2 * \text{througput} - \max(750 - \text{throughput}, 0) \\
&\quad -5 * \text{latency} \ - 9 * \max(\text{latency} \ - 75, 0)
\end{aligned}
$$

Bandwidth Enforcer (BwE) [2]

- Different classes have differen… requirements and different prioritie…

**How to decide fair shares?**

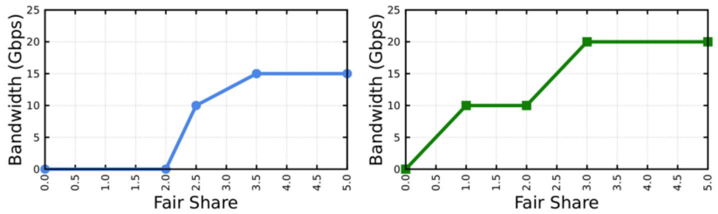



Fair share as concave functions


[1] Achieving High Utilization with Software-Driven WAN. [Hong et al., SIGCOMM'13]
[2] BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. [Kumar et al., SIGCOMM'15]

# The synthesis conundrum remains

“Maximize throughput”
“Minimize latency”




$$\text{Start} ::= \text{Start} + \text{Start}/0.5 \mid x \mid 0 \mid 1 \mid \text{Expr}$$

$$\text{Expr} ::= \text{Expr} + \text{Expr}/0.4 \mid x \mid 0 \mid 1$$

$$O(throughput, latency) = 2 * througput - \max(750 - throughput, 0) - 5 * latency - 9 * \max(latency - 75, 0)$$

Give me an optimal program!

Give me the optimization target!

# But didn’t we have PBE?

| Input | Output |
| --- | --- |
| Throughput: 10Gbps<br>Latency: 100ms<br>… | ?? |
| Throughput: 7Gbps<br>Latency: 300ms<br>… | ?? |

Let’s compare!

Throughput: 10Gbps
Latency: 100ms
…

vs.

Throughput: 7Gbps
Latency: 300ms
…

Throughput: 10Gbps
Latency: 100ms
…

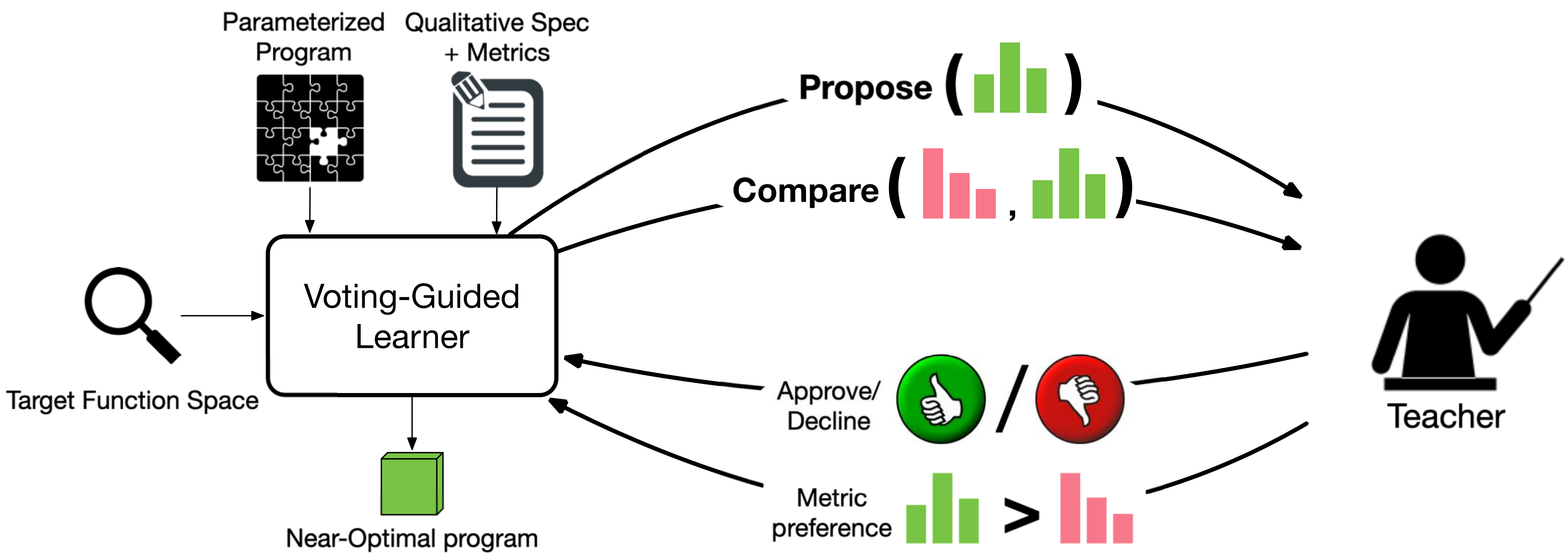
***“Comparison is the thief of joy.”***

--Theodore Roosevelt

# Net10Q: Comparative Synthesis for Network Design

**Learner's Goal:**

Spend **a budgeted number of queries** and to produce a **near-optimal** program *from the perspective of the teacher*



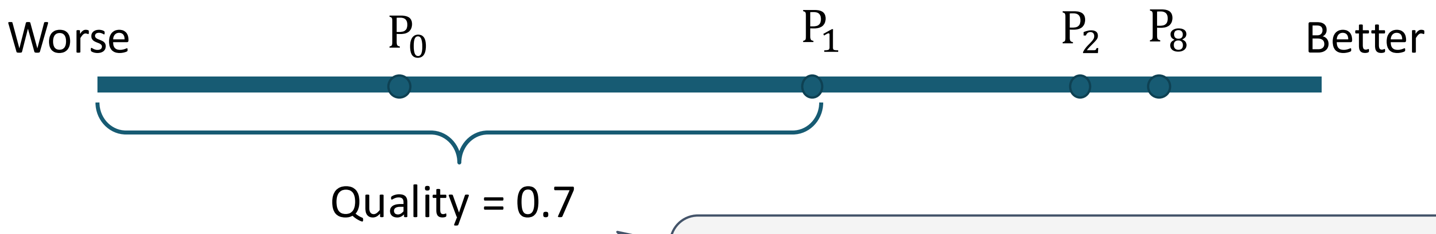

w/ Yanjun Wang, Chuan Jiang, Zixuan Li, Sanjay Rao. [HotNets'19 + POPL'23]

# Quality of Solution

How close a solution is to the ground truth optimal?

**Quality of Solution:**

The **“relative rank”** of the solution among all solutions

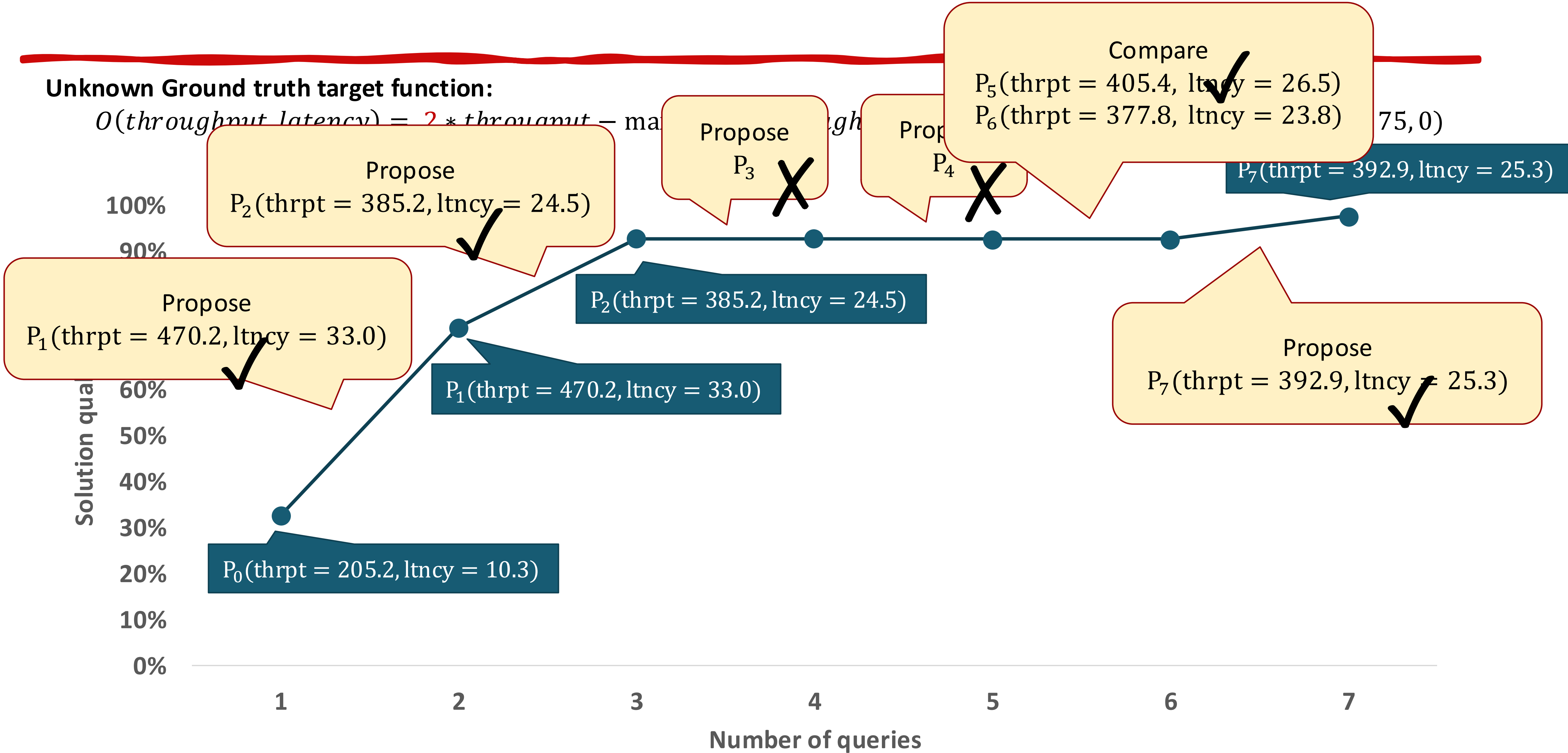



$P_1$ is better than or equal to 70% of possible solutions

Computing the exact quality can be expensive, we estimate the quality by **sampling**

# Example

**Unknown Ground truth target function:**



# How does the synthesizer work?

**A Voting-Guided Learning Algorithm**

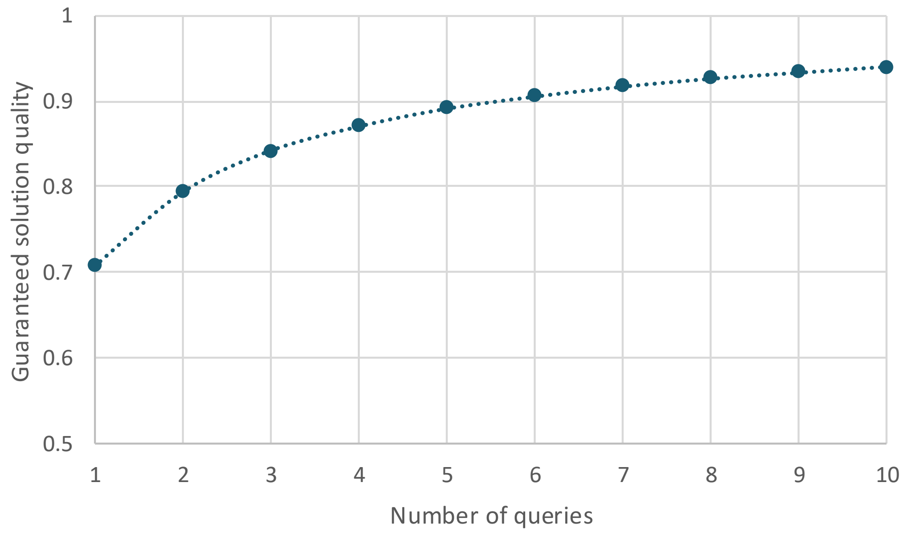
- Maintains a *Pareto candidate set*
- Each query can prune the search space, one way or the other
- Greedily prune the candidate set by making the *most informative* query (i.e., maximizing the worst-case space cut)
- Re-generate more candidates when the candidate set becomes too small

# Convergence Guarantee

*(How fast does the output solution approach the optimal?)*

**Theorem**

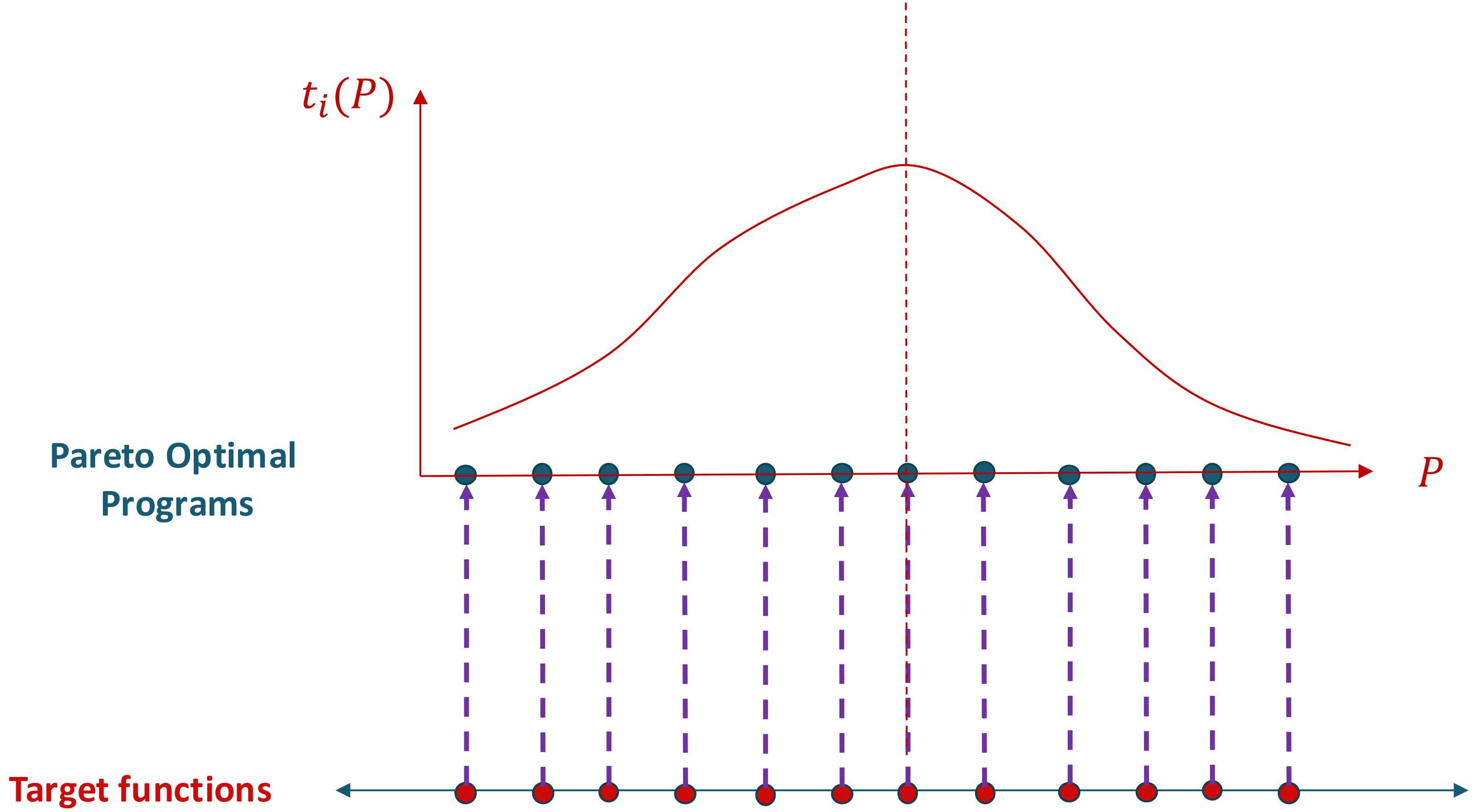
The voting-guided learning algorithm guarantees a ***logarithmic*** rate of convergence. (the median quality of solution is at least $2^{\frac{-1}{n+1}}$ after $n$ queries). The bound is tight.




**Proof idea**: Every query discards at least one randomly generated candidate.

# Can the algorithm converge faster?

*(Yes, when the search space is **sortable**.)*




**Example**

If there are two competing metrics (e.g., *throughput* and *latency*) such that for each metric continued improvement leads to diminishing marginal utility, the search space is *sortable*.

# Can the algorithm converge faster?

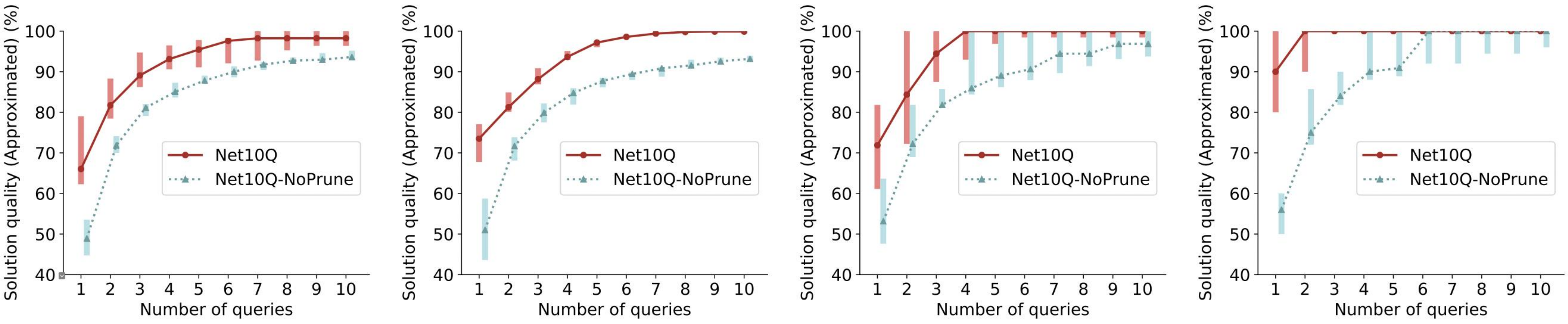
*(Yes, when the search space is* ***sortable****.)*

**Theorem**

When the target function space is *sortable*, the voting-guided learning algorithm guarantees a ***linear*** rate of convergence. (the median quality of solution is at least $1-\frac{1}{\Omega(1.5^n)}$ after $n$ queries). The bound is tight.

**Proof idea**: Every query discards at least one third of the candidates from the current PCS pool.

# Oracle-based Evaluation (Perfect Oracle)




**(a)** MCF. **(b)** BW. **(c)** NF. **(d)** OSPF.

***Net10Q*** performed constantly better than ***Net10Q-NoPrune***

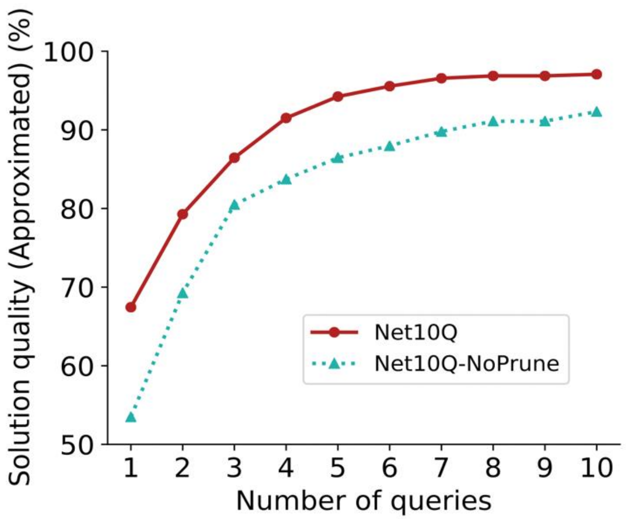
# Oracle-based Evaluation (Imperfect Oracle)

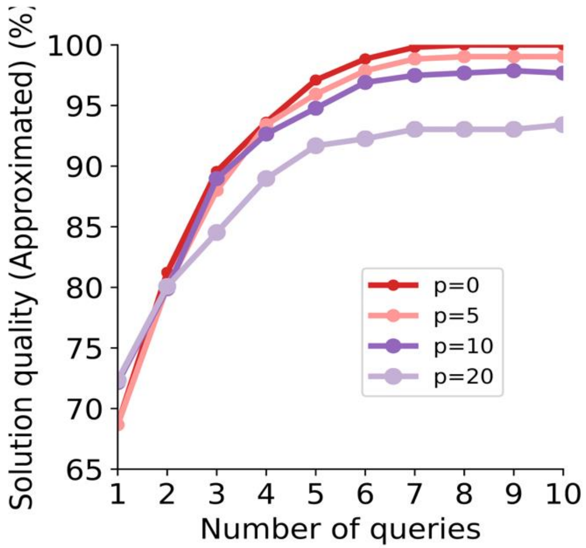
(BW on CWIX)

Imperfect oracle model:

assigns a **random** reward that is sampled from a normal distribution (tunable by p)

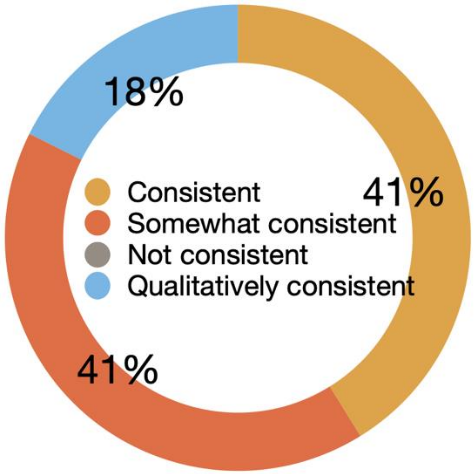
Net10Q vs. Net10Q-NoPrune (p=10)



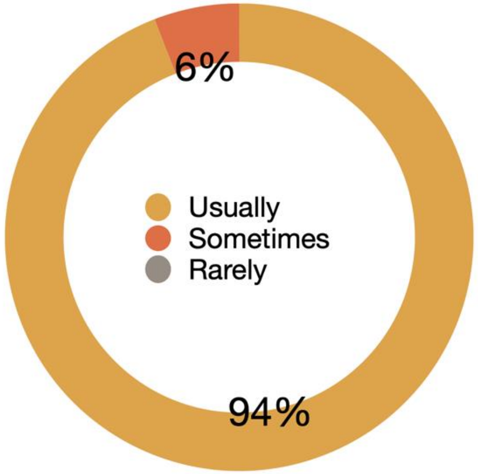




***Net10Q*** can handle moderate feedback inconsistency

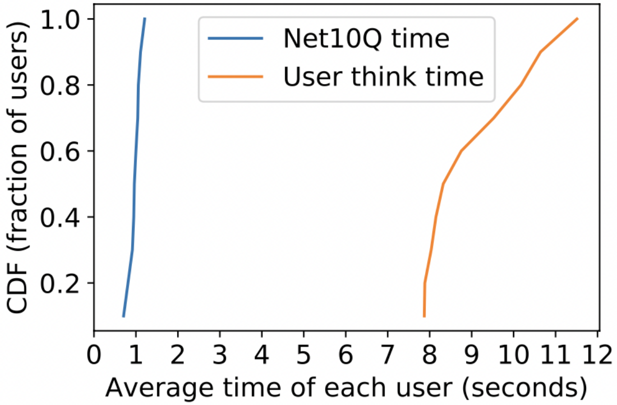
# User Feedback




Quality of recommendations




Was time taken acceptable?




Average time per query across users

**Qualitative Comments**

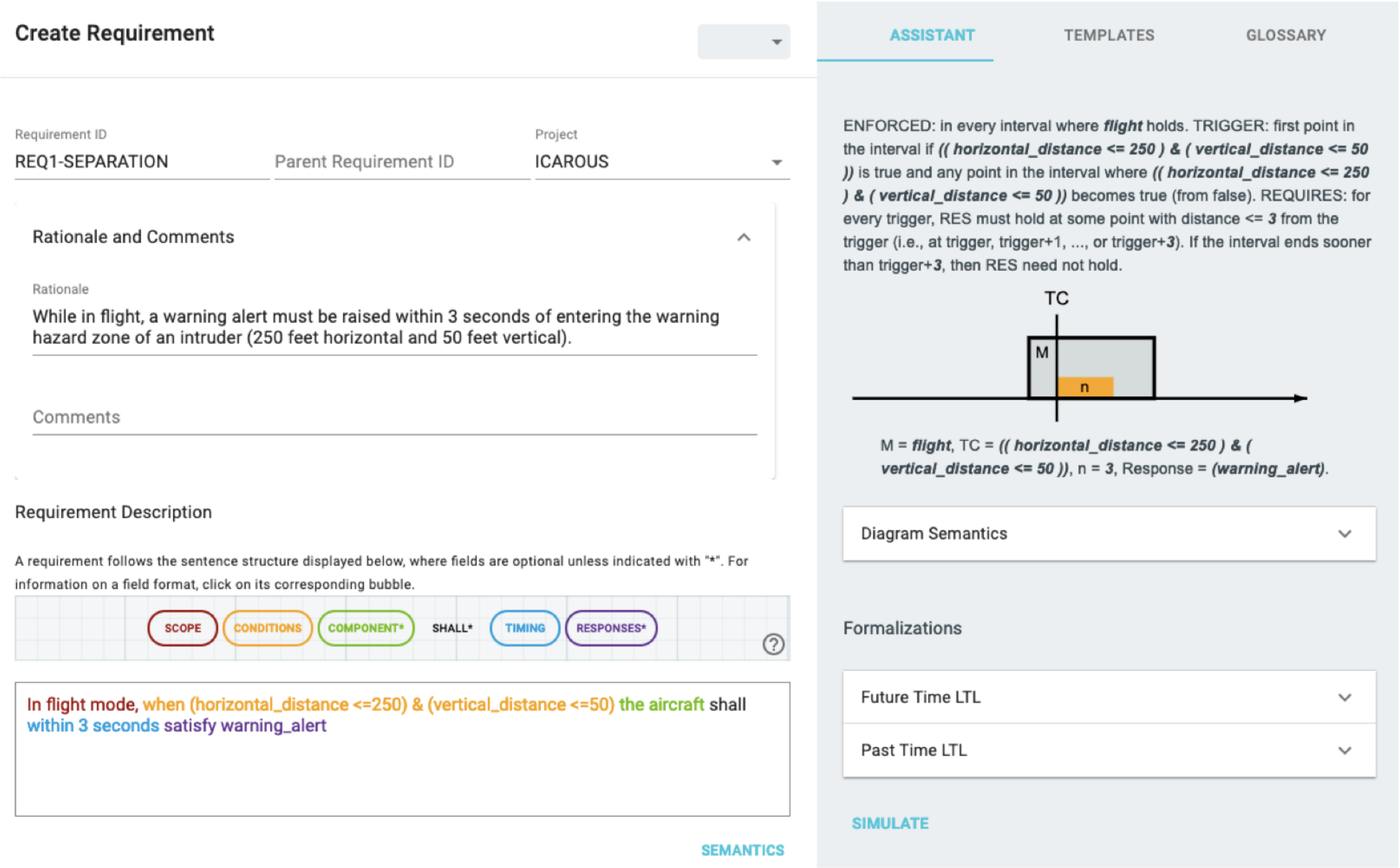
- Most users are satisfied with ***Net10Q***'s recommendations and response time.
- *"The study was well done in my opinion. It put the engineer/architect in a position to make a qualified decision to try and chose the most reasonable outcome."* – an expert user

# Can we learn formal specification from natural languages?

- Again, examples are not comprehensive, and good examples can be very expensive.
- Accurate specification may call for excessive examples
- *"Natural language will always remain the basic interpretation of, and reservoir for, the development of the artificial formalized languages of science."* – Doris Bradley

# Pre-LLM Age

**FRET:**
Controlled Natural Language to Metric Temporal Logic (MTL)




Conrad et al. A Compositional Proof Framework for FRETish Requirements[CPP'22]

# Pre-LLM Age

**Jdoctor:** translating Javadoc to JML-style specifications

```java
/**
 * @param x the number to check, must be positive
 * @return true if x is prime, false otherwise
 * @throws IllegalArgumentException if x is negative
 */
public boolean isPrime(int x) { ... }
```

```java
// Preconditions
assert x > 0;

// Exceptional Postconditions
if (x < 0) throw new IllegalArgumentException();

// Normal Postcondition
return isPrime(x);
```

Blasi et al. Translating Code Comments to Procedure Specifications [ISSTA'18]

# Pre-LLM Age

**Pros**

- Pattern matching for semi-structured languages, which is predictable, understandable and adaptable
- Low computational cost

Cons

- Limited generalization
- Limited handling of ambiguities
- Manually defined patterns

# NLP is the bread and butter of LLMs!

**Can Large Language Models Write Good Property-Based Tests?**

**Clarify When Necessary:**
**Resolving Ambiguity Through Interaction with LMs**

**Finding Inductive Loop Invariants using Large Language Models**

**Can Large Language Models Transform Natural Language Intent into Formal Method Postconditions?**

**Tell Me More! Towards Implicit User Intention Understanding of Language Model Driven Agents**

USER INTENT RECOGNITION AND SATISFACTION WITH LARGE LANGUAGE MODELS: A USER STUDY WITH CHATGPT

*SpecGen*: Automated Generation of Formal Program Specifications via Large Language Models

**Can Large Language Models Reason about Program Invariants?**

# Open Question: How to evaluate LLM-generated specs?

Shuvendu Lahiri proposed the notions of soundness and completeness with respect to a set of input-output tests $T$:

- **Soundness:** all tests in $T$ satisfy $\phi$
- **Complete measure:** if $(i, o')$ is a mutation from a test $(i, o) \in T$, how likely $(i, o')$ is *inconsistent* with $\phi$

**Problems**

- Higher completeness measure is not always better (e.g., $x > 1.04562$ is likely overfitting; $x > 1$ is more natural)
- What if test cases are not available (how about generate test cases?)

# Open Question: What's the best paradigm?

**Natural Language Description** ⇄ **Formal Specs**

**Examples** ⇄ **Programs**

# Workflow 1

**Natural Language Description**

↓

↓

**Programs**

# Workflow 2

**Natural Language Description**

↓

↓

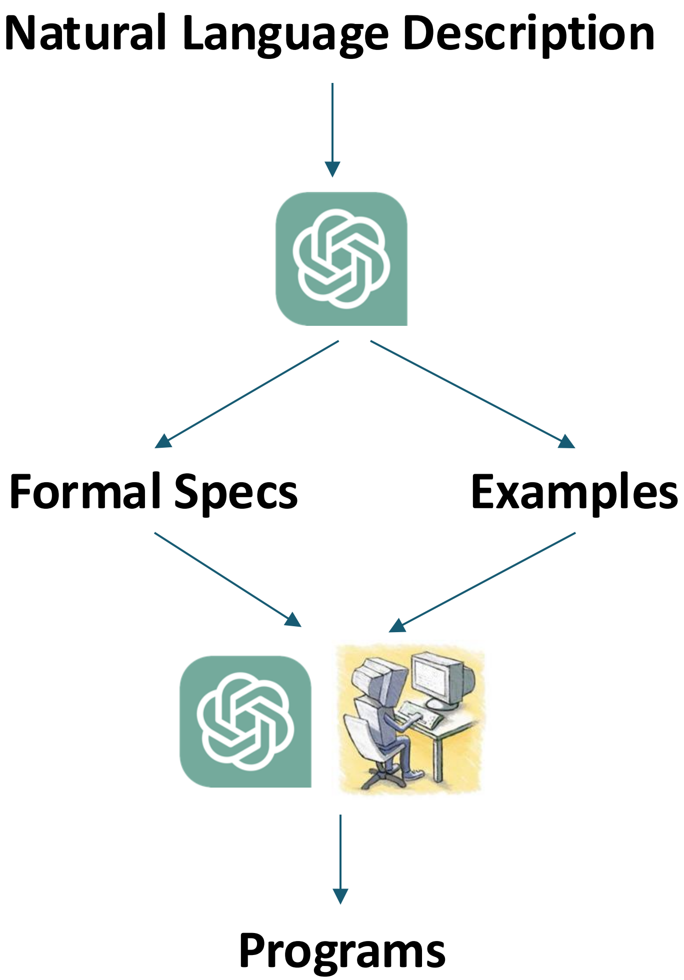
**Formal Specs**

↓

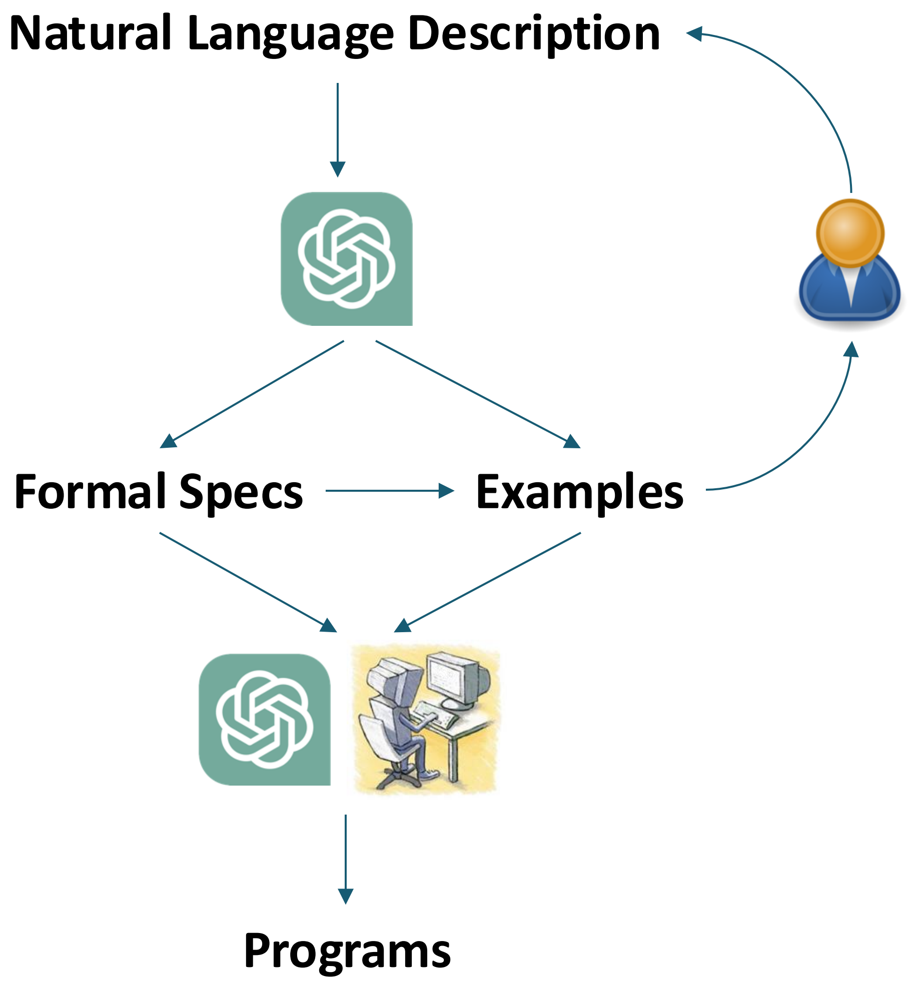
↓

**Programs**

# Workflow 3


Natural Language Description
Formal Specs
Examples
Programs

# Workflow 4


Natural Language Description
Formal Specs
Examples
Programs

# Workflow n?

*Figure out your own workflow for your project!*