EVIDEN

PISE

Protocol Inference Using Symbolic Execution and Automata Learning

Octobre 2025

PISE

Protocol Inference Using Symbolic Execution and Automata Learning

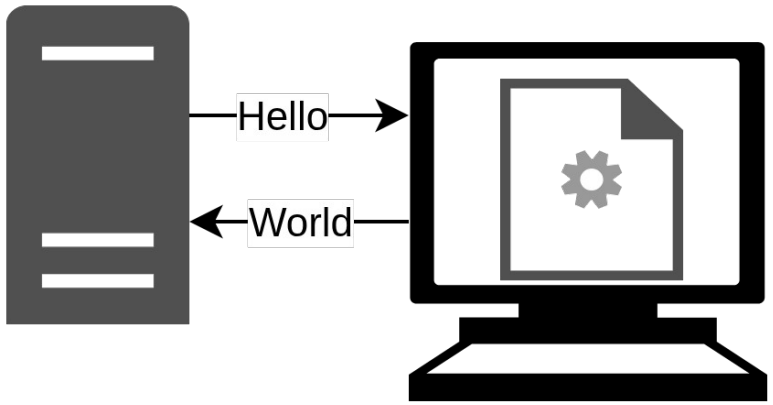
- Auteurs : Ron Marcovich, Orna Grumberg et Gabi Nakibly du Technion - Israel Institute of Technology
- Publié en 2023

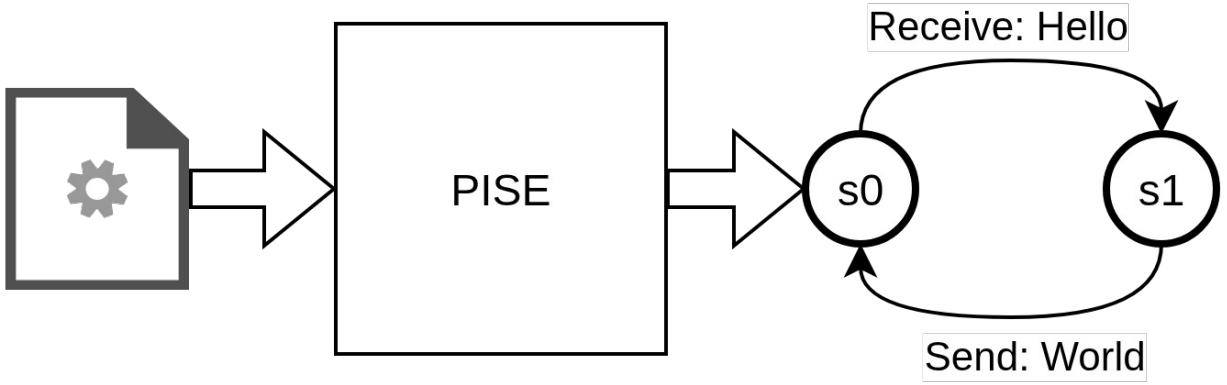
EVIDEN

Problématique

Inférence protocolaire

En partant seulement d'un binaire





EVIDEN

Comment ?

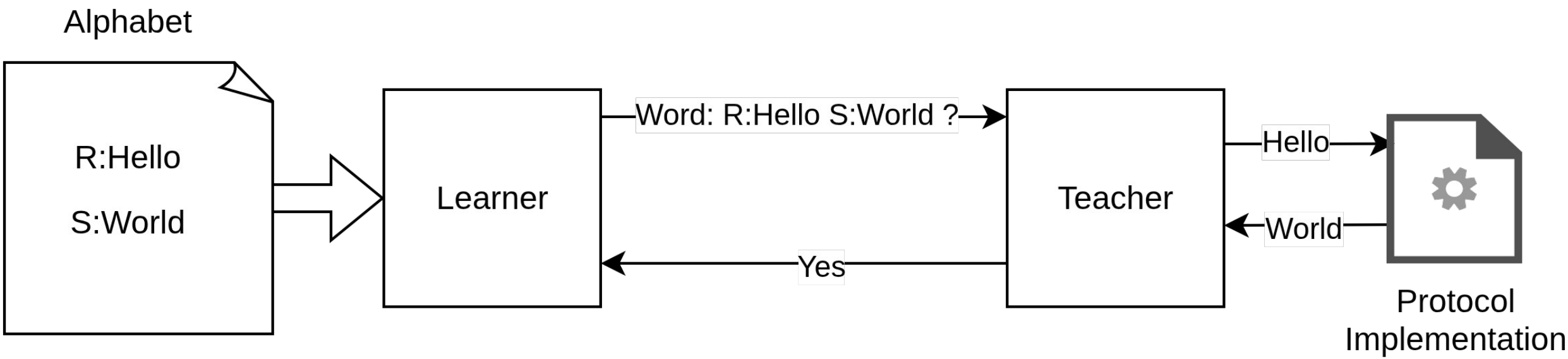
L’algorithme L*

Le fabuleux domaine des langages formels

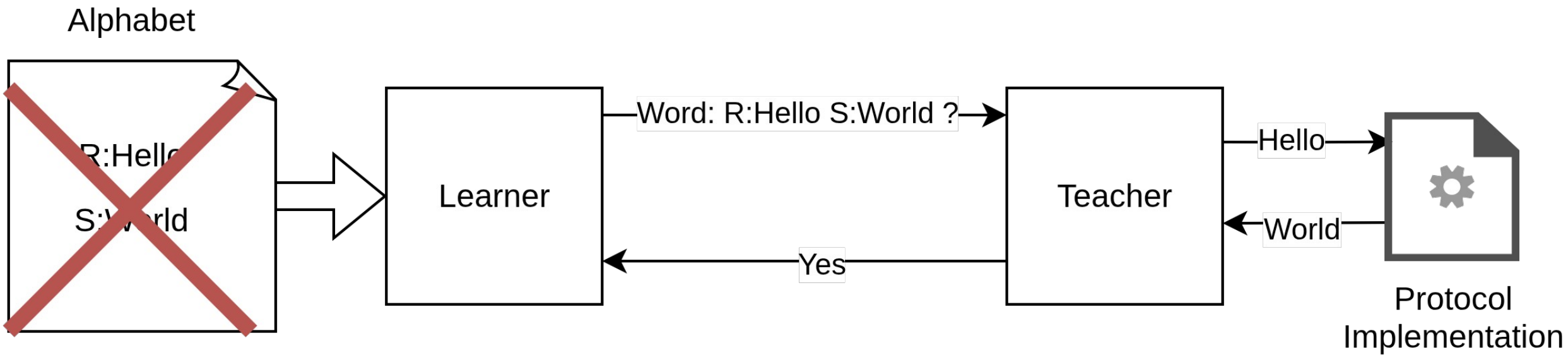
- Un langage formel L sur un alphabet A est un ensemble de mots, qui sont des séries de lettres tirées de l’alphabet.
  Exemple:A = {r0=“Receive:Hello”, s0=“Send:World”} et L = {r0, r0s0, r0s0r0, …}
- Un langage régulier est un langage qui peut être définit par une expression régulière (aka. regexp).
- Un langage est régulier si et seulement si il existe un automate déterministe fini (DFA) qui accepte ce langage.
- L’algorithme L* (Dana Angluin, 1987) permet de reconstruire l’automate pour un langage régulier à partir d’un oracle (Teacher) qui répond à deux types de requêtes:
  - Une requête d’appartenance: “Est-ce que ce mot est dans le langage ?”
    => Facile à répondre à partir d’un binaire
  - Une requête d’équivalence: “Est-ce que ce DFA est équivalent au langage”
    => Difficile à répondre à partir d’un binaire
- Une version probabiliste de L* permet de faire une approximation d’équivalence en générant une série de requête d’appartenance aléatoirement

L’algorithme L*

Pour un protocole



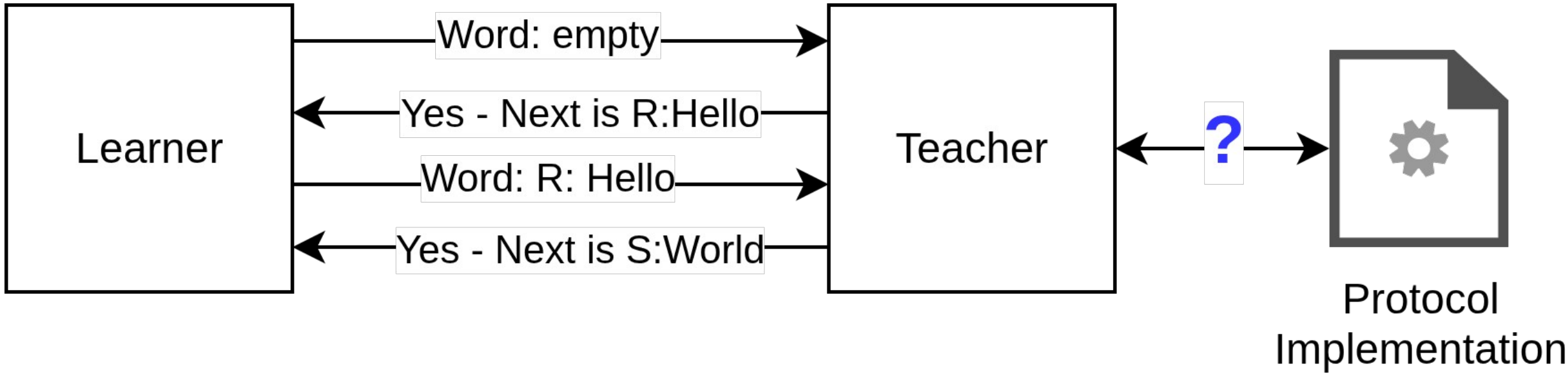
La problématique de PISE



La solution PISE

Un professeur visionnaire

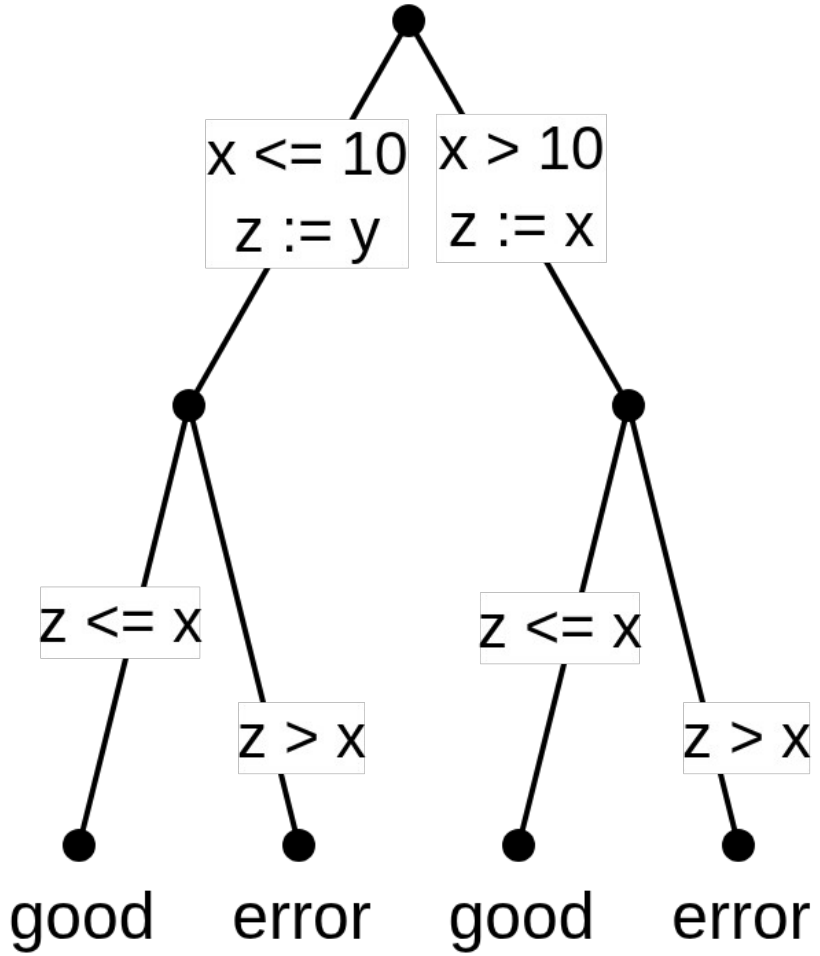
- PISE propose une version modifiée de l'algorithme L*
- Le professeur a un nouveau pouvoir :
  - Lorsque une requête d'appartenance est positive, le professeur donne la liste de toutes les lettres qui peuvent suivre le mot.
- L'apprenti est modifié pour construire son alphabet pendant l'apprentissage.



L’exécution symbolique

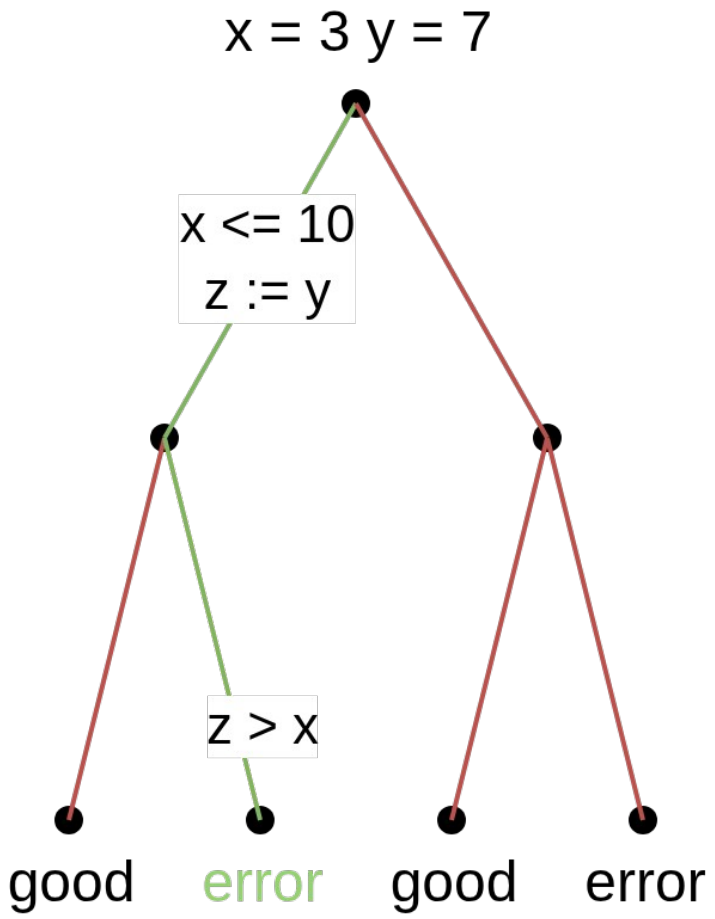
La botte secrète du professeur

```
void function(int x, int y) {
    int z;
    if (x > 10) {
        z = x;
    } else {
        z = y;
    }
    if (z > x) {
        print("error");
    } else {
        print("good");
    }
}
```



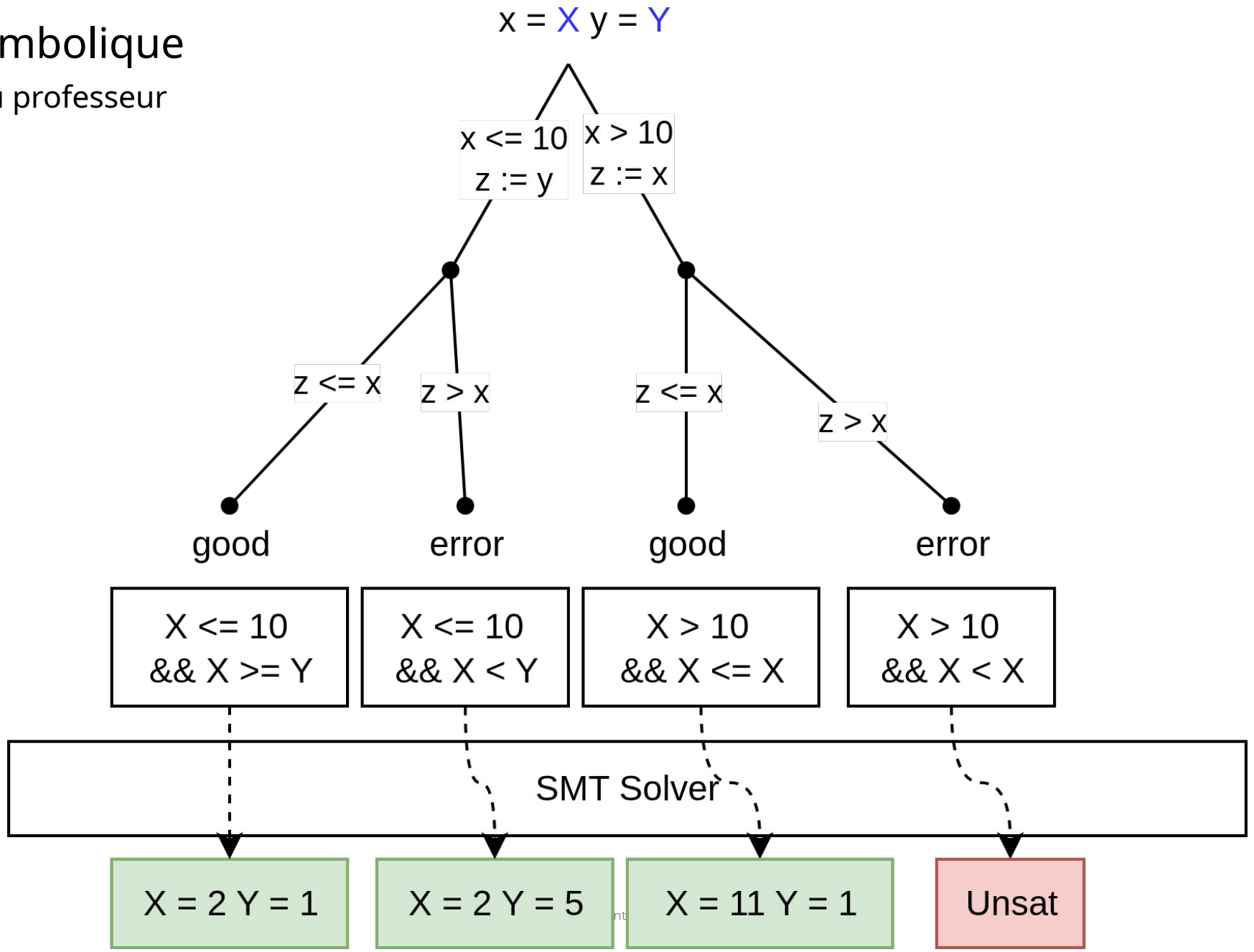
L’exécution symbolique

La botte secrète du professeur



L’exécution symbolique

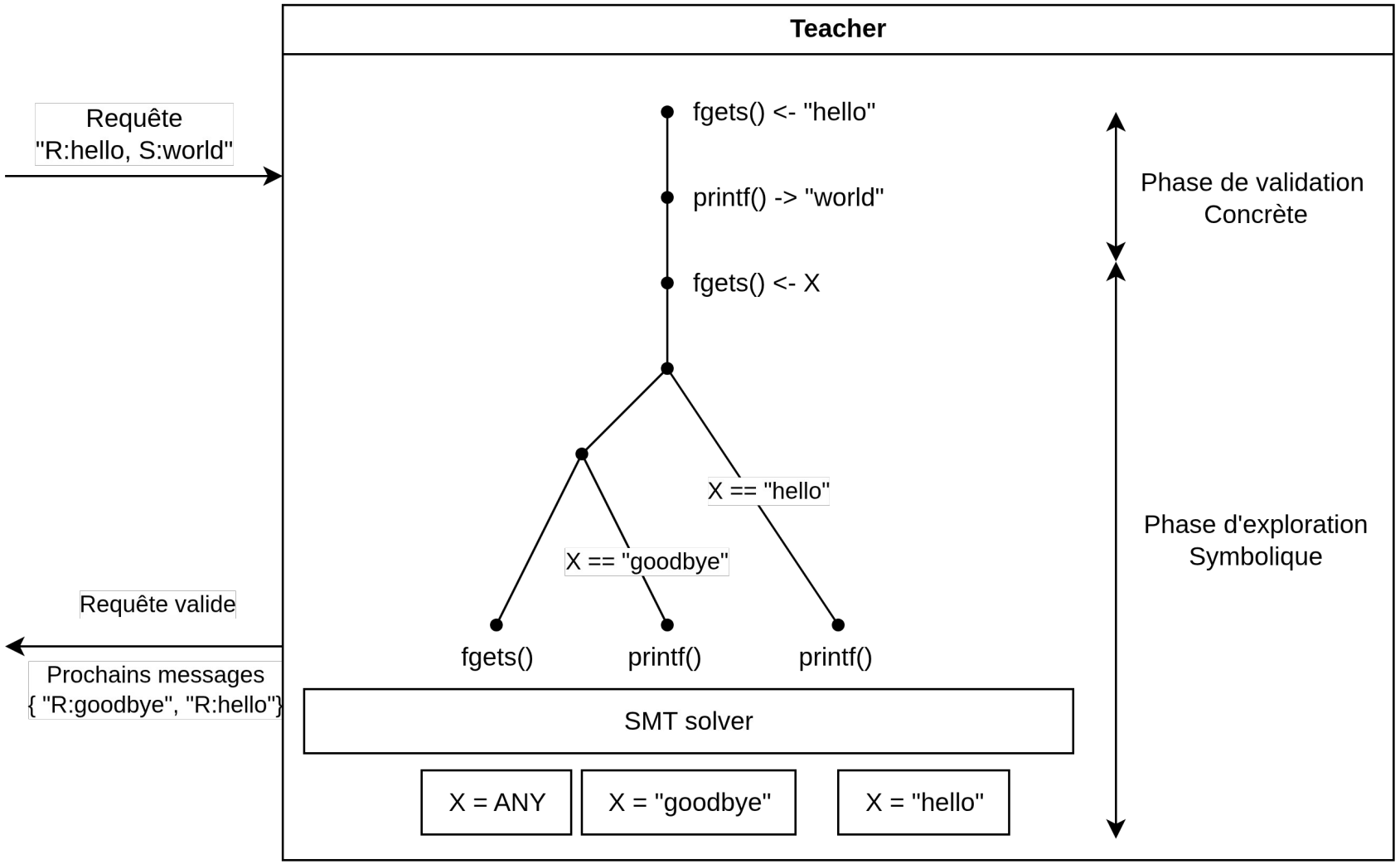
La botte secrète du professeur



Découvrir le prochain message d’un protocole

```c
int main() {
  char *buffer = malloc(sizeof(char[MAX_LEN]));
  while (1) {
    if (fgets(buffer, MAX_LEN, stdin) != NULL) {
      if (strcmp(buffer, "hello\n") == 0) {
        printf("world\n");
      } else if (strcmp(buffer, "goodbye\n") == 0) {
        printf("goodbye\n");
        break;
      }
    }
  }
  free(buffer);
}
```

Découvrir le prochain message d’un protocole



EVIDEN

A quoi ça sert ?

Les applications

- Reverse de protocoles inconnus :
  - Command & control de malwares
  - protocoles propriétaires

- Analyse de sécurité des implémentations, la machine d'état peut :
  - Réveler des non conformités à la spécification
  - Réveler si une implementation est vulnerable à une faille protocolaire
  - Permettre d'identifier une implementation (fingerprinting)

Les limites

- Requiert de connaître les points d'entrée et de sortie du binaire.
  Souvent standard : les syscalls read/write, send/recv

- L'exécution symbolique ne scale pas bien pour des programmes complexes
  - Problème d'explosion exponentielle du nombres de chemins possible
  - Le SMT solver est une operation lourde (SMT => NP complete)

- L'utilisation de code asynchrone, de multi-threading, de queues de réceptions/transmission perturbent l'analyse du protocole.

EVIDEN

Démonstration

Sources

- Publication de PISE : https://www.ndss-symposium.org/wp-content/uploads/2023/03/bar2023-23002-paper.pdf
- Presentation: https://www.youtube.com/watch?v=ou9NloRLmNk
- Code source: https://github.com/ron4548/pise
- Angr (moteur d'execution symbolique): https://github.com/angr/angr
- Learnlib (implémentation de L*): https://learnlib.de/7

EVIDEN

Questions ?