AI for verification and verification for AI

Overview and discussion

KIT, January 26th 2023
Lecture Overview

• Verification for AI
  • Sandboxing
  • Neural-network verification

• AI for Verification
  • Imitation learning of theorem proving
  • Reinforcement learning of theorem proving

• Large Language Models and the Future of Programming
Machine-Learning Primer

Supervised Learning as an Optimization Problem

**Problem:** learning an unknown function from many input/output examples.

\[
f_W(x) = W_3 \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x))
\]

\[
L(W) = \frac{1}{|D|} \sum_{x,y \in D} (y - f_W(x))^2
\]

\[
W_{t+1} \leftarrow W_t + \lambda \nabla_W L
\]

Reinforcement Learning

[Diagram of agent-environment interaction showing state, action, reward, and next state transitions, along with policy and value representations.]
AI for Verification
init → [(ctrl; plant)*)] safe
Sandboxing

**Approach:** do not trust the neural-network, sandbox it!

```plaintext
init \rightarrow [\{
  {?safeAccel; accel
  U brake};
  t:=0; {pos'=vel, vel'=acc}
}]* (pos < stopSign)
```
Closed-Loop Network Verification

\[
\text{init} \rightarrow [( \\
\quad u \ := \ NN(x) ; \\
\quad x \ := \ \text{plant}(x, u) ; )\star] \ \text{saf e}
\]
Reachability analysis of neural networks

\[ \text{NN}(x) = W_3 \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x)) \]

Reachability analysis of neural networks

$NN(x) = W_3 \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x))$

Open-Loop Neural Network Verification

Example: robustness verification

\[ \forall (x, y) \in D, \forall x', \|x - x'\|_\infty < \epsilon \rightarrow F(x') = y \]
Verification for AI
Can we train an AI to interact with a theorem prover?

```coq
Require Import List.
Require Import List.Notations.
Generalizable All Variables.

Fixpoint length `(l: list A) : nat :=
match l with
| [] => 0
| x::xs => 1 + length xs
end.

Lemma app_length:
forall {A:Type}(l1 l2: list A),
length(l1 ++ l2) = length l1 + length l2.
Proof.
  intro A; induction l1; intros l2.
  - trivial.
  - simpl. rewrite IHl1. reflexivity.
Qed.
```
LEARNING TO FIND PROOFS AND THEOREMS
BY LEARNING TO REFINE SEARCH STRATEGIES

Jonathan Laurent, André Platzer (NeurIPS 2022)
LOOP INVARIANT SYNTHESIS

- Training data unavailable and hard to generate!
- No pre-existing deep-learning agent capable of generalizing across instances.

To prove the final assertion, one must find a loop invariant that:

1. is true before the loop
2. is preserved by the loop body (when the loop guard holds)
3. implies the final assertion (when the loop guard does not hold)

Invariant: $x \geq y \land x \geq 1 \land y \geq 0$
A LANGUAGE FOR EXPRESSING STRATEGIES

We define a strategy language based on **choose** and **event** operators.

```python
def solver(
    init: Formula, guard: Formula,
    body: Program, post: Formula) -> Formula:
    def prove_inv(inv: Formula) -> List[Formula]:
        assert valid(Implies(init, inv))
        ind = Implies(And(guard, inv), wlp(body, inv))
        event(PROVE_INV_EVENT)
        match abduct(ind):
            case Valid:
                return [inv]
            case [*suggestions]:
                aux = choose(suggestions)
                return [inv] + prove_inv(aux)
        inv_cand = choose(abduct(Implies(Not(guard), post)))
        inv_conjuncts = prove_inv(inv_cand)
        return And(*inv_conjuncts)
```

▲ A solver strategy for invariant synthesis
GENERATING TRAINING PROBLEMS

• Generating interesting theorems is harder than proving those!

• Our approach: refining conditional generative strategies using RL.

```python
def teacher(rng: RandGen) -> Prog:
    cs = sample_constrs(rng)
    p = generate_prog(cs)
    p = transform(p, rng)
    p = hide_invariants(p)
    return p

def generate_prog(cs: Constrs):
    p = Prog("assume init;
    while (guard) {
        invariant inv_lin;
        invariant inv_aux;
        invariant inv_main;
        body;
    }"
    assert post;")

    p = refine_guard(p, cs)
    p = refine_inv(p, cs)
    p = refine_body(p, cs)
    assert valid(inv_preserved(p))
    p = refine_post(p, cs)
    assert valid(inv_post(p))
    p = refine_init(p, cs)
    assert valid(inv_init(p))
    penalize_violations(p, cs)
    return p

def transform(p: Prog, rng: RandGen):
    p = shuffle_formulas(p, rng)
    p = add_useless_init(p, rng)
    ...
    return p
```

▲ Outline of a teacher strategy for invariant synthesis
Large Language Models

and the future of programming
Language Models are Few-Shot Learners

Tom B. Brown*  Benjamin Mann*  Nick Ryder*  Melanie Subbiah*
Jared Kaplan†  Prafulla Dhariwal  Arvind Neelakantan  Pranav Shyam  Girish Sastry
Amanda Askell  Sandhini Agarwal  Ariel Herbert-Voss  Gretchen Krueger  Tom Henighan
Rewon Child  Aditya Ramesh  Daniel M. Ziegler  Jeffrey Wu  Clemens Winter
Christopher Hesse  Mark Chen  Eric Sigler  Mateusz Litwin  Scott Gray
Benjamin Chess  Jack Clark  Christopher Berner
Sam McCandlish  Alec Radford  Ilya Sutskever  Dario Amodei
Breakthrough: Large Language Models are Few Shot Learners

Fine-tuning
The model is trained via repeated gradient updates using a large corpus of example tasks.

1. sea otter => loutre de mer

gradient update

1. peppermint => menthe poivrée

gradient update

1. plush giraffe => girafe peluche

In-context learning

Zero-shot
The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

1. Translate English to French:
2. cheese => ........................................

Few-shot
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

1. Translate English to French:
2. sea otter => loutre de mer
3. peppermint => menthe poivrée
4. plush giraffe => girafe peluche
5. cheese => ........................................
Additional Slides
Minerva Examples
**Question:** Assume that the variance of the first $n$ natural numbers is 10, and the variance of the first $m$ even natural numbers is 16. Compute $m + n$.

**Model output:** Let $\sigma^2$ be the variance of the first $m$ even natural numbers, and $\tau^2$ be the variance of the first $n$ natural numbers. Then, we have $\tau^2 = \frac{n^2 - 1}{12}$ and $\sigma^2 = \frac{m^2 - 1}{3}$. We solve the system of equations:

\[
\frac{n^2 - 1}{12} = 10 \\
\frac{m^2 - 1}{3} = 16.
\]

This translates into the system

\[
\begin{align*}
    n^2 &= 121 \\
    m^2 &= 48 + 1 = 49.
\end{align*}
\]

Therefore, $n = \boxed{11}$ and $m = \boxed{7}$, so $n + m = \boxed{18}$.

**Question:** For every $a, b, b \neq a$ prove that

\[
\frac{a^2 + b^2}{2} > \left(\frac{a + b}{2}\right)^2.
\]

**Model output:**

\[
\frac{a^2 + b^2}{2} > \left(\frac{a + b}{2}\right)^2
\]

\[
\iff a^2 + b^2 > \frac{a^2 + b^2 + 2ab}{4}
\]

\[
\iff a^2 + b^2 > \frac{a^2 + b^2 + 2ab}{2}
\]

\[
\iff 2a^2 + 2b^2 > a^2 + b^2 + 2ab
\]

\[
\iff a^2 + b^2 > 2ab
\]

\[
\iff a^2 + b^2 - 2ab > 0
\]

\[
\iff (a - b)^2 > 0
\]

which is true, because the square of a real number is positive.
**Question:** A uniform solid disk starts from rest and rolls down an inclined plane without slipping. After some time, what fraction of the disk’s total kinetic energy is rotational kinetic energy? What of the following is the right choice? Explain your answer.

(A) $1/4$ (B) $1/3$ (C) $1/2$ (D) $2/3$

**Model output:** The disk has both translational and rotational kinetic energy. Let $v$ be the speed of the center of mass of the disk, $I$ be the moment of inertia of the disk, $M$ be the mass of the disk, and $R$ be the radius of the disk. The translational kinetic energy of the disk is given by

$$K_t = \frac{1}{2} M v^2.$$  

The rotational kinetic energy of the disk is given by

$$K_r = \frac{1}{2} I \omega^2 = \frac{1}{2} \left( \frac{1}{2} M R^2 \right) \left( \frac{v}{R} \right)^2 = \frac{1}{4} M v^2.$$  

Therefore, the ratio of the rotational kinetic energy to the total kinetic energy is

$$\frac{K_r}{K_t + K_r} = \frac{\frac{1}{4} M v^2}{\frac{1}{2} M v^2 + \frac{1}{4} M v^2} = \frac{\frac{1}{4} M v^2}{\frac{3}{4} M v^2} = \frac{1}{3}.$$  

Final Answer: The final answer is (B).
**Question:** A particle moves so that it is at \((3 \sin \frac{t}{4}, 3 \cos \frac{t}{4})\) at time \(t\). Find the speed of the particle, measured in unit of distance per unit of time.

**Target:** The parametric curve \((x, y) = (3 \sin \frac{t}{4}, 3 \cos \frac{t}{4})\) describes a circle with radius 3. Furthermore, it makes a full revolution at time \(t = 8\pi\). *(figure)* Therefore, the speed of the particle is \(\frac{8\pi}{8\pi} = \frac{3}{4}\).

**Model output:** The speed is the magnitude of the velocity vector. Therefore, we must first find the velocity vector, which is the derivative of the position vector. We have

\[
v = \frac{d}{dt} \left(3 \sin \frac{t}{4}, 3 \cos \frac{t}{4}\right) = 3 \left(\cos \frac{t}{4}, -\sin \frac{t}{4}\right) \frac{1}{4} = \left(\frac{3}{4} \cos \frac{t}{4}, -\frac{3}{4} \sin \frac{t}{4}\right).
\]

Therefore, the magnitude of the velocity vector is

\[
|v| = \sqrt{\left(\frac{3}{4} \cos \frac{t}{4}\right)^2 + \left(-\frac{3}{4} \sin \frac{t}{4}\right)^2} = \sqrt{\frac{9}{16} \cos^2 \frac{t}{4} + \frac{9}{16} \sin^2 \frac{t}{4}} = \frac{3}{4} \sqrt{\cos^2 \frac{t}{4} + \sin^2 \frac{t}{4}} = \frac{3}{4}.
\]

Therefore, the speed of the particle is \(\frac{3}{4}\).
ChatGPT Examples
I am preparing a lecture on using formal methods to verify safety-critical systems that rely on neural networks. What should I cover?

When preparing a lecture on using formal methods to verify safety-critical systems that rely on neural networks, you could cover the following topics:

1. Introduction to neural networks and their applications in safety-critical systems, such as autonomous vehicles and medical devices.
2. Overview of formal methods and how they can be used to ensure the safety and reliability of systems that rely on neural networks.
3. Techniques for formally verifying the properties of neural networks, such as model checking and theorem proving.
4. Methods for testing neural networks, such as test case generation and testing with formal specifications.
5. Challenges and limitations of using formal methods to verify safety-critical systems that rely on neural networks, such as scalability and interpretability of the resulting models.
6. Applications of formal methods to real-world safety-critical systems that rely on neural networks and case studies.
7. Future directions in the use of formal methods to verify safety-critical systems that rely on neural networks, such as the integration of machine learning and formal methods.
8. Tools and frameworks available for formal verification of neural networks, such as DeepSafe, DeepPoly, DeepSpec, etc.

Note: As the field is evolving rapidly, it is recommended to keep yourself updated with the recent developments and trends in the field.
Can you write me an OCaml function that computes the factorial of a number?

Yes, here is an OCaml function that computes the factorial of a given number:

```ocaml
let rec factorial n =  
  if n = 0 then 1  
  else n * factorial (n - 1)
```

You can use it by calling the function with the number you want to compute the factorial of, for example:

```ocaml
print_int (factorial 5);;
```

This will output the value 120, which is the factorial of 5.
What will happen if I pass a negative number to this function?

The function as written will enter into an infinite recursion if a negative number is passed as an argument. The reason for this is that the `n - 1` in the recursive call of `factorial` will never reach 0 and the function will keep calling itself indefinitely.

It is better to add an additional check inside the function to handle negative numbers or non-integer values. One way to do this is to check if the argument is less than or equal to 0 and return 1 if it is.

```ocaml
let rec factorial n =
  if n <= 0 then 1
  else n * factorial (n - 1)
```

By doing this, if you pass negative number or zero to the function, it will return 1 instead of entering into an infinite recursion.
Can you add 201 and 102 in base 3?

Yes, in base 3 the sum would be 303.
Let me teach you how to add numbers in base 3.

For example, let’s compute $R=A+B$ with $A=[1, 0, 1]$ and $B=[2, 1, 1]$.

Initially, carry=0.

$R[0] = (A[0] + B[0] + \text{carry}) \mod 3 = (1 + 2 + 0) \mod 3 = 0.$
Also, $A[0] + B[0] = 3 >= 3$ and so we set carry = 1.

Also, $A[1] + B[1] + \text{carry} = 2 < 3$ and so we set carry=0.

Finally,

All elements of A and B are exhausted and so the answer is $R=[0, 2, 2]$.

Now, can you try and compute $R=A+B$ in base 3 with $A=[1, 0, 1]$ and $B=[2, 0, 1]$?
Sure, here is how I would compute $R=\mathbf{A}+\mathbf{B}$ in base 3 with $\mathbf{A}=[1, 0, 1]$ and $\mathbf{B}=[2, 0, 1]$:

Initially, carry=0.

$R[0] = (A[0] + B[0] + \text{carry}) \mod 3 = (1 + 2 + 0) \mod 3 = 0.$
Also, $A[0] + B[0] = 3 \geq 3$ and so we set carry = 1.

Also, $A[1] + B[1] + \text{carry} = 1 < 3$ and so we set carry=0.

Finally,


All elements of $\mathbf{A}$ and $\mathbf{B}$ are exhausted and so the answer is $R=[0, 1, 2]$. 

What about \([0, 0, 1] + [0, 0, 2]\)?

To compute \(R=A+B\) in base 3 with \(A=[0, 0, 1]\) and \(B=[0, 0, 2]\), we can follow the steps outlined above:

Initially, carry=0.

\[
R[0] = (A[0] + B[0] + \text{carry}) \mod 3 = (0 + 0 + 0) \mod 3 = 0.
\]
Also, \(A[0] + B[0] = 0 < 3\) and so we set \(\text{carry} = 0\).

\[
R[1] = (A[1] + B[1] + \text{carry}) \mod 3 = (0 + 0 + 0) \mod 3 = 0.
\]
Also, \(A[1] + B[1] + \text{carry} = 0 < 3\) and so we set \(\text{carry} = 0\).

Finally,

\[
\]
Also, \(A[2] + B[2] + \text{carry} = 3 \geq 3\) so we set \(\text{carry} = 1\).

All elements of \(A\) and \(B\) are exhausted, but we have a non-zero carry, so we append it to the result. The final answer is \(R=[0, 0, 0, 1]\).
Can you write a python function describing an algorithm to add numbers in base 3?

Sure, here is a Python function that describes an algorithm to add two numbers in base 3:

```python
def add_base3(A, B):
    carry = 0
    R = []
    for i in range(max(len(A), len(B))):
        a = A[i] if i < len(A) else 0
        b = B[i] if i < len(B) else 0
        r = (a + b + carry) % 3
        carry = 1 if a + b + carry >= 3 else 0
        R.append(r)
    if carry:
        R.append(carry)
    return R
```

Here's how it works:

1. It initializes a variable `carry` to 0. This variable will be used to keep track of any "carry" from one digit to the next.
2. It initializes an empty list `R` to store the result.
3. It uses a for loop to iterate over the digits of the numbers, starting with the least significant digit (at index 0) and going to the most significant digit.