

Large Language Models for AI Planning

Efficiency and correctness
through code generation

Michael Katz
Principal Research Scientist
IBM Research



It's a Team Effort!



Michael Katz



Harsha Kokel



Daniel Cao



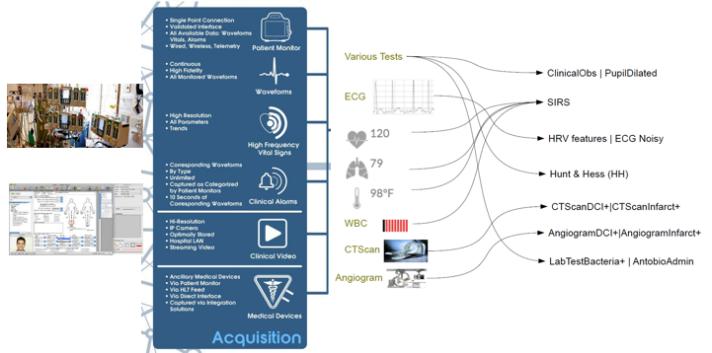
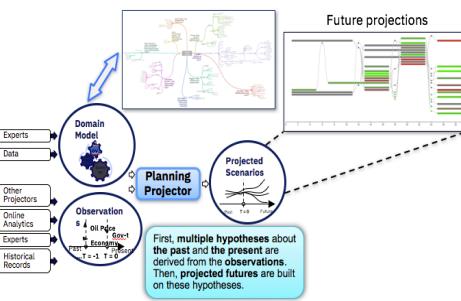
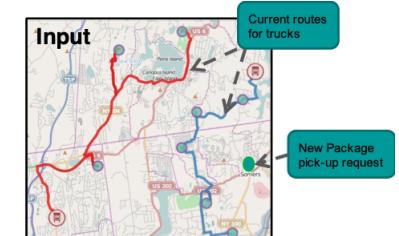
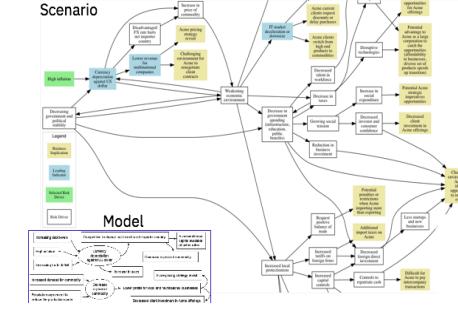
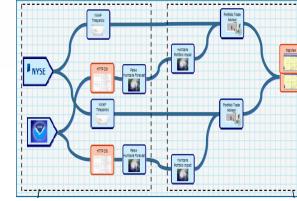
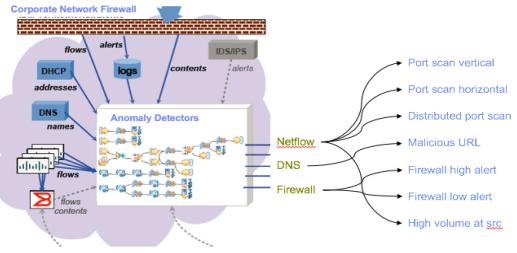
Kavitha Srinivas



Shirin Sohrabi

What is AI Planning

A subfield of Artificial Intelligence focused on **automating the decision-making process**.



Planning problem

Planning problem

Mathematical Model:

Planning problem

Mathematical Model:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- non-negative action costs $c(a, s)$

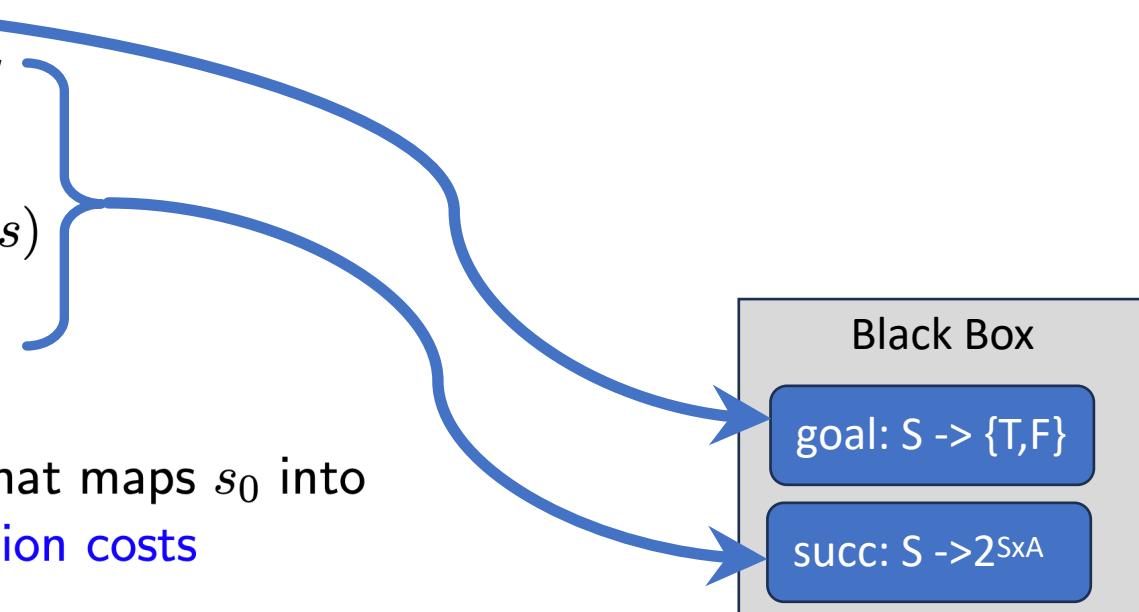
A solution is a sequence of applicable actions that maps s_0 into S_G , and it is optimal if it minimizes sum of action costs

Planning problem

Mathematical Model:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- non-negative action costs $c(a, s)$

A solution is a sequence of applicable actions that maps s_0 into S_G , and it is optimal if it minimizes sum of action costs

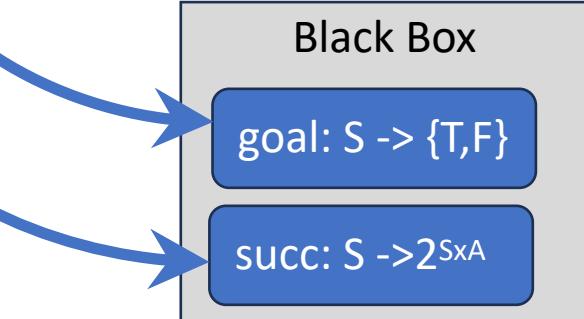
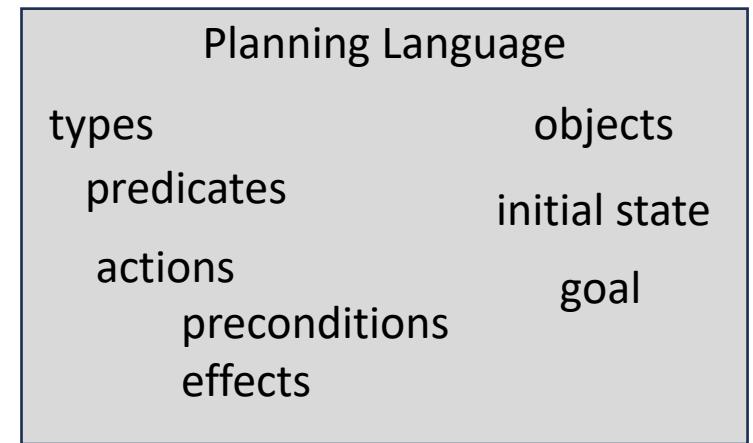


Planning problem

Mathematical Model:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- non-negative action costs $c(a, s)$

A solution is a sequence of applicable actions that maps s_0 into S_G , and it is optimal if it minimizes sum of action costs



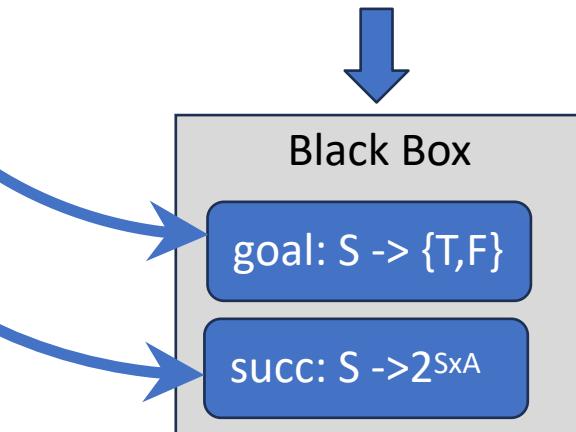
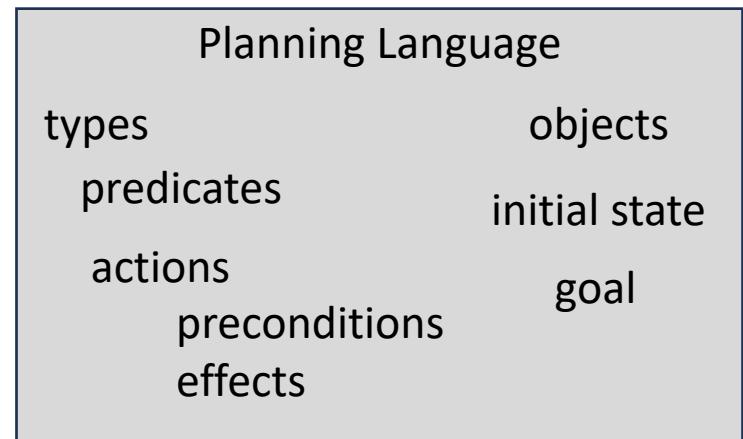
Planning problem

The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24.

Mathematical Model:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- non-negative action costs $c(a, s)$

A solution is a sequence of applicable actions that maps s_0 into S_G , and it is optimal if it minimizes sum of action costs



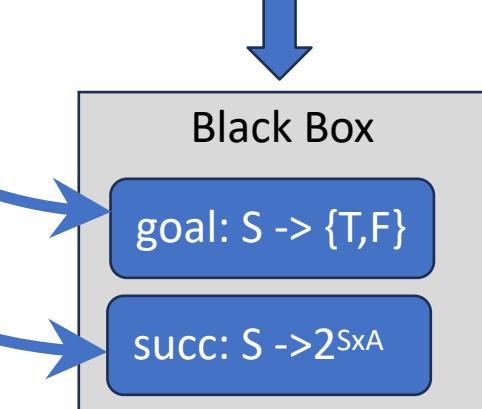
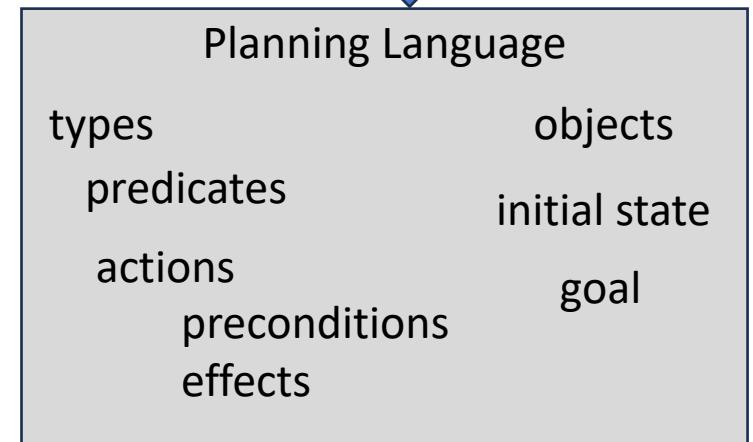
Planning problem

The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24.

Mathematical Model:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- non-negative action costs $c(a, s)$

A solution is a sequence of applicable actions that maps s_0 into S_G , and it is optimal if it minimizes sum of action costs



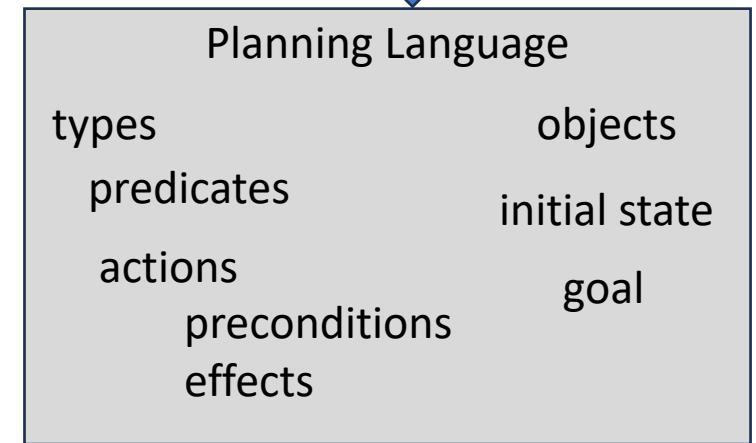
Planning problem

The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24.

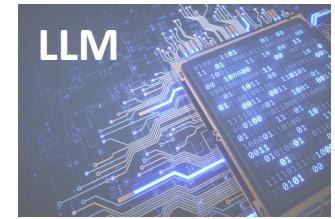
Mathematical Model:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- non-negative action costs $c(a, s)$

A solution is a sequence of applicable actions that maps s_0 into S_G , and it is optimal if it minimizes sum of action costs



Recent Trends in Planning from text with LLMs

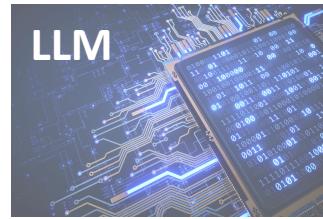


Recent Trends in Planning from text with LLMs

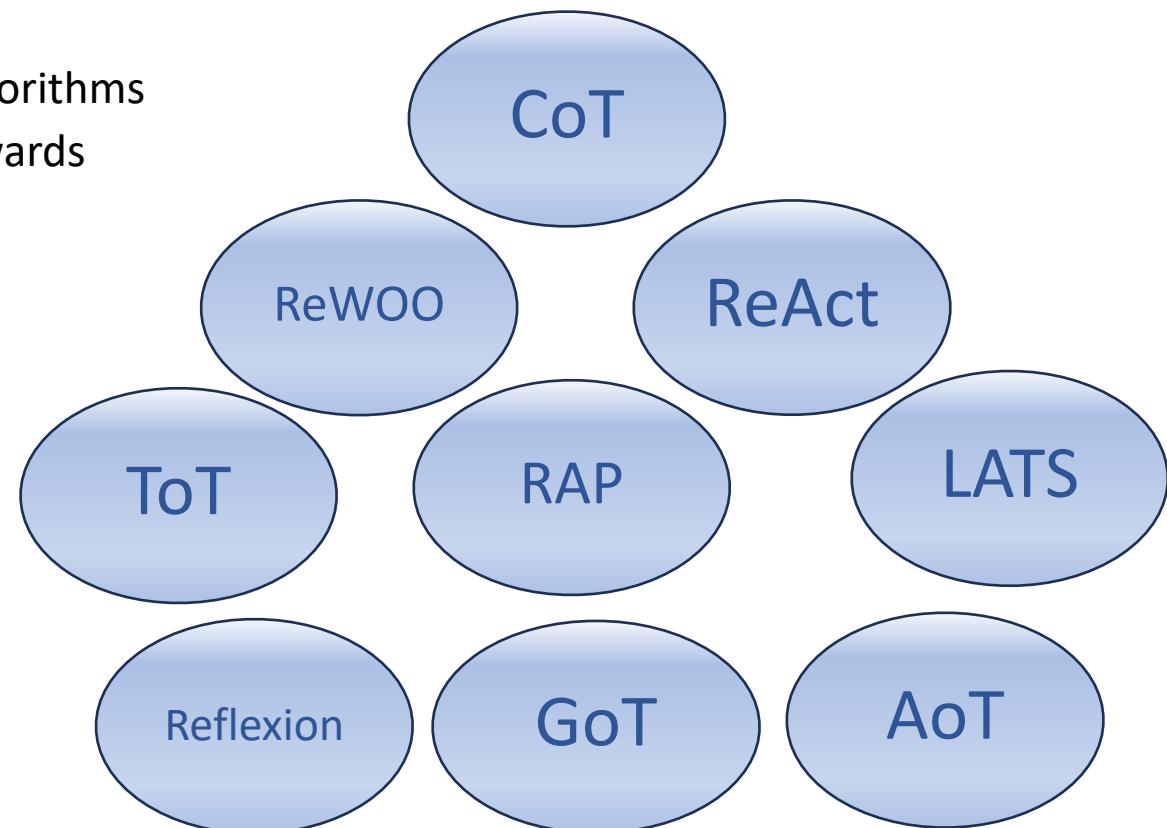


- Generate the entire plan at once

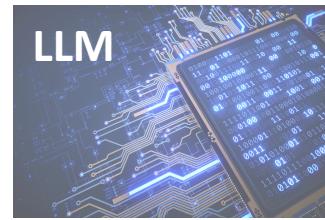
Recent Trends in Planning from text with LLMs



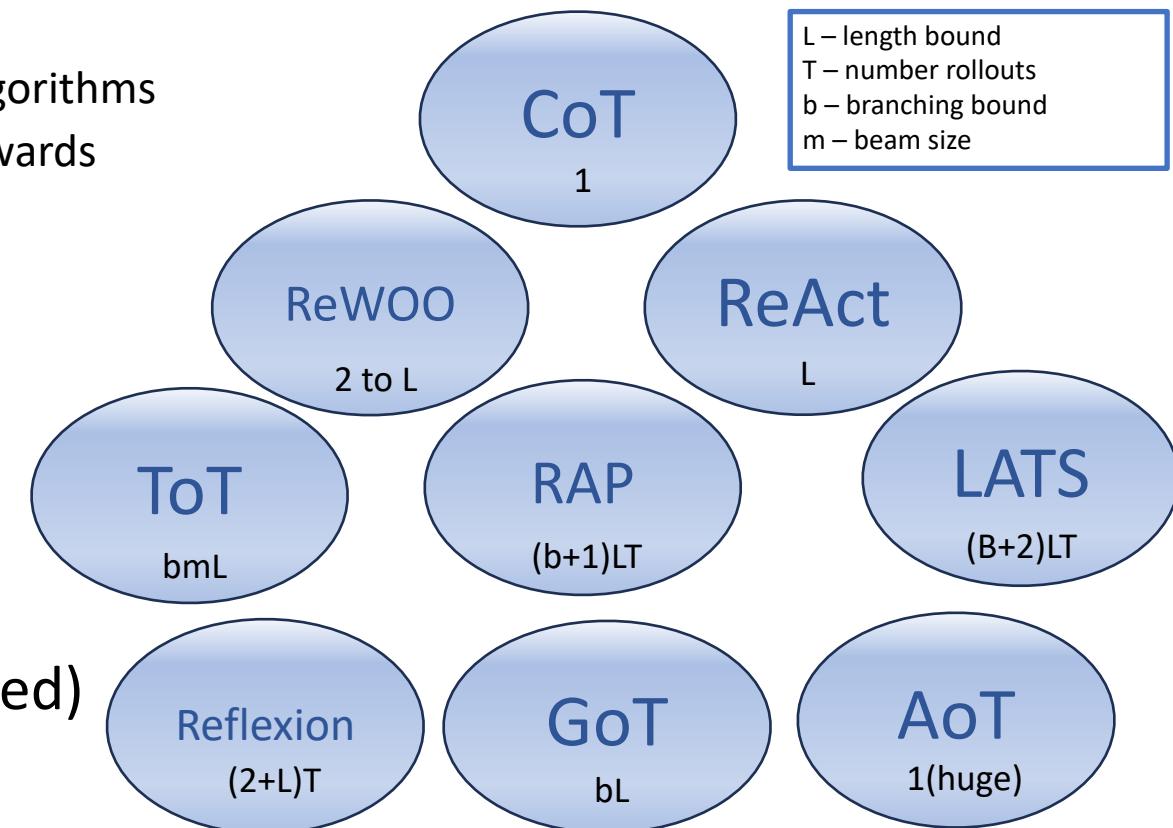
- Generate the entire plan at once
- Generate one action at a time (explore some portion of the state space)
 - Generate (a few) paths (CoT/ReAct/ReWOO)
 - Perform (limited) search, with the help of existing search algorithms
 - Use LLMs for generating successors, goal test, heuristics/rewards
 - Use LLMs for generating search spaces
 - Restrict the algorithms to limit the number of calls to LLMs



Recent Trends in Planning from text with LLMs



- Generate the entire plan at once
- Generate one action at a time (explore some portion of the state space)
 - Generate (a few) paths (CoT/ReAct/ReWOO)
 - Perform (limited) search, with the help of existing search algorithms
 - Use LLMs for generating successors, goal test, heuristics/rewards
 - Use LLMs for generating search spaces
 - Restrict the algorithms to limit the number of calls to LLMs
- No guarantees of soundness or completeness
- Limited solution validation
- Reported success rates in 70-80% (not reproduced)
- Not clear how the success is measured
- No computational complexity investigation (in terms of LLM calls)



Example:

24 game

Example:

24 game

- Rules of the game:
 - Input: list of 4 integers
 - Output: arithmetic expression that involves all 4 numbers and equals 24.

Example:

24 game

- Rules of the game:
 - Input: list of 4 integers
 - Output: arithmetic expression that involves all 4 numbers and equals 24.
- Relatively small search space: 4573 states.

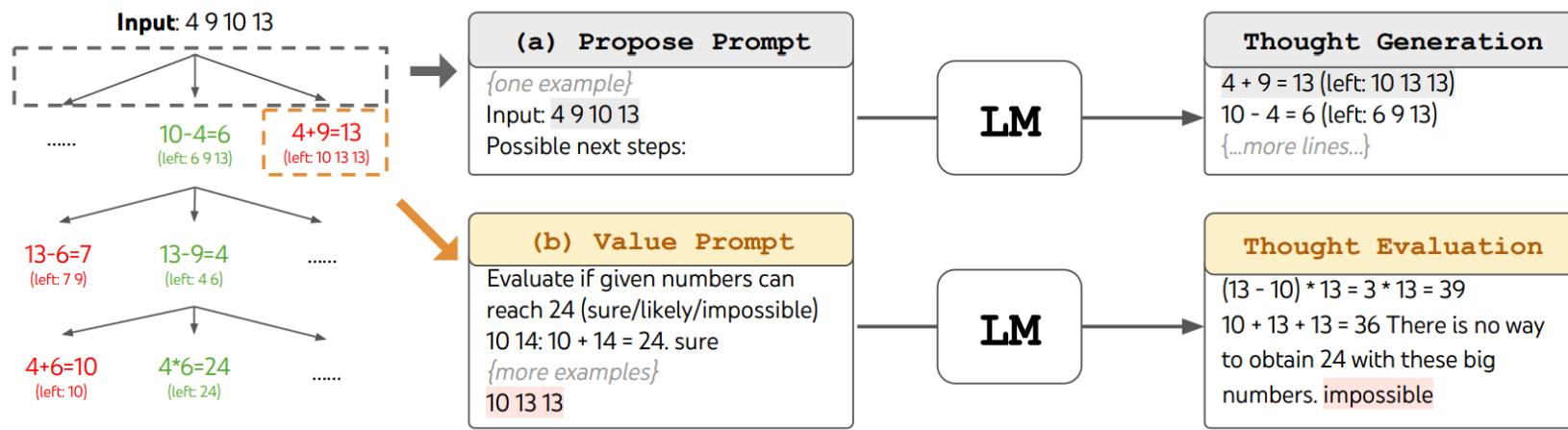
Example:

24 game

- Rules of the game:
 - Input: list of 4 integers
 - Output: arithmetic expression that involves all 4 numbers and equals 24.
- Relatively small search space: 4573 states.
- ToT: perform search, get successors by calling LLM, check dead-ends by calling LLM.

Example: 24 game

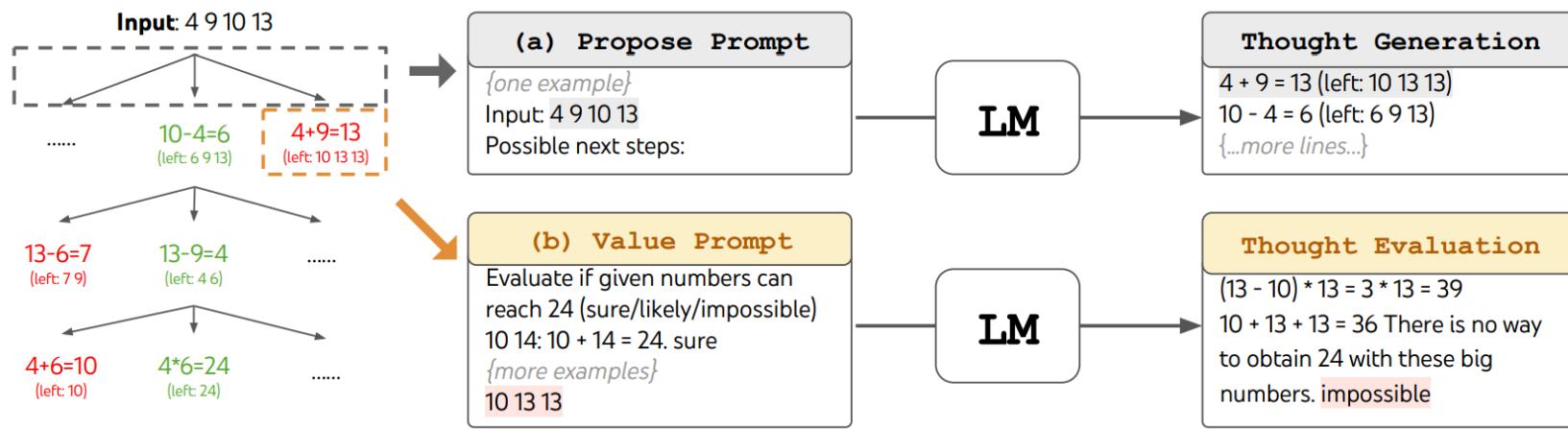
- Rules of the game:
 - Input: list of 4 integers
 - Output: arithmetic expression that involves all 4 numbers and equals 24.
- Relatively small search space: 4573 states.
- ToT: perform search, get successors by calling LLM, check dead-ends by calling LLM.



From: Yao et. al., NeurIPS 2023, Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Example: 24 game

- Rules of the game:
 - Input: list of 4 integers
 - Output: arithmetic expression that involves all 4 numbers and equals 24.
- Relatively small search space: 4573 states.
- ToT: perform search, get successors by calling LLM, check dead-ends by calling LLM.



From: Yao et. al., NeurIPS 2023, Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Algorithm 1 ToT-BFS($x, p_\theta, G, k, V, T, b$)

Require: Input x , LM p_θ , thought generator $G()$ & size limit k , states evaluator $V()$, step limit T , breadth limit b .

```

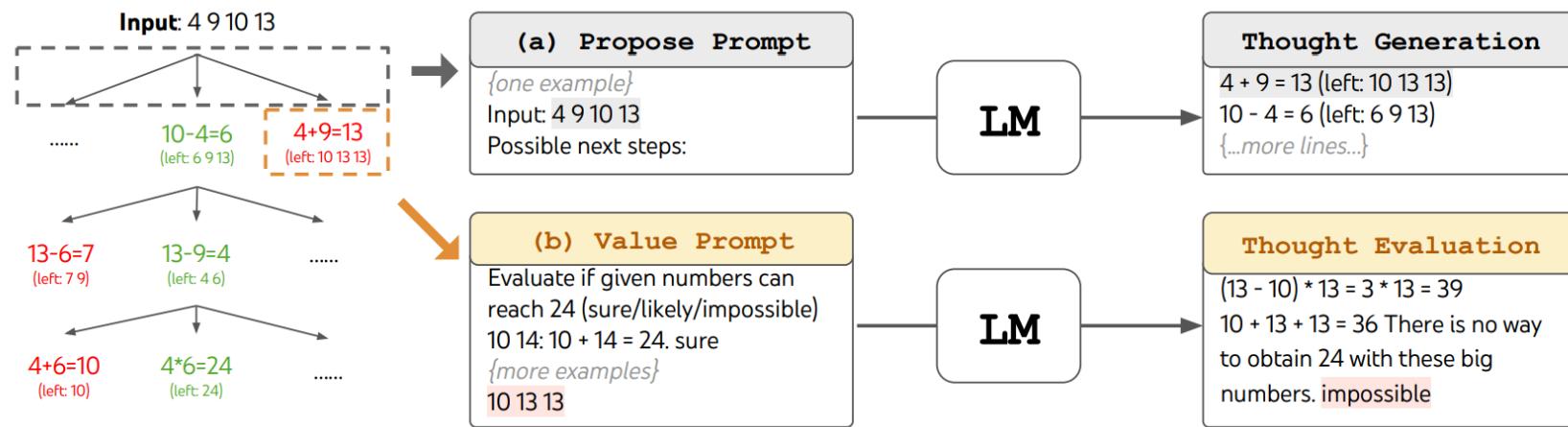
 $S_0 \leftarrow \{x\}$ 
for  $t = 1, \dots, T$  do
     $S'_t \leftarrow \{[s, z] \mid s \in S_{t-1}, z_t \in G(p_\theta, s, k)\}$ 
     $V_t \leftarrow V(p_\theta, S'_t)$ 
     $S_t \leftarrow \arg \max_{S \subset S'_t, |S|=b} \sum_{s \in S} V_t(s)$ 
end for
return  $G(p_\theta, \arg \max_{s \in S_T} V_T(s), 1)$ 

```

From: Yao et. al., NeurIPS 2023

Example: 24 game

- Rules of the game:
 - Input: list of 4 integers
 - Output: arithmetic expression that involves all 4 numbers and equals 24.
- Relatively small search space: 4573 states.
- ToT: perform search, get successors by calling LLM, check dead-ends by calling LLM.
- Shows empirically that it works better than CoT (higher success rate)
- 74% accuracy reported
- They also report 30.8% solved rate on the only unsolvable task [validation issues]



From: Yao et. al., NeurIPS 2023, Tree of Thoughts: Deliberate Problem Solving with Large Language Models

From: Yao et. al., NeurIPS 2023, Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Algorithm 1 ToT-BFS($x, p_\theta, G, k, V, T, b$)

Require: Input x , LM p_θ , thought generator $G()$ & size limit k , states evaluator $V()$, step limit T , breadth limit b .

```

 $S_0 \leftarrow \{x\}$ 
for  $t = 1, \dots, T$  do
     $S'_t \leftarrow \{[s, z] \mid s \in S_{t-1}, z_t \in G(p_\theta, s, k)\}$ 
     $V_t \leftarrow V(p_\theta, S'_t)$ 
     $S_t \leftarrow \arg \max_{S \subset S'_t, |S|=b} \sum_{s \in S} V_t(s)$ 
end for
return  $G(p_\theta, \arg \max_{s \in S_T} V_T(s), 1)$ 

```

From: Yao et. al., NeurIPS 2023

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

- 1.2 queries to LLM on average to obtain a valid successor function

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

- 1.2 queries to LLM on average to obtain a valid successor function
- Code can be validated/tested

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

- 1.2 queries to LLM on average to obtain a valid successor function
- Code can be validated/tested
- Any search algorithm can be used: sound and complete algorithm + correct successor function = guaranteed solution correctness

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

- 1.2 queries to LLM on average to obtain a valid successor function
- Code can be validated/tested
- Any search algorithm can be used: sound and complete algorithm + correct successor function = guaranteed solution correctness
- Fast: all the 1362 games in the suite were solved in under 2s

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Thought of Search (ToS)

(K. et. al., NeurIPS 2024)

Black-box planning

- 1.2 queries to LLM on average to obtain a valid successor function
- Code can be validated/tested
- Any search algorithm can be used: sound and complete algorithm + correct successor function = guaranteed solution correctness
- Fast: all the 1362 games in the suite were solved in under 2s
- Deployment of the solution does not require reliance on the particular language model / version /stability.

User: The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

ChatGPT: Below is the Python code that defines how to generate the successor states for a given state in the 24 Game. A state in the 24 Game is a list of numbers. The successor states are generated by taking two numbers from the state, applying an operation (addition, subtraction, multiplication, or division if the divisor isn't zero), and then forming a new list with the result and the remaining unused numbers.

```
import itertools

def generate_successor_states(state):
    successors = []
    n = len(state)
    for (i, j) in itertools.combinations(range(n), 2):
        a, b = state[i], state[j]
        remaining = [state[k] for k in range(n) if k != i and k != j]

        # Generate all possible new states
        successors.append(remaining + [a + b])
        successors.append(remaining + [a * b])
        if a > b:
            successors.append(remaining + [a - b])
        else:
            successors.append(remaining + [b - a])
        if b != 0:
            successors.append(remaining + [a / b])
        if a != 0:
            successors.append(remaining + [b / a])

    return successors
```

This function takes a state (a list of numbers), iterates over all pairs of numbers, applies the operations to these pairs, and then forms new lists with the result of the operation and the other numbers that were not used in the operation. This ensures that each successor state is a valid step towards finding a solution to reach 24, using all four numbers exactly once along with any of the allowed operations.

User: Provide only the python code that tests whether a state is a goal state.

ChatGPT: Here's the Python code that tests whether a given state is a goal state in the 24 Game. A goal state is achieved if the list contains exactly one number, and that number is 24.

```
def is_goal_state(state):
    return len(state) == 1 and state[0] == 24
```

This function checks if the length of the state list is 1 and whether the sole element in the list is the number 24.

Search

```
def bfs(state, successor_states, is_goal, state_representation):
    expansions, evaluations = 0, 0
    s = state
    Q = [tuple((s, None))]
    Closed = dict()
    while len(Q) > 0:
        s, parent = Q[0][0], Q[0][1]
        del Q[0]
        c = state_representation(s)
        if c in Closed:
            continue
        Closed[c] = parent
        if is_goal(s):
            plan = reconstruct_plan(s, Closed, state_representation)
            return plan, expansions, evaluations
        expansions += 1
        for t in successor_states(s):
            Q.append(tuple((t, s)))
            evaluations += 1
    return None, expansions, evaluations
```

Search

Successor
function

```
def bfs(state, successor_states, is_goal, state_representation):
    expansions, evaluations = 0, 0
    s = state
    Q = [tuple((s, None))]
    Closed = dict()
    while len(Q) > 0:
        s, parent = Q[0][0], Q[0][1]
        del Q[0]
        c = state_representation(s)
        if c in Closed:
            continue
        Closed[c] = parent
        if is_goal(s):
            plan = reconstruct_plan(s, Closed, state_representation)
            return plan, expansions, evaluations
        expansions += 1
        for t in successor_states(s):
            Q.append(tuple((t, s)))
            evaluations += 1
    return None, expansions, evaluations
```

Search

Successor
function

Goal test
function

```
def bfs(state, successor_states, is_goal, state_representation):
    expansions, evaluations = 0, 0
    s = state
    Q = [tuple((s, None))]
    Closed = dict()
    while len(Q) > 0:
        s, parent = Q[0][0], Q[0][1]
        del Q[0]
        c = state_representation(s)
        if c in Closed:
            continue
        Closed[c] = parent
        if is_goal(s):
            plan = reconstruct_plan(s, Closed, state_representation)
            return plan, expansions, evaluations
        expansions += 1
        for t in successor_states(s):
            Q.append(tuple((t, s)))
            evaluations += 1
    return None, expansions, evaluations
```

Search

Successor
function

Goal test
function

State
representation
(stringify)

```
def bfs(state, successor_states, is_goal, state_representation):
    expansions, evaluations = 0, 0
    s = state
    Q = [tuple((s, None))]
    Closed = dict()
    while len(Q) > 0:
        s, parent = Q[0][0], Q[0][1]
        del Q[0]
        c = state_representation(s)
        if c in Closed:
            continue
        Closed[c] = parent
        if is_goal(s):
            plan = reconstruct_plan(s, Closed, state_representation)
            return plan, expansions, evaluations
        expansions += 1
        for t in successor_states(s):
            Q.append(tuple((t, s)))
            evaluations += 1
    return None, expansions, evaluations
```

Search

```
def bfs(state, successor_states, is_goal, state_representation):
    expansions, evaluations = 0, 0
    s = state
    Q = [tuple((s, None))]
    Closed = dict()
    while len(Q) > 0:
        s, parent = Q[0][0], Q[0][1]
        del Q[0]
        c = state_representation(s)
        if c in Closed:
            continue
        Closed[c] = parent
        if is_goal(s):
            plan = reconstruct_plan(s, Closed, state_representation)
            return plan, expansions, evaluations
        expansions += 1
        for t in successor_states(s):
            Q.append(tuple((t, s)))
            evaluations += 1
    return None, expansions, evaluations
```

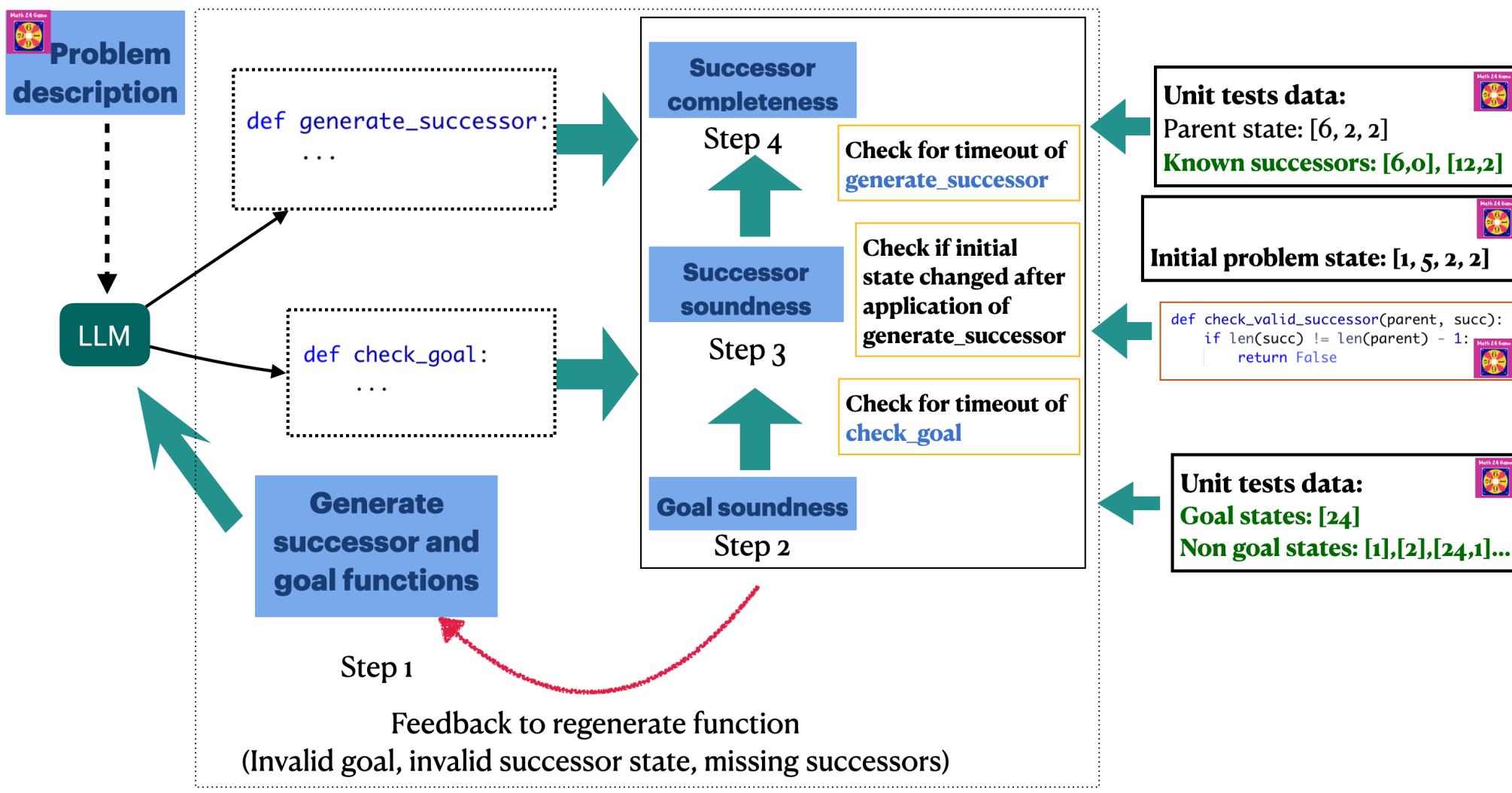
Experiments

Approach	Complexity	24Game		Crossword		BlocksWorld		PrOntoQA	
		States	Calls	States	Calls	States	Calls	States	Calls
IO	$O(D)$	0.02%	1362	4e-9%	20	0.5%	502	4%	4000
CoT	$O(D)$	0.02%	1362	4e-9%	20	0.5%	502	4%	4000
ReAct	$O(LD)$	0.07%	4086	4e-8%	200	7.8%	8032	24.6%	24K
ReWOO	$O(LD)$	0.07%	4086	4e-8%	200	7.8%	8032	24.6%	24K
RAP	$O(TbLD)$	3.3%	245K	2e-6%	12K	388%	482K	1229%	1.44M
ToT	$O(bmLD)$	1.6%	102K	1e-6%	5K	194%	201K	615%	600K
GoT	$O(bLD)$	0.3%	20K	2e-7%	1K	39%	40K	122%	120K
Reflection	$O(LTD)$	0.7%	68K	4e-7%	2.4K	77.6%	90K	245%	320K
LATS	$O(TbLD)$	3.3%	286K	2e-6%	14K	388%	562K	1229%	1.68M
ToS (ours)	$O(1)$	27.0%	2.2	3e-4%	3.8	125%	3.8	175%	2.6

Table 1: Comparison of various approaches on evaluated tasks. The number of tasks is D , the length bound of the search/rollout/plan is L , number of rollouts is T , beam size is m , and branching bound is b . The summed number of states over the D tasks and the projected number of LLM evaluations are given per approach and problem.

- 100% instances solved
- Small number of iterations with frontier language models

Automating Thought of Search (AutoToS): A Journey Towards Soundness and Completeness



Human involvement

Goal states:

[24]

Non-goal states:

[], [3], [24, 1], [1, 6, 4], [1, 1, 4, 6]

Successors:

`[[1, 1, 1, 8], [[0.125, 1, 1], [1, 1, 9], [1, 1, 8], [0, 1, 8], [1, 2, 8], [1, 1, 7], [-7, 1, 1]]]
[[6, 6, 6, 6], [[1.0, 6, 6], [6, 6, 12], [0, 6, 6], [6, 6, 36]]]`

Partial
soundness
test:

```
def validate_transition_complex(self, s, t):
    if len(s) - len(t) != 1:
        feedback = search pprint("Invalid transformation: length mismatch – the length of a successor must be
one less than the parent.")
        feedback += search pprint("Let's think step by step. First think through in words why the successor
function produced a successor that had a length that was not exactly one less than the parent. Then
provide the complete Python code for the revised successor function that ensures the length of a
successor is exactly one less than the parent.")
        feedback += search pprint("Remember how you fixed the previous mistakes, if any. Keep the same
function signature.")
    return False, feedback
return True, ""
```

Search

```
def bfs(state, successor_states, is_goal, state_representation, verify_successor, perform_checks = True, timeout=600):
    start_time = time.time()
    expansions, evaluations = 0, 0
    error_detected = False ; feedback = "" ; Closed = dict()
    Q = [tuple((state, None))]
    while len(Q) > 0:
        if perform_checks and timeout and time.time() - start_time > timeout:
            feedback += pprint("Timeout occurred during search with the successor function.\nPlease provide a different implementation of the successor function.\nKeep the same function signature."); break
        s, parent = Q[0][0], Q[0][1] ; del Q[0]
        c = state_representation(s)
        if c in Closed: continue
        Closed[c] = parent
        try: _is_goal = is_goal(s)
        except KeyboardInterrupt:
            feedback += pprint("Goal test function execution took too long.\nPlease provide a different implementation of the goal test function.\nKeep the same function signature."); break
        if _is_goal: return reconstruct_plan(s, Closed, state_representation), expansions, evaluations, feedback
        expansions += 1
        if perform_checks: before_succ_run = copy.deepcopy(s)
        try: successors = successor_states(s)
        except KeyboardInterrupt:
            feedback += pprint("Successor function execution took too long.\nPlease provide a different implementation of the successor function.\nKeep the same function signature."); break
        if perform_checks and state_representation(s) != state_representation(before_succ_run):
            feedback += pprint("Input state should not change as a result of applying the successor function.\nState before successor function applied:", before_succ_run,
                               "\nState after successor function applied:", s, "\nLet's think step by step. First, think of the possible reasons why the input state may change as a result of applying the successor function, such as the use of shallow copy.\nThen, provide the complete Python code for the revised successor function that returns a list of successor states and ensure it does not change the input state.\nRemember how you fixed the previous mistakes, if any. Keep the same function signature."); break
        for t in successors:
            if perform_checks:
                res, succ_feedback = verify_successor(s, t)
                if not res:
                    feedback += pprint(succ_feedback, "\nInput state:", s, "\nExample wrong successor state:", t)
                    error_detected = True ; break
                Q.append(tuple((t, s)))
                evaluations += 1
            if error_detected: break
    return None, expansions, evaluations, feedback
```

Making LLMs produce usable search components code

System prompt

You are a Python coding assistant. Help me generate my Python functions based on the task descriptions. Please always generate only a single function and keep all imports in it. If you need to define any additional functions, define them as inner functions. Do not generate examples of how to invoke the function. Please do not add any print statements outside the function. Provide the complete function and do not include any ellipsis notation.

Step 1: Initial prompt

The 24 Game is a mathematical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. The game is played with a list of four numbers, and the player must use all four numbers exactly once, using any combination of addition, subtraction, multiplication, or division, to arrive at the number 24. If a state is defined by a list of numbers (4 or less), how are successor states defined? Provide only the python code that returns a list of successor states for a state.

Provide only the python code that tests whether a state is a goal state.

Making LLMs produce usable search components code

Step 2: Goal function check

The goal test function failed on the following input state [24, 1], incorrectly reporting it as a goal state. First think step by step what it means for a state to be a goal state in this domain. Then think through in words why the goal test function incorrectly reported input state: [24, 1] as a goal state. Now, revise the goal test function and ensure it returns false for the input state. Remember how you fixed the previous mistakes, if any. Keep the same function signature.

Step 3: Successor function soundness check

Invalid transformation: length mismatch - the length of a successor must be one less than the parent. Let's think step by step. First think through in words why the successor function produced a successor that had a length that was not exactly one less than the parent. Then provide the complete Python code for the revised successor function that ensures the length of a successor is exactly one less than the parent. Remember how you fixed the previous mistakes, if any. Keep the same function signature.

Input state: [1, 1, 4, 6] Example wrong successor state: [6, 5]

Step 4: Successor function completeness check

Successor function when run on the state [1, 1, 4, 6] failed to produce all successors. Missing successors are: [[1, 4, 7], [-5, 1, 4], [1, 1, 2], [1, 5, 6], [0.25, 1, 6], [-3, 1, 6], [0.1666666666666666, 1, 4], [1, 3, 6], [1, 4, 5], [1, 1, 1.5]] First think step by step why the successor function failed to produce all successors of the state. Then, fix the successor function. Remember how you fixed the previous mistakes, if any. Keep the same function signature.

Experiments

		24 Game	PrOntoQA	Sokoban	Crossword	BlocksWorld
AutoToS	GPT-4o-mini	8.8	4.8	6.4	9.6	10.0
	GPT-4o	3.4	2.6	2.2	5.8	2.0
	Llama3.1-405b	3.4	2.0	2.6	4.0	3.2
	Llama3.1-70b	7.4	2.0	8.2	6.2	5.8
	DeepSeek-CoderV2	4.4	2.0	2.8	6.6	4.2
	ToS GPT-4	2.2	2.6	NA	3.8	3.8

Table 1: The average number of calls to the language model per domain.

Experiments

	24 Game	PrOntoQA	Sokoban	Crossword	BlocksWorld
AutoToS	GPT-4o-mini	8.8	4.8	6.4	9.6
	GPT-4o	3.4	2.6	2.2	5.8
	Llama3.1-405b	3.4	2.0	2.6	4.0
	Llama3.1-70b	7.4	2.0	8.2	6.2
	DeepSeek-CoderV2	4.4	2.0	2.8	6.6
ToS GPT-4	2.2	2.6	NA	3.8	3.8

Table 1: The average number of calls to the language model per domain.

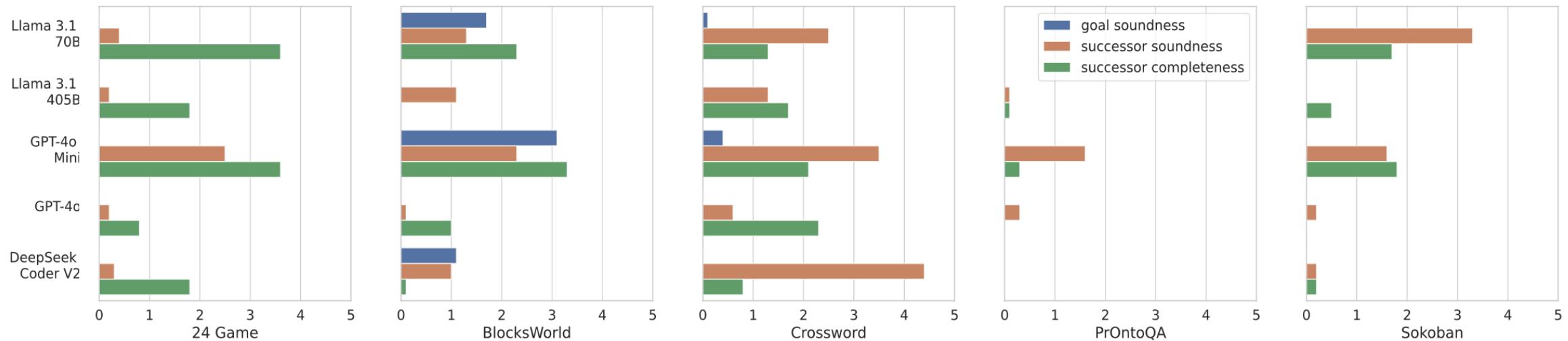
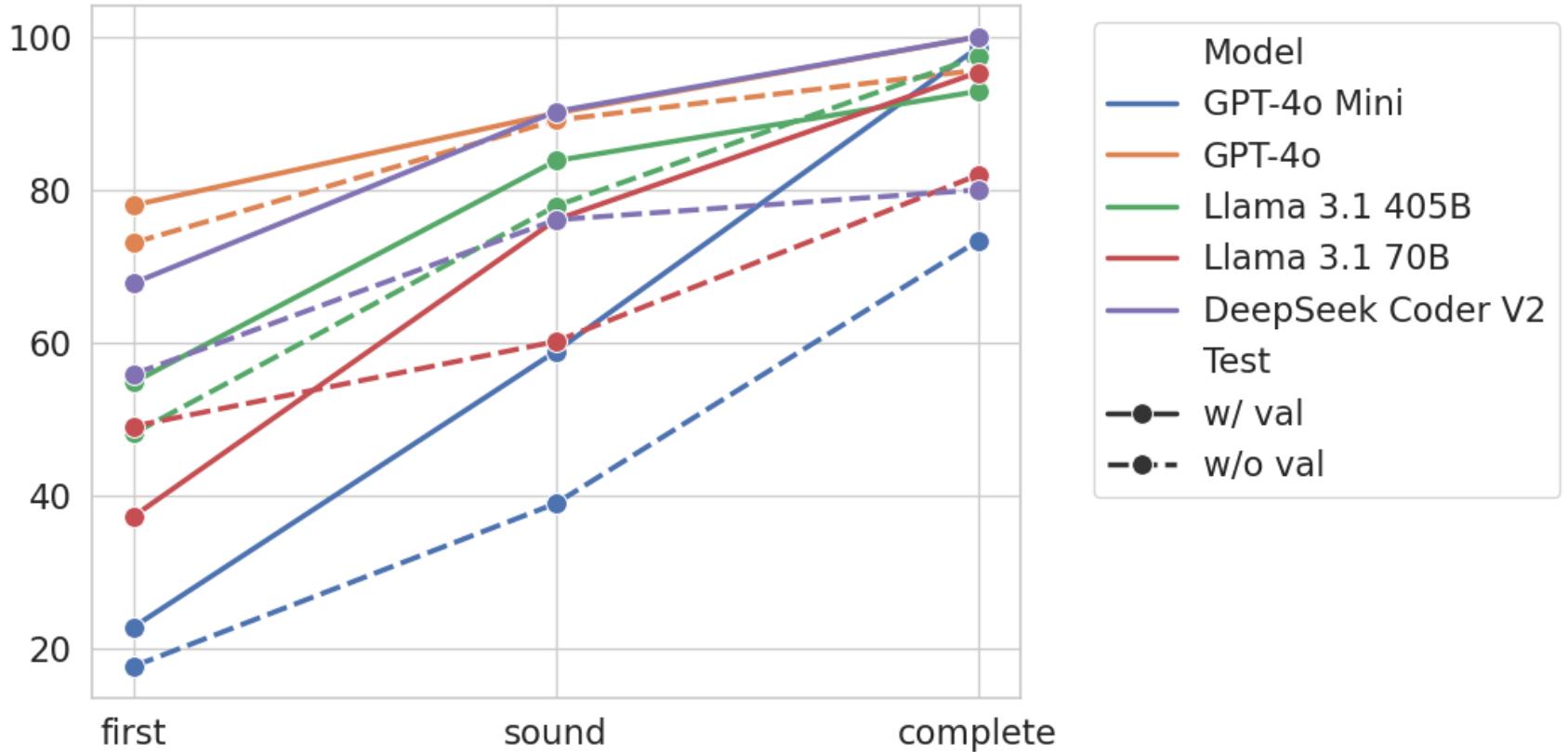
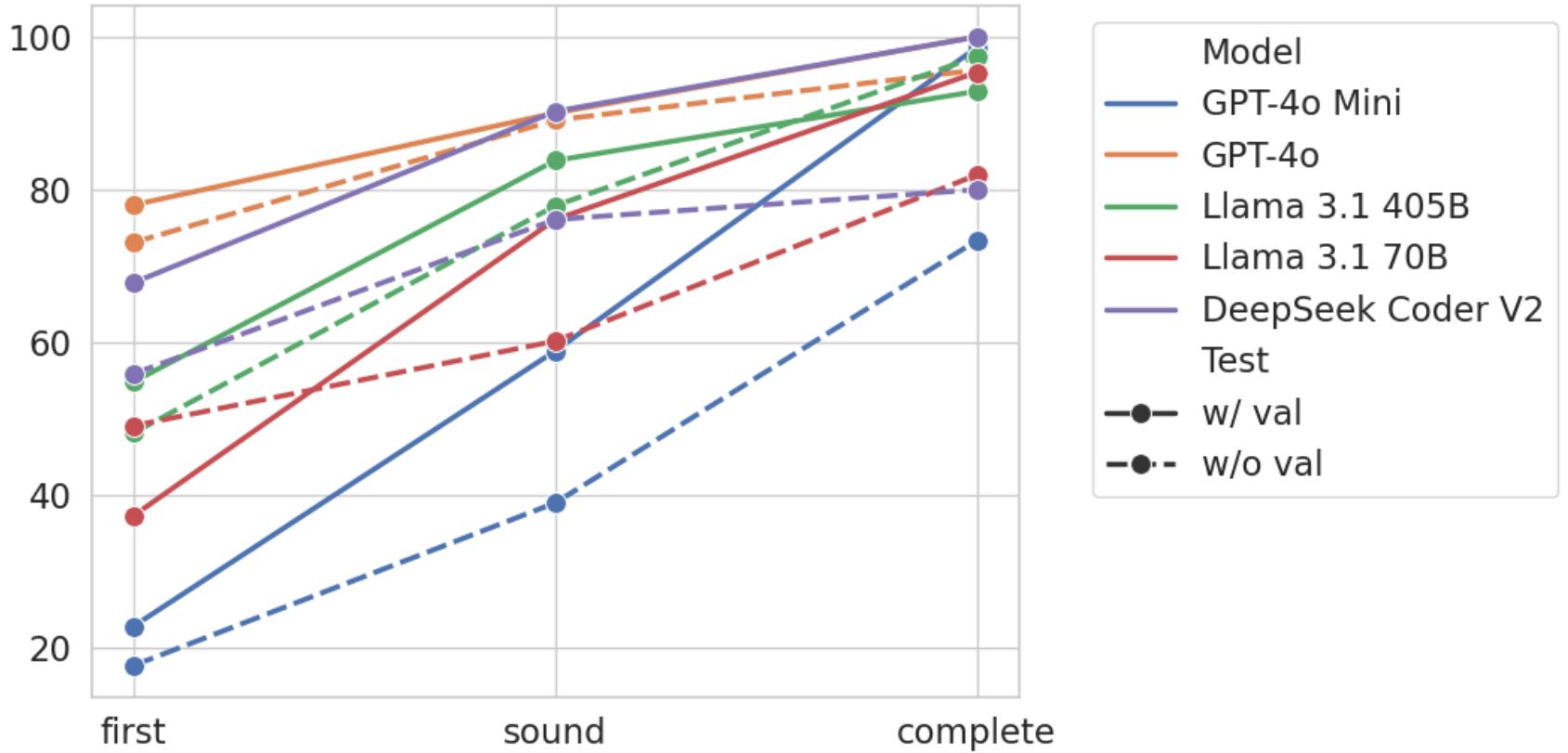


Figure 3: Average number of feedback calls for goal soundness, successor soundness, and successor completeness.

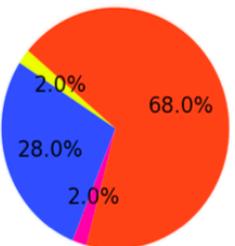
Experiments



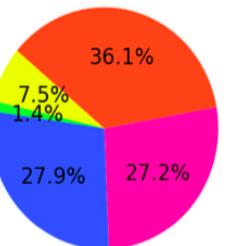
Experiments



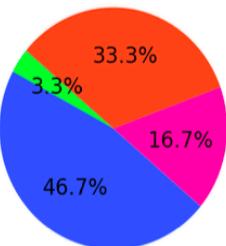
DeepSeek-Coder-V2



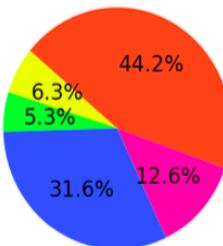
GPT-4o-mini



GPT-4o



Llama 3.1 70B



Llama 3.1 405B

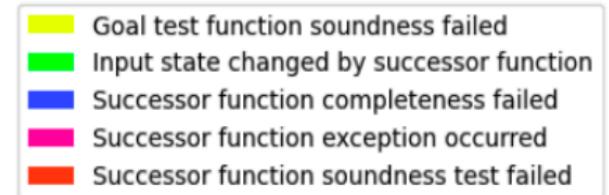
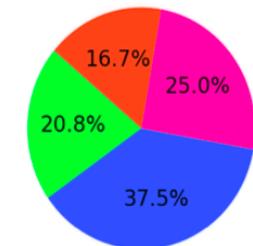
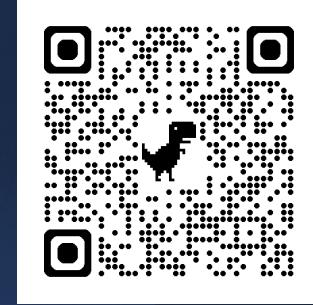


Figure 4: Partition of the errors in the generated code.

Future Directions



- Heuristic functions, search pruning, duplicate detection, ...
- Some implementations are much slower than others – choose when deploying
- Further automating the process – unit tests, better textual description,...

- Be aware: a single 24 game search of ToT with GPT-4o (single run) consumes ~20kWh (according to estimates found online).

The Carbon Impact
of Large Language
Models:
AI's Growing
Environmental Cost

