

Project 1: Regular Languages and Pursuit-Evasion

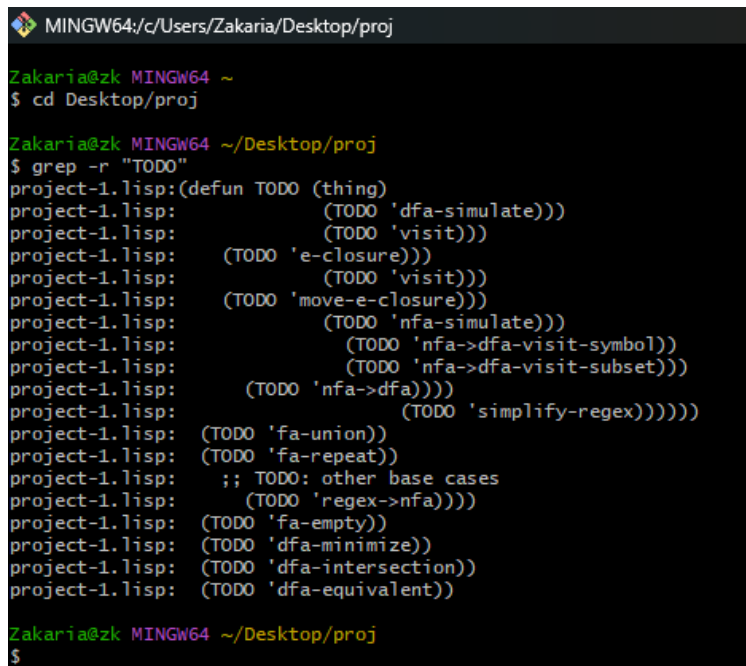
CSCI 561

November 2, 2024

Application: Text Processing with Grep

grep is a unix utility that searches text for lines matching a regular expression.

1. **Using Grep:** Use the standard unix (or GNU) `grep` and the implementation in this project to search the starter code for all TODOs.



```
MINGW64:/c/Users/Zakaria/Desktop/proj

Zakaria@zk MINGW64 ~
$ cd Desktop/proj

Zakaria@zk MINGW64 ~/Desktop/proj
$ grep -r "TODO"
project-1.lisp:(defun TODO (thing)
project-1.lisp:      (TODO 'dfa-simulate)))
project-1.lisp:      (TODO 'visit)))
project-1.lisp:      (TODO 'e-closure)))
project-1.lisp:      (TODO 'visit)))
project-1.lisp:      (TODO 'move-e-closure)))
project-1.lisp:      (TODO 'nfa-simulate)))
project-1.lisp:      (TODO 'nfa->dfa-visit-symbol)))
project-1.lisp:      (TODO 'nfa->dfa-visit-subset)))
project-1.lisp:      (TODO 'nfa->dfa)))
project-1.lisp:      (TODO 'simplify-regex))))))
project-1.lisp:      (TODO 'fa-union))
project-1.lisp:      (TODO 'fa-repeat))
project-1.lisp:      ;; TODO: other base cases
project-1.lisp:      (TODO 'regex->nfa)))
project-1.lisp:      (TODO 'fa-empty))
project-1.lisp:      (TODO 'dfa-minimize))
project-1.lisp:      (TODO 'dfa-intersection))
project-1.lisp:      (TODO 'dfa-equivalent))

Zakaria@zk MINGW64 ~/Desktop/proj
$
```

Figure 1: Histogram

2. **NFA vs. DFA Simulation:** The original `grep` utility (as written by Ken Thompson) converted the provided regular expression to an NFA and then simulated the NFA on the input text. Why would Thompson have used NFA simulation instead of DFA simulation?

Ken Thompson used NFA simulation instead of DFA simulation in the original `grep` utility due to the advantages of NFA simulate over making and simulating a DFA. Additionally, Thompson's algorithm allows for direct and efficient conversion from a regular expression to an NFA that simulates the string in linear time, making it ideal for fast text-search applications like `grep` where we care about run time performance. An algorithm first constructing a DFA from an NFA formed by a regular expression needs to pay an upfront computational cost that is $O(2^{|Q|})$ in converting from an NFA to a DFA before reaping the benefits of less complex DFA simulation, which may not be worthwhile when run on only a few strings. [Lecture-04a-NFA-to-DFA-Handout.pdf](#)

3. **Time Complexity:** To simulate an NFA $N = (Q, \Sigma, \delta, q_0, F)$ on string σ using the algorithm in the original grep (which is roughly equivalent to the NFA simulation algorithm covered in lecture) requires $O(|Q| * |\sigma|)$ time. However, many “regex” engines (including the popular implementation in Perl) exhibit worst-case complexity that is exponential in $|\sigma|$.

- (a) Prove that NFA simulation is possible in $O(|Q| * |\sigma|)$ time.

Answer: Here is our NFA simulation algorithm from class:

```
def simulate(input):
    state = e-closure({startState})
    for currentChar in input:
        nextState = empty
        for s in state:
            nextState = nextState U table[s, currentChar]
        state = e-closure(nextState)

    if state == empty:
        break

    if state (intersection) acceptStates == empty:
        return reject
    return accept
```

We can see that the overall time complexity is $O(|Q| * |\sigma|)$, since the “for currentChar in input:” loop still runs once per character in input, contributing a factor of $|\sigma|$ to the complexity. Inside the loop, for each character, it iterates over each state in “state”, potentially up to $|Q|$ states in the worst case. The operations inside the inner loop (unioning the next states and applying the e-closure) are still bounded by $O(|Q|)$, since each state transition or e-transition can only go to a finite number of other states. Thus, the total complexity will be $O(|Q| * |\sigma|)$.

- (b) Why is there such a difference in performance between Thompson’s grep and some other “regex” engines?

Thompson’s grep uses an NFA simulation algorithm that relies on tracking the set of all reachable states at any given step of the string input. This maintains a linear complexity with respect to the input length of the string by tracking all possible states simultaneously without needing to backtrack. It does mean there is added complexity by needing to, in the worst case, keep track of all states in the NFA, which is linear in complexity.

Backtracking-based regex engines use a depth-first search approach. They evaluate one possible path of the NFA for the given string at a time and backtrack if they encounter a mismatch. A benefit of this approach is the simplicity with which it can be programmed, and only needing to focus on being in one state at a time of the NFA. This can lead to exponential complexity in worst case, as the number of nodes in a tree of depth n is 2^n . A back-tracking approach may end up traversing every single path in the tree before deciding on the string, and therefore be exponential in complexity as it goes over every single node. [”Regular Expression Matching Can Be Simple And Fast \(but is slow in Java, Perl, PHP, Python, Ruby, ...\)”](#)

Application: Discrete Event Systems Model of Pursuit-Evasion

Pursuit-evasion games are scenarios with multiple agents where one agent attempts to avoid capture by another. Consider a variation of pursuit-evasion games as follows:

- Two agents share a grid environment: a human (evader) and wumpus (pursuer).
- The human and wumpus alternate moves on the grid. The wumpus moves each turn up, down, left or right. The human can move up, down, left, right, or remain in place. You may assume that the human moves first.
- If the wumpus and human ever occupy the same grid cell, the wumpus eats the human.
- If the human reaches a designated grid cell, they escape.

Answer the following questions using your implementation of finite automata operations for support.

1. For the map in 2, construct a discrete event system model. Assume that the human's movements are controllable and that the wumpus's movements are not controllable.
2. For your DES model of 2, construct a specification for the human to avoid the wumpus and escape.
 - (a) Can the human guarantee to avoid being eaten, no matter what the wumpus does? Prove yes or no via automata operations.
 - (b) Can the human always to escape in finite time (fixed number of steps), no matter what the wumpus does? Prove yes or no.
3. Design a map where the wumpus can always eat the human and prove via a DES model that this is the case.
4. Design a map where the human can always escape and prove via a DES model that this is the case.

W		E
	O	
H		

Figure 2: Example wumpus-world map. W represents the initial location of the wumpus. H represents the initial location of the human. E represents the escape location. O represents an obstacle that neither the human nor wumpus can move into.

Answers

1. State Definitions with Turn-Taking

Each state in the DES model captures the human's and wumpus's positions on a grid and specifies whose turn it is to move. The starting positions are $H(0,0)$ for the human and $W(0,2)$ for the wumpus, with the initial state represented as $H(0,0)$, $W(0,2)$, H_turn . The model defines the following types of states:

- **Initial State:** Starting state with $H(0,0)$, $W(0,2)$, H_turn .
- **Human Turn State:** Any state where it is the human's turn, represented as $H(x,y)$, $W(u,v)$, H_turn .
- **Wumpus Turn State:** Any state where it is the wumpus's turn, represented as $H(x,y)$, $W(u,v)$, W_turn .
- **Escape State:** A final state where the human reaches the escape cell $H(2,2)$, regardless of the wumpus's position.
- **Caught State:** A final state where the human and wumpus occupy the same cell, indicating capture.

Each state thus represents a unique configuration of positions and turn, e.g., $H(1,2)$, $W(0,1)$, H_turn .

2. Events and Transitions

Events represent possible moves by the human and wumpus, alternating turns:

- **Controllable Events (Human's Moves):** The human's moves (up, down, left, right) lead to states where it becomes the wumpus's turn, e.g., from $H(0,0)$, $W(0,2)$, H_turn , a H_Down move results in $H(1,0)$, $W(0,2)$, W_turn . Moves into obstacle cells are invalid.
- **Uncontrollable Events (Wumpus's Moves):** The wumpus moves randomly in four directions, leading to states where it becomes the human's turn, e.g., W_Right from $H(1,0)$, $W(1,0)$, W_turn changes to $H(1,0)$, $W(1,1)$, H_turn . *Moves into obstacles are also invalid.*

3. Special Transitions

The model includes transitions to two final states based on collision or successful escape:

- **Collision (Caught State):** Transition to the "caught" state if the human and wumpus occupy the same cell.
- **Escape:** Transition to the "escape" state if the human reaches the escape cell $H(2,2)$.

4. Graph Representation with Turn-Taking

- **Nodes:** Represent unique states defined by positions and turn, e.g., $H(0,0)$, $W(0,2)$, H_turn .
- **Edges:** Label transitions with move events (e.g., H_Right for human, W_Left for wumpus), alternating turns.
- **Start and Final States:** The start state has a bold outline, while final states (**Escape** and **Caught**) are marked with double circles. Transitions into obstacle cells are omitted to indicate invalid moves.

4. Graph Representation with Turn-Taking

The DES model is represented as a directed graph, where each node represents a unique state with turn-taking between the human and wumpus. The graph structure and design are defined as follows:

- **Nodes:**
 - Each node represents a state, labeled by the human's and wumpus's positions, such as $H(1,2)$ $W(0,1)$.
 - Key states are color-coded:

- * The **Start State** (S) at $H(0,0)$, $W(0,2)$ is shown in green.
- * The **Escape State** (E) at $H(2,2)$, regardless of wumpus position, is blue.
- * The **Caught State** (C), where the human and wumpus share a cell, is marked in red.

- **Edges and Transitions:**

- Each edge corresponds to a move by either the human or wumpus, labeled according to the direction (e.g., **H moves Down** or **W moves Left**).
- The human's moves are controllable (up, down, left, right) and transition to states where it becomes the wumpus's turn. For example, from $H(0,0)$ $W(0,2)$, a **H moves Down** transition leads to $H(1,0)$ $W(0,2)$.
- The wumpus's moves are uncontrollable and random, also in four directions. A wumpus move, such as **W moves Right**, transitions to a new state like $H(1,0)$ $W(1,1)$, where it's the human's turn.

- **Invalid Moves:**

- Moves leading to the obstacle cell at $(1,1)$ are omitted, as these are invalid and do not produce transitions.

- **Special Transitions to Final States:**

- The graph includes edges to the final **Escape** or **Caught** states as applicable:
 - * **Escape:** If the human reaches $H(2,2)$, the model transitions to **E** with a label **H escapes**.
 - * **Caught:** If the human and wumpus occupy the same cell, the model transitions to **C** with the label **Caught**.

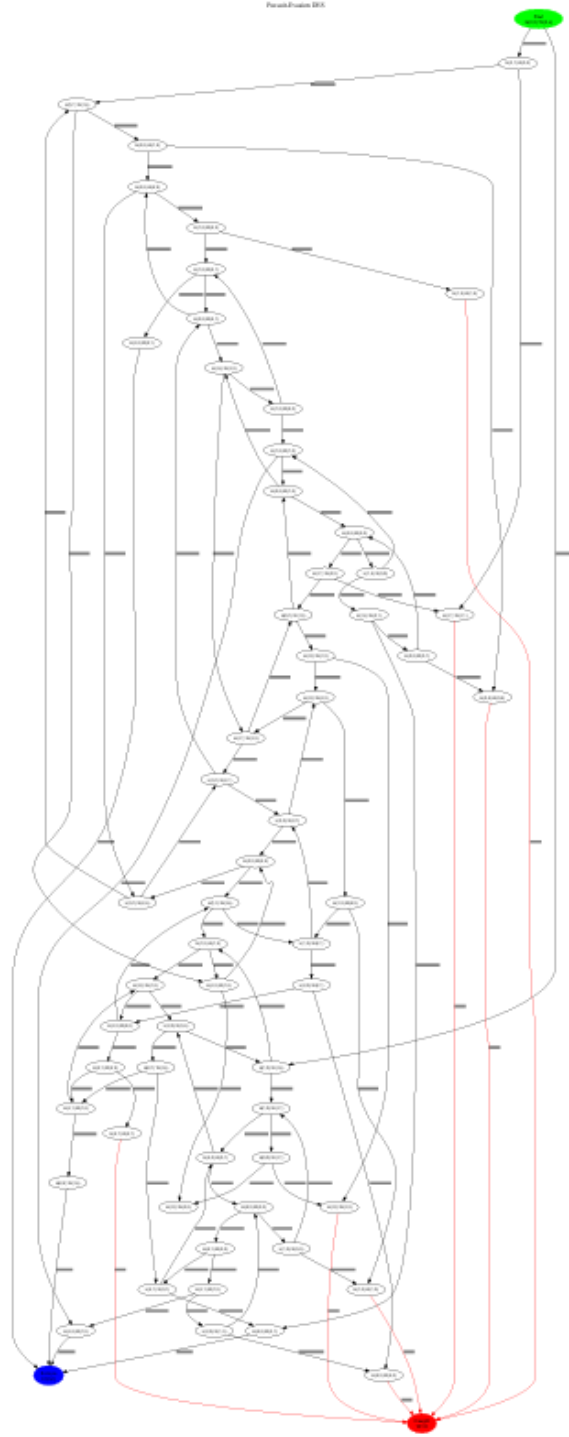


Figure 3: Pursuit-Evasion DES

PS: For a high-resolution version of the image, please refer to the following link: [link](#)

Summary: This DES model captures all valid states and transitions while respecting movement constraints due to obstacles and alternating turns between the human and wumpus. It includes paths leading to both the escape and eaten states, facilitating an analysis of the human's strategy to avoid the wumpus and reach the escape. The random movement of the wumpus and alternating turns make this an NFA model.

2. Specification for the Human to Avoid the Wumpus and Escape

To achieve the goals of avoiding the wumpus and reaching the escape cell, we construct two formal specifications in the form of finite automata. These specifications enforce conditions for safe movement and successful escape within the DES model.

(a) Specification to Avoid Being Eaten

Avoidance Condition: The human must never occupy the same cell as the wumpus. This requires that all states where the human and wumpus positions coincide are avoided.

Specification Automaton for Avoidance:

- Define states in the automaton to match the DES model states but exclude any transitions leading to configurations where the difference between the x-coordinates or y-coordinates between the human and the wumpus is less than 1. In other words, if the human is at $H(x, y)$ and the wumpus at $W(u, v)$, do not allow any moves that cause $|x - u| + |y - v| \leq 1$. For if the human were to enter such a square where $|x - u| + |y - v| \leq 1$, then the wumpus could move into the human's square on the wumpus' turn.
- Label **safe states** as those where for the human at $H(x, y)$ and the wumpus at $W(u, v)$, $|x - u| + |y - v| > 1$, and allow transitions only between these states.

The avoidance specification automaton will thus allow only paths that keep the human in safe states, avoiding any possible overlap with the wumpus.

(b) Specification for Finite Escape

Answer: No, No escape specification exists that can get the human to the exit in finite time for the given grid the human and wumpus reside on. This is because the wumpus can move to the exit with finite probability in 2 turns, compared to a minimum of 4 turns for the human.

3. Map Design Where the Wumpus Can Always Eat the Human

To construct a grid where the wumpus can always intercept the human, we design a layout and create a DES model that guarantees the human cannot reach an escape cell without encountering the wumpus. Strategic obstacle placement and central positioning of the wumpus restrict the human's movement options and force them into paths that lead to capture.

1. Map Design

We use a 3×3 grid with the following layout:

- **(0,1)**: Starting position of the human.
- **(2,0)**: Escape cell.
- **(1,1)**: Starting position of the wumpus, located centrally.
- **(0,0)**, **(0,2)**, **(1,0)**, and **(2,1)**: Obstacles that restrict the human's movement options.

Obstacle (0,0)	H(0,1)	Obstacle (0,2)
Obstacle (1,0)	W(1,1)	Empty
Escape (2,0)	Obstacle (2,1)	Empty

2. DES Model Construction

Each state in the DES model is represented as $H(x,y)$, $W(u,v)$, **turn**. The model enforces:

- **Human's constrained moves**: up, down, left, and right. Obstacles limit possible movement paths, directing the human toward the wumpus.
- **Wumpus's targeted moves**: The wumpus starts in the center and moves to intercept any available human path toward the escape cell.

3. Specification Automaton for Guaranteed Capture

To demonstrate that capture is inevitable, we define a specification automaton with:

- **Safe states**: States where $H(x,y) \neq W(x,y)$, ensuring that the human and wumpus do not occupy the same cell until the capture.
- **Capture state**: A state where the human and wumpus occupy the same cell, marking the human's capture.

Transitions: The automaton restricts transitions to ensure that:

- Paths leading to the capture state are prioritized, where the wumpus intercepts the human before reaching the escape cell.
- Obstacles prevent any alternative paths to the escape, ensuring that each route leads to capture.

Conclusion: The specification automaton confirms that with this layout, no matter what path the human takes, they will inevitably reach the capture state where $H(x,y) = W(x,y)$. This proves that the wumpus always intercepts the human, eliminating any possible escape routes in this configuration.

4. Map Design Where the Human Can Always Escape

To construct a grid where the human can guarantee escape, we designed a layout and a DES model that ensures the human can reach the escape cell without being intercepted by the wumpus. The obstacles and initial positioning create a separation that allows the human to move safely to the escape.

1. Map Design

We use a 3×3 grid with the following layout:

- **(0, 0):** Starting position of the human.
- **(2, 0):** Escape cell.
- **(0, 2):** Starting position of the wumpus.
- **(0, 1), (1, 1), and (2, 1):** Obstacles that limit the wumpus's movement options and create a safe path for the human.

H(0,0)	Obstacle (0,1)	W(0,2)
Empty	Obstacle (1,1)	Empty
Escape (2,0)	Obstacle (2,1)	Empty

2. DES Model Construction

In this DES model, each state is represented as $H(x,y)$, $W(u,v)$, **turn**. Given that the human's movements are controllable and the wumpus's movements are constrained (avoiding paths that lead to the human), the model captures:

- **Human's controllable moves:** up, down, left, and right, allowing them to navigate the path to the escape cell.
- **Wumpus's constrained moves:** up, down, left, or right, with obstacle restrictions preventing it from reaching the human's path.

3. Specification Automaton for Guaranteed Escape

To verify that escape is always achievable, we define a specification automaton with:

- **Safe states:** All states where $H(x,y) \neq W(x,y)$, ensuring the human and wumpus never occupy the same cell.
- **Accepting (Escape) state:** A final state where the human reaches the escape cell $H(2,0)$.

Transitions:

- The automaton includes only paths leading to the escape state without entering any overlap states (where $H(x,y) = W(x,y)$).
- Ensure that at least one path exists from the initial state $H(0,0)$, $W(0,2)$ to the escape state $H(2,0)$ within a finite number of moves.

Through exhaustive generation and testing of possible states, we confirm that there are no caught cases, as all paths avoid any overlap between the human and wumpus positions. This verification guarantees that the human will reach the escape state without interception.

PS: For a high-resolution version of the DES model, please refer to the following [link](#).

Answer: Yes, this specification automaton and the constrained movement rule demonstrate that, no matter what the wumpus does, the human can always reach the escape state without interception, guaranteeing escape.