discussion_jrelling.pdf – discusses output and code

Output Discussion:

1. a plus Machines

DTM:
Successfully accepts strings like aaa_ in 4 steps with 4 transitions.
Operates with a consistent average nondeterminism of 1.0, reflecting the lack of branching.
NTM:
Also accepts aaa_ but uses 7 transitions, demonstrating average nondeterminism of 1.75.
Rejects _ (empty string) with 0 transitions and no branching.
Comparison:
The DTM follows a single linear computation path, while the NTM's nondeterminism slightly increases the number of transitions but allows it to explore more possibilities concurrently. For simple inputs, both perform similarly.

2. a*b*c* Machines

DTM:
Accepts aaabbbccc_ in 10 steps with an average nondeterminism of 1.0.
NTM:
Demonstrates greater flexibility, accepting _ (empty tape) in only 2 steps.
For aaabbbccc_, it also accepts the string but with 44 transitions and an average nondeterminism of 4.4, indicating significant branching.
Rejects invalid strings like abcabc_ and abcd_ in 4 steps but with 14 transitions, with average nondeterminism of 3.5.
Comparison:
The NTM's ability to explore multiple paths simultaneously gives it an advantage in handling short inputs efficiently. However, for longer strings, its nondeterminism results in significantly higher transitions and complexity compared to the DTM.

3. ({w | w has the same number of 0's and 1's})

DTM:
Accepts 000111_ in 31 steps, with all transitions contributing to a consistent nondeterminism of 1.0.
Rejects 00011_ in 20 steps with slightly reduced nondeterminism (0.95) due to efficient halting.
NTM:
Accepts 000111_ with 41 transitions and 1.32 average nondeterminism.
Rejects 00011_ in a similar number of steps as the DTM, but with slightly more transitions (22).
Efficiently handles balanced strings like 010101_ with 25 steps, using moderate nondeterminism (1.52).
Comparison:
The deterministic machine is slower but straightforward, reflecting its singular computational path. In contrast, the NTM explores multiple configurations in parallel, resulting in slightly more steps and transitions but improved adaptability for irregular input patterns.

Code Discussion:

The traceTM_jrelling.py script simulates deterministic and nondeterministic turing machines by modeling execution as a tree of configurations. Each configuration represents a state, tape content, and its position, starting from an initial state and expanding to new levels based on valid transitions. Nondeterminism is modeled by allowing multiple transitions from a single configuration, resulting in branching paths in the tree. The execution halts when a string is accepted, rejected, or the maximum depth is reached.

The code tracks nondeterminism by counting transitions across all levels and calculating the average number of transitions per level, representing the degree of branching. Accepted strings trace their path back through parent configurations to visualize the computation. This approach combines a breadth-first search over the configuration tree with clear values for execution depth, total transitions, and nondeterminism.