Non-Deterministic Machines

Robert M. Keller
Harvey Mudd College
13 April 2003

What is Non-Determinism?

- Can a machine have choice or free-will?
- Non-determinism supposes it can.

Uses of Non-Determinism?

- Modeling asynchronous events, wherein we don’t know the order in which things happen.
- Modeling certain kinds of parallel computing.
- As a mathematical contrivance to establish certain results.

Non-Deterministic Turing Machines

- Consider the transition rules of a Turing machine:

$$a, 0 \rightarrow b, 1, L$$
$$b, 0 \rightarrow c, 0, R$$
$$\ldots$$

Non-Deterministic Turing Machines

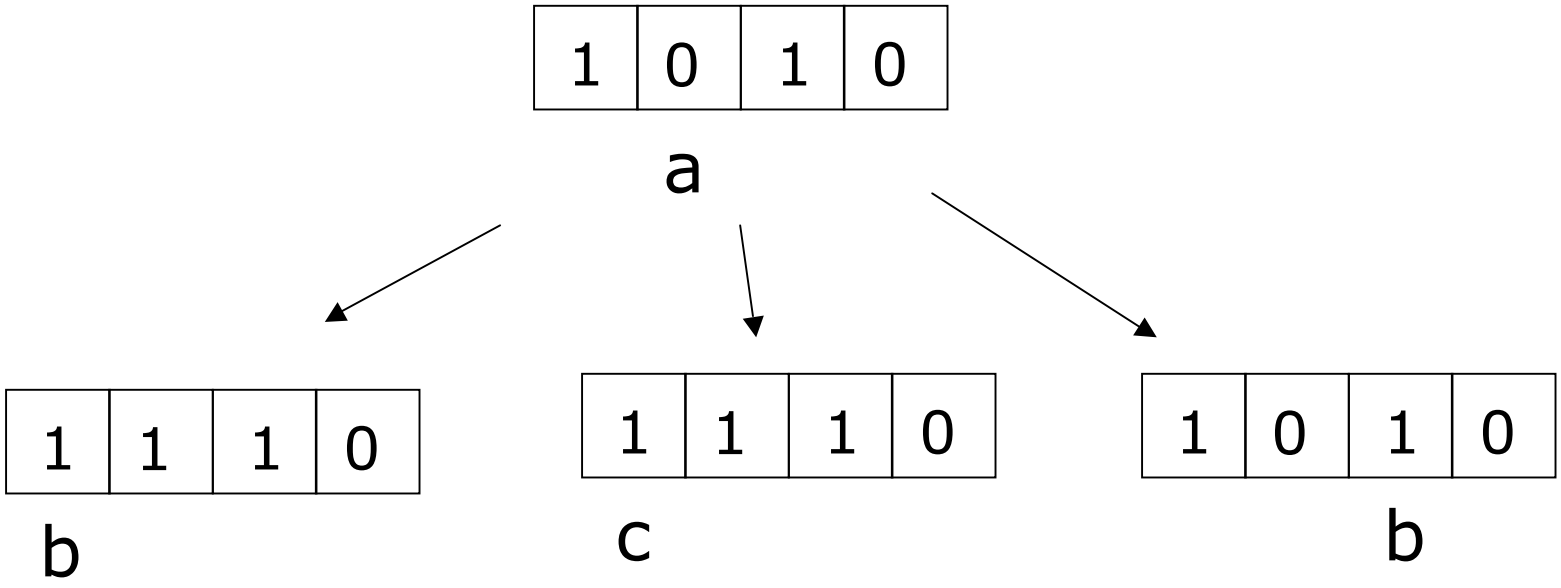
- Suppose we have 2 or more rules with the same LHS, but different RHS's.

  a, 0 → b, 1, L
  a, 0 → c, 1, L
  a, 0 → b, 0, R

- If the machine were to get to this state and symbol combination, the next state would be **ambiguous**.

- If the machine had free-choice, it could go either way.

Non-Deterministic Choice



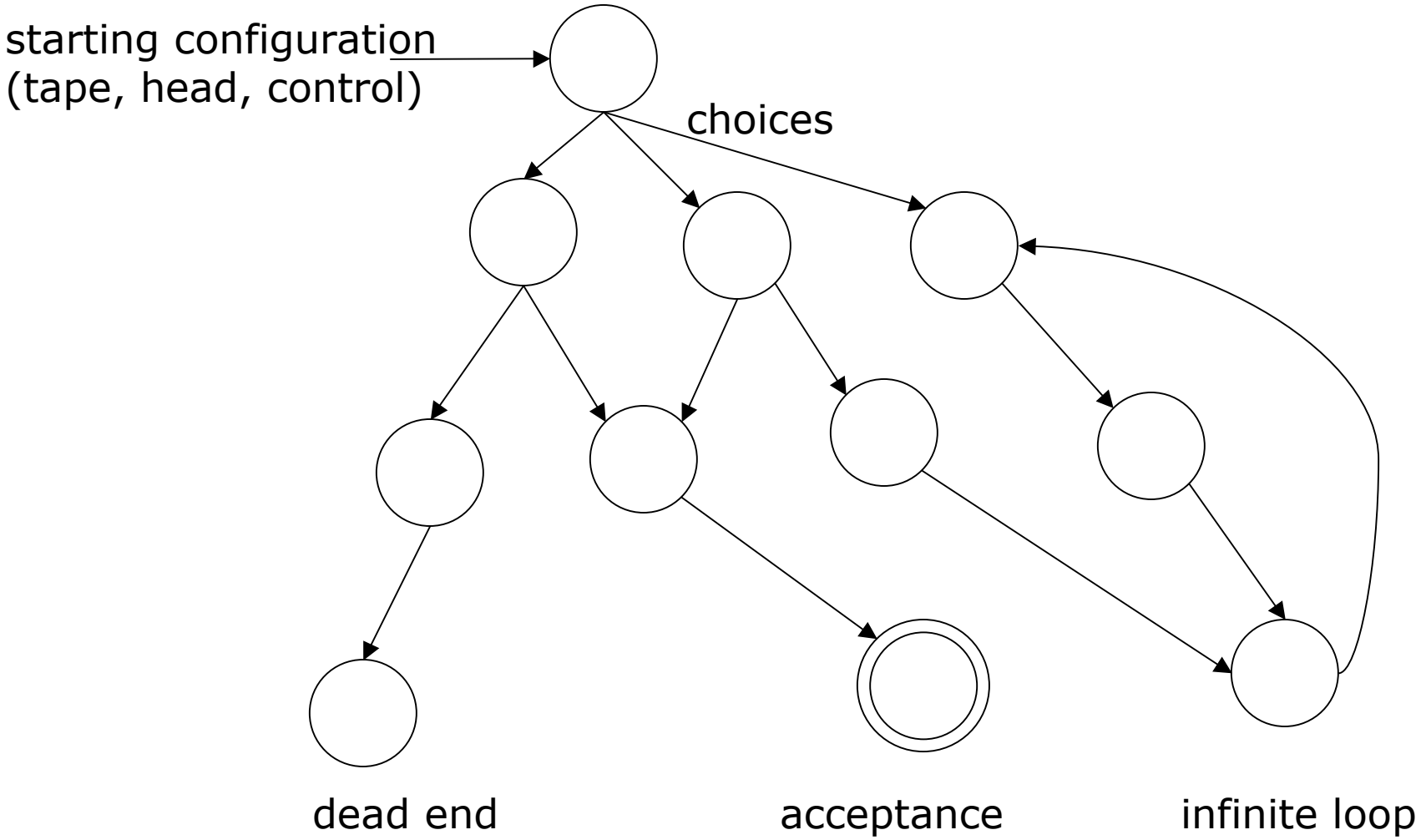
a, 0 → b, 1, L
a, 0 → c, 1, L
a, 0 → b, 0, R

The Use of Non-Determinism

- Most of our uses will be in the case of acceptors, rather than function computors.

- We say that a non-deterministic TM accepts a string if **there exists** a path from the starting configuration to an accepting state.

Acceptance


starting configuration
(tape, head, control)
choices
dead end
acceptance
infinite loop

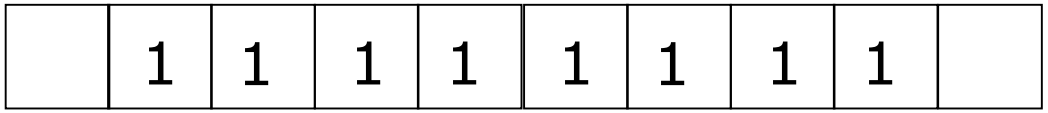
Guessing

- Using the metaphor that the machine "wants" to accept a string if possible, the choices can be thought of as "guesses" as to which way to go to get to an accepting state.

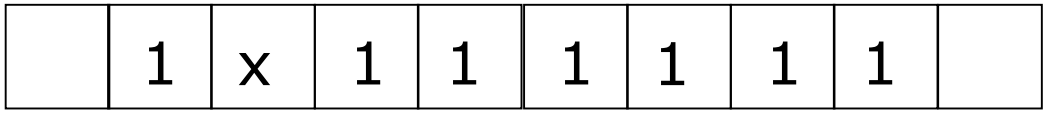
Economies of Non-Determinism

- With non-deterministic acceptance, we can get by with smaller machines.
- Example: A machine that accepts strings having length of composite (non-prime) numbers.
- The machine can move its head right, guessing when to stop.
- Then it marks off multiples of the cells (at least 2) it has traversed.
- If the marking comes out even, then it accepts.

Composite-Length ND Acceptor

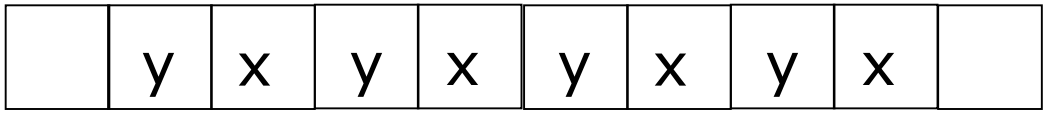


a ⟶



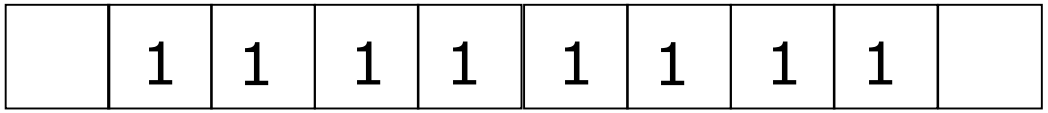
b

multiples of 2

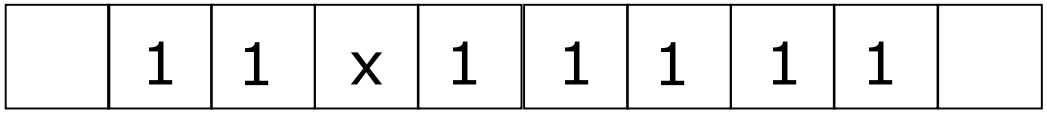


c accept

Composite-Length ND Acceptor

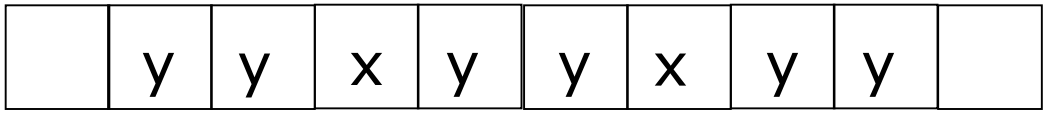


a ⟶



b

multiples of 3



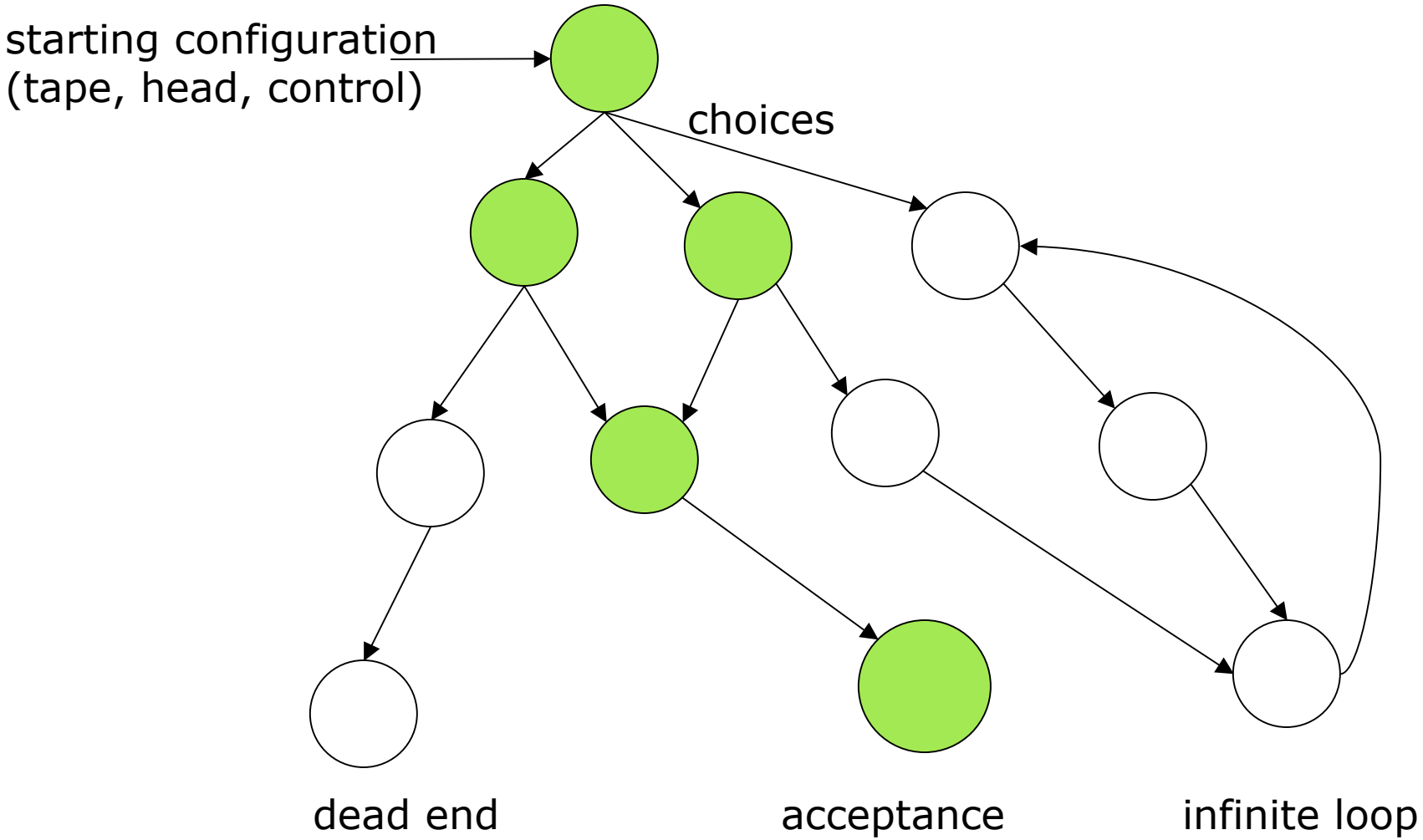
d don’t accept
(no problem!)

Asymmetry of Non-Deterministic Acceptance

- If the string in question is **not** in the language, there must be **no** sequence of moves that will lead to an accepting state.

- Having a sequence to a rejecting state is not enough.

Only the green configurations lead to acceptance.



Does Non-Determinism Change Computability?

- No. A non-deterministic machine can be simulated by a deterministic one.
- This is done by a dove-tailing construction.
- The simulating machine accepts exactly in the case that one of the simulations accepts.

Does Non-Determinism Change Complexity?

- Yes. A non-deterministic machine can often accept in a much shorter sequence due to guess.
- The question of P = NP is whether a non–determininstic machine operating in polynomial time on some sequence has a corresponding deterministic machine that also operates in polynomial time.
- Probably this is not the case, but it has never been disproved.

Grammars

Grammars Define Languages

- A grammar is a formal system that is by nature non-deterministic.
- A grammar generates a set of strings.
- In many cases, the specification by a grammar is more efficient that specifying a machine.

Definition of Grammars

- A grammar consists of 4 parts:
  - Terminal alphabet Σ
  - Auxiliary alphabet A (such that $A \cap \Sigma = \varnothing$)
  - A finite set of productions $\rightarrow$
  - Start symbol $S \in A$
- Each production has the form $x \rightarrow y$, where $x \in (\Sigma \cup A)^*-\{\Lambda\}$, $y \in (\Sigma \cup A)^*$.
- **Note**: Auxiliaries are also called "non-terminals" and "variables".

Derivation in a Grammar

- A derivation in of a grammar is a sequence of strings $x_0 \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \ldots$ each in $(\Sigma \cup A)^*$ where:
  - x_0 = S, the start symbol
  - For each i, $x_i \Rightarrow x_{i+1}$ provided that there are string u, v, x_i = u, v, v', w such that
    - x_i = uvw,
    - x_{i+1} = uv'w,
    - v $\rightarrow$ v' is a production

- The **language generated by a grammar** is the set of strings $x \in \Sigma^*$ such that there is a derivation that ends with x.

Example of a Grammar

- Alphabet: {a, b}
- Auxiliary alphabet: {S}
- Start Symbol: S
- Productions:
  - S → ab
  - S → aSb
  - S → SS
- Example derivations of strings in the language:
  - S ⇒ ab
  - S ⇒ aSb ⇒ aabb
  - S ⇒ aSb ⇒ aaSbb ⇒ aaabbb
  - S ⇒ SS ⇒ abS ⇒ abab
  - S ⇒ SS ⇒ SSS ⇒ ababab
  - S ⇒ SS ⇒ aSbS ⇒ aabbS ⇒ aabbaSb ⇒ aabbaabb

Example:
Grammar for Additive Arithmetic Expressions

- The start symbol is A.
- The terminals are {a, b, c, +}.
- The auxiliares are {A, V}.
- The productions are:
  - $A \rightarrow V$
  - $A \rightarrow V + A$
  - $V \rightarrow a$
  - $V \rightarrow b$
  - $V \rightarrow c$
- Sample derivations:
  1. $A \Rightarrow V \Rightarrow a$
  2. $A \Rightarrow V \Rightarrow c$
  3. $A \Rightarrow V + A \Rightarrow c + A \Rightarrow c + V \Rightarrow c + a$
  4. $A \Rightarrow V + A \Rightarrow c + A \Rightarrow c + V + A \Rightarrow c + b + A \Rightarrow c + b + V \Rightarrow c + b + a$

Another Example of a Grammar for the Language $\{a^n b^n c^n \mid n \in \omega, n > 0\}$

- The grammar:
  - Terminal alphabet $\Sigma = \{a, b, c\}$
  - Auxiliary alphabet {S, A, B, C, F}
  - Start symbol S
  - Productions $\rightarrow$:
    - $S \rightarrow FE$ $\quad E \rightarrow ABCE$ $\quad E \rightarrow \Lambda$
    - $BA \rightarrow AB$ $\quad CA \rightarrow AC$ $\quad CB \rightarrow BC$
    - $FA \rightarrow a$ $\quad aA \rightarrow aa$ $\quad aB \rightarrow ab$
    - $bB \rightarrow bb$ $\quad bC \rightarrow bc$ $\quad cC \rightarrow cc$

A derivation (underlines show symbols replaced):
$S \Rightarrow \underline{FE} \Rightarrow FABC\underline{E} \Rightarrow FABCABC\underline{E} \Rightarrow FAB\underline{CA}BC \Rightarrow FA\underline{BA}CBC \Rightarrow FAAB\underline{CB}C \Rightarrow \underline{FA}ABBCC \Rightarrow \underline{aA}BBCC \Rightarrow a\underline{aB}BCC \Rightarrow aa\underline{bB}CC \Rightarrow aab\underline{bC}C \Rightarrow aabb\underline{cC} \Rightarrow aabbcc$

Types of Grammars

- Grammars are classified by the kinds of productions they allow:
  - Type 0: no restriction
  - Type 1: length of LHS $\leq$ length of RHS (Context-Sensitive), plus $S \rightarrow \Lambda$ where S is the start symbol.
  - Type 2: LHS is a single auxiliary only (Context-Free)
  - Type 3: LHS is a single auxiliary, and RHS is either Λ, or σA where A is an auxiliary and $\sigma \in \Sigma$.
- These types are called the "Chomsky Hierarchy", after linguist Noam Chomsky, who first named them.

Types of Languages

- Each type of grammar generates a language of that type.
- Every type i language is a type j language for i < j.

Type 0 Languages

- Type 0 languages are equivalent to **recursively-enumerable** languages.
- For any grammar, there is a TM that accepts the language that the grammar generates.
- Any TM's recognized strings can be generated by a corresponding grammar.

Use of Machines

- Sometimes it is easier to construct a machine for a language than it is to construct a grammar, and vice-versa.
- Using non-determinism in the machine should not be overlooked.