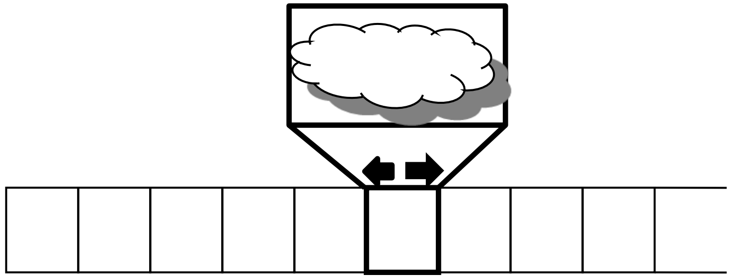
CS3102 Theory of Computation

Decidable

$w \rightarrow$
Input



$\Rightarrow$ ✓ Accept & halt | ✗ Reject & halt | Never runs forever

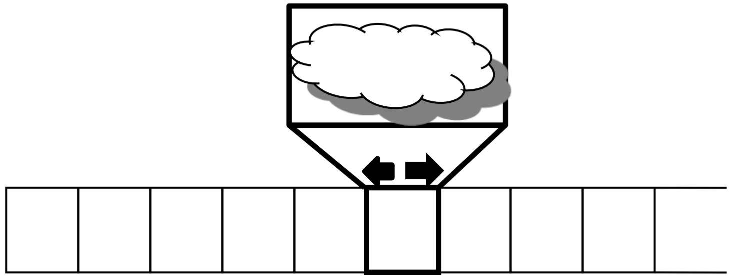
A language is decidable iff it is exactly the set of strings accepted by some always-halting TM.

$w \in \Sigma^*$	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb	aaaa	…
$M(w) \Rightarrow$	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	…
$L(M) =$	$\{\ a,$		$aa,$				$aaa,$								$aaaa$	$\ldots\}$

M must always halt on every input.

Recognizable

$w \rightarrow$ Input



$\Longrightarrow$ ✓ Accept & halt | × Reject & halt $\equiv$ ∞ Run forever

A language is Turing-recognizable iff it is exactly the set of strings accepted by some Turing machine.

$w \in \Sigma^*$	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb	$aaaa$	…
$M(w) \Rightarrow$	✓	×	✓	∞	×	∞	✓	∞	∞	×	×	×	∞	×	✓	…
$L(M) =$	$\{\ a,$		$aa,$				$aaa,$								$aaaa$	$\dots\}$

M can run forever on an input, which is implicitly a reject (since it is not an accept).

$HALT_{TM}$ is Undecidable

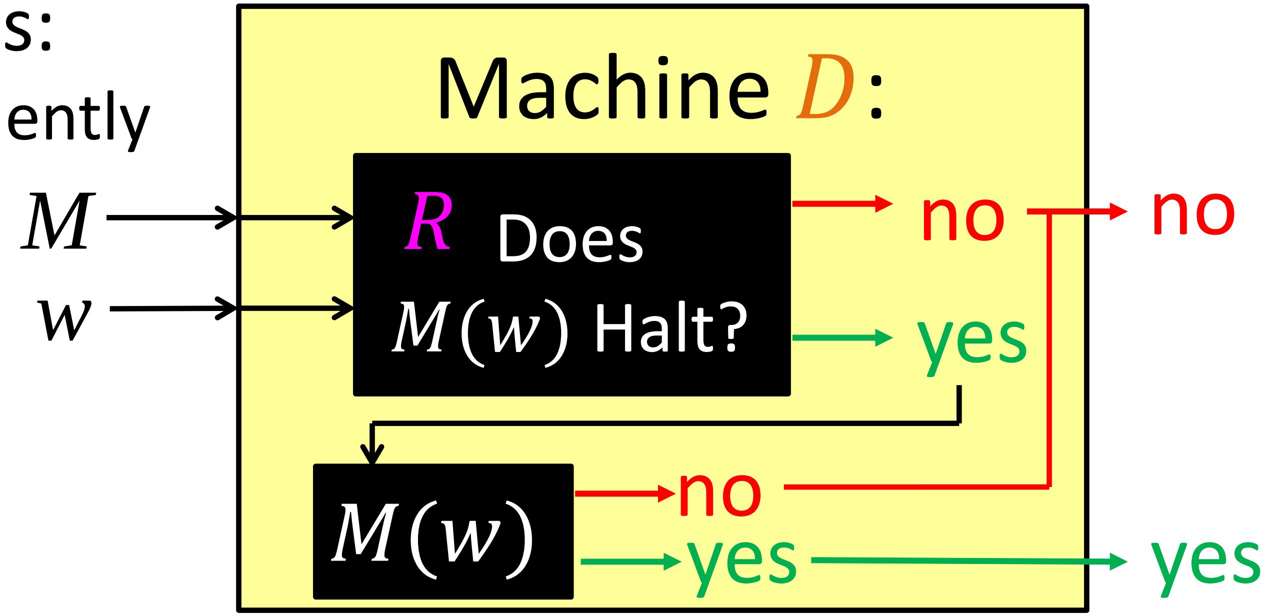
- $HALT_{TM} = \{< M, w > | M$ is a TM description and M halts on input $w\}$
  - All machine description input pairs in which the machine halts on input
- To show $HALT_{TM}$ is undecidable show A_{TM} isn't harder than $HALT_{TM}$
- Want to use a solver for $\underline{\quad HALT_{TM} \quad}$ to build a solver for $\underline{\quad A_{TM} \quad\quad}$
- $\underline{\quad A_{TM} \quad}$ reduces to $\underline{\quad HALT_{TM} \quad}$

A_{TM}

$HALT_{TM}$

Deciding A_{TM} with $\mathrm{HALT_{TM}}$

- Assume $HALT_{TM}$ is decidable.
- Then some TM R can decide $HALT_{TM}$.
- We can use R to build a machine D that decides A_{TM}:
  - Call R on $< M, w >$
  - If R rejects, it means M doesn't halt: **reject**
  - If R accepts, it means M halts:
    - Call M on w, respond equivalently



Any TM that decides $HALT_{TM}$ could be used to build a TM that decides A_{TM} (which is impossible) thus no TM exists that can decide $HALT_{TM}$

Another example: REG_{TM}

- $REG_{TM} = \{M \mid L(M) \text{ is regular}\}$
- How do we show that REG_{TM} is undecidable?
  - Reduce some language we already know is undecidable to REG_{TM}
  - Use REG_{TM} to solve $HALT_{TM}$

$REG_{TM} \geq HALT_{TM}$

- Given a potential instance of $HALT_{TM}$ (i.e. M, w), create a new turing machine M' whose language is regular if and only if $M(w)$ halts
- If I knew whether or not $L(M')$ was regular, I knew whether or not $M(w)$ halted

Psuedocode for M'

```
public static boolean mPrime(String x){
        if(x∈ a^n b^n){
                return true;
        }
        y = M(w);       If this line terminates
        return true;    We can only get to this line
}
```

If $M(w)$ halts then $L(M') = \Sigma^*$

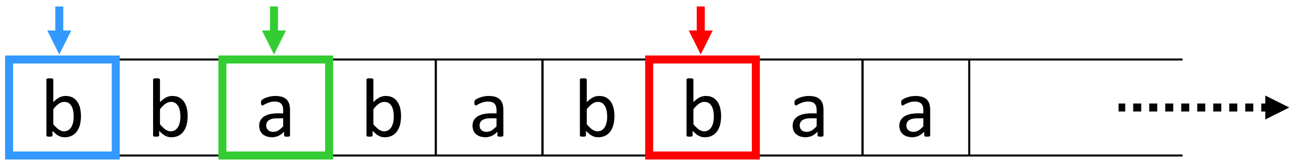
If $M(w)$ runs forever then $L(M') = a^n b^n$

How to solve $HALT_{TM}$ with REG_{TM}

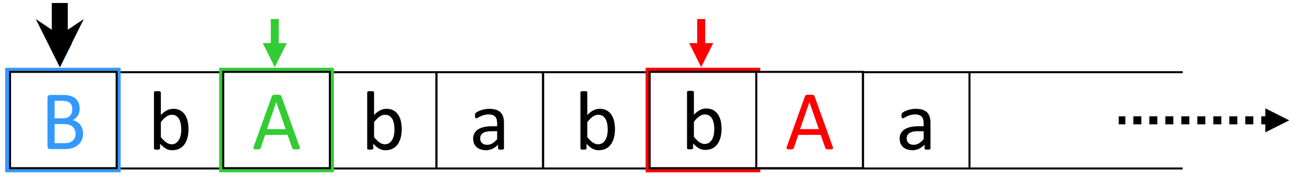
```
public static boolean halt(M,w){
        mPrime = make_mPrime(M,w);
        return reg(mPrime);
}
```

Turing Machine "Enhancements"

Multiple heads:



Idea: Mark heads locations on tape and simulate

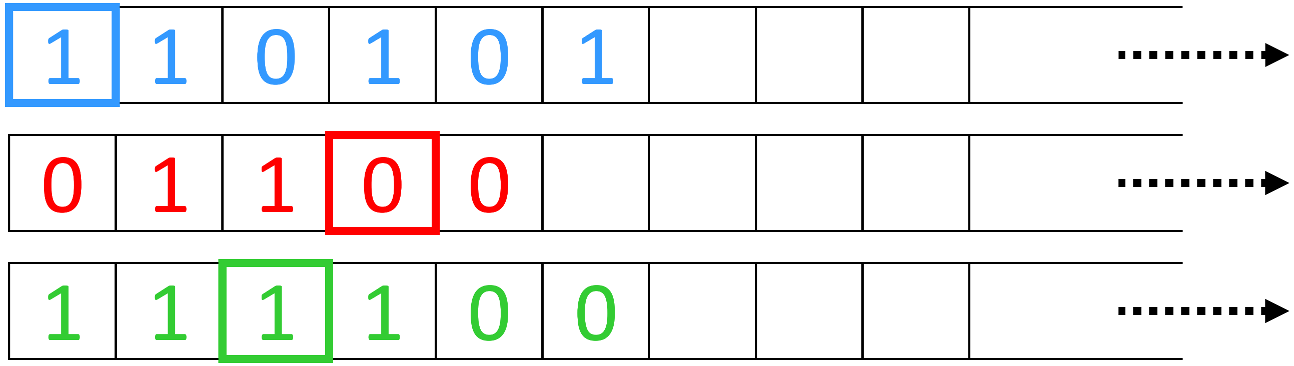


Modified δ' processes each "virtual" head independently:

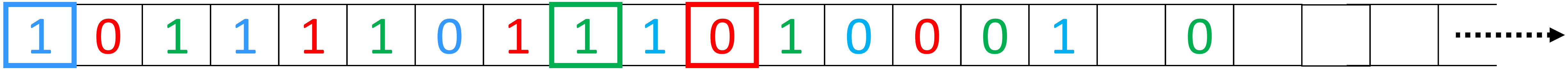
- Each move of δ is simulated by a long scan & update
- δ' updates & marks all "virtual" head positions

Turing Machine “Enhancements”

Multiple tapes:



Idea: Interlace multiple tapes into a single tape

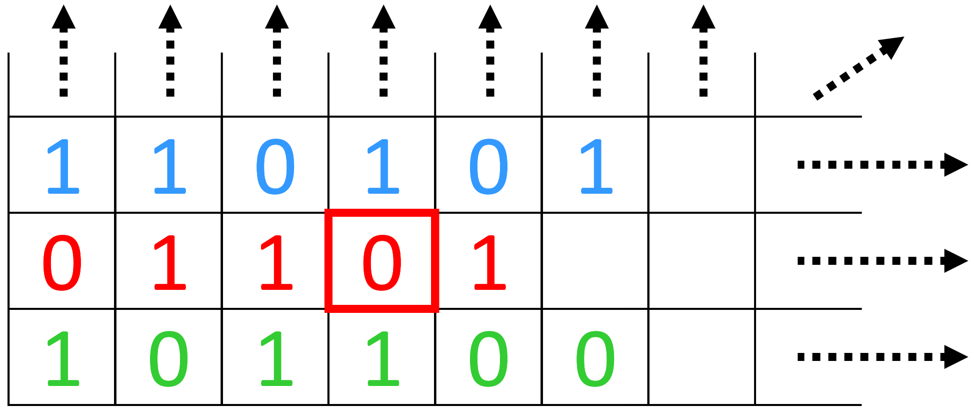


Modified δ' processes each “virtual” tape independently:

- Each move of δ is simulated by a long scan & update
- δ' updates R/W head positions on all “virtual tapes”

Turing Machine “Enhancements”

Two-dimensional tape:



Idea: Flatten 2-D tape into a 1-D tape

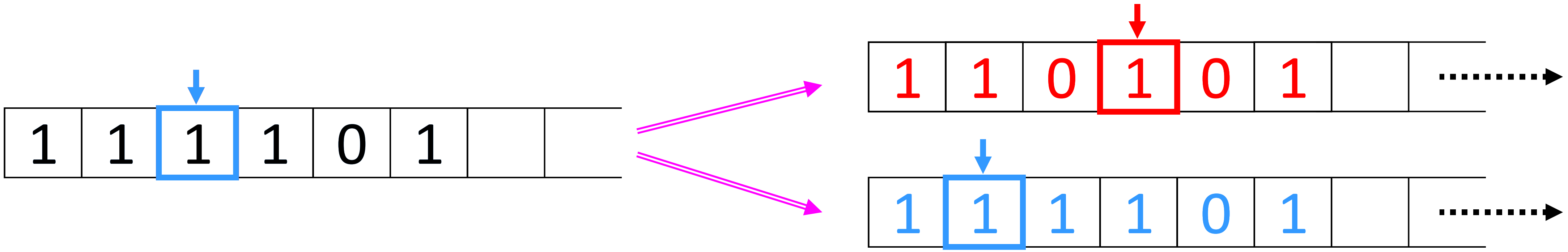
1	0	1	1	0	0	$	0	1	1	$\boxed{0}$	1	$	1	1	0	1	0	1	$	

Modified 1-D δ' simulates the original 2-D δ:

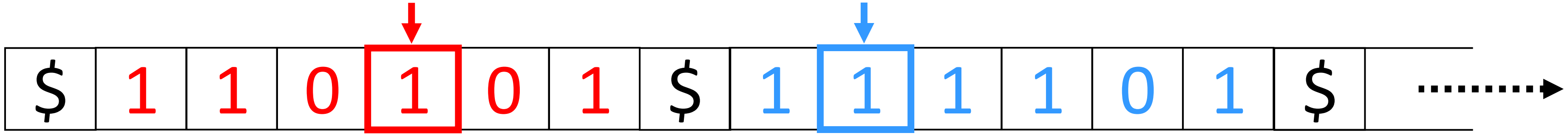
- Left/right δ moves: δ' moves horizontally
- Up/down δ moves: δ' jumps between tape sections

Turing Machine "Enhancements"

Non-determinism:



Idea: Parallel-simulate non-deterministic threads



Modified deterministic δ' simulates the original ND δ:

- Each ND move by δ spawns another independent "thread"
- All current threads are simulated "in parallel"

Enumerators

- An enumerator for language L is a TM which prints all strings in L onto its tape
- Lexicographic enumerator
  - Prints them in lexicographic order
- A language is recognizable if and only if it has an enumerator
- A language is decidable if and only if it has a lexicographic enumerator

Enumerable = Recognizable

Lexicographically Enumerable = Decidable

Closure properties of Recognizable

- Closed under:
  - Union
  - Intersection
  - Concatenation
  - Kleene
- Not closed under:
  - Complement

Not closed under Complement

- If L and $\overline{L}$ are both recognizable, then L is decidable.
- To determine if $w \in L$:
  - Run w on recognizer for L for 5 steps
  - Run w on recognizer for $\overline{L}$ for 5 steps
  - Repeat until one of them accepts

Some Non-Recognizable Languages

- $COHALT = \{< M, w >| M \text{ does not halt on } w\}$
- $ALL_{TM} = \{< M > | L(M) = \Sigma^*\}$
  - Reduce $COHALT$ to ALL_{TM}

Some Non-Recognizable Languages

- $COHALT = \{< M, w > | M \text{ does not halt on } w\}$
- $ALL_{TM} = \{< M > | L(M) = \Sigma^*\}$
  - Reduce $COHALT$ to ALL_{TM}

Reduce $COHALT$ to ALL_{TM}

- Use a recognizer for ALL_{TM} to recognize $COHALT$
- Given a $COHALT$ instance (i.e. M, w), build a new machine M' such that $L(M') = \Sigma^*$ if and only if $M(w)$ runs forever

Psuedocode for M'

```
public static boolean mPrime(string x){
        count = 0;
        while(M(w) hasn't halted){
                if(count > x){
                        return true;
                }
                run M(w) for 1 step;
        }
        return false;
}
```

If $M(w)$ halts, there is a longest string I can accept

The only way to accept all strings is for $M(w)$ to run forever

Reduction

- Given an instance M, w of $COHALT$
- Use M, w to build M'
- As the recognizer for ALL_{TM} if $L(M') = \Sigma^*$
- Its answer tells us if $M(w)$ runs forever