Tadeusz Dudkiewicz (td370782)

1.1

Yes, machines fulfilling this restriction recognize same class of languages as regular Turing machines. Obviously they can't recognize more languages as they are Turing machines with the restriction. It's enough to show that each regular Turing machine can be simulated on the restricted machine.

I'm going to simulate a regular tape on two restricted tapes. On the first one I'll keep content of cells from the orginal tape which are on the left side of a head and on the second one content of cells on the right side of the head (in a reversed order). Head on 1st tape is over last symbol in left side of orginal tape, head on 2nd tape is over the last non-blank symbol. When simulating, the symbol under head on the 1st tape corresponds to symbol under the head on the original tape.

I'll use special 'new blank' symbol to replace all occurances (in reads and writes) of blank symbol in the orginal machine. I'll also asume there is a special symbol 'beginning of the tape' which head can read, but can't move left after reading it.

Example orginal tape:

```
> A B C D E F
      ^
```

Corresponding restricted tapes:

```
> A B C
      ^

> F E D
      ^
```

When in the orginal machine the head moves to the right writing symbol A on the tape, the new machine writes symbol A on 1st tape. If head on 2nd tape has a symbol different from blank 'begging of the tape' under it the restricted machine writes this symbol on blank under head on 1st tape (without moving this head) and moves had on 2nd tape to the left (wrting blank over that symbol). When there is 'beginning of the tape' under head on 2nd tape the restricted machine writes 'new blank' symbol on 1st tape.

When in the orginal machine the head moves to the left writing symbol A on the tape, the new machine moves head over 2nd tape one position to the right and then writes symbol A on it. Head on 1st tape just moves to the left (writing blank over symbol A).

Apart from two tapes for each working tape of orginal machine the new machine has two additional tapes for input tape. Before running the machine input tape is copied into this pair of tapes (moving head to the left of input and than writing it backwards on 2nd tape of pair) and any subsequent reads of input tapes work on this pair of tapes, so it is possible to move head to the left when reading the input.

1.2

I'm going to assume that representation of set with repetitions is correct (e.g. "0$1#0$0#0$1" is a correct rectangle with A = 0 and B = 0, 1). If it's incorrect it's easy to check for duplicated vectors before running my solution using constant number of counters (each of which needs at most log(input size) number of bits):

```
for starting position i (from 0 to size of input):
  if i isn't a start of a vector:
    continue with next value of i

  for starting position j (from 0 to size of input):
    if j isn't a start of a vector or i == j:
      continue with next value of j

    for length k (from 0):
      if on both i+k and j+k there is \$ or \# or blank then:
        reject word (a duplicated vector was found)
      if on i+k and j+k there are different symbols then:
        exit this inner loop (vectors are different)
```

where by testing whether i isn't a start of a vector I mean checking whether i != 0 and there isn't # on position i-1.

The first step should be checking wheather input is valid, i.e if between each pair of symbols $, # or the end of input there is at least one 0 or 1 (this can be easly done by looking at consecutive symbols on input tape), there are no leading zeros (does any 00 follows $ or #) and wheather between all # there is the same number of $ (which can be done by counting number of $ before first # and then comparing it to number of $ between subsequent # and the end of input - with two counters and in logarithmic space).

Machine also accepts empty set (empty input), as it's a valid rectangle.

In over cases I'm going to iterate over all possible lengths of first vectors l and check whether it's possible that first set was composed of vectors of length l. I'll do this by iterating over all vectors which have different prefix of length l (where l isn't a length in symbols, but the number of integers in it's represenation) as 1st vector. Then for each such prefix I'm going to iterate over all vectors with same prefix as 1st vector, take their suffixes and check if there exists a vector with tested prefix and this suffix (and vice-versa). Obviosuly relation of 'have same set of suffixes' is transitive and by checking 1st prefix with others I'll ensure that all prefixes have same suffixes sets and thus given word represents a rectangle wich A set composed of vectors of length l.

This solution uses counters each taking value from range from 0 to length of input and thus uses logarithmic space. Since the number of counters is constant the solution runs in a logarithmic space. Every loop is either bounded by length of input.

Pseudocode:

```
# Subroutines that are inlined in the machine:

samePrefix(i, j, length):
  check_length = offset = 0
  while check_length < length:
    if on both offset+i and offset+j there is $:
      increase check_length
    elif on both offset+i and offset+j there are # or blank then:
      if check_length + 1 != l then:
        reject a word (different sizes of sets)
      increase check_length
    elif on offset+i and offset+j there are different symbols then:
      return false
    increase offset
  return true


sameSufix(i, j, skip_length):
  i_offset = j_offset = i_skipped = j_skipped = 0

  # skip skip_length vectors on i
  while i_skipped < skip_length:
    if on i+i_offset there is $ then:
      increase i_skipped
    elif on i+i_offset there is # or blank then:
      reject word (bad size of a set)
    increase i_offset

  # skip skip_length vectors on j
  while j_skipped < skip_length:
    if on j+j_offset there is $ then:
      increase j_skipped
    elif on j+j_offset there is # or blank then:
      reject word (bad size of a set)
    increase j_offset

  # check whether rest (until # or bank) is the same
  for offset (from 0 to size of input):
    if on both i+i_offset+offset and j+j_offset+offset there is # or blank then:
      return ture (both sets have ended)
    elif on i+i_offset+offset or on j+j_offset+offset there is # or blank then:
      return false (different sizes of sets or lengths of some vector)
    elif on i+i_offset+offset and j+j_offset+offset there are different symbols then:
      return false (different vectors)
  return true
```

```
# Main pseudocode

# compute number of vectors in the first set:
max_length = 0
for position i (starting from 0):
  if on i there is $ on the tape:
    increase max_length
  if on i there is # or blank on the tape:
    increase max_length
    break this loop

# try all possible solutions
for possible length of vectors in first set l (from 1 to max_length - 1):
  for starting position i (from 0 to size of input):
    if i isn't a start of a vector or samePrefix(0, i, l):
      continue with next value of i

    for starting position j (from 0 to size of input):
      if j isn't a start of a vector or not samePrefix(0, j, l):
        countine with next value of j
      found = 0
      for starting position k (from 0 to size of input):
        if k isn't a start of a vector:
          continue with next value of k
        if samePrefix(i, k, l) and sameSufix(j, k, l):
          found = 1
          break this loop
      if found = 0 then:
        continue the outermost loop with next value of l

    for starting position j (from 0 to size of input):
      if j isn't a start of a vector or not samePrefix(i, j, l):
        continue with next value of j
      found = 0
      for starting position k (from 0 to size of input):
        if k isn't a start of a vector:
          continue with next value of k
        if samePrefix(0, k, l) and sameSufix(j, k, l):
          found = 1
          break this loop
      if found = 0 then:
        continue the outermost loop with next value of l
  accept word (all checks passed for current value of l)
reject word (none value of l passed all tests)
```