COMP30026 Models of Computation

Recognisable and Decidable Languages

Lecture Week 11. Par

Semester 2, 2021

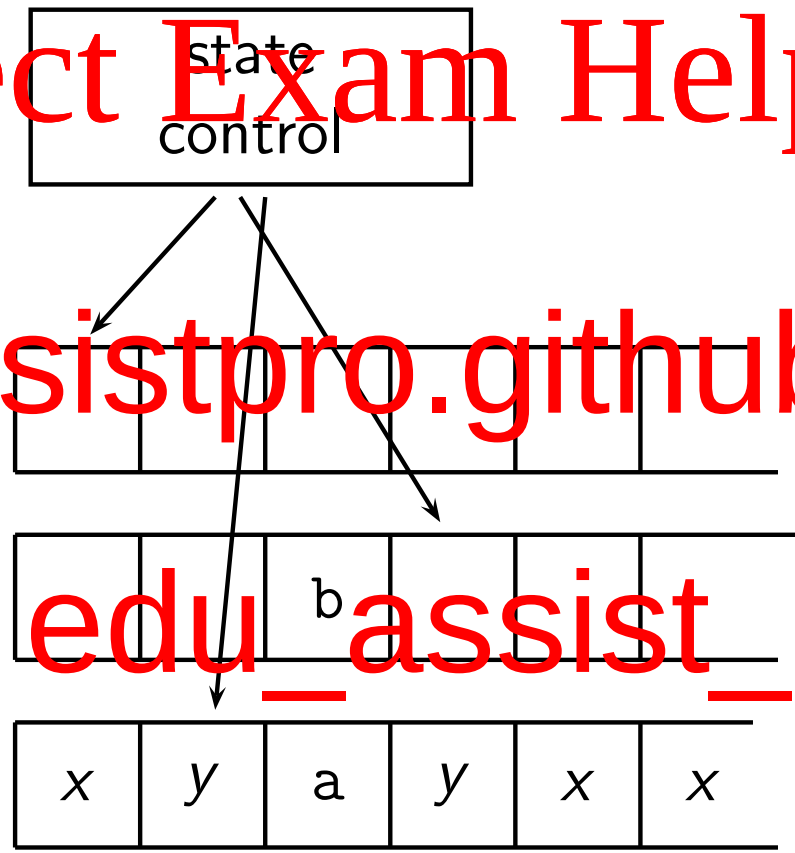
Multitape Machines

A multitape Turing machine has k tapes. It takes its input on tape 1, other tapes are blank.

The tran
type

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$$

It specifies how the k tape heads behave when the machine is in state q_i, reading $a_1, \ldots a_k$:



$$\delta(q_i, a_1, \ldots, a_k) = (q_j, (b_1, \ldots, b_k), (d_1, \ldots, d_k))$$

Simulating a Multitape Machine

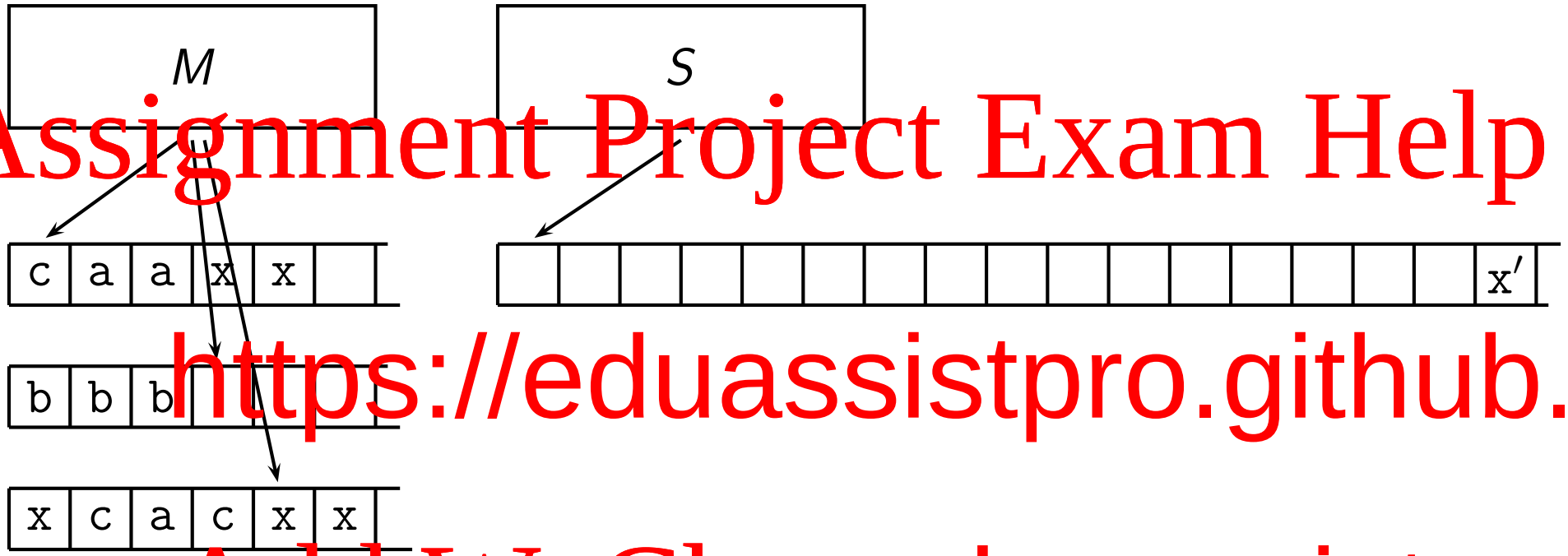
Theorem: A language is Turing recognisable iff some multitape Turing machine recognises it.

Proof sk M by a standar

Suppose the multitape machine's tape alphabet i

The standard machine has tape alphabet s a separator, not in $\Gamma \cup \Gamma'$, and there is some one- between elements in Γ and elements in Γ'.

Simulating a Multitape Machine



S reorganises input $x_1 x_2 \cdots x_n$ into $\# \underbrace{\sqcup' \#}_{k-1 \text{ times}}$

Note how elements of Γ' represent "marked" elements from Γ.

Simulating a Multitape Machine

Simulating an M move, S scans its tape to determine the marked symbols M's transiti

If a "virtual head" of M moves to a $\#$, S shifts that symbol, and every symbol after it, one cell to the right. In the vacant c writes $\sqcup$.

Nondeterministic Turing Machines

A nondeterministic Turing machine has a transition function of type

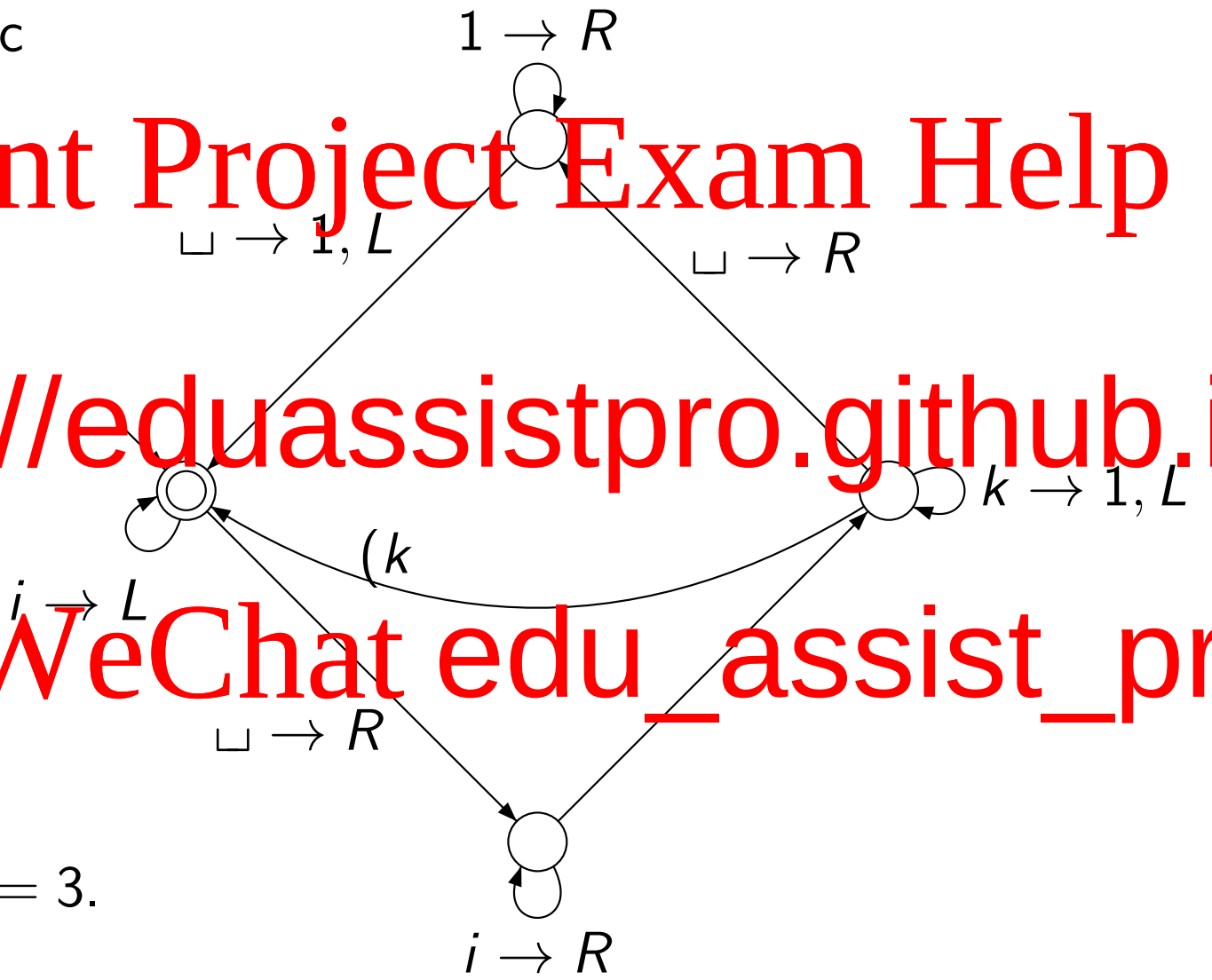
If some

accepts its input.

This is the same type of nondeterminism that an NF

Simulating a Nondeterministic Turing Machine

First, a deterministic machine to generate $\{1, \ldots, k\}^*$, in order of increasing length.



Try running it for $k = 3$.

Simulating a Nondeterministic Turing Machine

Theorem: A language is Turing recognisable iff some nondeterministic Turing machine recognises it.

Proof sketch: We need to show that every nondeterministic Turing machine N can be simulated by a deterministic Turing machine D.

We show h

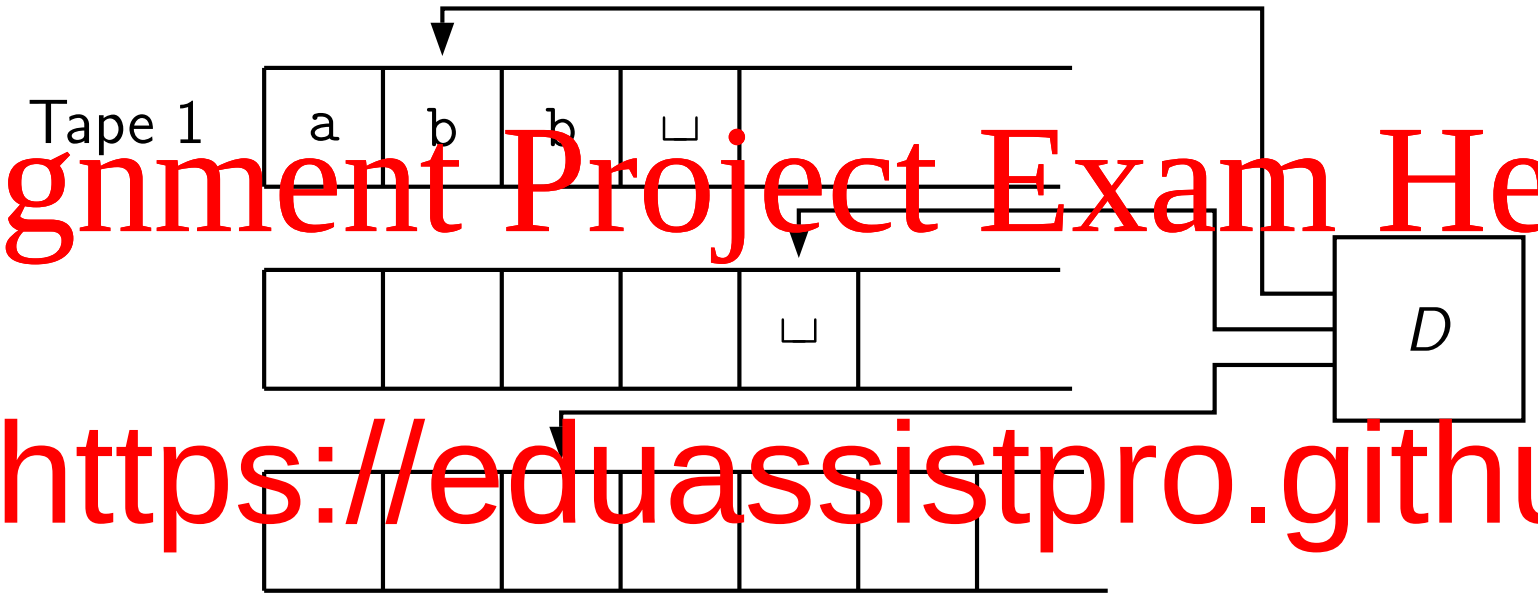
Let k be the largest number of choices, according to N's transition function, for any state/symbol combination.

Tape 1 contains the input.

Tape 3 holds longer and longer sequences from $\{1, \ldots, k\}^*$.

Tape 2 is used to simulate N's behaviour for each fixed sequence of choices given by tape 3.

Simulating a Nondeterministic Turing Machine



1. Initially tape 1 contains input w. The
2. Overwrite tape 2 by w.
3. Use tape 2 to simulate N. Tape 3 dictates how N should make its choices. If N says accept, accept. If the "choice list" on tape 3 gets exhausted, go to step 4.
4. Generate the next choice list on tape 3. Go to step 2.

Enumerators

The Turing machine we built to generate all strings in $\{1, \ldots, k\}^*$ is an example of an enumerator.

We could i rint all the stri .

For an enumerator to enumerate a language , it must eventually print w.

The reason why we also call Turing recognisable la ursively enumerable is the following theorem.

Enumerators

Thm: L is Turing recognisable iff some enumerator enumerates L.

Proof: Let E enumerate L. Then we can build a Turing machine recognising L as follows:

1. Let
2. Si

Enumerators

Thm: L is Turing recognisable iff some enumerator enumerates L.

Proof: Let E enumerate L. Then we can build a Turing machine recognising L as follows:

1. Let
2. Si

Conversely, let M recognise L. Then we can b E by elaborating the enumerator from a few slides bac enumerate Σ^*, producing $s_1, s_2, \ldots$

Enumerators

Thm: L is Turing recognisable iff some enumerator enumerates L.

Proof: Let E enumerate L. Then we can build a Turing machine recognising L as follows:

1. Let
2. Si

scept.

Conversely, let M recognise L. Then we can b E by elaborating the enumerator from a few slides bac enumerate Σ^*, producing $s_1, s_2, \ldots$. Her

1. Let $i = 1$.
2. Simulate M for i steps on each of $s_1, \ldots s_i$.
3. For each accepting computation, print that s.
4. Increment i and go to step 2.

Partial Functions

We have silently assumed that the domain of a function is known, so that $f : X \to Y$ means that $f(x)$ is defined for each $x \in X$.

In compu o deal with func

We write $f : X \hookrightarrow Y$ to say that f has a d
X, but $f(x)$ may be undefined for some

Note that a total function $f : X \to Y$ i
$f : X \hookrightarrow Y$.

Partial Functions: Example 1

The function f defined by

$$f(n) = \begin{cases} 42 & \text{if } n = 0 \\ f(n-2) & \text{if } n \neq 0 \end{cases}$$

is a par

In a natura gative.
Its range is $\{42\}$.

In this case, it is not too hard to determine the set of values for which f is defined. So we could also choose to say th total function $X \to \mathbb{Z}$, where $X = \{n \in \mathbb{Z} \mid n \geq 0 \wedge n \text{ is even}\}$.

However, it is not always easy, or even possible, to determine a function's domain.

Partial Functions: Example 2

The function c defined by

$$1 \quad \text{if } n = 0 \text{ or } n = 1$$

is a partial f

Will the evaluation of $c(n)$ terminate for all

Partial Functions: Example 2

The function c defined by

$$c(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } n = 1 \end{cases}$$

is a partial f

Will the evaluation of $c(n)$ terminate for all n?

It is not known whether c is total.

This is the so-called $3n + 1$ problem, or Collatz's problem.

Program Termination

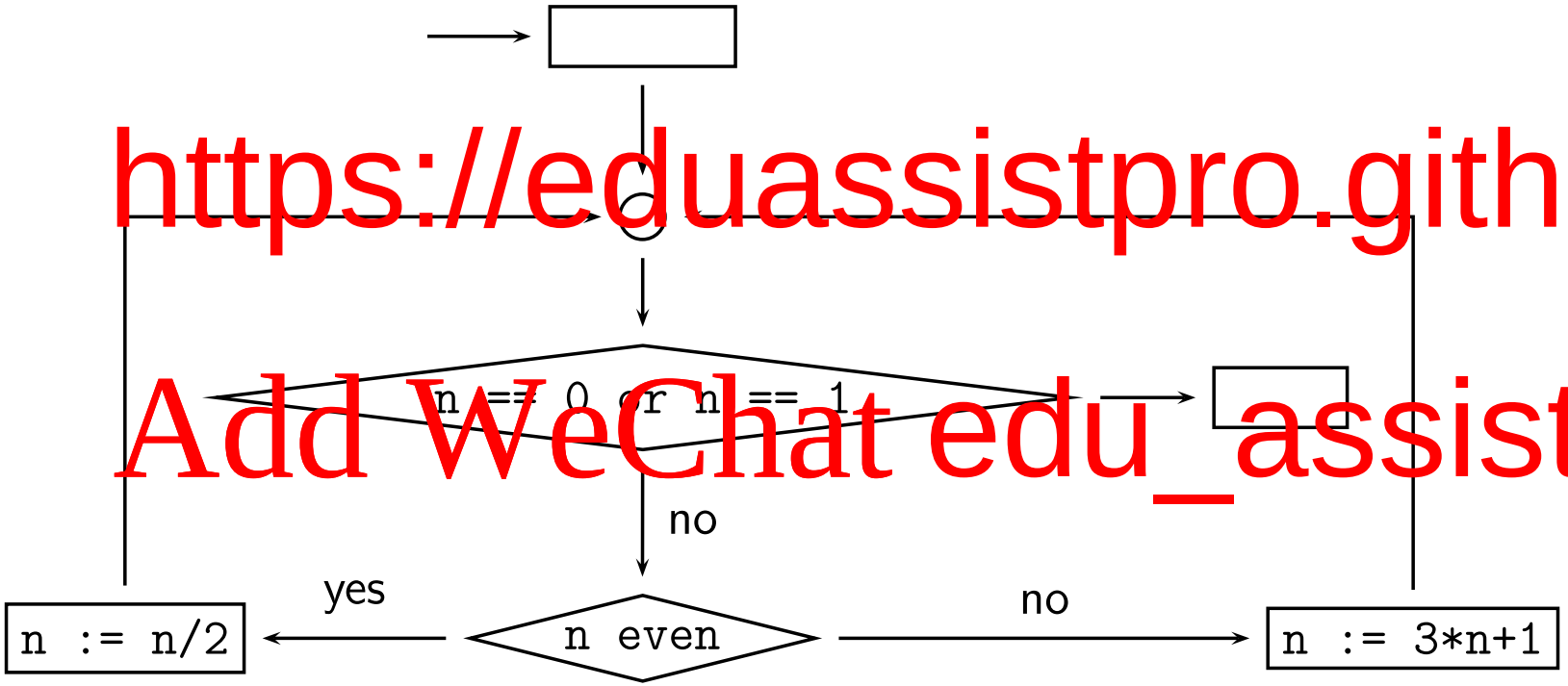
Here is a Haskell program that produces the list of n-values generated in successive recursive calls:

```
c :: Integ
c 0 = [1]
c 1 = [1]
c n = n : c (if even n then n `div` 2 else 3*n+1)
```


Colatz's sequence for 27: 27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107
412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1.

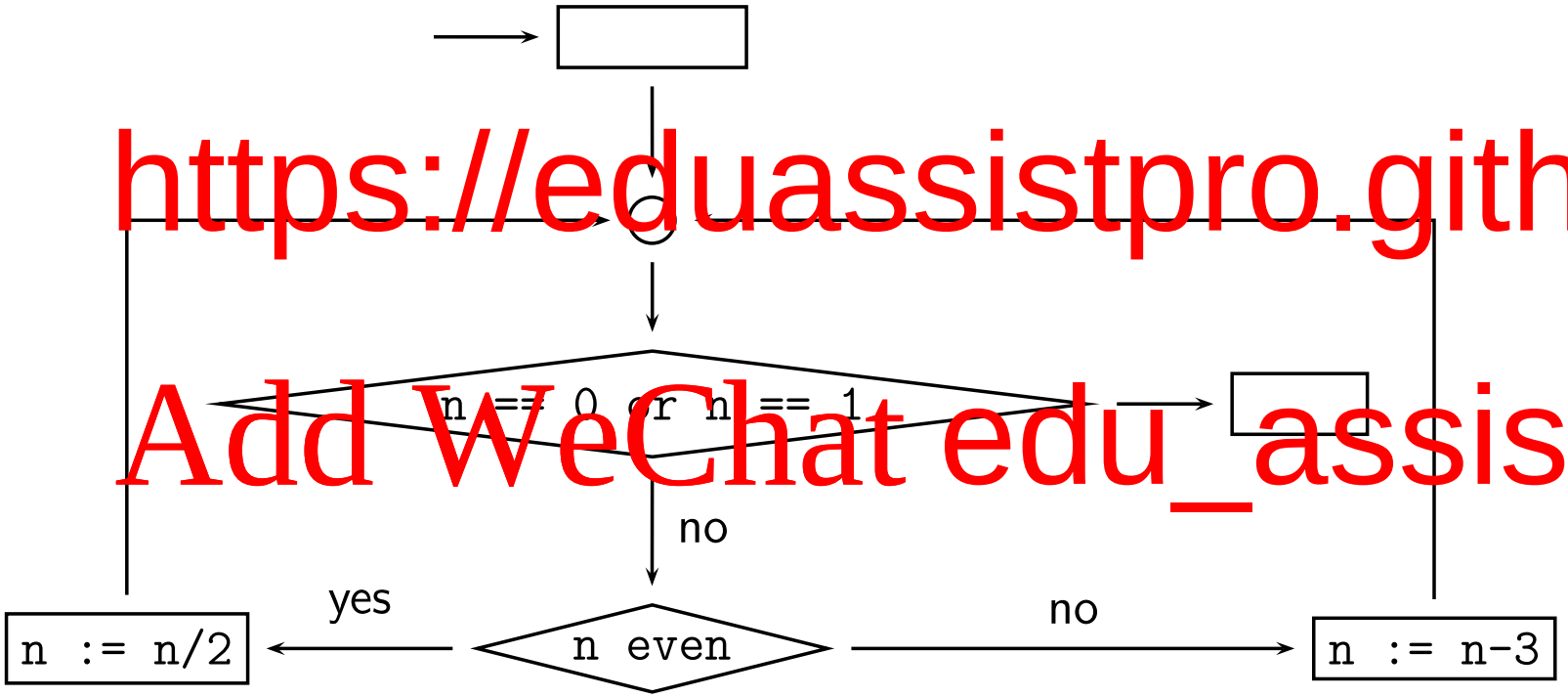
Program Termination

Here it is as a flowchart program:



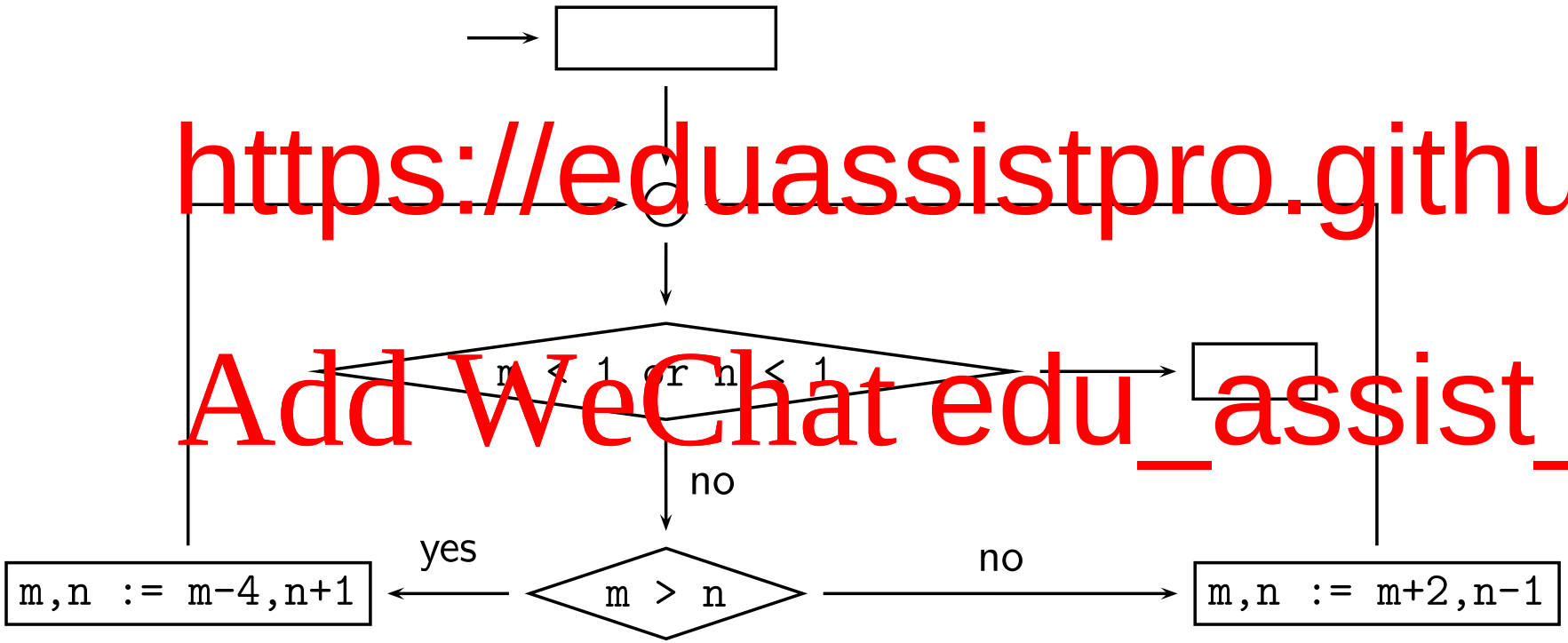
Quiz 1: Does It Terminate?

Here is a variant. Does this halt for all positive input?



Quiz 2: Does It Terminate?

And how about this? Does it halt for all input?



Proving Termination

Suppose that, for each loop in a program, we can find some "measu

1. the
2. the

Then the program must terminate for all input, bec ral number cannot be made smaller indefinitely.

The Termination Question

Termination of algorithms is a tricky problem (and the general problem is undecidable).

For example, we suggested earlier algorithms for translating propositional formulas to, say, DNF, including rules like

$$\neg\neg\alpha \rightsquigarrow \alpha$$

$$\alpha \wedge (\beta \vee \gamma) \rightsquigarrow (\alpha \wedge$$

$$(\beta \vee \gamma) \wedge \alpha \rightsquigarrow (\beta \wedge$$

Note that some of the rules decrease the size of a term, while others increase it (and some duplicate certain subterms).

So why should this process terminate?

Quiz 3: A Marble Game

You have a bag of red and white marbles, plus a box of red marbles.

Repeat this process:

- If th
  oth
- If they are of different colours, put them back.
- If they are of the same colour, discard both, but put one red marble (from the box) in the bag.

Does this terminate?

Well-Founded Orderings

The binary relation $\prec$ over some set X is well-founded iff there is no infinite sequence of X-elements $x_1, x_2, x_3, \ldots$ such that

We say th

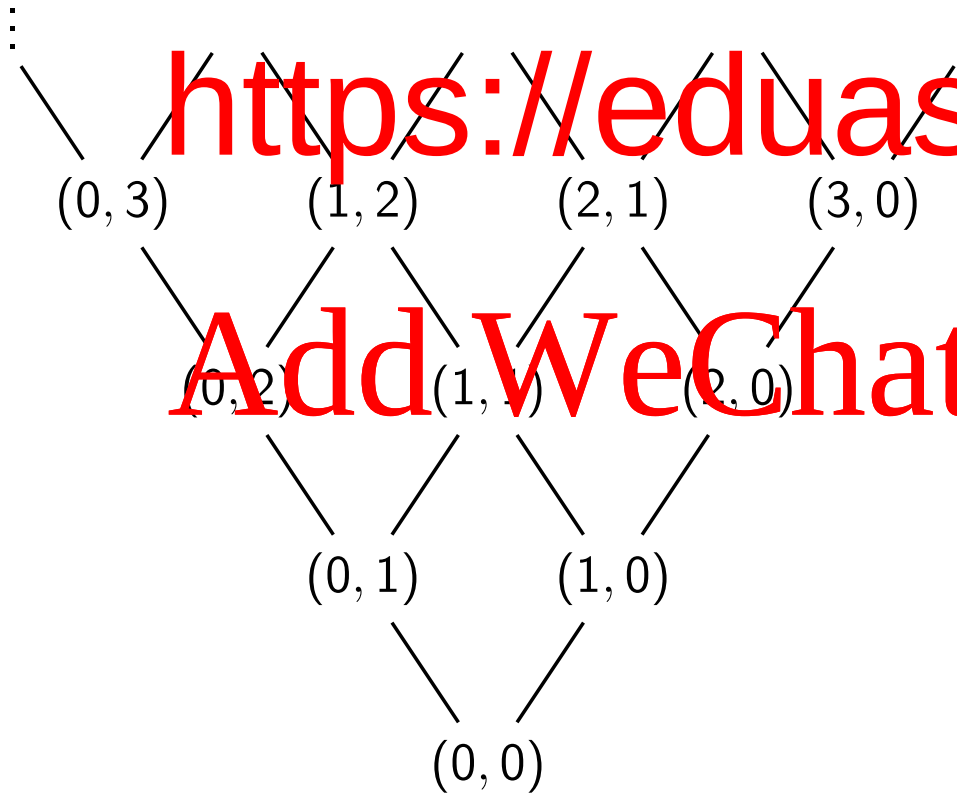
For example, $(\mathbb{N}, <)$ is a well-founded structu

Given a finite number of well-founded structures $(X_1, \prec_1), \ldots, (X_n, \prec_n)$, we can obtain well-founded orderings of $X = X_1 \times \cdots \times X_n$ in a number of different ways.

Ordering Pairs: Component-Wise Ordering

$$(x_1, x_2) \preceq (y_1, y_2) \text{ iff } x_1 \leq y_1 \wedge x_2 \leq y_2$$

$$p \prec q \text{ iff } p \preceq q \wedge p \neq q$$



Values increase as you travel up along edges.

Ordering Pairs: Lexicographic Ordering

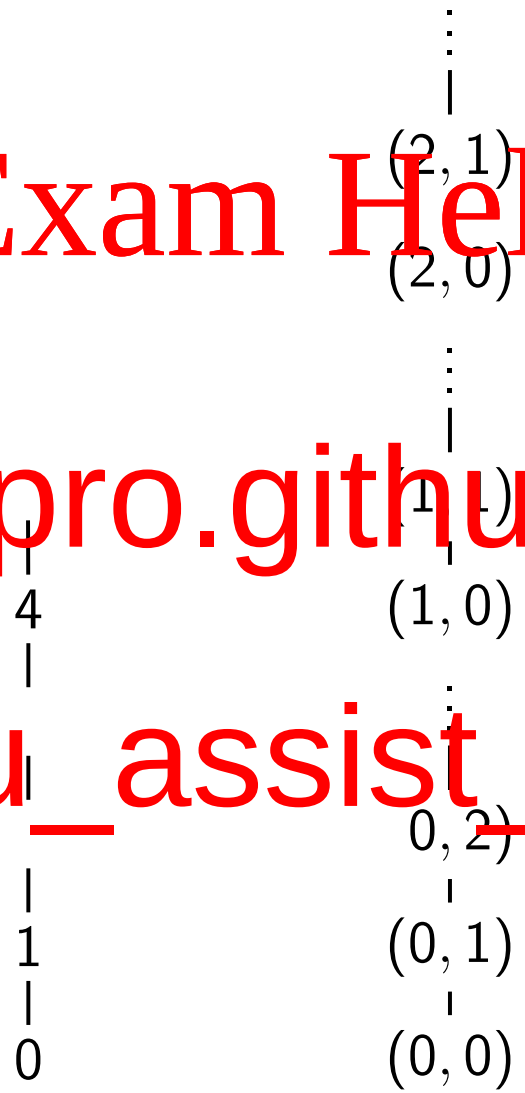
On the left:

On the right:

lexicogr

iff $m \leq m' \wedge (m = m' \Rightarrow n \leq n')$.

Define again $p \prec q$ iff $p \preceq q \wedge p \neq q$.



Well-Founded Orderings on Tuples

Theore

extensio

Theore

to tuples.

Component-Wise Ordering of Tuples

Ordering $X = X_1 \times \cdots \times X_n$ component-wise:

n

If each (i i

Example using component-wise ordering:

$$(2,2,2) \succ (2,2,1) \succ (2,1,1) \succ (2, \quad ,0)$$

Lexicographic Ordering of Tuples

Ordering $X = X_1 \times \ldots \times X_n$ lexicographically:

$$\left(\underset{n}{\qquad} \underset{i-1}{\qquad} y_j \right)$$

If each $(X_i$ $_i$ X

Example using lexicographic ordering:

$(2, 2, 2) \succ (2, 1, 42) \succ (1, 3, 1000) \succ (1, 3, 999) \succ (1, 3, 0) \succ (0, 0, 15)$

Well-Founded Induction

There is an induction principle to go with well-founded relations.

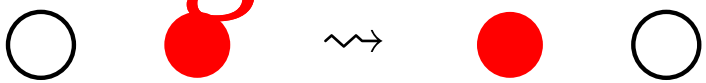
Given a we
"for all

We proceed in one step:

- Assume that $S(x')$ holds for all x' and use that to establish $S(x)$.

Example 1: The Dutch Flag

We have 12 pebbles laid out in a row. Consider three rewrite rules


(for a white left of a red, swap)

To see that rewriting terminates, use together with lexicographic ordering on 12-tuples.

Example 2: Ackermann's Function

The following is a definition of Ackermann's function:

$$ack(0, y) = y + 1$$

It grows in

For example, $ack(3, 3) = 61$ and $ack(4$

However, lexicographic well-founded indu
that the function is total—as a Haskell program it will terminate for all input (possibly after a very long time).

Example 2: Ackermann's Function

In each recursive call, the argument (x, y) decreases strictly.

$$ack(x+1, y+1) \quad = \quad ack(x, \quad ack(x+1, y) \quad)$$