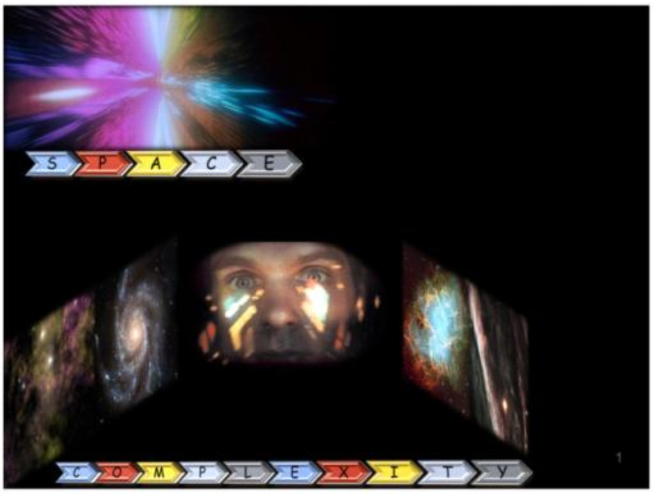
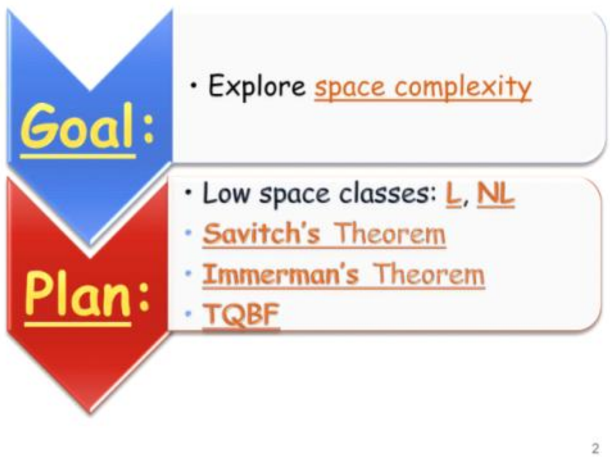
Space Complexity Notes



In this presentation we take a closer look at complexity classes in which the bound is on the amount of memory it takes to compute the problem.

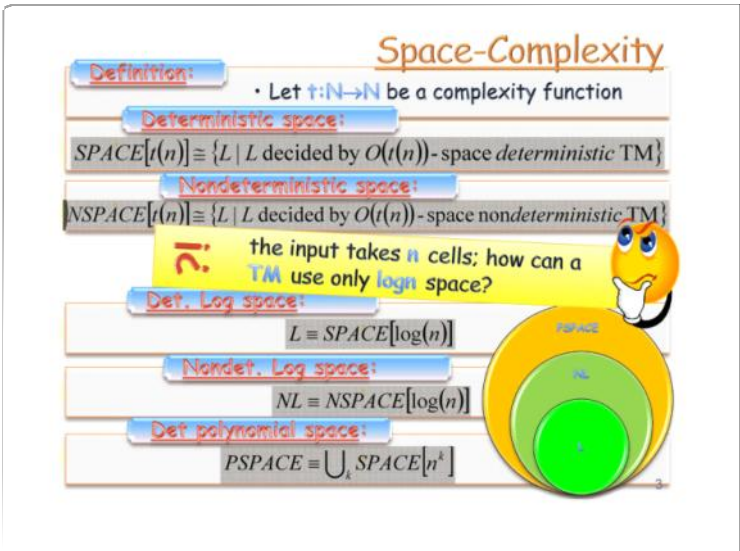


In particular, we'll look at low complexity classes, such as

- LOGSPACE
- and non-deterministic LOGSPACE.

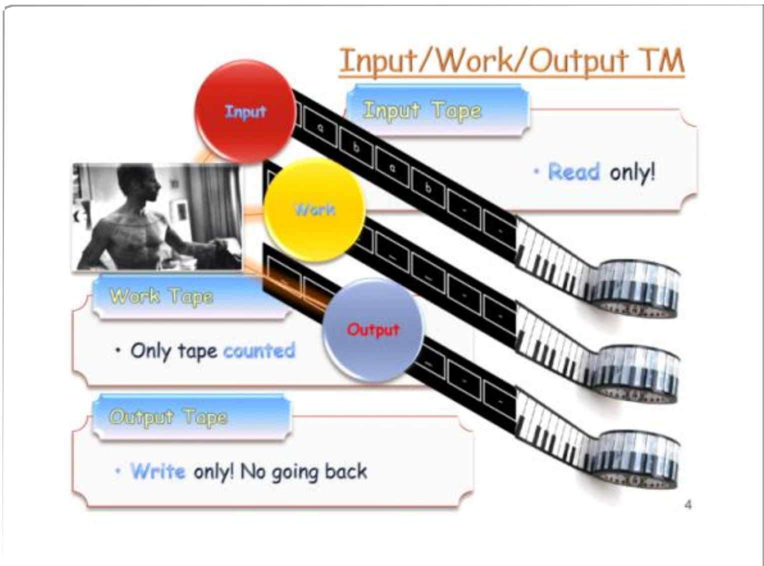
Among others, we prove three fundamental theorems regarding those classes.

[Space Complexity](#)
[Savitch's Theorem](#)
[Immerman's Theorem](#)
[TQBF](#)



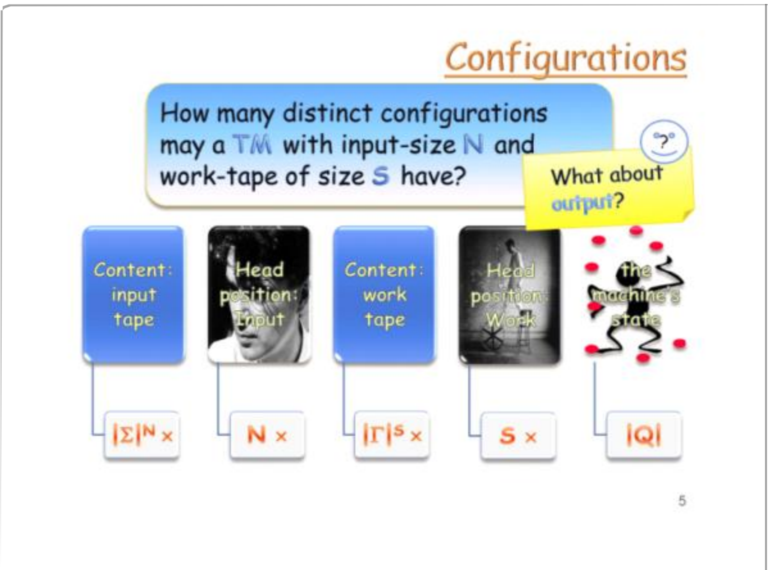
Let us recall our definition of space complexity classes.

It is quite straightforward, however, we need to clarify what it mean for an algorithm to use sub linear space.



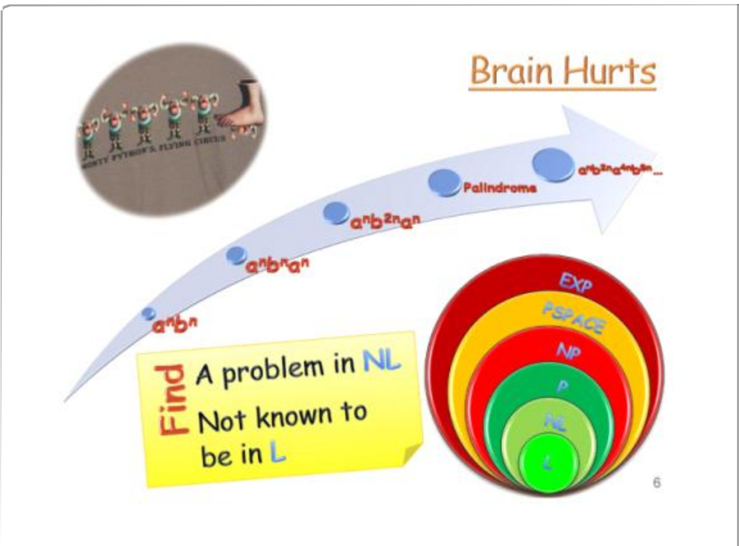
For that purpose, we change a little our model of computation to consist of

- An input tape, which is read only,
- An output tape, which is write only,
- And a work tape, which is the only one counted for purposes of complexity bounds.



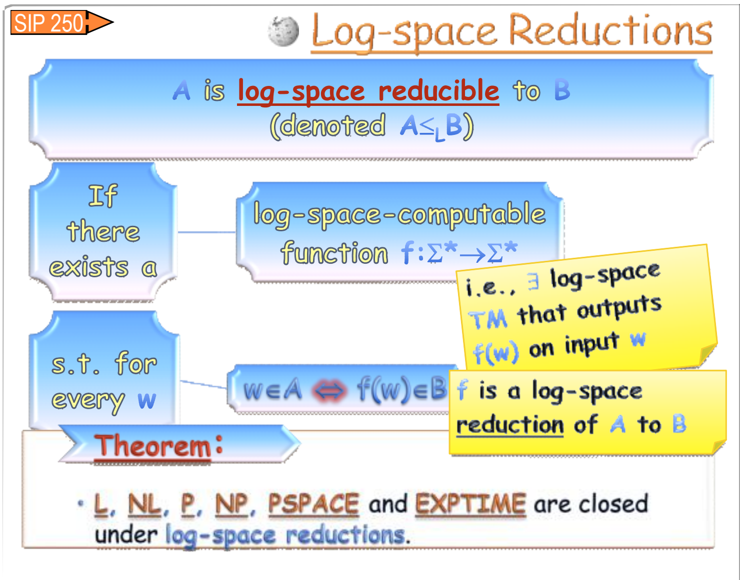
Let us now figure out how many configurations such a machine has:

- The location of the heads on the input tape and on the work tape are counted.
- Both the content of the output tape and the location of the head on it are not considered in counting the configurations.
- The content of only the work tape is counted.

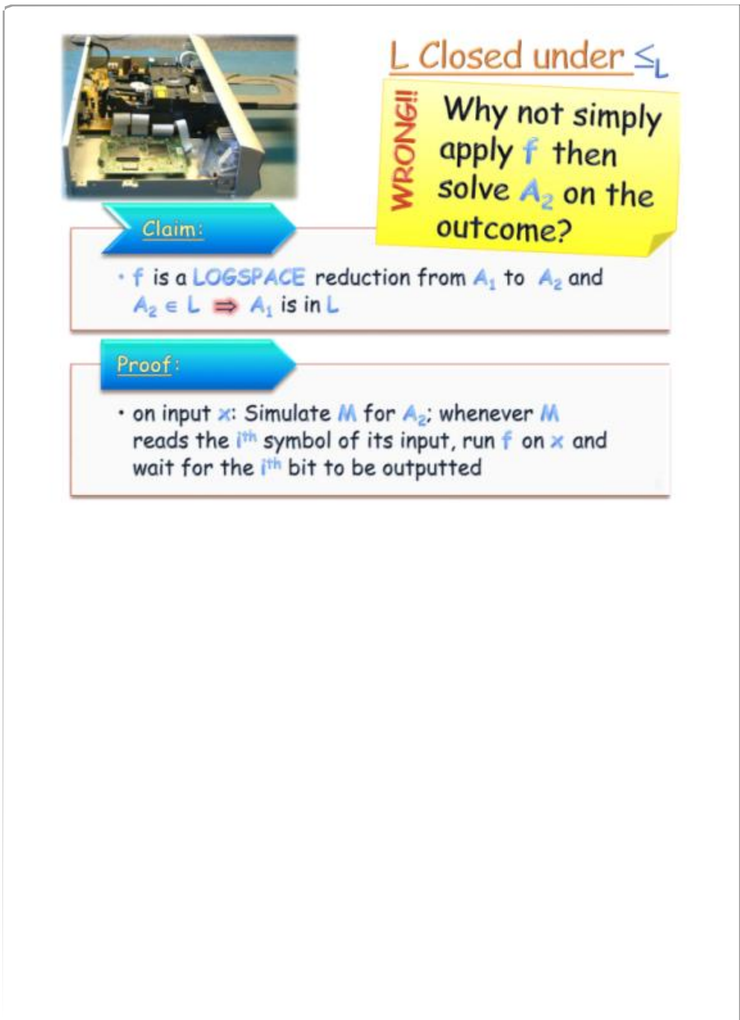


Try to put the following computational problems in as small a class as you can.

Try also to come up with a problem that is in non-deterministic LOGSPACE, however is not known to be in LOGSPACE.



We can now define LOGSPACE reductions: they're the same as Karp reductions, with the added restriction that the reduction-function must be computed using only logarithmic memory.



Let us now see that these reductions can indeed be applied appropriately.

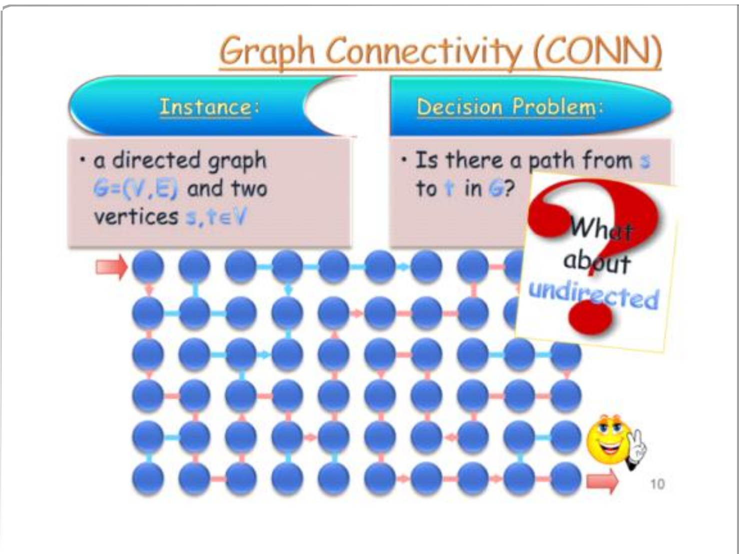
Think of the following scenario: you have a little chip that can play a DVD in a given format. You have a DVD encoded with a different format.

You have another little chip that can convert the format the DVD is written in to the format the other chip can read.

Is it possible to combine the two and build a machine that can play the DVD?

The wrong solution would be to store the output of the first chip and apply the second chip to that -there is simply not enough memory for that solution to work.

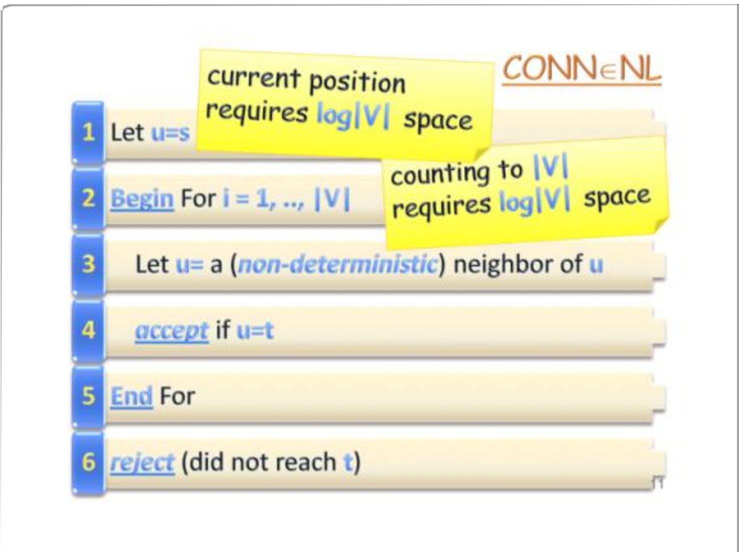
The correct solution is to run the second chip and give it the appropriate bits of the output of the first chip; if necessary, restart the first chip, and let it read the DVD from start.



Let us now formally define the connectivity problem:

Given a graph, a start vertex, and a target vertex, is there a path from start to target?

Q: Do you think the same problem, however on an undirected graph, is easier?

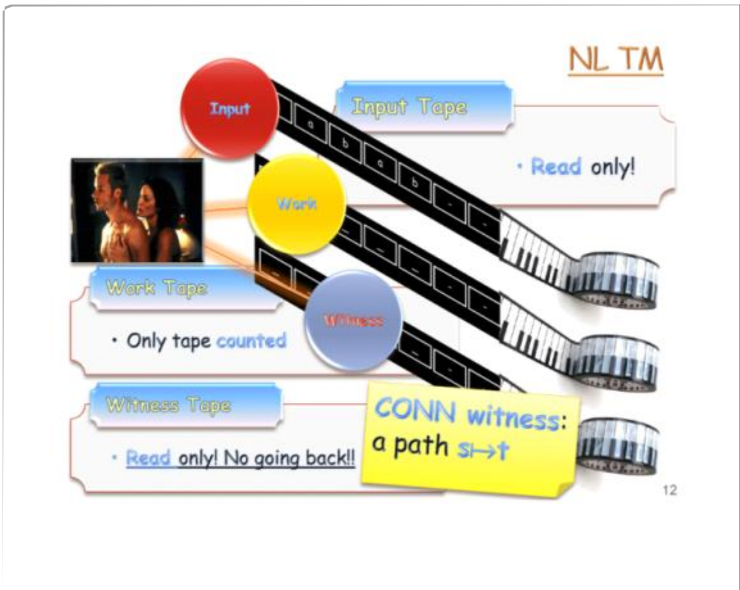


Let us first see that connectivity is in non-deterministic LOGSPACE.

A non-deterministic algorithm for connectivity maintains a pointer to a vertex of the graph.
Initially it points to the start vertex.

At every stage, the algorithm chooses an edge going out of the vertex it points to, and direct its pointer to the vertex the edge leads to.

If it reaches the target, it <u>accepts</u>.
If it went too many stages, it <u>rejects</u>.

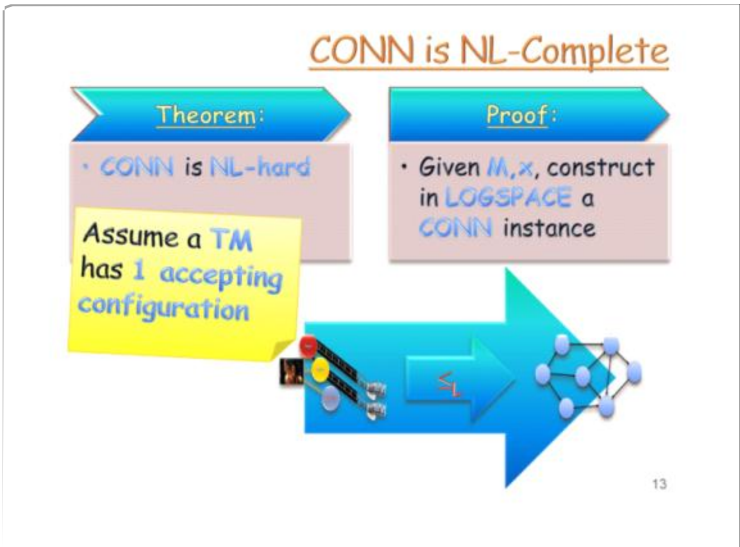


An alternative formulation of non-deterministic space bounded machines is by introducing the <u>witness</u> tape.

The machine can only read that tape and moreover must read it bit by bit and never go back.

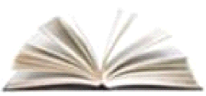
It is enough that there exists one possible assignment to the content of the tape that causes the machine to accept, for the input to be accepted.

Q: What complexity class do we get if we allow the machine to go back on the witness tape?

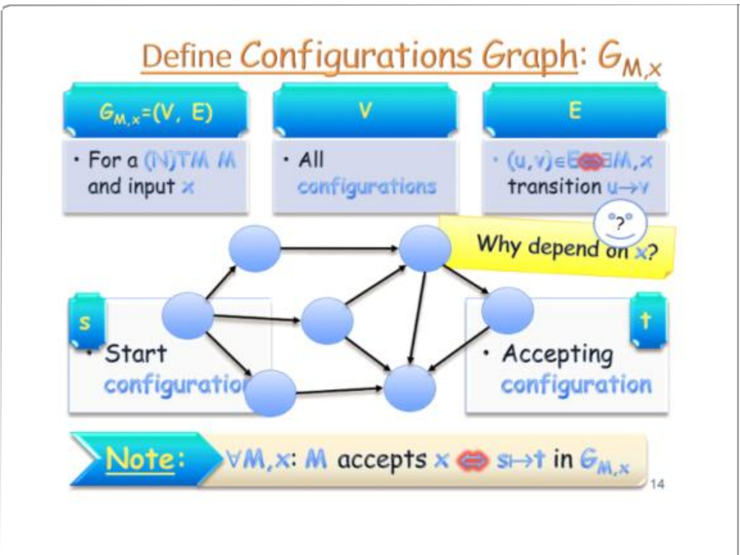


It turns out that connectivity is non-deterministic LOGSPACE complete.

We will show how to construct the connectivity instance given a machine M and input X, so that the machine accepts its input if and only if the instance is in CONN.



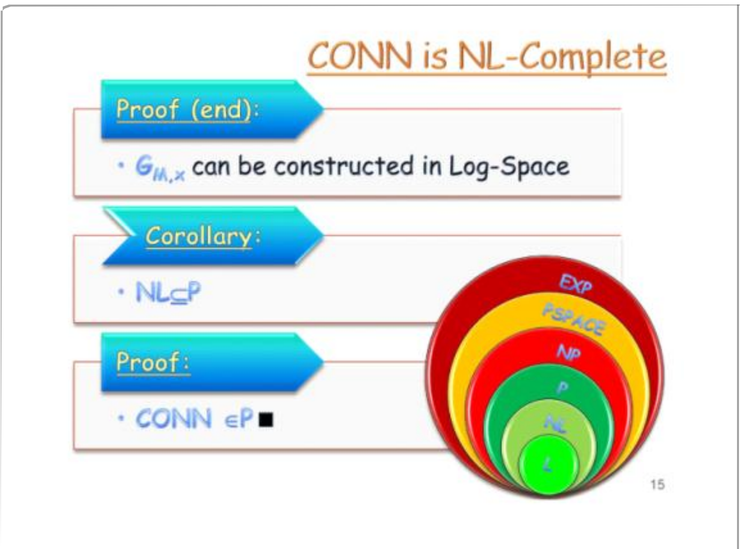
[NL Completeness](#)



For that purpose let us introduce the configurations' graph:

- vertexes correspond to configurations,
- edges to transitions,
- the start vertex correspond to the start configuration,
- and the target vertex corresponds to the accepting configuration.

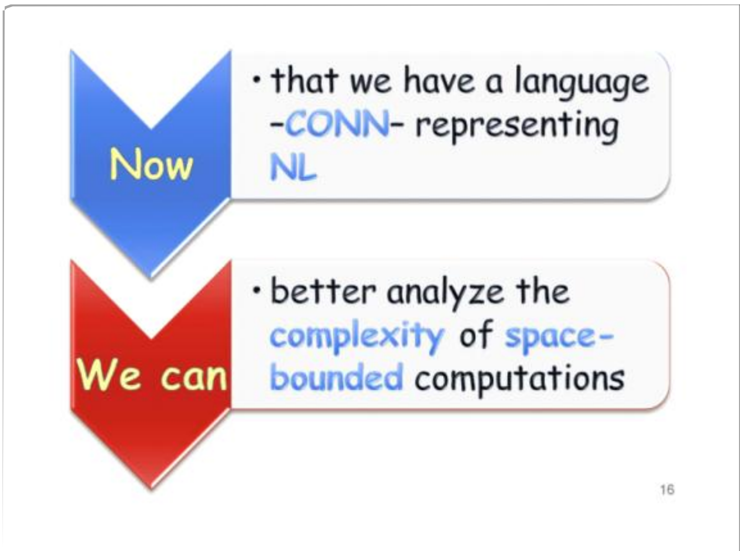
An accepting computation of the machine corresponds to a path from start to target, while such a path clearly corresponds to accepting computation.



Given a non-deterministic LOGSPACE machine, its configuration graph can be computed with logarithmic memory:

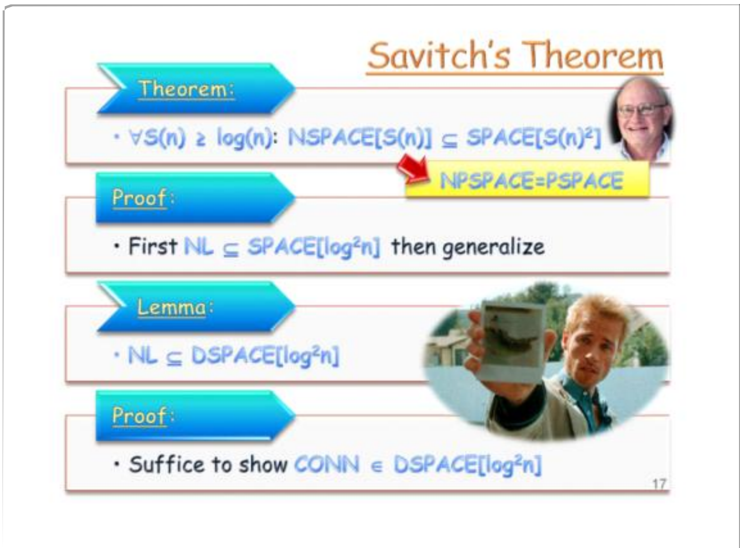
The algorithm simply needs to compute, given two configurations, whether there is a transition from one to the other.

As a corollary we get that non-deterministic LOGSPACE is contained in P.



The fact that connectivity is NL-complete is fundamental in analyzing space complexity classes:

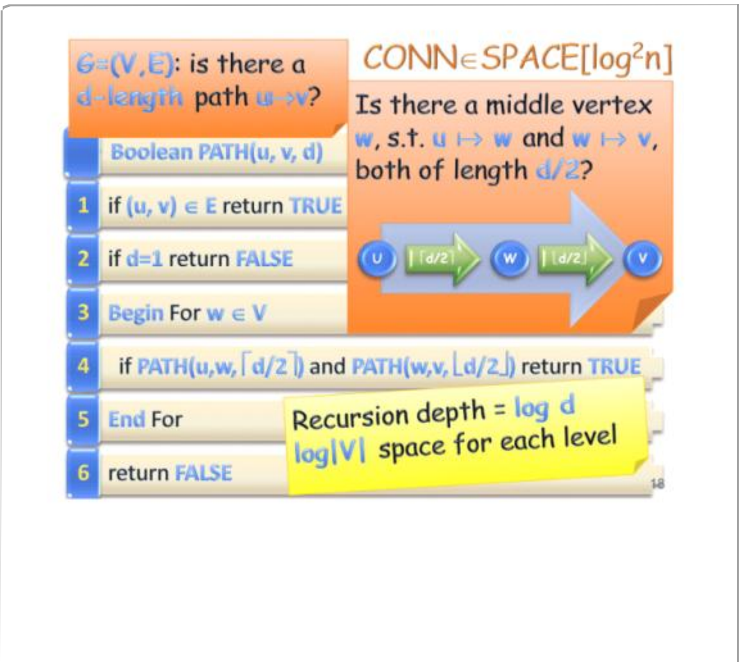
It is crucial in the proof of the following two fundamental theorems we prove.



The first is a theorem by Savitch concerning the overhead involved in converting a non-deterministic computation to a deterministic one.

It turns out that the overhead in terms of space is not that large, it is in fact quadratic.

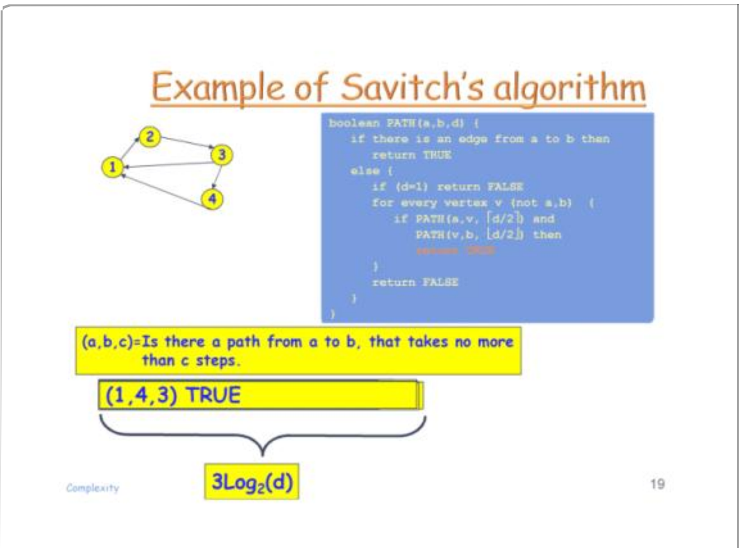
To prove that theorem, we will start with the special case of NL, and proceed to show a general technique of how to extend such statements for small classes to larger classes.



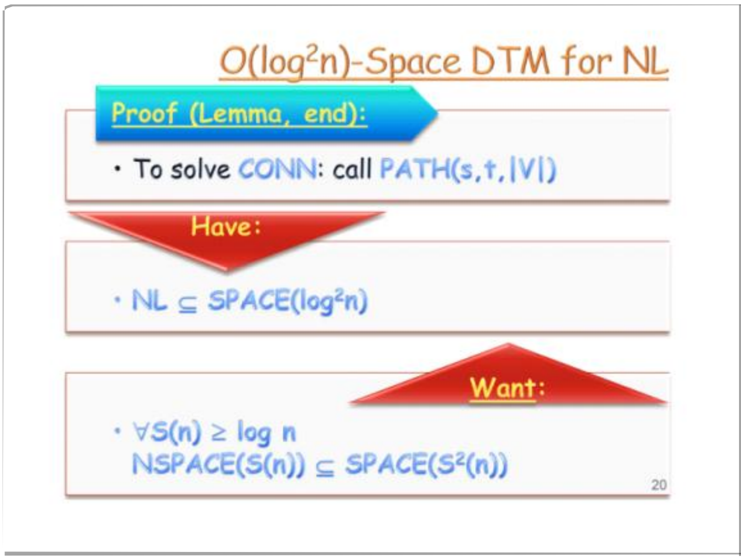
Savitch's deterministic simulation algorithm for connectivity is recursive:

To decide if there is a path of length d, it goes over all possible vertexes for the middle of the path, and call itself to decide whether the appropriate paths of half the lengths exist: one from the start vertex to the middle vertex, and another from the middle of vertex to the target vertex.

The recursion depth is logarithmic in the length of the path, and at each level the algorithm maintains a pointer to one vertex.

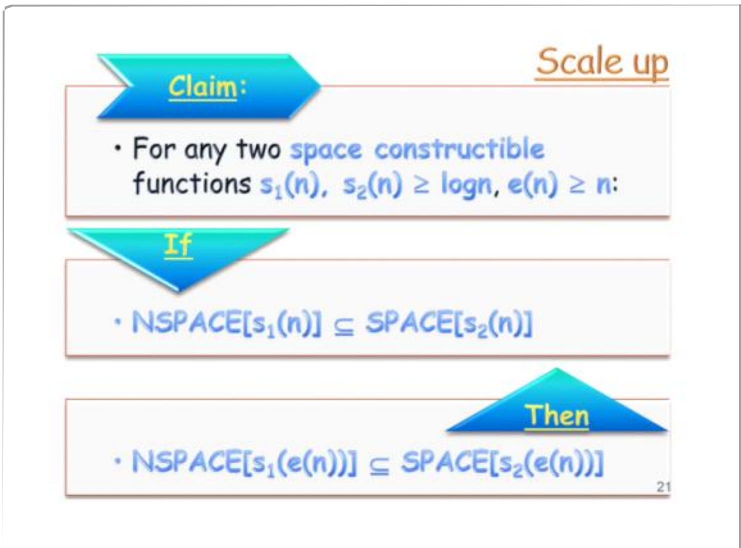


Here is a simulation of the algorithm on a simple example.



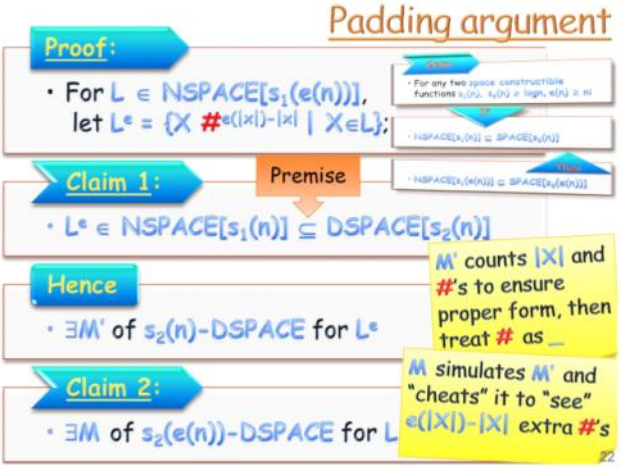
To solve connectivity, one can simply apply the algorithm with the number of vertexes as the length of the path.

Now that we have proven the Theorem for NL, we need to extend it to general classes. Namely, show that for every space bound, the cost of translating a non-deterministic algorithm to a deterministic one is quadratic.



We show a more general principle, that any such relation between models and bounds can be scaled up with a super linear extension function. The extension function scales up both bounds.

This technique is simple yet tricky and is referred to as the <u>padding argument</u>.

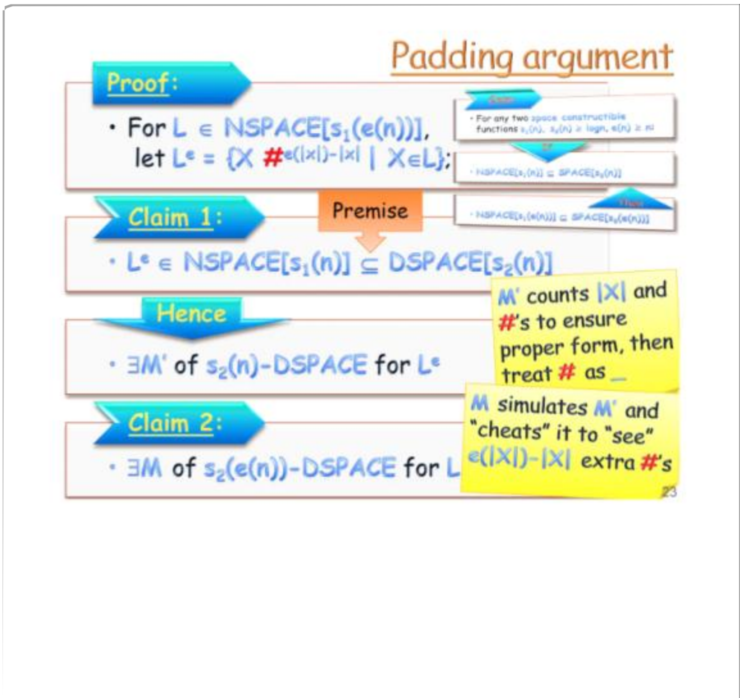


The padding argument goes as follows:

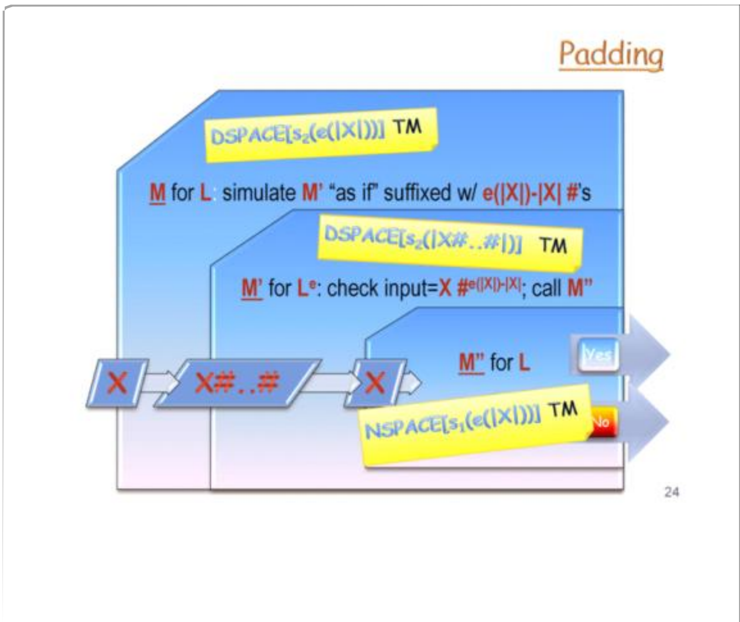
Given a language L, accepted by a non-deterministic TM, define the language L_e that comprises all strings in L padded with the appropriate number of #.
That padding makes the language L_e in the appropriate non-deterministic class.
Now, one can apply the containment of the premise and obtain a determined TM for L_e.
This deterministic TM verifies that the number of #'s is appropriate with respect to the size of the "real" input.

One can in turn, given only the real input, simulate this machine maintaining a counter of the number of #'s, and letting the TM work as if the appropriate number of #'s is appended to the real input.


Padding argument



The padding argument goes as follows: given a language L, accepted by a non-deterministic TM, define the language Le that comprises all strings in L padded with the appropriate number of #. That padding makes the language Le in the appropriate non-deterministic class. Now, one can apply the containment of the premise and obtain a determined TM for Le. This deterministic TM verifies that the number of #'s is appropriate with respect to the size of the "real" input. One can in turn, given only the real input, simulate this machine maintaining a counter of the number of #'s, and letting the TM work as if the appropriate number of #'s is appended to the real input.

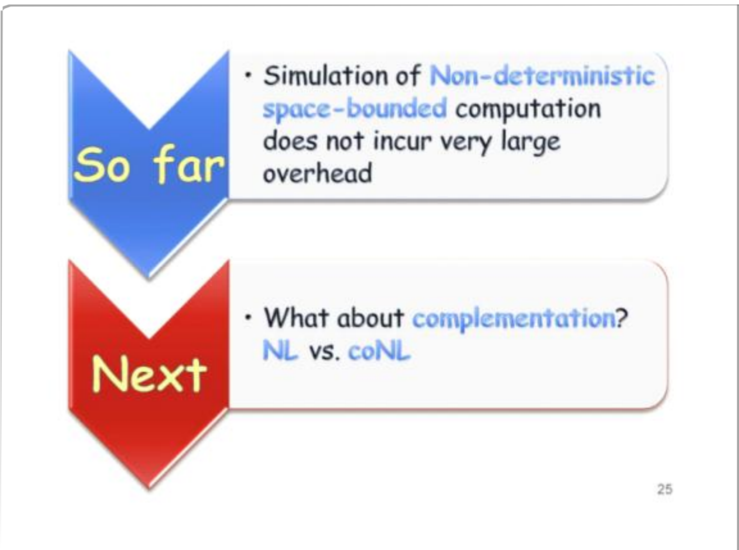


Here's an illustration of the construction:

We start with a TM M" for L, which can be converted into a TM for L_e (checking that the number of #'s is appropriate can be carried out in LOGSPACE), which by the assumption of the premise can be made deterministic --- that's the TM M'.

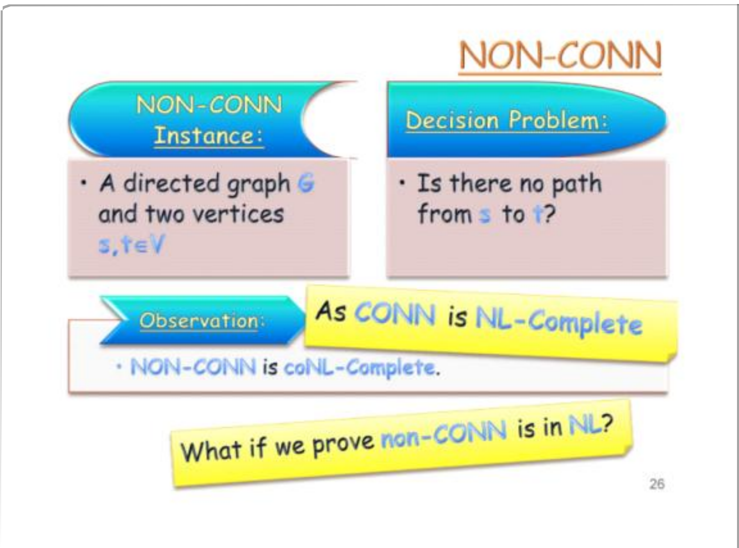
M is a TM for L of appropriate space that simulates M', and if M' wonders off to the # section, it maintains a pointer (it has enough space to do so) to where it is and simulates it as if the #'s are there.

This completes the proof of Savitch's theorem.



We have just seen that enhancing space-bounded computation with non determinism does not make it so much stronger.

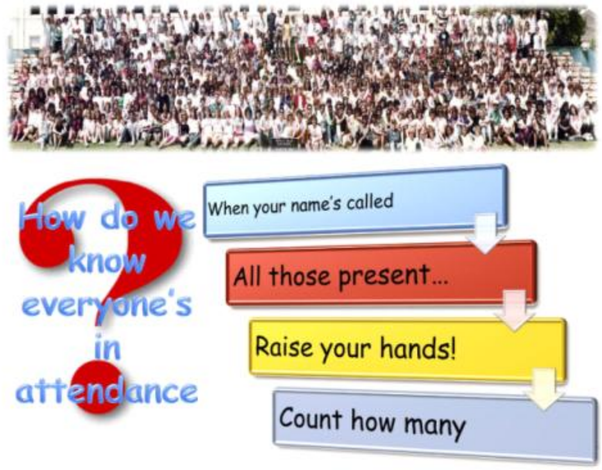
Next, we look at another aspect by which non determinism for space bounded computations has a limited effect.



Let us first define the non-connectivity problem, which is simply the complement of the connectivity problem.

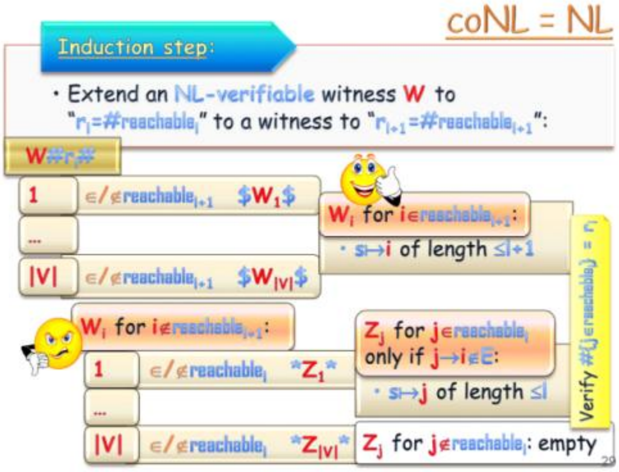
Non-connectivity is clearly coNL-complete, therefore, it represents the entire coNL class.

It follows, that if we show non-connectivity is in NL, we've proven NL=coNL.



To show that non-connectivity is in NL, we can use the witness formulation of NL, where the TM for L reads a witness of membership from left to write and verifies it indeed proves the input is in L.

Given G let us define the set of reachable vertexes, namely those that can be reached by a directed path from the start vertex s.
To show there is no path from s to t, we can show that the size of the reachable set is the same for G and for G only where all edges going into t are removed.
Hence, it is enough to verify a proof showing what is the number of reachable vertexes of a given graph (first have a proof for G, store that number, then verify a proof for the altered graph, and compare the two numbers).
To verify that indeed the number of reachable vertexes is as claimed, the witness can be constructed inductively over the length of the path.
There is obviously exactly one vertex reachable within 0 steps.
We'll next see how to extend a witness, *proving the number of reachable vertexes after l steps is R_l*, into a witness for l+1, and so that if the prefix can be verified by a LOGSPACE TM then so is the entire witness.



W is the witness, proving that the number of reachable vertexes after l steps is R_l.
Let us append to it an array of sub-witnesses, one for each vertex of the graph: the ith segment would first specify whether the ith vertex is or is not reachable within l+1 steps. Next, depending on that bit (and separated by \$ signs) are the corresponding witnesses. Assuming all sub-witnesses are true, the verifier can count to see how many vertexes are reachable within l+1 steps.
In case vertex i is reachable within l+1 steps, the witness would simply be a path from start to vertex i of length at most l+1.
In case vertex i is not reachable within l+1 steps, the sub-witness dedicated for that ith vertex would itself be an array with every segment corresponding to a vertex of the graph. The bit for each vertex j corresponds to whether vertex j is reachable within l steps. Clearly, no vertex j reachable within l steps can have an edge to vertex i; the witness for vertex j reachable within l steps, would be simply a path from start to j of length at most l.
If vertex j is not reachable within l steps the jth sub-witness is left empty.
All sub-witnesses are clearly proving what they claim, and exist --- except for the witness that vertex j is not reachable within l steps.
How then can the verifier be sure that's true?
The answer is the crux of the entire argument and is as follows: the NL TM verifies that the <u>number of vertexes</u> listed as reachable within l steps is <u>exactly R_l</u>, the number proven in W to be the number of reachable vertexes within l steps!

N.D. Algorithm for $reach_S(v, l)$

$reach_s(v, l)$

1. length = l; u = s

2. while (length > 0) {
    3. if u = v return 'YES'
    4. else, for all (u' $\in$ V) {
        5. if (u, u') $\in$ E nondeterministic switch:
            5.1 u = u'; --length; break
            5.2 continue
    }
}
6. return 'NO'

Takes up logarithmic space

This N.D. algorithm might never stop

Complexity 30

N.D. Algorithm for CR_S

CR_s (d)

1. count = 0
2. for all u $\in$ V {
    3. $count_{d-1}$ = 0
    4. for all v $\in$ V {
        5. nondeterministic switch:
            5.1 if reach(v, d - 1) then ++$count_{d-1}$ else *fail*
                if (v,u) $\in$ E then ++count; break
            5.2 continue

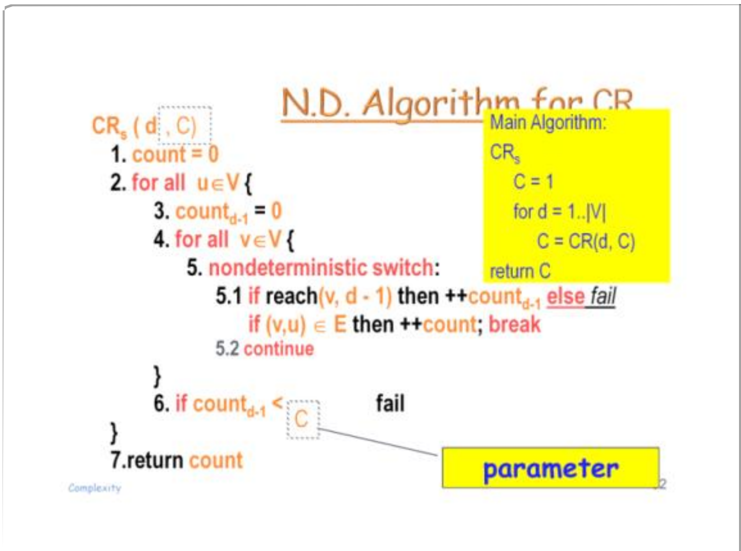
Assume (v,v) $\in$ E

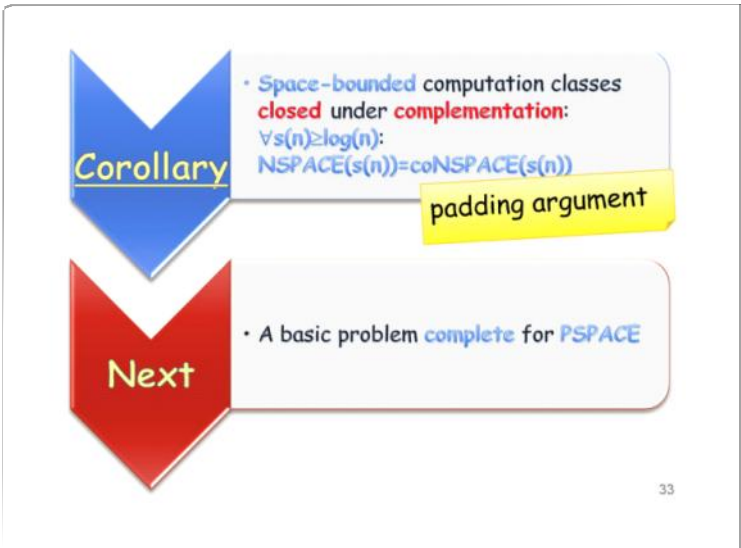
    }
    6. if $count_{d-1}$ < CR_s (d-1) fail

Recursive call!

}
7.return count

Complexity 31





[PSPACE Completeness]

TQBF

Instance:

- a fully quantified Boolean formula ϕ

Decision Problem:

- Is ϕ true?

EG $\forall x \exists y \forall z[(x \vee \neg y \vee z) \wedge (\neg x \vee y)]$

Theorem:

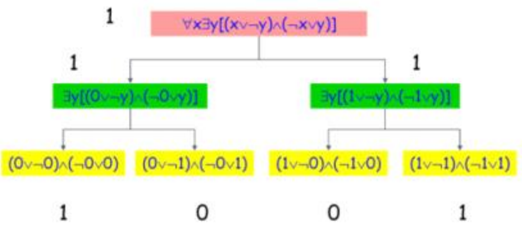
- TQBF $\in$ PSPACE

Proof:

- A poly-space algorithm A that evaluates ϕ:
  if ϕ is quantifier-free return its value
  if $\phi = \forall x.\psi(x,..)$ return $A(\psi(0,..)) \wedge A(\psi(1,..))$
  if $\phi = \exists x.\psi(x,..)$ return $A(\psi(0,..)) \vee A(\psi(1,..))$

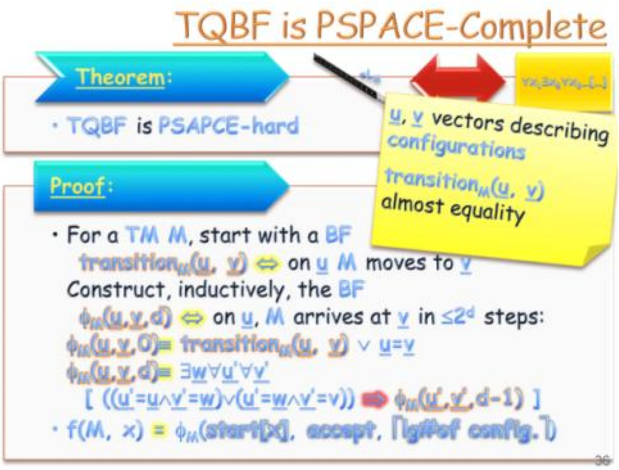
34

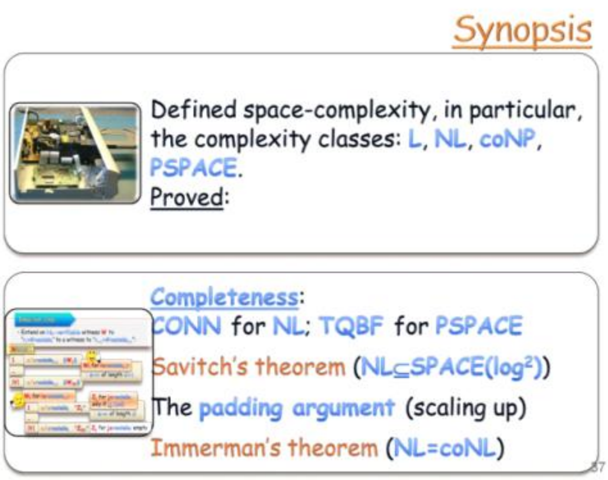
Algorithm for TQBF

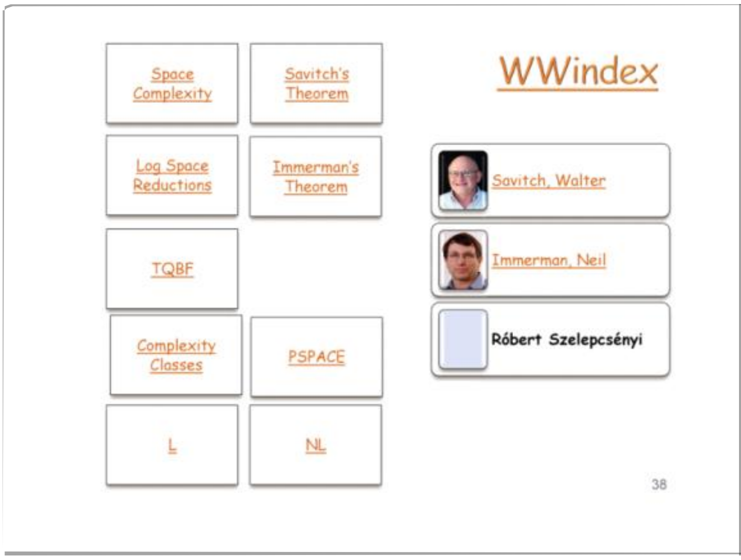


Complexity

35


TQBF is PSPACE-Complete
Theorem:
• TQBF is PSAPCE-hard
u, v vectors describing configurations
transitionM(u, v)
almost equality
Proof:
• For a TM M, start with a BF
transitionM(u, v) ⇔ on u M moves to v
Construct, inductively, the BF
φM(u,v,d) ⇔ on u, M arrives at v in ≤2d steps:
φM(u,v,0)≡ transitionM(u, v) ∨ u=v
φM(u,v,d)≡ ∃w∀u'∀v'
[((u'=u∧v'=w)∨(u'=w∧v'=v)) ⇒ φM(u',v',d-1)]
• f(M, x) ≡ φM(start[x], accept, ⌈lg#of config.⌉)
36


Synopsis
Defined space-complexity, in particular, the complexity classes: L, NL, coNP, PSPACE.
Proved:
Completeness:
CONN for NL; TQBF for PSPACE
Savitch's theorem (NL⊆SPACE(log²))
The padding argument (scaling up)
Immerman's theorem (NL=coNL)
37



- Space Complexity
- Savitch's Theorem
- Log Space Reductions
- Immerman's Theorem
- TQBF
- Complexity Classes
- L
- NL
- PSPACE