The Immerman-Szelepcsenyi Theorem

Relating NL and co-NL

NL and co-NL

Quick review: See if you can remember

- Definition of co-NP
- A complete problem for co-NP
- A problem in NP ⋂ co-NP

NL and co-NL

Quick review: See if you can remember

- Definition of co-NP
- A complete problem for co-NP
- A problem in NP $\cap$ co-NP

Can you suggest the following?

- Definition of co-NL
- A complete problem for co-NL
- A problem in NL $\cap$ co-NL

NL and co-NL

- It’s widely conjectured that NP ≠ co-NP
- In contrast, NL = co-NL !

  (Proved independently by Immerman and Szelepcesnyi in 1987)

Immerman-Szelepcesnyi Theorem

- Recall that PATH is complete for NL:

  PATH = { (G,x,y) | y is reachable from x in G }
- Let NoPATH =

  { (G,x,y) | y is *not* reachable from x in G }
- NoPATH is complete for co-NL

- Immerman-Szelepcesnyi: NoPATH is in NL
- Corollary: NL = co-NL

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

Notation: for an instance (G,x,y) of NoPATH

- Let the nodes of G be 1, 2, …n
- Let C_i denote the set of nodes of G that are reachable from x via a path of length at most i
- y is reachable from x if and only if y is in C_{n-1}

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

We'll build up to the proof by developing nondeterministic algorithms for several handy checks

- Check0(v,j): is v in C_j?

- Nondeterministic algorithm correctness: Some algorithm execution returns true if the answer is yes, and there is no such execution if the answer is no

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

Check0(v,j): is v in C_j? (v is a node of G, $1 \leqslant j \leqslant n-1$)

```
If v = x Return true
Else
   p = x
   For k = 1 to j
      Nondeterministically choose a node p'
      If ( (p,p') is not in G) Return false
      If (p' = v) Return true
      p = p'
   Return false
```

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

We'll build up to the proof by developing nondeterministic algorithms for several handy checks

- Check0(v, j): is v in C_j?
- Check1(v, j, c_j): is v is *not* in C_j, given that $c_j = |C_j|$?

NoPATH = $\{(G,x,y) \mid x \not\to y \text{ in } G\}$ is in NL

Check1(v, j, c_j): is v is *not* in C_j, given that $c_j = |C_j|$?

Idea: find c_j nodes that *are* in C_j, and make sure none is v

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

Check1(v, j, c_j): is v is *not* in C_j, given that $c_j = |C_j|$?

Idea: find c_j nodes that *are* in C_j, and make sure none is v

```
count  = 0 //the number of nodes found in C_j so far
For k = 1 to n
   If (k ≠  v)
      If Check0(k,j)  // k is in C_j
            count  = count + 1
If (count  = c_j) Return true
Return false
```

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

Check1(v, j, c_j): is v is *not* in C_j, given that $c_j = |C_j|$?

Correctness:

- If v is not in C_j, there is an accepting execution, where on each iteration of the For loop when k is in C_j, Check0(k,j) returns true. In this case, count is incremented exactly c_j times.

- If v is in C_j then there is no accepting execution: count can be incremented at most c_j-1 times and so the algorithm returns false.

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

We'll build up to the proof by developing nondeterministic algorithms for several handy checks

- Check0(v, j): is v in C_j?
- Check1(v, j, c_j): is v is not in C_j given $c_j = |C_j|$?
- Check2(v, j, c_{j-1}): is v is not in C_j given $c_{j-1} = |C_{j-1}|$?
  - Try modifying Check1

NoPATH = $\{(G,x,y) \mid x \not\to y \text{ in } G\}$ is in NL

Check2(v, j, c_{j-1}): is v is not in C_j given $c_{j-1} = |C_{j-1}|$?

Idea: find c_{j-1} nodes in C_{j-1}, and make sure there is no edge to v from any of these

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

Check2(v, j, c_{j-1}): is v is not in C_j given $c_{j-1} = |C_{j-1}|$?

Idea: find c_{j-1} nodes in C_{j-1}, and make sure there is no edge to v from any of these

```
count  = 0 //the number of nodes found in C_{j-1} so far
For k = 1 to n
   If (k ≠  v)
      If Check0(k,j-1)  // k is in C_{j-1}
            count  = count + 1
             If (k,v) is in G, Return false
If (count  = c_{j-1}) Return true
Return false
```

NoPATH = $\{(G,x,y) \mid x \not\to y \text{ in } G\}$ is in NL

We'll build up to the proof by developing nondeterministic algorithms for several handy checks

- Check0(v, j): is v in C_j?
- Check1(v, j, c_j): is v is not in C_j given $c_j = |C_j|$?
- Check2(v, j, c_{j-1}): is v is not in C_j given $c_{j-1} = |C_{j-1}|$?
- Check3(j, c_j, c_{j-1}): is $c_j = |C_j|$ given $c_{j-1} = |C_{j-1}|$?

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

Check3(j, c_j, c_{j-1}): is $c_j = |C_j|$ given $c_{j-1} = |C_{j-1}|$?

```
count  = 0 //the number of nodes found in Cj so far
For k = 1 to n
    If Check0(k,j)
        count  = count + 1
    Else If not Check2(k, j, cj-1)
        Return false
If (count  = cj) Return true
Return false
```

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

Check3(j, c_j, c_{j-1}): is $c_j = |C_j|$ given $c_{j-1} = |C_{j-1}|$?

Correctness:

- If $c_j = |C_j|$ there is an accepting execution, when exactly c_j of the Check0(v,j) tests return true and the remaining tests to Check2(v, j, c_{j-1}) also return true. On this execution, the for loop terminates and count = c_j

- If $c_j \neq |C_j|$ either one of the Check2 tests returns false or at the end, count $\neq c_j$

NoPATH = $\{(G,x,y) \mid x \not\rightarrow y \text{ in } G\}$ is in NL

We'll build up to the proof by developing nondeterministic algorithms for several handy checks

- Check0(v, j): is v in C_j?
- Check1(v, j, c_j): is v is not in C_j given $c_j = |C_j|$?
- Check2(v, j, c_{j-1}): is v is not in C_j given $c_{j-1} = |C_{j-1}|$?
- Check3(j, c_j, c_{j-1}): is $c_j = |C_j|$ given $c_{j-1} = |C_{j-1}|$?

Let's use these checks to build an algorithm for NoPATH.

NoPATH = $\{(G,x,y) \mid x \nrightarrow y \text{ in } G\}$ is in NL

NoPATH(G,x,y)

c[0] = 1 //only one node is reachable in 0 steps from x

For i = 1 to n-1 // calculate c_i

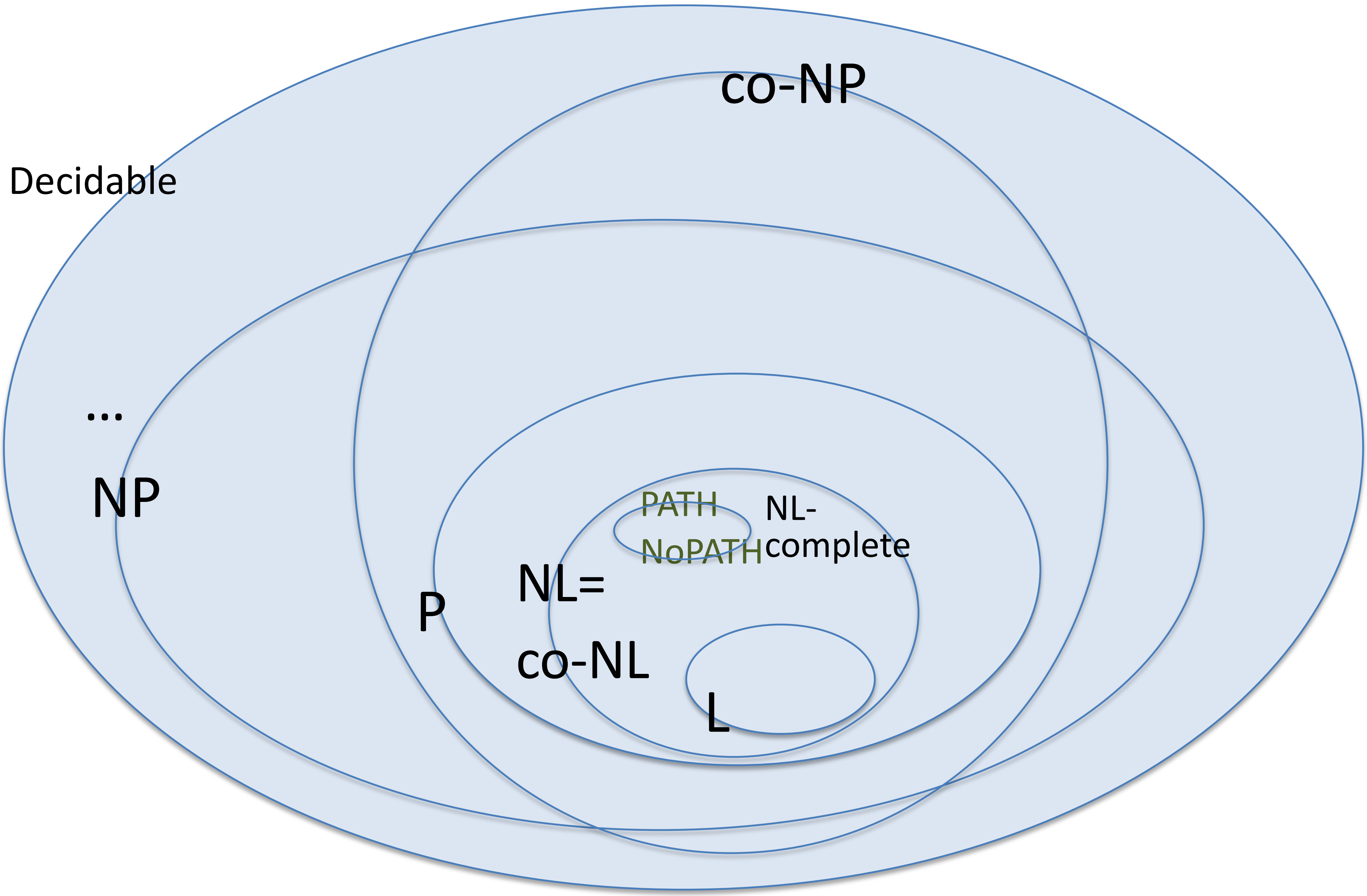
$c_i = 1$

While (not Check3(i, c_i, c_{i-1})) and ($c_i < n$)

$c_i = c_i + 1$

If Check1(y,n-1,c_{n-1}) Return true

Return false


co-NP
Decidable
...
NP
PATH
NoPATH
NL-
complete
NL=
co-NL
P
L

Next Class

- Introduction to Randomized Complexity Classes
- Reading: Arora-Barak 7.1-7.3