Logarithmically Bounded Space

More new problems that are representative of space bounded complexity classes

Recall Space Bounded Complexity Classes

- A TM has a *read-only input tape* and *work tapes*.
- SPACE($s(n)$) languages accepted by deterministic halting TMs that use $O(s(n))$ work tape cells on inputs of length n.
- NSPACE($s(n)$): replace "deterministic" by "nondeterministic".

Addition

- **Instance**: Binary numbers x, y and z
- **Problem**: Is x + y = z?

- Addition is in L (log space) … why?

Addition

- **Instance**: Binary numbers x, y and z
- **Problem**: Is x + y = z?

- Addition is in L (log space) … why?
- Given x and y, can we *compute* the sum x+y in log space?
- Let‘s be clear about accounting for space in log space function computation

Space bounded function computation

- A TM has
  - a *read-only input tape,*
  - a *write-only output tape:* output bits are written from left to right,
  - *work tapes*.
- We charge for
  - space used by the work tapes: #cells that the tape head has visited
  - space to store the index of input and output tape heads

- A function is *log-space computable* if some (deterministic) TM that computes the function uses log n space on inputs of length n.

Addition

- **Instance**: Binary numbers x, y and z
- **Problem**: Is $x + y = z$?

- Given x and y, can we *compute* the sum $x+y$ in log space?

Addition

- **Instance**: Binary numbers x, y and z
- **Problem**: Is x + y = z?

- Given x and y, can we *compute* the sum x+y in log space?
  - A log-space TM, M, can compute reverse(x+y), i.e., the bits of the sum in reverse order
  - A log-space TM, M´, can output the reverse of its input
  - We can compose M and M´ to compute x + y

How to compose log-space TMs?

- On input w, we want to run TM M and then run TM M´ on the output produced by M
- We can‘t store the output produced by M

How to compose log-space TMs?

- On input w, we want to run TM M and then run TM M´ on the output produced by M
- We can‘t store the output produced by M
- Instead, whenever M´ needs to read the *i*th output bit of M, it re-runs M up until it produces the needed bit, discarding output bits produced prior to bit *i*

Path

- **Instance**: A directed graph $G = (V,E)$ and two nodes s and f
- **Problem**: Is there a path in G from s to f?

Path

- **Instance**: A directed graph G = (V,E) and two nodes s and f
- **Problem**: Is there a path in G from s to f?

- PATH is in P – use breadth-first or depth-first search – but these algorithms use polynomial space too.
- Is there a more space-efficient solution? What if you can use nondeterminism?

Path

- **Instance**: A directed graph G = (V,E) and two nodes s and f
- **Problem**: Is there a path in G from s to f?

- PATH is in NL (nondeterministic log space): simply guess a path from s to f and verify that each edge of the guessed path is in the graph
- To save space, store only the current node along the path

Space Bounded Complexity Classes

- L = SPACE(log n)
- NL = NSPACE(log n)

- Explain why the class L is in the class P

Space Bounded Complexity Classes

- L = SPACE(log n)
- NL = NSPACE(log n)

- Explain why the class L is in the class P
- What about NL?

Path is NL-Complete

- We need a new notion of reduction $L \leq_{\log} L'$
- Function f: $L \rightarrow L'$ is a *log-space, many-one reduction* if
  - f is log-space computable and
  - x is in L iff f(x) is in L'
- We already saw that Path is in NL
- We'll show a log-space reduction from any language in NL to Path

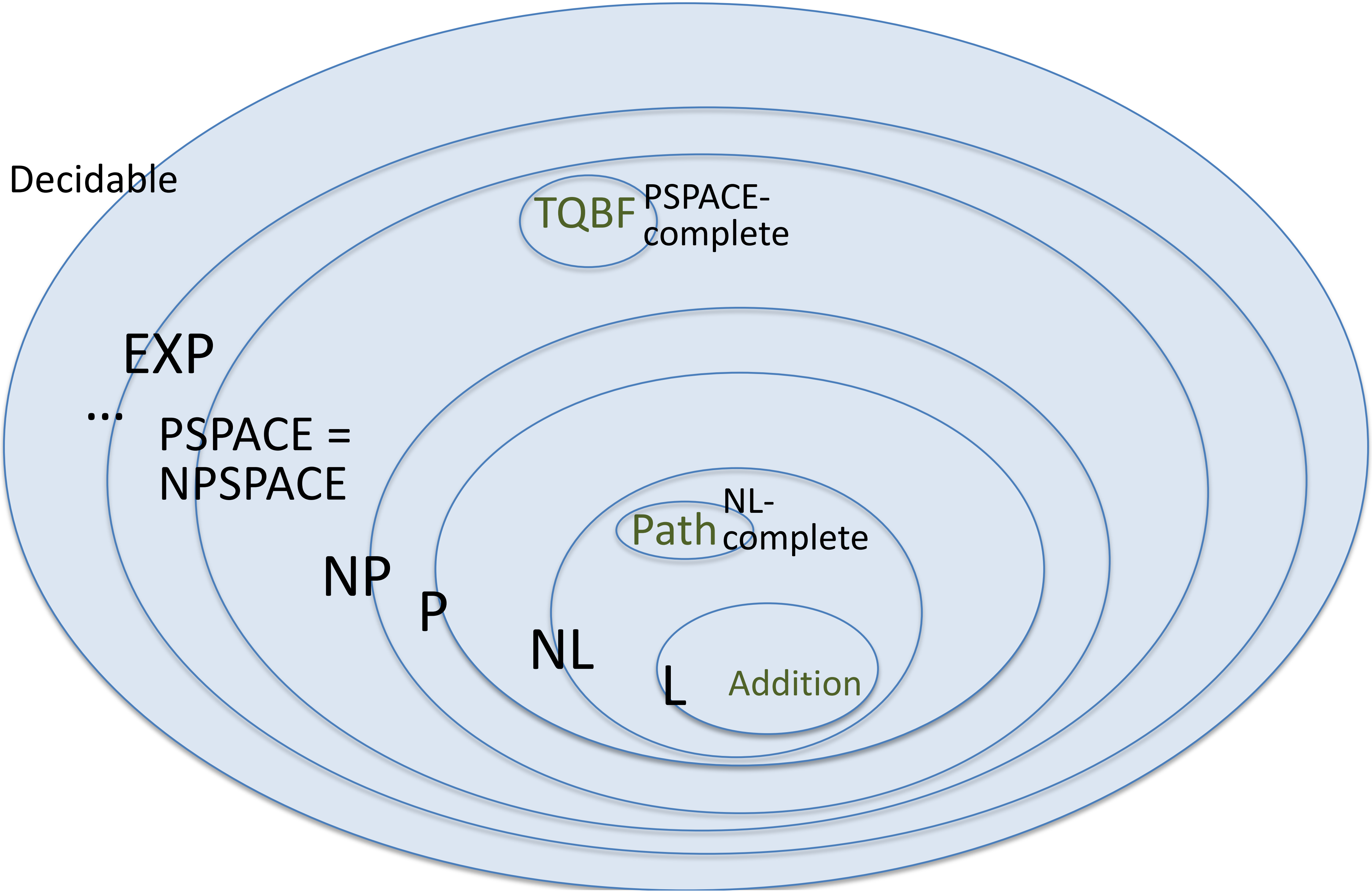
Path is NL-Complete

- Let L be in NL, accepted by NTM M that has a unique accepting configuration acc(w) on input w
- We need a log-space reduction from L to Path

Path is NL-Complete

- Let *L* be in NL, accepted by NTM M that has a unique accepting configuration acc(*w*) on input *w*
- We need a log-space reduction from *L* to Path
- The reduction simply computes the configuration graph G of M on *w* and outputs (G, init(w), acc(w))

Complexity Classes



P vs NL: Insight on Parallel Algorithms

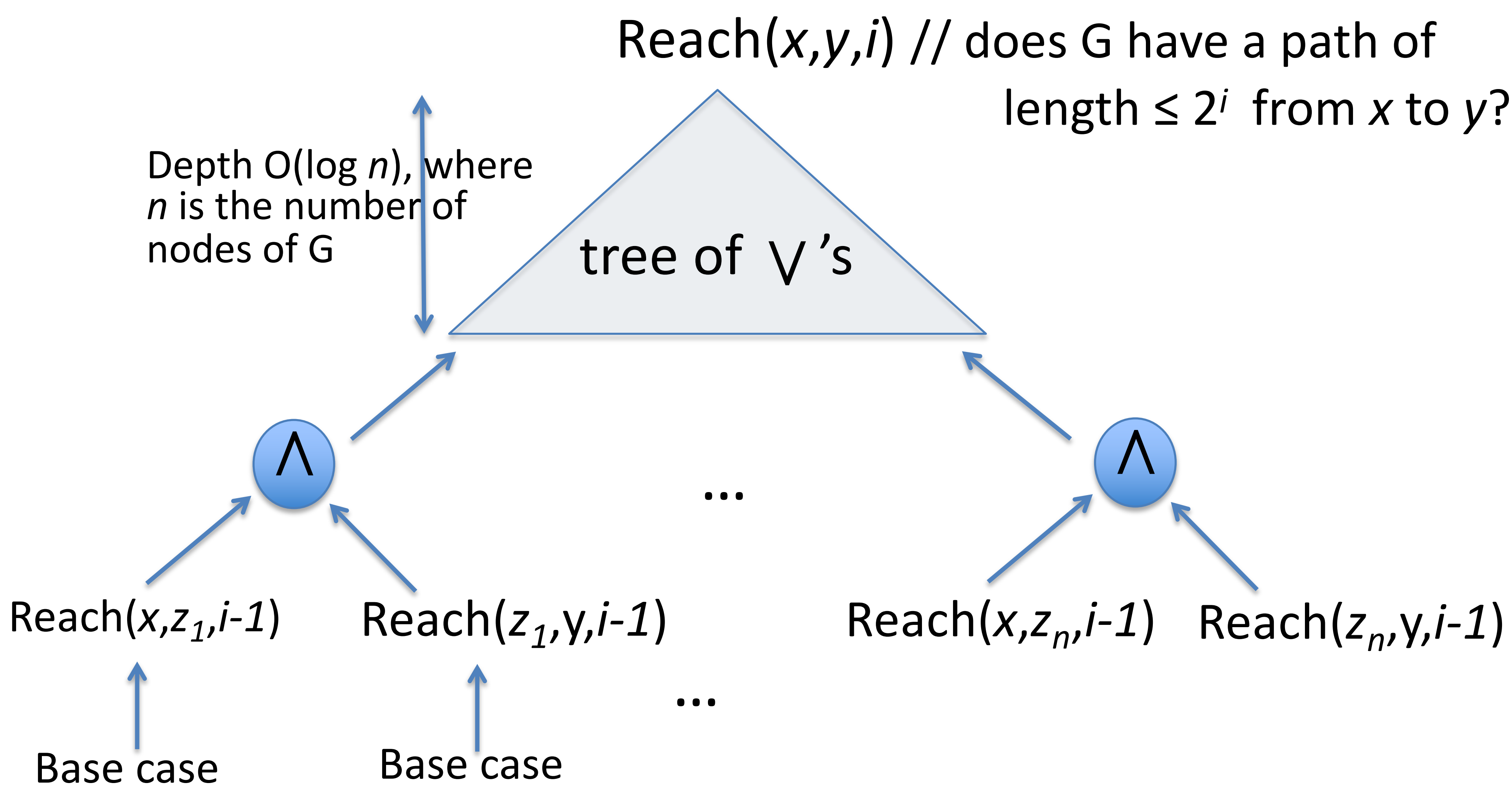
- Problems in NL have “fast” parallel algorithms: bounded fan-in circuits with
  - depth (parallel time) polylog(n)
  - size (# parallel processors) poly(n)

A Fast Parallel Algorithm for Path

```
Reach(x,y,i)  // does G have a path of length ≤ 2^i  from x to y?
    If i = 0 then
        If (x = y) or ( (x,y) is an edge of G) Return True
        Else Return False
    Else
        For each node z of G
            If (Reach (x, z, i-1) and Reach(z, y, i-1) )
                Return True
        Return False
```

- Parallelize the Reach algorithm
  - Do the tests for all nodes z in parallel
  - Do the two Reach computations in parallel

A Fast Parallel Algorithm for Path



A Fast Parallel Algorithm for Path

- Circuit depth at each recursion level: $O(\log n)$
- Circuit depth for base case: $O(\log n)$
- $O(\log n)$ levels of recursion

- Total circuit depth for DGR: $O(\log^2 n)$
- Total number of gates (processors): $\text{poly}(n)$

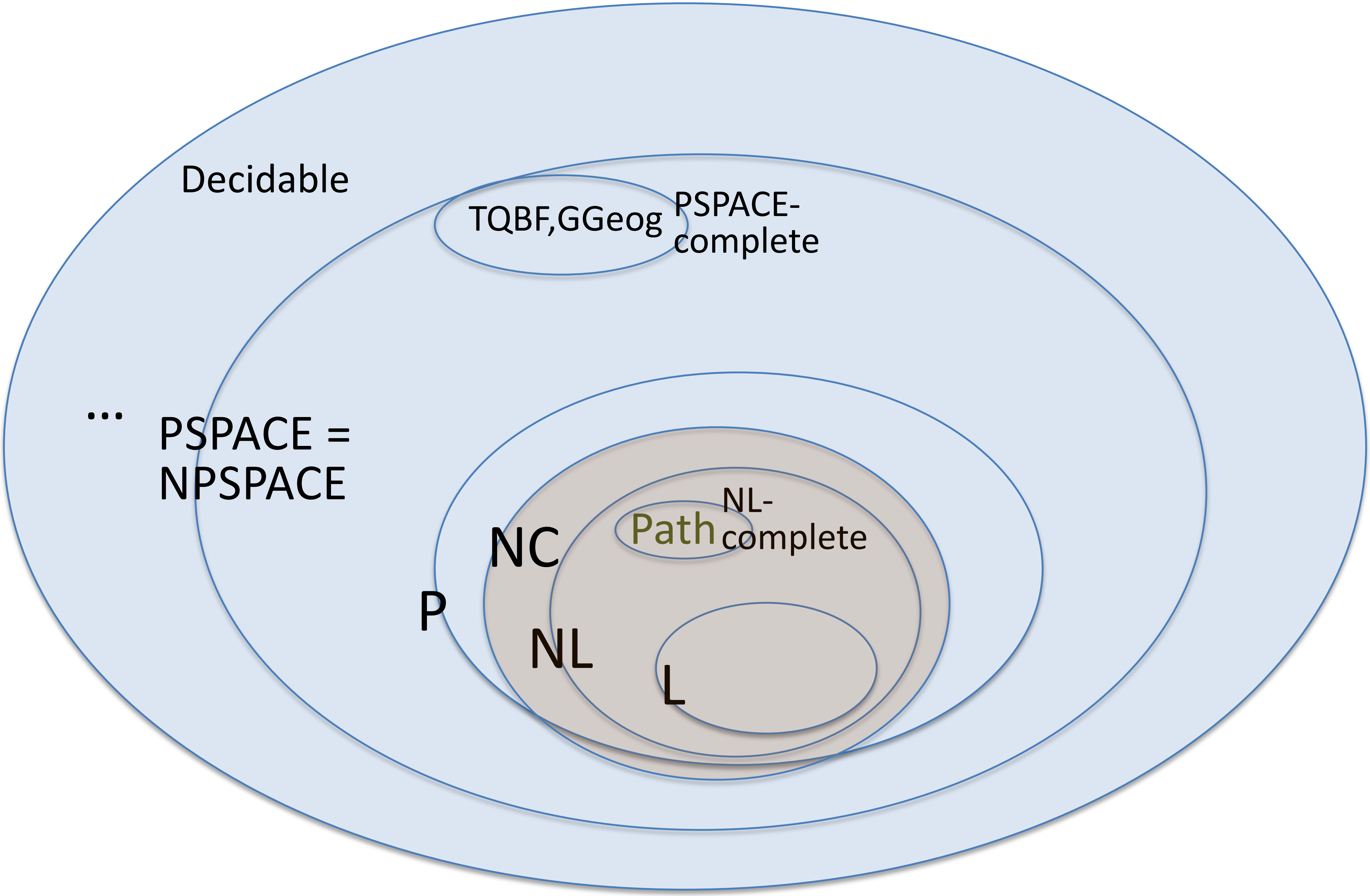
P vs NL: Insight on Parallel Algorithms

- Problems in NL have “fast” parallel algorithms: bounded fan-in circuits with
  - depth (parallel time) polylog(n)
  - size (# parallel processors) poly(n)
- Not all problems in P have fast parallel algorithms
- Problems that are $\leq_{log}$-complete for P are the “hardest” problems, with respect to their space and parallel time requirements

Nick's Class

- NC(d(n),p(n)) is the class of languages that have bounded fan-in circuits of depth O(d(n)) and with O(p(n)) gates.
- $NC = \bigcup_{c,d} NC(\log^c n, n^d)$

- Path is in NC($\log^2 n$, poly(n))
- (See Arora-Barak Chapter 6.5)


Decidable
TQBF,GGeog
PSPACE-
complete
...
PSPACE =
NPSPACE
NL-
Path
complete
NC
P
NL
L

Summary

- Log space bounded complexity classes
- A new complexity class of problems with “fast” parallel algorithms: NC
- Log-space reductions ($\leq_{log}$) help identify which problems in P have fast parallel algorithms and which don’t

Next Class

- The Immerman-Szelepcsényi theorem: NL = co-NL
- Reading: Arora-Barak 4.3, 4.4