Formale Methoden der Informatik

Block 1: Foundations of Complexity Theory

5. Other Important Complexity Classes

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

21 October, 2013

Outline

The Class L (1)

Definition of L

L is the class of all problems that can be solved by a program that uses logarithmic space, i.e. $\mathcal{P} \in \mathrm{L}$ if there is a program such that for all instances I of $\mathcal{P}$, the program uses at most $\mathrm{O}(log_2|I|)$ bits of read/write memory.

- The running time is, in principle, unlimited!
  (but it can be polynomially bounded, as we shall see).
- Logarithmic space = really little space, e.g. $log_2(65536) = 16$.
- By rephrasing $\mathrm{O}(log_2|I|)$, we have that a program can use at most $c \cdot log_2|I| + d$ bits of the read/write memory, where c, d are constants.
- Thus the read/write memory of a program is limited to constantly many
  - pointers (addresses of memory),
  - counters, or
  - Boolean flags.

The Class L (2)

- We must assume that an input to a logarithmic space program is stored in the read-only memory!
  - Clearly, $c \cdot log_2 n + d < n$ for a sufficiently large n.
  - Thus without the assumption, on a large input the program would not be able even to traverse its input.

- An example of a problem in L:

FIND-NODE

INSTANCE: A natural number n, and a tree T where each node is labeled with a natural number.

QUESTION: Does T contain a node labeled with n?

- We have **FIND-NODE** $\in$ L because we can traverse T in depth-first manner using only three pointers to nodes in T: *current*, *next*, and *aux*.

Solving **FIND-NODE** in Logarithmic Space

We assume the following methods (implementable to run in log. space):

- $root(T)$ returns the root of T
  - $root(T) = nil$ if T is empty
- $firstChild(T, x)$ returns the first child of a node x in T
  - $firstChild(T, x) = nil$ if such a child does not exist
- $rightSibling(T, x)$ returns the right sibling of a node x in T
  - $rightSibling(T, x) = nil$ if such a sibling does not exist
- $parent(T, x)$ returns the parent of a node x in T
  - $parent(T, x) = nil$ if x is the root in T
- $isChildOf(T, x_1, x_2)$ returns **true** iff x_1 is a child of x_2 in T
- $isLeaf(T, x)$ returns **true** iff $firstChild(T, x) = nil$
- $hasRightSibling(T, x)$ returns **true** iff $rightSibling(T, x) \neq nil$
- $labelling(T, x)$ returns the label of x in T

```
// space efficient implementation of depth-first traversal

Boolean find(Tree T, Integer val)
  current = root(T);
  if labelling(T, current) = val then return true;
  next = firstChild(T, current);

  while (next != nil) {

    if isChildOf(T, next, current) and !isLeaf(T, next) then {
      current := next; next := firstChild(T, next) }

    else if isChildOf(T, next, current) and isLeaf(T, next) then {
      aux := current; current := next; next := aux }

    else if isChildOf(T, current, next) and hasRightSibling(T, current) then
      { aux := current; current := next; next := rightSibling(T, aux) }

    else if isChildOf(T, current, next) and !hasRightSibling(T, current) {
      current := next; next := parent(T, next) }

    if labelling(T, current) = val then return true
  }
  return false
```

L and P

Theorem

$\mathsf{L} \subseteq \mathsf{P}$, *i.e. every problem solvable in logarithmic space is also solvable in polynomial time.*

Proof sketch

Recall that a logarithmic space program Π_1 on input I uses at most $c \cdot log_2|I| + d$ bits of read/write memory (including the program counter, registers, etc), where c, d are constants. Note that each stage of the execution of Π_1 on I is uniquely described by the content of the memory. Thus observe that we have $2^{c \cdot log_2|I|+d}$ possible configurations in which Π_1 may be. Observe that $2^{c \cdot log_2|I|+d} = (2^{\log_2|I|})^c \cdot 2^d = |I|^c \cdot 2^d$. Since c, d are constants, we get that there are only polynomially many different configurations in which Π_1 may be.

Based on the above observation, we can write a polynomial time program Π_2 that lists all the possible configurations of Π_1 on I, and then decides whether Π_1 returns *true* or *false* on I.

The Class PSPACE

PSPACE

PSPACE is the class of all problems that can be solved by a program that uses polynomial space, i.e. $\mathcal{P} \in$ PSPACE if there is a program Π such that: for all instances I of $\mathcal{P}$, Π uses at most $\mathrm{O}(|I|^k)$ bits of memory, where k is a constant .

- The class is quite wide and powerful, and many real world problems fall into this class.
- "polynomial space" means that the program execution may be in exponentially many different states (vs. polynomially many states in case of L)
- PSPACE-complete problems are considered intractable (we'll see NP $\subseteq$ PSPACE).

Example of a Problem in PSPACE: Tic-Tac-Toe

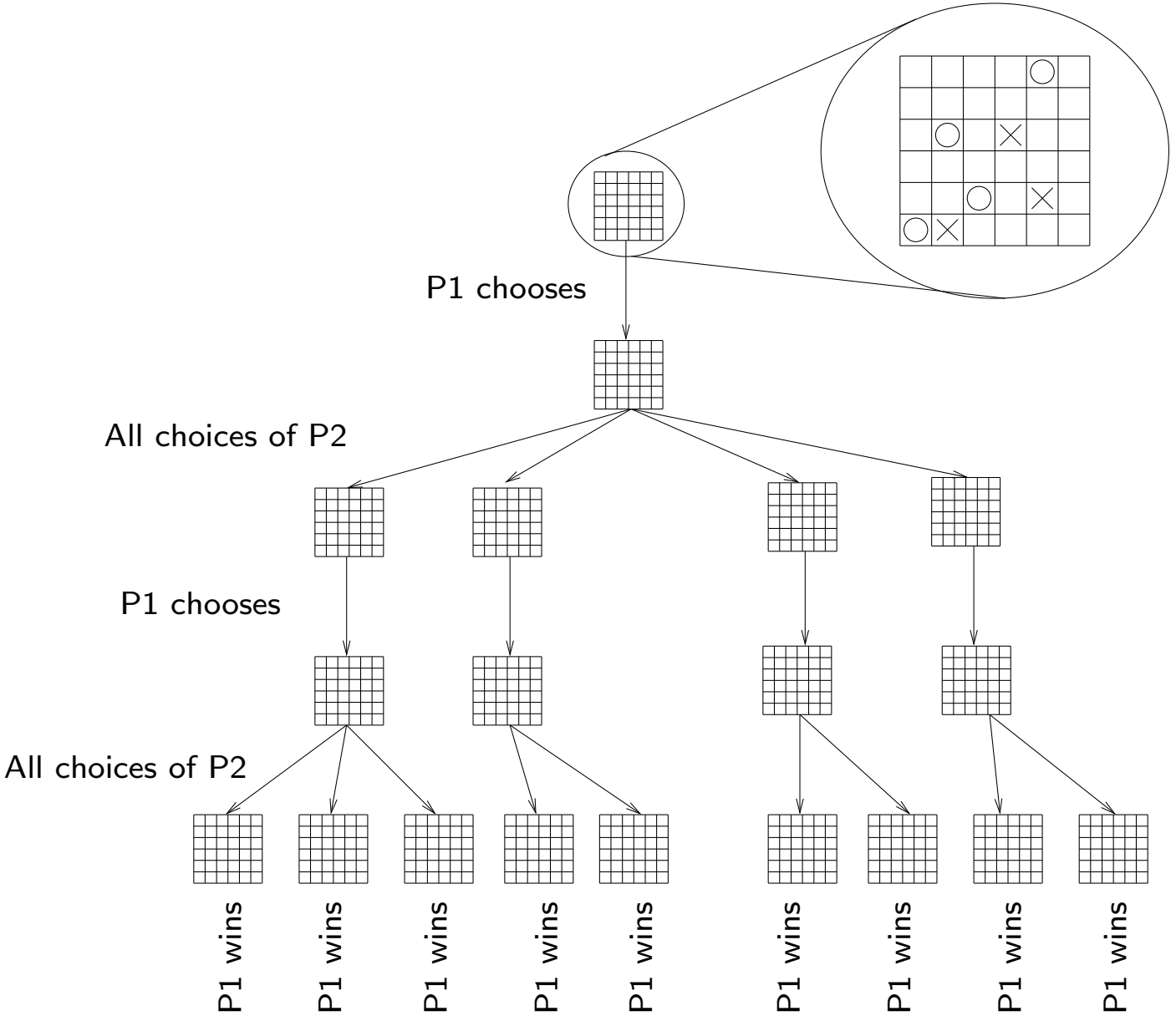
- Recall the classic Tic-Tac-Toe game (played on a 3×3 board)
- An (n, k)-Tic-Tac-Toe is a generalization played on an $n \times n$ board.
- The first player to obtain a row of k is the winner.
- Note that each game ends after at most n^2 steps.

TTT-WINNING-STRATEGY

INSTANCE: Natural numbers n, k, and a configuration C of an $n \times n$ board.

QUESTION: Does the player 1 have a winning strategy for the (n, k)-Tic-Tac-Toe game starting from the configuration C?

Winning Strategies In Generalized Tic-Tac-Toe



PSPACE: Summary

TTT-WINNING-STRATEGY

This problem is in PSPACE (and is PSPACE-complete):

- it suffices to have a stack of at most n^2 configurations;
- each configuration has polynomial size.

Observations

- Alternation is typical for PSPACE-complete problems, i.e.:
  there exists a choice for player P1, s.t.
  for all choices of player P2,
  there exists a choice for player P1, ...
- Problems in PSPACE can be solved by polynomially deep nested loops: In case of TTT-WINNING-STRATEGY, the nesting is bounded by n^2.

Other Problems in PSPACE

First-order queries

Evaluating a first-order sentence over a structure.

SQL evaluation

Evaluating an SQL query over a database.

The two above are essentially the same problem!

Universality of a regular expression

Checking if a regular expression E matches all possible strings.

PSPACE vs. P

Theorem

P $\subseteq$ PSPACE, *i.e. every problem solvable in polynomial time is also solvable in polynomial space.*

Proof sketch

Trivial, any program running in polynomial time never uses more than polynomial space, i.e. if Π is a polynomial time program to solve $\mathcal{P} \in$ P, then Π also runs in polynomial space and thus proves $\mathcal{P} \in$ PSPACE.

PSPACE vs. NP

Theorem

NP $\subseteq$ PSPACE.

Proof sketch

Assume a problem $\mathcal{P} \in$ NP. Then there is a certificate relation R for $\mathcal{P}$ that is polynomially balanced and polynomially decidable.

We can devise a program that solves instances I of $\mathcal{P}$ in polynomial space:

- Traverse the candidate certificates one by one (keep only one in memory). Since R is polynomially balanced, it is safe to concentrate on candidates of polynomial size.
- For each candidate C, check $(I, C) \in R$. Since R is polynomially decidable, each test requires at most polynomial time, and thus also only polynomial space.

The Class EXPTIME

EXPTIME

EXPTIME is the class of all problems that can be solved in exponential time, i.e. $\mathcal{P} \in$ EXPTIME if there is a program such that for all instances I of $\mathcal{P}$, the program runs in $\mathrm{O}(2^{|I|^k})$, where k is a constant.

The following inclusions are known:

- PSPACE $\subseteq$ EXPTIME (same argument as for L $\subseteq$ P)
- P $\subsetneq$ EXPTIME (a nontrivial diagonalization argument)

Examples of problems in EXPTIME:

- existence of winning strategies in GO generalized to $n \times n$ boards (reason: there exists no polynomial bound on the length of the game),
- evaluation of DATALOG queries (SQL extended with recursion).

Discussion

There are many more complexity classes. For instance:

- 2-EXPTIME = problems solvable in $\mathrm{O}(2^{2^{|I|^k}})$ time
- 3-EXPTIME = problems solvable in $\mathrm{O}(2^{2^{2^{|I|^k}}})$ time
- ...
- EXPSPACE = problems solvable in $\mathrm{O}(2^{|I|^k})$ space
- 2-EXPSPACE = problems solvable in $\mathrm{O}(2^{2^{|I|^k}})$ space
- 3-EXPSPACE = problems solvable in $\mathrm{O}(2^{2^{2^{|I|^k}}})$ space
- ...

Learning Objectives

- Understanding the definitions of L, PSPACE and EXPTIME
- Being aware of the main inclusions between P, NP, and the three classes above.