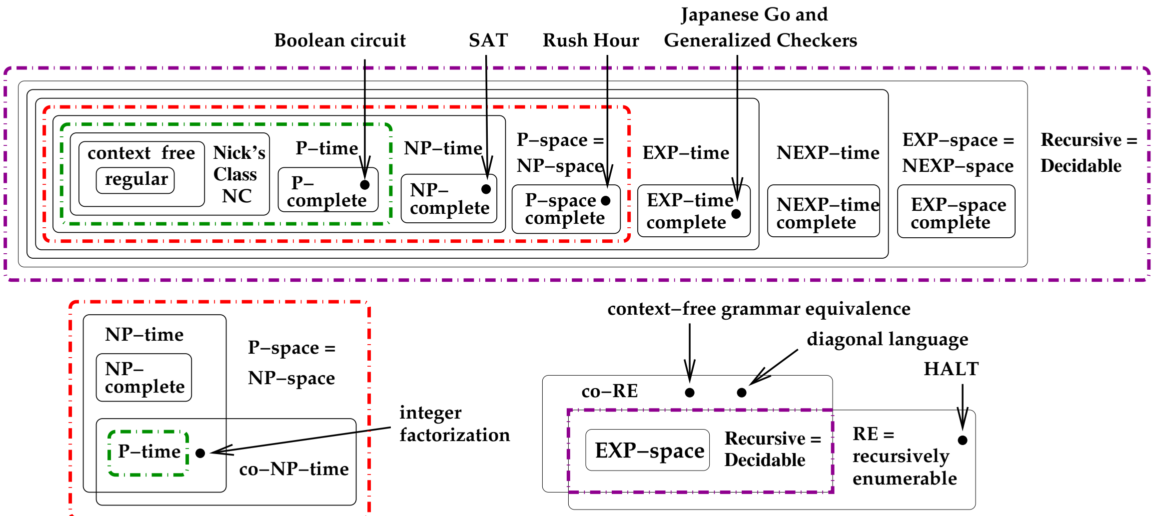
Computational Classes

The Euler diagram below illustrates the complexity classes that we discuss in CS 456/656. The diagram shows the widely believed relationshop between the complexity classes. Some of the classes are defined by grammar classes, and others are defined by computational complexity. Although "everyone" believes that $\mathcal{P}$-time is not equal to $\mathcal{NP}$-time, and that $\mathcal{NC}$ (Nick's Class) is not equal to $\mathcal{P}$-time, those claims have not been proven. Some inequalities are known: for example, it is known that $\mathcal{P}$-time is not equal to EXP-time, and that $\mathcal{P}$-space is not equal to EXP-space.



Polynomially and Exponentially Bounded Functions

An increasing function f on integers is *polynomially bounded* if there is some $k > 0$ such that $f(n) = O(n^k)$. Write $\mathcal{P}$ for the class of polynomial functions. We say that a function f is *exponentially bounded* if $f(n) = O\left(2^{g(n)}\right)$ for some $g \in \mathcal{P}$. Let EXP be the class of exponentially bounded functions. Clearly, $\mathcal{P} \subseteq$ EXP.

We say that a function f is *polylogarithmically bounded* if there is some constant k such that $f(n) = O(\log^k n)$.

Is there Anything Between Polynomial and Exponential?

Does there exist a function which grows faster than any polynomial function, but slower than any strictly exponential function? The answer is yes; for example, if F is any solution to the recurrence $F(n) = F(n-1) + F(n/2) + 1$, then $F \notin \mathcal{P}$, but F grows more slowly than 2^k for any positive constant k.

$\mathcal{P}$-Time and EXP-Time

$\mathcal{P}$-time and EXP-time (typically written as simply $\mathcal{P}$ and EXP) are the classes of languages (problems) that can be decided (solved) by a deterministic machine in time which is polynomially bounded, or exponentially bounded.

$\mathcal{NP}$-Time, NEXP-Time, $\mathcal{P}$-Space, and EXP-Space

These are the classes of languages which can be accepted in polynomial tiem, or exponential time, by a non-deterministic machine, and the classes of languages which can be decided by a deterministice machine which uses polynomially bounded space or exponentially bounded space.

Nick's Class

Nick's class ($\mathcal{NC}$) is defined to be the class of languages which can be decided (or problems that can be solved) in polylogarithmic time using polynomially many processors. Many processors running in parallel can be emulated by a single processor emulating the many processors in round-robin fashion. Thus $\mathcal{NC} \subseteq \mathcal{P}$-time.

"–Complete" Subclasses

Recall that an $\mathcal{NP}$-complete problem is the hardest problem in the class $\mathcal{NP}$-time in some sense. In what sense? From a practical perspctive, the partition problem appears far easier than SAT, for example. The perspective is that polynomial time computation is considered "trivial," and if you view all $\mathcal{P}$ computations to have zero hardness, what's left for any two $\mathcal{NP}$ problems is equally hard. The same concept guides the definitions of all "-complete" subclasses in our diagram.

A language L in $\mathcal{NP}$ is $\mathcal{NP}$-complete if every language in $\mathcal{NP}$ reduces to L in polynomial time.

A language L in $\mathcal{P}$-space is $\mathcal{P}$-space-complete if every language in $\mathcal{P}$-space reduces to L in polynomial time. Sliding block puzzles, such as Rush Hour, are $\mathcal{P}$-space-complete.

A language L in EXP-time is EXP-time-complete if every language in EXP-time reduces to L in polynomial time. 2-person board games such as generalized checkers and generalized chess are EXP-time complete. That is, the set of all generalized chess configurations from which White can force a win is EXP-time-complete.

A language L in NEXP-time is NEXP-time-complete if every language in NEXP-time reduces to L in polynomial time.

A language L in EXP-space is EXP-space-complete if every language in EXP-space reduces to L in polynomial time.

$\mathcal{P}$-completeness is defined differently. A $\mathcal{P}$-time language is $\mathcal{P}$-complete if every $\mathcal{P}$-time language can be reduced to L by an $\mathcal{NC}$ function. CVP is $\mathcal{P}$-complete.

Shortcut Definitions

1. $\mathcal{NC}$, Nick's Class, is the class of all languages which can be decided by polynomially many processors working in parallel, in polylogarithmic time.

2. Many important problems are in Nick's Class, such as finding maxima or minima, addition, subtraction, and multiplication on numerals, matrix multiplication.

3. The *circuit value problem*, CVP, is a dynamic programming problem. Dynamic programming is generally $\mathcal{P}$–TIME but not known to be $\mathcal{NC}$. Many DP problems, including CVP, are $\mathcal{P}$-complete.

4. The question of whether $\mathcal{NC} = \mathcal{P}$–TIME is open. As computers are more and more parallelized, this question is of enormous practical importance.

5. $\mathcal{P}$–TIME, usually abbreviated to just $\mathcal{P}$, is the class of all languages which can be decided in polynomial time by a deterministic machine.

6. The class $\mathcal{NP}$–TIME, usually, abbreviated to just $\mathcal{NP}$, can be defined in two ways.

   (a) L is $\mathcal{NP}$–TIME if it can be accepted by some non-deterministic machine M in polynomial time, provided M makes the correct choice at every branch of the computation.

   (b) L is $\mathcal{NP}$–TIME if every member of L can be verified in polynomial time. What this means is that there is a machine V, called the verifier, such that if $w \in L$, there is some string c, called a certificate, such that V, given w and c, verifies that $w \in L$ in polynomial time, and such that, if $w \notin L$, V cannot verify w regardless of the certificate.

7. $\mathcal{P}$–SPACE is the class of all languages which can be decided by a deterministic machine which uses polynomial space but can take any amount of time.

8. A language is called recursive, or decidable, if it can be decided by some deterministic machine. There is no limit on time or space.

9. A language L is called recursively enumerable if some deterministic machine enumerates L; equivalently, if some machine accepts L.

10. A machine with a given size memory is a finite automaton. But most of the machines we discuss have unlimited memory. What does that mean? It means that the number of bits a machine remembers at any given time is finite, but that it can call on additional memory at any time, and there is no limit to the number of times it can obtain more memory.