P, NP, oracles and computation over structures

A couple of results that narrows down the possibilities with regard to the question $P \stackrel{?}{=} NP$ and related ones:

Thm 14.1 If $P \neq NP$ then there exists $L \in NP - P$ which is not NP-complete.

Thm 14.3 If some unary language $L \subseteq \{0\}^*$ is NP-complete, then $P = NP$.

Oracles

Def. A *TM with oracle*, denoted $M^?$ where $?$ can be replaced by any language, is like an ordinary TM, *except* that it has

- a special string, called the *query string* and
- three special states, $q_?$ (the query state), q_{yes} and q_{no} (the answer states).

Computation with an oracle:

Let $A \subseteq \Sigma^*$ be a language, also called oracle in this context.

A computation of M^A, on input x say, proceeds as usual except for when M^A enters the query state q_A:

Then M^A goes to q_{yes} or to q_{no} depending on whether the query string belongs to A or not. This is counted as one step in the computation.

By $M^A(x)$ we denote the output of $M^?$ with oracle A and input x.

A nondeterministic TM with oracle (an NTM with oracle) functions like a deterministic one; but we don't require that the transition relation is a function.

For any language/oracle A, the classes P^A and NP^A are defined as P and NP, except that we consider TM's or NTM's with oracle A.

Thm 14.4 There is an oracle A such that $P^A = NP^A$.

Thm 14.5 There is an oracle B such that $P^B \neq NP^B$.

Proof of 14.4 By Exercise 8.4.4 (part of the examination) there exists a PSPACE-complete language A.

The Corollary to Savitch theorem (7.5) immediately implies (as noted earlier) that

(1) $PSPACE = NPSPACE$.

As A is PSPACE-complete every $L \in PSPACE$ can be decided by a TM (with oracle A) in polyn. time by reducing L (in polyn. time) to A and then instantly deciding whether the result belongs to A with the oracle. Thus

(2) $PSPACE \subseteq P^A$

Clearly

(3) $P^A \subseteq NP^A$.

Since $A \in PSPACE$, every polyn. time NTM with oracle A can be simulated by a polyn. space NTM. Hence

(4) $NP^A \subseteq NPSPACE$

By taking (2), (3), (4), (1) in this order we get

$$PSPACE \subseteq P^A \subseteq NP^A \subseteq NPSPACE = PSPACE,$$

so $P^A = NP^A$. □

The idea of the proof of Thm 14.5

Let $M_1^?, M_2^?, \ldots$ be an enumeration of polyn. time TM's with oracle, such that for every polyn. time TM $M^?$ with oracle and every oracle A, there are infinitely many i's such that $M^A(x) = M_i^A(x)$ for all inputs x.

Next, one proceeds by a kind of diagonalization to define an oracle B such that if

$$L = \{0^n : \text{there is } x \in B \text{ with } |x| = n\}$$

then $L \in NP^B - P^B$.

Such L belongs to NP^B regardless of what B is, because $0^n \in L$ $\iff$ we can guess x with $|x| = n$ and check (instantly) whether $x \in B$ with B as an oracle.

But B must be chosen so that $L \notin P^B$.

We construct B as $B = \bigcup_{i \in \mathbb{N}} B_i$ where B_i is the set of strings in B with length $\leq i$. (so $B_i \subseteq B_{i+1}$).

At each stage i we make sure that B_i is defined in such a way that

$$L(M_i^B) \neq L.$$

As we make sure that this holds for every i we get $L \notin P^B$.

See the course book for the details. □

Computation over structures

By structure we mean a first-order structure, i.e. a set (or "universe/domain") together with some relations, functions and distinguished elements from that set, called "constants" which are named by "constant symbols". Here we assume that there are only finitely many functions, relations and constants, so a structure can be written as

$$\mathcal{U} = (U, \underbrace{R_1, \ldots, R_\alpha}_{\text{relations}}, \underbrace{f_1, \ldots, f_\beta}_{\text{functions}}, \underbrace{c_1, \ldots, c_\gamma}_{\text{constants}}).$$

(U: universe)

We now explain what we mean by a Turing machine (TM) over such $\mathcal{U}$.

A TM over $\mathcal{U} = (U, R_1, \ldots, R_\alpha, f_1, \ldots, f_\beta, c_1, \ldots, c_\gamma)$ is equiped with

- one or more strings/tapes, and
- one or more cursors on each string/tape. Each cursor scans a cell on the tape.

Every cell is either empty/blank or contains an element from U.

Input for computations will be strings of elements from U placed in the beginning of a tape assigned for inputs.

A <u>language</u> (or <u>problem</u>) is, in this setting, a set of strings (or finite sequences) of elements from U.

A TM over $\mathcal{U} = (U, R_1, \ldots, R_\alpha, f_1, \ldots, f_\beta, c_1, \ldots, c_\gamma)$ can perform the following things:

1. Move cursors a step left/right.

2. For every constant c_i, check whether the content of a cell scanned by a cursor is c_i:

   If $(x_m) = c_i$ then .... else ....

   content of cell scanned by cursor nr. m.

3. For every constant c_i, write c_i in the cell scanned by a cursor:

   $x_m := c_i$.

4. If the cells scanned by cursors nr. $n_1, \ldots, n_k$ contain $a_1, \ldots, a_k \in U$ and f_i is k-ary, then write $f_i(a_1, \ldots, a_k)$ in the cell scanned by cursor nr. m:

   $$x_m := f_i(x_{n_1}, \ldots, x_{n_k}).$$

5. If the contents of the cells scanned by cursors nr. $n_1, \ldots, n_k$ are $a_1, \ldots, a_k \in U$ and R_i is a k-ary relation, check whether $R_i(a_1, \ldots, a_k)$ holds in $\mathcal{U}$ ($\mathcal{U} \models R_i(a_1, \ldots, a_k)$?):

If $\mathcal{U} \models R_i(x_{n_1}, \ldots, x_{n_k})$ then ....
else ....

A <u>TM M over $\mathcal{U}$</u> is a finite set of instructions (and states) which,

given a state and cursor positions, tells which of the above actions is to be taken and which state to move to.

Ordinary TM's with a binary alphabet correspond to

TM's over $(\{0,1\}, , , 0, 1)$.

Erwin Engeler, in Algorithmic properties of structures (Math. Systems theory 1967) seem to be the first to consider computation over structures.

Computation over

$$(\mathbb{R}, <, +, -, \cdot, 0, 1)$$

and over rings in general has been studied extensively, by Blum, Shub, Smale and others.

Poizat (1994) introduced/reinvented the study of computation over first-order structures in general, and in particular with respect to the $P \stackrel{?}{=} NP$ problem.

Nondeterministic computation

Over structures, this can be defined in at least two reasonable ways.

N_1: nondeterministic mode of the 1st kind.

The machine functions as a deterministic one, except that from a given state and cursor positions there may be more than one possibility for which state to move to and which action to take.

N_2: nondeterministic mode of the 2nd kind. (over $\mathcal{U}$ with universe U).

The machine functions as a deterministic one, except that it can guess any elements from U and write them on a particular tape assigned for guessing. (In all other aspects it is deterministic.)

For any structure $\mathcal{U}$, the complexity classes $P_{\mathcal{U}}$, $N_1P_{\mathcal{U}}$ and $N_2P_{\mathcal{U}}$ are defined as the classes P and NP, just making the straight forward adaptions to the respective mode of computation. (So $P_{\mathcal{U}} \subseteq N_1P_{\mathcal{U}}$, $P_{\mathcal{U}} \subseteq N_2P_{\mathcal{U}}$.)

Some results

- If $\mathcal{U}$ has at least two distinct constants, then $N_1P_{\mathcal{U}} \subseteq N_2P_{\mathcal{U}}$.

- For every ring
$$\mathcal{R} = (R, ,^+, ^-, ^\cdot, 0, 1, c_1, \ldots, c_n)$$
without elimination of quantifiers
$$P_{\mathcal{R}} \neq N_2P_{\mathcal{R}}.$$

- For every Boolean ring
$R = (R, , +, -, \cdot, 0, 1, c_1, \ldots, c_n)$,
$P_R \neq N_1 P_R$ (and hence $P_R \neq N_2 P_R$).

- For every infinite abelian group
$G = (G, , +, -, 0)$
there is an expansion by constants
$G' = (G, , +, -, 0, c_1, \ldots, c_n)$
such that $P_{G'} \neq N_1 P_{G'}$, and
we may assume that $n \geq 1$ so also
$P_{G'} \neq N_2 P_{G'}$.

- There is a structure $\mathcal{U}$ such that
$P_{\mathcal{U}} = N_1 P_{\mathcal{U}} = N_2 P_{\mathcal{U}}$.