6. Complexity Results

Overview

6.1 Complexity Classes

6.2 Problem Statements and Assumptions

6.3 Results: Overview

6.1 Complexity Classes

6.1.1 Decision Problems

Definition 6.1 (Decision Problems)
Formally, a decision problem Π *consists of a pair* (D,Y) *of a set* D *of* problem instances, *which are strings from a language over a finite alphabet, and a subset* Y *of* D *which are called* "Yes"-instances. *The strings in* $D \setminus Y$ *are called* "No"-instances.*An algorithm correctly solves a decision problem* $\Pi = (D,Y)$, *if it outputs, given an instance* I *from* D, *"Yes" if* $I \in Y$ *holds, and outputs "No" otherwise.*

Definition 6.2 (Deterministic Complexity Classes)
The class P *contains all decision problems* Π *that are solvable in polynomial time on a deterministic Turing machine (DTM), i.e., there exists a DTM* M*, which solves* Π *and such that the running time of* M *is bounded by a polynomial in the length* $|I|$ *of the input* I*.*

Similarly, the class PSPACE *contains all decision problems which are solvable by some DTM in polynomial work space, i.e., the number of tape cells visited by the workhead of* M *is bounded by a polynomial in the length of the input* I*.*

For example, the problem ACYCLIC $= (D,Y)$ where D consists of the encodings of directed graphs (according to some standard encoding) and Y contains the (encodings of) acyclic graphs in D, can be solved in polynomial time. Informally, we say that the problem, given a graph, deciding whether the graph is acyclic, is possible in polynomial time, and thus the problem is in P.

A decision problem $\Pi = (D, Y)$ is solved by a nondeterministic Turing machine (NTM), if for every instance I of Π there is a run of M such that M outputs "Yes" if and only if I is a yes-instance. Note that no symmetric behavior is required for no-instances–M need even not halt on such instances.

Definition 6.3 (Nondeterministic Turing Machines)
A decision problem Π is solvable in nondeterministic polynomial time, if Π is solved by some NTM M such that a polynomial exists which bounds for each "Yes"-instance the time of the shortest run of M which outputs "Yes." The class NP contains all decision problems which are solvable in nondeterministic polynomial time.

For example, the well-known satisfiability problem SAT $= (D, Y)$, whose problem instances D are propositional formulas $\phi = \bigwedge_{i=1}^{m} C_i$ in conjunctive normal form (CNF), and whose "Yes"-instances are the satisfiable formulas in D, is a in NP.

Definition 6.4 (Problem Reduction, Hardness, and Completeness)
A decision problem Π is polynomial-time reducible to a decision problem Π', if there is a function f which is computable in polynomial time, such that for every instance I of Π, $f(I)$ is an instance of Π' and $f(I)$ is a "Yes"-instance of Π' just if I is a "Yes"-instance of Π.

A problem Π is said to be hard *for a class of problems C (i.e., C-*hard*), if every problem in C is polynomial-time reducible to Π. Furthermore, a problem Π is* complete *for C, if Π is hard for C and Π belongs to C.*

Definition 6.5 (Oracle Turing Machine)
An oracle Turing machine (OTM) is a Turing machine equipped with a query tape, on which queries to a fixed problem can be posed. Any such query is answered upon entering a designated query state in one step.

For a complexity class C, we denote by P^C (resp., NP^C) the class of all decision problems which can be solved by a deterministic (resp., nondeterministic) OTM wich the help of an oracle in C.

In particular, P^{NP} (resp., $\mathrm{NP}^{\mathrm{NP}}$) is the class of problems solvable in polynomial time on a deterministic (resp., nondeterministic) Turing machine, if an oracle for a problem in NP may be used.

1. All classes of the polynomial hierarchy have complete problems, including canonical complete problems in terms of evaluating quantified Boolean formulas.
2. A *quantified Boolean formula* (QBF) is a generalized propositional formula, in which each propositional variable x_i ranges over $\{ true, false \}$ and is governed either by an existential ($\exists$) or a universal ($\forall$) quantifier.
3. The truth value of such a formula is obtained by eliminating all quantifiers in the obvious way and evaluating the resulting variable-free formula.
4. For example, $\forall y_1, y_2 \exists x_1, x_2 (x_1 \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge y_1)$ is a QBF. It evaluates to *false*, since on assigning e.g. *false* to both y_1 and y_2, the remaining formula is unsatisfiable.

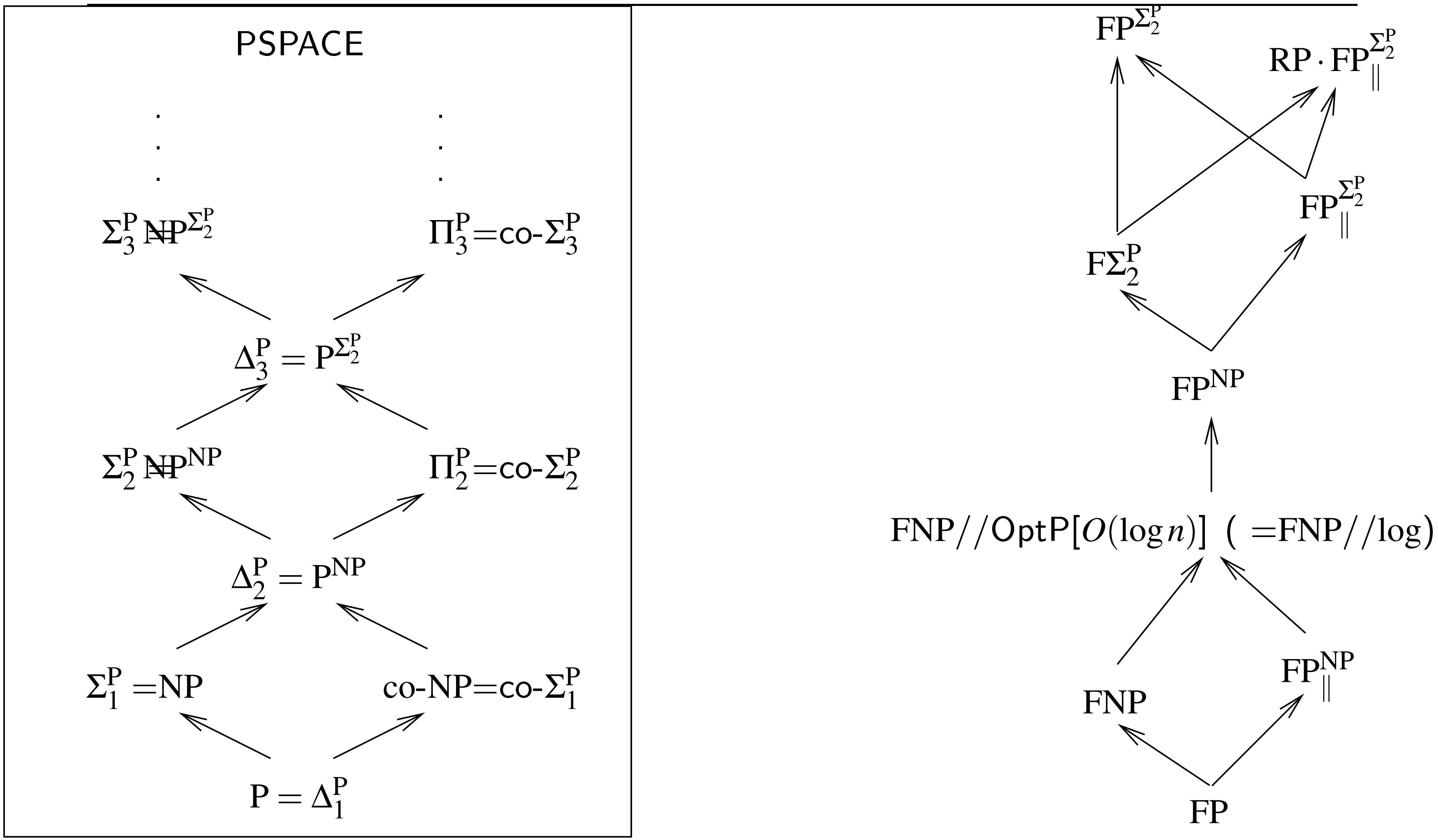

Figure 6.1: Decision (left) and Search (right) Problem Complexity Classes

6.1.2 Search Problems

In our analysis, we will also encounter problems which are not decision problems, but return a value as the solution to a problem instance, rather then reporting "Yes" or "No."

Definition 6.6 (Search Problem)
A search problem $\Pi = (D, S)$ *is a generalization of a decision problem, in which for every instance* $I \in D$ *of the problem a (possibly empty) finite set* $S(I)$ *of solutions exists. To solve such a problem, an algorithm must (possibly nondeterministically) compute the solutions* $S(I)$ *in its computation branches, if* $S(I)$ *is not empty. Classes of search problems, which are also referred to as* function classes, *are defined in analogy to classes of decision problems.*

Thus, while the decision problem SAT asks whether an instance I, given by a set of propositional clauses, is satisfiable, the set $S(I)$ of solutions for the corresponding search problem FSAT consists of all satisfying valuations for I.

Definition 6.7 (Search Problem Transformation)
A search problem Π_1 is polynomial-time transformable to Π_2, if (i) from every instance I of Π_1, an instance $f(I)$ of Π_2 is constructible in polynomial time, such that $f(I)$ has some solution precisely if I has, and (ii) from every solution S of $f(I)$, a solution of I can be constructed in time polynomial in the size of S and I.

Definition 6.8 (Hardness and Completeness)
As in the case of decisions problems, a search problem Π *is hard for a class of search problems* C*, if every problem in* C *is polynomial-time transformable into* Π*. A search problem* Π *is complete for* C*, if* Π *is hard for* C *and belongs to* C*.*

We now describe some the search problem classes that will shall encounter in this chapter. The search problem counterparts of the classes C in the polynomial hierarchy are often denoted by a prefixed "F"; some of them appear in Figure 6.1.

The Classes FP**,** $\mathrm{FP}^{\mathrm{NP}}$**, and** $\mathrm{FP}^{\Sigma_2^{\mathrm{P}}}$**:** These are the classes of functions computable by a deterministic Turing machine in polynomial time with no oracle, NP-oracle, and Σ_2^{P}-oracle, respectively.

The Classes FNP **and** $\mathrm{F}\Sigma_2^{\mathrm{P}}$**:** FNP (resp., $\mathrm{F}\Sigma_2^{\mathrm{P}}$) contains the multi-valued functions whose solutions can be computed by a nondeterministic transducer in polynomial time (resp., in polynomial time with an NP-oracle), such that a given solution candidate can be checked in polynomial time (resp., the check is in $\mathrm{co\text{-}NP}$).

The Class $\mathrm{FNP}//\mathrm{log}$: $\mathrm{FNP}//\mathrm{log}$ is short for the class $\mathrm{FNP}//\mathrm{OptP}[O(\log n)]$ (Chen and Toda 1995).

Intuitively, it is the class of problems such that a solution for an instance I can be nondeterministically computed by a transducer in polynomial time, if the optimal value $\mathrm{opt}(I)$ of an NP optimization problem on I (an integer) is known, where $\mathrm{opt}(I)$ (represented in binary) must have $O(\log |I|)$ bits.

E.g., computing the largest set S of pairwise connected nodes in a given graph G (i.e., a maximum clique) is a problem in $\mathrm{FNP}//\mathrm{log}$ (observe that different maximum cliques may exist).

The class $\mathrm{FNP}//\mathrm{log}$ reduces to $\mathrm{FP}^{\mathrm{NP}}$ and roughly amounts to a randomized version of $\mathrm{FP}^{\mathrm{NP}}_{||}$.

The Class $\mathrm{RP} \cdot \mathrm{FP}_{\|}^{\Sigma_2^P}$**:** The class $\mathrm{RP} \cdot \mathrm{FP}_{\|}^{\Sigma_2^P}$ (Chen and Toda 1995) contains informally those problems for which a solution on input I can be found by a random polynomial time algorithm with very high probability, by using a problem in $\mathrm{FP}_{\|}^{\Sigma_2^P}$ as single-call subroutine.

6.2 Problem Statements and Assumptions

In our analysis, we study four types of complexity problems. The problems are generic for the type of status set semantics which the agent is willing to base its decisions on. Let Sem be any kind of status sets.

Consistency: The problem is deciding whether an agent program $\mathcal{P}$ is consistent on an agent state $\mathcal{O}_\mathcal{S}$, i.e., whether $\mathcal{P}$ possesses a Sem-status set on $\mathcal{O}_\mathcal{S}$.

Recognition: Decide whether a given status set S is a Sem-status set of an agent program $\mathcal{P}$ on an agent state $\mathcal{O}_\mathcal{S}$.

Computation: Compute some arbitrary Sem-status set of an agent program $\mathcal{P}$ on a given agent state $O_{\mathcal{S}}$.

Action Reasoning: This comprises elementary reasoning about an agent's behavior in a certain situation, and comes in two different variants. The problem is deciding, from an agent program $\mathcal{P}$ and an agent state $O_{\mathcal{S}}$, whether the agent takes an action α under Sem-status sets, under the

- *possibility variant*, i.e., whether α is executed according to some Sem-status set, or the
- *certainty variant*, i.e., whether α is executed according to every Sem-status set.

Assumptions on Problem Instances

System Background: We assume that that the environment in which agent programs are evaluated does not change. That is, we assume that there is a background which consists of a software code $\mathcal{S}$, an action base $\mathcal{AB}$, a set of action constraints $\mathcal{AC}$, and a set of integrity constraints $\mathcal{IC}$, each of which is fixed and does not change over time.

Fixed Agent Program: Furthermore, we assume that the agent program $\mathcal{P}$ to be evaluated is fixed.

- The above setting corresponds to what is commonly called the *data complexity* of a program in the database field (Vardi 1982).
- If we would consider varying programs where the agent state is *fixed* (resp., *varying*), we would have *expression (or program) complexity and combined complexity*, which are typically one exponential higher than data complexity.

Assumptions about Software Code

- Evaluating an agent program built on top of arbitrary software code is not effectively possible.
- In all available commercial software packages, calls to a function are completed in a finite amount of time.
- Our aim is to analyze the cost of agent decision making *per se*, and to see under which circumstances this is efficiently possible.

Domain Closure Assumption: All objects considered for grounding the program rules, evaluation of the action preconditions, the conditions of the actions constraints and the integrity constraints must be from $O_{\mathcal{S}}$, or they must be constructible from objects therein by operations from a fixed (finite) set in a number of steps that is bounded by some constant, and such that each operation is efficiently executable (i.e., in polynomial time) and involves only a number of objects bounded by some other constant.

Polynomial Code Calls: We further assume that the evaluation time of code condition calls χ over a state $O_{\mathcal{S}}$, for any particular legal assignment of objects, is bounded by a polynomial in the size of $O_{\mathcal{S}}$. Moreover, we assume that given an agent state $O_{\mathcal{S}}$ and a set of ground actions $\mathcal{A}$, the state $O'_{\mathcal{S}}$ which results under weakly-concurrent execution of $\mathcal{A}$ on $O_{\mathcal{S}}$ (see Definition 4.6 on page 132) is constructible in polynomial time.

Domain Closure implies that the number of objects which may be relevant to a fixed agent program $\mathcal{P}$ on a given state $O_{\mathcal{S}}$ is bounded by a polynomial in the number of objects in $O_{\mathcal{S}}$.

Also, each such object can be generated in polynomial time.

In particular, this also means that the number of ground rules of $\mathcal{P}$ which are relevant is polynomial in the size of $O_{\mathcal{S}}$, measured by the number of objects that it contains.

As a consequence of these assumptions,

- the action and integrity constraints are evaluable on an agent state $O_{\mathcal{S}}$ under the generalized active domain semantics in polynomial time, and
- the integrity constraints on the agent state $O'_{\mathcal{S}}$ resulting from the execution of a set of actions A grounded in the active domain, are checkable in polynomial time in the size of $O_{\mathcal{S}}$.

6.3 Results

Sem-status set	consistency	computation	recognition	action reasoning: possible	action reasoning: certain
feasible	P	FP	P	NP	co-NP
rational $\equiv$ reasonable $\equiv$ F-preferred rational $\equiv$ F-preferred reasonable	P	FP	P	P	P
weak rational $\equiv$ weak reasonable	P	FP	P	NP	co-NP

Table 6.1: Complexity of Positive Agent Programs, $IC = \emptyset$

				action reasoning	
Sem-status set	**consistency**	**computation**	**recognition**	**possible**	**certain**
feasible	NP	FNP	P	NP	co-NP
rational	NP	FNP//log	co-NP	Σ_2^P	co-NP
reasonable	NP	FNP	P	NP	co-NP
F-preferred rational	NP	FNP//log	co-NP	Σ_2^P	Π_2^P
F-preferred reasonable	NP	FNP//log	co-NP	Σ_2^P	Π_2^P
weak rational	NP	FNP//log	co-NP	Σ_2^P	Π_2^P
weak reasonable	NP	FNP//log	co-NP	Σ_2^P	Π_2^P

Table 6.2: Complexity of Agent Programs with Negation, $IC = \emptyset$

				action reasoning	
Sem-status set	**consistency**	**computation**	**recognition**	**possible**	**certain**
feasible	NP	FNP	P	NP	co-NP
rational $\equiv$ reasonable $\equiv$ F-preferred rational $\equiv$ F-preferred reasonable	P	FP	P	P	P
weak rational $\equiv$ weak reasonable	NP	FNP//log *	co-NP	NP	Π_2^P

* ... hard for both FNP and $FP_{\|}^{NP}$

Table 6.3: Complexity of Positive Agent Programs, arbitrary IC

				action reasoning	
Sem-status set	**consistency**	**computation**	**recognition**	**possible**	**certain**
feasible	NP	FNP	P	NP	co-NP
rational	Σ_2^P	$F\Sigma_2^P$	co-NP	Σ_2^P	Π_2^P
reasonable	NP	FNP	P	NP	co-NP
F-preferred rational	Σ_2^P	$FP^{\Sigma_2^P} \cap RP \cdot FP_{\parallel}^{\Sigma_2^P}$	Π_2^P	Σ_3^P	Π_3^P
F-preferred reasonable	NP	FNP//log	co-NP	Σ_2^P	Π_2^P
weak rational	Σ_2^P	$FP^{\Sigma_2^P} \cap RP \cdot FP_{\parallel}^{\Sigma_2^P}$	Π_2^P	Σ_3^P	Π_3^P
weak reasonable	NP	FNP//log	co-NP	Σ_2^P	Π_2^P

Table 6.4: Complexity of Agent Programs with Negation, arbitrary IC