# I Don't Want to Think About it Now: Decision Theory With Costly Computation

Joseph Y. Halpern*
Cornell University
halpern@cs.cornell.edu

Rafael Pass†
Cornell University
rafael@cs.cornell.edu

**Abstract**

Computation plays a major role in decision making. Even if an agent is willing to ascribe a probability to all states and a utility to all outcomes, and maximize expected utility, doing so might present serious computational problems. Moreover, computing the outcome of a given act might be difficult. In a companion paper we develop a framework for game theory with costly computation, where the objects of choice are Turing machines. Here we apply that framework to decision theory. We show how well-known phenomena like *first-impression-matters biases* (i.e., people tend to put more weight on evidence they hear early on), *belief polarization* (two people with different prior beliefs, hearing the same evidence, can end up with diametrically opposed conclusions), and the *status quo* bias (people are much more likely to stick with what they already have) can be easily captured in that framework. Finally, we use the framework to define some new notions: *value of computational information* (a computational variant of *value of information*) and *computational value of conversation*.

## 1 Introduction

Computation plays a major role in decision making. Even if an agent is willing to ascribe a probability to all states and a utility to all outcomes, and maximize expected utility—that is, to follow the standard prescription of rationality as recommended by Savage [1954], doing so might present serious computational problems. Computing the relevant probabilities

*Supported in part by NSF grants IIS-0534064, IIS-0812045, and IIS-0911036, and by AFOSR grants FA9550-08-1-0438 and FA9550-09-1-0266, and ARO grant W911NF-09-1-0281.

†Supported in part by a Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR Award FA9550-08-1-0197, and BSF Grant 2006317.

might be difficult, as might computing the relevant utilities. Work on Bayesian networks [Pearl 1988] and other representations of probability, and related work on representing utilities [Bacchus and Grove 1995; Boutilier, Brafman, Domshlak, Hoos, and Poole 2004] can be viewed as attempts to ameliorate these computational problems. Our focus is on the complexity of computing the outcome of an act in a given state. Consider the following simple example, taken from [Halpern and Pass 2010].

Suppose that a decision maker (DM) is given an input $n$, and is asked whether it is prime. The DM gets a payoff of \$1,000 if he gives the correct answer and loses \$1,000 if he gives the wrong answer. However, he also has the option of playing safe, and saying "pass", in which case he gets a payoff of \$1. Clearly, many DMs would say "pass" on all but simple inputs, where the answer is obvious, although what counts as a "simple" input may depend on the DM.[1]

In [Halpern and Pass 2010], we introduced a model of game theory with costly computation. Here we apply that framework to decision theory. We assume that the DM can be viewed as choosing an algorithm (i.e., a Turing machine); with each Turing machine (TM) $M$ and input, we associate its *complexity*. The complexity can represent, for example, the running time of $M$ on that input, the space used, the complexity of $M$ (e.g., how many states it has), or the difficulty of finding $M$ (some algorithms are more obvious than others). We deliberately keep the complexity function abstract, to allow for the possibility of representing a number of different intuitions. The DM's utility can then depend, not just on the payoff, but on the complexity.

The DM's goal is to choose the "best" TM; the one that will give him the greatest expected utility, taking both the payoff and complexity into account. To make this choice, the DM must have beliefs about the TM's running time and the "goodness" of the TM's output. For example, if the TM outputs "prime" on some input $n$, then TM must have beliefs about how likely $n$ is to actually be prime. As this example suggests, we actually need here to deal with what philosophers have called "impossible" possible worlds [Hintikka 1975; Rantala 1982]. If $n$ is a prime, then this is a mathematical fact; there can be no state where $n$ is not prime; nevertheless, since we want to allow for DMs that are resource-bounded and cannot compute whether $n$ is prime, we want it to be possible for the DM to believe that $n$ is not prime. Similarly, if the complexity function is supposed to measure running time, then the actual running time of a TM $M$ on input $t$ is a fact of mathematics; nevertheless, we want to allow the DM to have false beliefs about $M$'s running time. We capture such false beliefs by having both the utility function and the complexity function depend on the state of nature.

As we show here, using these simple ideas leads to quite a powerful framework. For ex-

[1] While primality testing is now known to be in polynomial time [Agrawal, Keyal, and Saxena 2004], and there are computationally-efficient randomized algorithms that that give the correct answer with extremely high probability [Rabin 1980; Solovay and Strassen 1977], we can assume that the DM has no access to a computer.

ample, many concerns expressed by the emerging field of *behavioral economics* (pioneered by Kahneman and Tversky [Kahneman, Slovic, and Tversky 1982]) can be accounted for by simple assumptions about players' cost of computation. To illustrate this point, we show that *first-impression-matters biases* [Rabin 1998], that is, that people tend to put more weight on evidence they hear early on, can be easily captured using computational assumptions. We can similarly explain *belief polarization* [**?**]—that two people, hearing the same information (but with possibly different prior beliefs) can end up with diametrically opposed conclusions. Finally, we can also use the framework to formalize one of the intuitions for the well-known *status quo* bias [Samuelson and Zeckhauser 1998]: people are much more likely to stick with what they already have.

As a final application, we use the framework to define a new notion: *value of computational information*. To explain it, we first recall *value of information*, a standard notion in decision analysis. Value of information is meant to be a measure of how much a DM should be willing to pay to receive new information. The idea is that, before receiving the information, the DM has a probability on a set of relevant events and chooses the action that maximizes his expected utility, given that probability. If he receives new information, he can update his probabilities (by conditioning on the information) and again choose the action that maximizes expected utility. The difference between the expected utility before and after receiving the information is the value of the information.

In many cases, a DM seems to be receiving valuable information that is not about what seem to be the relevant events. This means that we cannot do a value of computation calculation, at least not in the obvious way. For example, suppose that the DM is interested in learning a secret, which we assume for simplicity is a number between 1 and 1000. A priori, suppose that the DM takes each number to be equally likely, and so has probability .001. Learning the secret has utility, say, \$1,000,000; not learning it has utility 0. The number is locked in a safe, whose combination is a 40-digit binary numbers. What is the value to the DM of learning the first 20 digits of the combination? As far as value of information goes, it seems that the value is 0. The events relevant to the expected utility are the possible values of the secret; learning the combination does not change the probabilities of the numbers at all. This is true even if we put the possible combinations of the lock into the sample space. On the other hand, it is clear that people may well be willing to pay for learning the first 20 digits. It converts an infeasible problem (trying $2^{40}$ combinations by brute force) to a feasible problem (trying $2^{20}$ combinations).

Although this example is clearly contrived, there are many far more realistic situations where people are clearly willing to pay for information to improve computation. For example, companies pay to learn about a manufacturing process that will speed up production; people buy books on speedreading; and faster algorithms for search clearly are considered valuable. We show that we can use our computational framework to make the notion of *value of computational information* precise, in a way that makes it a special case of value

of information.[2] In addition, we define a notion of *computational value of conversation*, where the DM can communicate interactively with an informed observer before making a decision (as opposed to just getting some information). Interestingly, the notion of *zero knowledge* [Goldwasser, Micali, and Rackoff 1989] gets an elegant interpretation in this framework. Roughly speaking, a zero-knowledge algorithm for membership in a language $L$ is one where there is no added value of conversation in running the algorithm beyond what there would be in learning whether an input $x$ is in $L$, no matter what random variable is of interest to the DM.

In the next section we define our computational framework carefully, and show how it delivers reasonable results in a number of examples. In Section 3, we consider the value of computational information. We conclude with a discussion of related work in Section 4.

# 2 A computational framework

The framework we use here for adding computation to decision theory is essentially a single-agent version of what were called in [Halpern and Pass 2010] *Bayesian machine games*. In a standard Bayesian game, each player has a *type* in some set $T$, and then makes a single move. Player $i$'s type can be viewed as describing $i$'s initial information; some facts that $i$ knows about the world. In the number-in-the-safe example, there is essentially only one type, since the DM gets no information. In the case of the manufacturing process, the type could be the configuration of the system; manufacturing processes typically apply to a number of configurations. We assume that an agent's move consists of choosing a Turing machine. As we said in the introduction, associated with each Turing machine and type is its complexity. Given as input a type, the Turing machine outputs an action. The utility of a player depends on the type *profile* (i.e., the types of all the players), the action profile, and the complexity profile. (While typically all that matters to player $i$ is the complexity of his algorithm, it may, for example, matter to him that his algorithm is faster than that of player $j$.)

Turning to decision theory, we take a *standard decision problem with types* to be characterized by a tuple $(S, T, A, \Pr, u)$, where $S$ is a state space, $T$ is a set of types, $A$ is a set of actions, $\Pr$ is a probability distribution on $S \times T$ (there may be correlation between states and types), and $u : S \times T \times A \to \mathbb{R}$, where $u(s, t, a)$ is the DM's utility if he performs action $a$ in state $s$ and has type $t$.[3] (It is not typical to consider a decision maker's type in standard decision theory, but it does not hurt to add it; it will prove useful once we consider computation.) For each action $a$, we can consider the random variable $u_a$ defined on $S$

[2] Our notion of value of computational information is related to, but not quite the same as, the notion of *value of computation* introduced by Horvitz [1987, 2001]; see Section 4.

[3] In [Halpern and Pass 2010], we did not have a state space $S$, but we assumed that nature had a type. Nature's type can be identified with the state.

by taking $u_a(s,t) = u(s,t,a)$. The expected utility of action $a$, denoted $E_{\Pr}[u_a]$, is just the expected value of the random variable $u_a$ with respect to the probability distribution $\Pr$; that is, $E_{\Pr}[u_a] = \sum_{(s,t)\in S\times T} \Pr(s,t)u(s,t,a)$. We assume that the DM is an expected utility maximizer, so he chooses an action $a$ with the largest expected utility.

To combine the ideas of Bayesian machine games and decision problems, we consider *computational decision problems*. In a computational decision problem, just like in a computational Bayesian machine game, the DM chooses a Turing machine. We assume that the action performed by the TM depends on the type. We denote by $M(t)$ the output of the machine on input the type $t$. To capture the DM's uncertainty about the TM's output, we use an *output function* $\mathcal{O} : \mathbf{M} \times S \times T \to I\!N$, where $\mathbf{M}$ denotes the set of Turing Machines; $\mathcal{O}(M,s,t)$ is used to describe what the DM thinks the output of $M(t)$ is in state $s$. To simplify the presentation, we abuse notation and use $M(s,t)$ to denote $\mathcal{O}(M,s,t)$.

The DM's utility will depend on the state $s$, his type $t$, and the action $M(s,t)$, as is standard; in addition, it will depend on the "complexity" of $M$ given input $t$. The complexity of a machine can represent, for example, the running time or space usage of $M$, or the complexity of $M$ itself, or some combination of these factors. For example, Rubinstein [1986] considers what can be viewed as special case of our model, where the DM chooses a finite automaton (and has no type); the complexity of $M$ is the number of states in the description of the automaton. To capture the cost of computation formally, we use a *complexity function* $\mathcal{C} : \mathbf{M} \times S \times T \to I\!N$, to describe the complexity of a TM given an input type and state. (As we shall see, by allowing the state to be included as an argument to $\mathcal{C}$, we can capture the DM's uncertainty about the complexity.)

We define a *computational decision problem* to be a tuple $\mathcal{D} = (S, T, A, \Pr, \mathcal{M}, \mathcal{C}, \mathcal{O}, u)$, where $S$, $T$, $A$, and $\Pr$ are as in the definition of a standard decision problem, $\mathcal{M} \subseteq \mathbf{M}$ is a set of TMs (intuitively, the set that the DM can choose among), $\mathcal{O}$ is an output function, $\mathcal{C}$ is a complexity measure, and $u : S \times T \times A \times I\!N \to I\!R$. The expected utility of a TM $M$ in the decision problem $\mathcal{D}$, denoted $U_{\mathcal{D}}(M)$, is $\sum_{s\in S, t\in T} \Pr(s,t)u(s,t,\mathcal{O}(M,s,t),\mathcal{C}(M,s,t))$. Note that now the utility function gets the complexity of $M$ as an argument. For ease of exposition here, we restrict to deterministic TMs for most of the paper; we need to consider randomized TMs for our results on zero knowledge.

**Example 2.1** Consider the primality-testing problem discussed in the introduction. Formally, suppose that the DM's type is just a natural number $< 2^{40}$, and the DM must determine whether the type is prime. The DM can choose either 0 (the number is not prime), 1 (the number is prime), or 2 (pass). If $M$ is a TM, then $M(s,t)$ is $M$'s output in state $s$ on input $t$. The state $s$ here is used to capture the DM's uncertainty about the output. So if the DM believes that the DM will output pass with probability $2/3$, then the set of states such that $M(s,t) = 2$ has probability $2/3$. Let $\mathcal{C}(s,t,M)$ be 0 if $M$ computes the answer within $2^{20}$ steps on input $t$, and 10 otherwise. (Think of $2^{20}$ steps as representing representing a hard deadline.) Here the state $s$ encodes the DM's uncertainty about the running time of

$M$. For example, if the DM does not know the running time of $M$, but ascribes probability $2/3$ to $M$ finishing in less than $2^{20}$ steps on input $t$, then the set of states $s$ such that $\mathcal{C}(s, t, M) = 0$ has probability $2/3$. Finally, let utility $u(s, t, a, c) = 10 - c$ if $a$ is either 0 or 1, and this is the correct answer in state $s$ (that is, $t$ is viewed as prime in state $s$ and $a = 1$, or $t$ is not viewed as prime in state $s$ and $a = 0$), and $u(s, t, 2, c) = 1 - c$. Now the state $s$ is used to encode the DM's uncertainty about the correctness of $M$'s answer. (Note that we are allowing "impossible" states, where $t$ is viewed as prime in state $s$ even though it is in fact composite; this is needed to model the DM's uncertainty.) Thus, if the DM is sure that $M$ always gives the correct output, then $u(s, t, a, c) = 10 - c$ for all states $s$ and $a \in \{0, 1\}$.

We can also consider a variant of this problem, where the DM is given a specific input $t$ and is asked if $t$ is prime. Although there is obviously a right answer (the number is prime or it's not), the DM might still have uncertainty regarding whether a particular TM $M$ gives the right answer, the running time of $M$, and the output of $M$. ∎

**Example 2.2** Consider the number-in-the-safe example from the introduction. Here there is only a single type, $t_0$; we can think of the state space $S$ as consisting of pairs $(s_1, s_2, s_3)$, where $s_1$ is the number in the safe, $s_2$ is the combination, and $s_3$ encodes the DM's beliefs about the complexity and correctness of TMs. An algorithm in this case is just a sequence of combinations to try and a stopping rule. Suppose that the agent gets utility $10 - \mathcal{C}((s_1, s_2, s_3), t_0, M)$ if $s_2$ (the actual combination) is one of the numbers generated by $M$ before it halts, and $0 - \mathcal{C}((s_1, s_2, s_3), t_0, M)$ otherwise, where $\mathcal{C}((s_1, s_2, s_3), t_0, M)$ is 0 if $M$ halts within $2^{20}$ steps in state $(s_1, s_2, s_3)$, and 10 otherwise. ∎

**Example 2.3 (Biases in information processing)** Psychologists have observed many systematic biases in the way that individuals update their beliefs as new information is received (see [Rabin 1998] for a survey). In particular, a "first-impressions-matter" bias has been observed: individuals put too much weight on initial signals and less weight on later signals. As they become more convinced that their beliefs are correct, many individuals even seem to simply ignore all information once they reach a confidence threshold. Several papers in behavioral economics have focused on identifying and modeling some of these biases (see, e.g., [Rabin 1998] and the references therein, [Mullainathan 2002], and [Rabin and Schrag 1999]). In particular, Mullainathan [2002] makes a potential connection between memory and biased information processing, using a model that makes several explicit (psychology-based) assumptions on the memory process (e.g., that the agent's ability to recall a past event depends on how often he has recalled the event in the past). More recently, Wilson [2002] presents an elegant model of bounded rationality, where agents are described by finite automata, which (among other things) can explain why agents eventually choose to ignore new information; her analysis, however, is very complex and holds only in the limit (specifically, in the limit as the probability $\nu$ that a given round is the last round goes to 0).

As we now show, the first-impression-matters bias can be easily explained if we assume that there is a small cost for "absorbing" new information. Consider the following simple game (which is very similar to the one studied by Mullainathan [2002] and Wilson [2002]). The state of nature is a bit $b$ that is 1 with probability $1/2$. An agent receives as his type a sequence of independent samples $s_1, s_2, \ldots, s_n$ where $s_i = b$ with probability $\rho > 1/2$. The samples corresponds to signals the agents receive about $b$. An agent is supposed to output a guess $b'$ for the bit $b$. If the guess is correct, he receives $1 - mc$ as utility, and $-mc$ otherwise, where $m$ is the number of bits of the type he read, and $c$ is the cost of reading a single bit ($c$ should be thought of the cost of absorbing/interpreting information). It seems reasonable to assume that $c > 0$; signals usually require some effort to decode (such as reading a newspaper article, or attentively watching a movie). If $c > 0$, it easily follows by the Chernoff bound that after reading a certain (fixed) number of signals $s_1, \ldots, s_i$, the agents will have a sufficiently good estimate of $\rho$ that the marginal cost of reading one extra signal $s_{i+1}$ is higher than the expected gain of finding out the value of $s_{i+1}$. That is, after processing a certain number of signals, agents will eventually disregard all future signals and base their output guess only on the initial sequence. We omit the straightforward details.

Essentially the same approach allows us to capture belief polarization. Suppose for simplicity that two agents start out with slightly different beliefs regarding the value of some random variable $X$ (think of $X$ as representing something like "O.J. Simpson is guilty"), and get the same sequence $s_1, s_2, \ldots, s_n$ of evidence regarding the value of $X$. (Thus, now the type consists of the initial belief, which can for example be modeled as a probability or a sequence of evidence received earlier, and the new sequence of evidence). Both agents update their beliefs by conditioning. As before, there is a cost of processing a piece of evidence, so once a DM gets sufficient evidence for either $X = 0$ or $X = 1$, he will stop processing any further evidence. If the initial evidence supports $X = 0$, but the later evidence supports $X = 1$ even more strongly, the agent that was initially inclined towards $X = 0$ may raise his beliefs to be above threshold, and thus stop processing, believing that $X = 0$, while the agent initially inclined towards $X = 1$ will continue processing and eventually believe that $X = 1$. ∎

**Example 2.4 (Status quo bias)** The status quo bias is well known. To take just one example, Samuelson and Zeckhauser [1998] observed that when Harvard University professors were offered the possibility of enrolling in some new health-care options, older faculty, who were already enrolled in a plan, enrolled in the new option much less often than new faculty. Assuming that all faculty evaluate the plans in essentially the same way, this can be viewed as an instance of a status quo bias. Samuelson and Zeckhauser suggested a number of explanations for this phenomenon, one of which was computational. As they point out, the choice to undertake a careful analysis of the options is itself a decision. Someone who is already enrolled in a plan and is relatively happy with it can rationally decide that it is

not worth the cost of analysis (and thus just stick with her current plan), while someone who is not yet enrolled is more likely to decide that the analysis is worthwhile. This explanation can be readily modeled in our framework. An agent's type can be taken to be a description of the alternatives. A TM decides how many alternatives to analyze. There is a cost to analyzing an alternative, and we require that the decision made be among the alternatives analyzed or the status quo. (We assume that the status quo has already been analyzed, through experience.) If the status quo already offers an acceptable return, then a rational agent may well decide not to analyze any new alternatives. Interestingly, Samuelson and Zeckhauser found that, in some cases, the status quo bias is even more pronounced when there are more alternatives. We can capture this phenomenon if we assume that, for example, that there is an initial cost to analyzing, and the initial cost itself depends in part on how many alternatives there are to analyze (so that it is more expensive to analyze only three alternatives if there are five alternatives altogether than if there only three alternatives). This would be reasonable if there is some setup cost in order to start the analysis, and the setup depends on the number of items to be analyzed. ∎

# 3 Value of computational information

## 3.1 Value of information: a review

Before talking about value of computational information, we briefly review value of information. Consider a standard decision problem. To deal with value of information, we consider a partition of the state space $S$. The question is what it would be worth to the DM to find out which cell in the partition the true state is in. (Think of the cells in the partition as corresponding to the possible realizations of a random variable $X$, and the value of information as corresponding to the value of learning the actual realization of $X$.) Of course, the value may depend on the DM's type $t$. To compute the value of information, we compute the expected expected utility of the best action given type $t$ conditional on receiving the information, and compare it to the expected utility of the best action for type $t$ before finding out the information. We talk about "expected expected utility" here because we need to take into account how likely the DM is to discover that he is in a particular cell.

**Example 3.1** Suppose that an investor can buy either a stock or bond. There are two states of the world, $s_1$ and $s_2$, and a single type $t_0$. A priori, the investor thinks $s_1$ has probability $2/3$ and $s_2$ has probability $1/3$. Buying the bond gives him a guaranteed utility of 1 (in both $s_1$ and $s_2$). In state $s_1$, buying the stock gives a utility of 3; in state $s_2$, buying the stock gives a utility of $-4$. Clearly, a priori, buying the stock has an expected utility of $2/3$, so buying the bond has a higher expected utility. What is the value of learning the true state (which corresponds to the partition $\{\{s_1\}, \{s_2\}\}$)? Clearly if the true state is $s_1$, buying the stock is the best action, and has (expected) utility 3; in state $s_2$, buying the bond is the best

action, and has expected utility 1. Thus, the expected expected utility of the information is $(2/3)3 + (1/3)1 = 7/3$ (since with probability $2/3$ the DM expects to learn that it is state $s_1$ and with probability $1/3$ the DM expects to learn that it is $s_2$), and so the value of information is $7/3 - 1 = 4/3$. ∎

We leave it to the reader to write the obvious formal definition of value of information in type $t$.

## 3.2 Value of computational information

In our framework, it is easy to model the value of computational information: it is just a special case of value of information. Formally, given a standard decision problem $(S, T, A, \Pr, u)$, we must first extend it to a computational decision problem $(S', T, A, \Pr, \mathcal{M}, \mathcal{C}, \mathcal{O}, u')$. $\mathcal{M}$ is some appropriate set of TMs; each TM in $\mathcal{M}$ outputs an action in $A$ given an element of $S' \times T$. As discussed in Section 2, we need a richer state space to capture the DM's uncertainty regarding the output of the TM and the running time of the TM chosen. We can take $S'$ to have the form $S \times S''$, where $s'' \in S''$ determines the running time and output of each TM $M \in \mathcal{M}$. Similarly, $u'((s, s''), t_0, M((s, s''), t_0), \mathcal{C}((s, s''), t_0, M))$ depends on $u(s, M((s, s''), t))$ and $\mathcal{C}((s, s''), t, M)$. (For example, we can assume that $u'((s, s''), t_0, M'((s, s''), t_0), \mathcal{C}((s, s''), t_0, M)) = u(s, M(s, t)) - \mathcal{C}((s, s''), t, M)$, but we do not require this.)

In this setting, value of computational information essentially becomes a special case of value of information. The only difference is that since the machine set $\mathcal{M}$ might be infinite, there might not exist a machine with maximal expected utility. So, instead of comparing the expected utilities of the best machines (before and after receiving the information), we compare the *supremum* of the expected utilities of machine $M \in \mathcal{M}$ (before and after receiving the information). More precisely, given a partition $Q$ of the state of nature, for every cell $q \in Q$, let $\Pr_q$ denote the distribution $\Pr$ conditioned on event that the state of nature is part of the cell $q$. and let the random variable $\mathbf{q}(s, t)$ denote the cell of $s$. The value of computational information (of learning what cell $q \in Q$ the state of nature is in) is

$$E_{\Pr}\left[\sup_{M \in \mathcal{M}} E_{\Pr_{\mathbf{q}}}[u'_M]\right] - \sup_{M \in \mathcal{M}} E_{\Pr}[u'_M]. \tag{1}$$

That is, on the left-hand side, we compute the expected expected utility by summing $\Pr(s, t) \sup_{M \in \mathcal{M}} E_{\Pr_{\mathbf{q}(s,t)}}[u'_M]$ over all pairs $(s, t) \in S' \times T$. Effectively, this means that the DM chooses the best TM for each cell, after being informed what the cell is. We discuss this issue in more detail in Section 3.3.

Using this formalism, we can consider the value of learning that a particular TM $M$ is a "good" algorithm for the problem at hand (i.e., either learning that it always gives the correct answer, or always runs quickly), since this is just an event, just like learning

the value of some random variable $X$ is an event in a standard decision problem. In a computational decision problem, the DM has a prior probability on $M$ being good, and can compute the expected increase in utility resulting from learning that $M$ is good.

**Example 3.2** Consider the primality-testing problem from Example 2.1, viewed as a computational decision problem $(S, T, A, \Pr, \mathcal{M}, \mathcal{C}, \mathcal{O}, u')$. Given the utility function, for simplicity, we restrict $\mathcal{M}$ to to be a finite set of TMs that all halt within $2^{20}$ steps. Thus, the DM is certain of the complexity of all TMs in $\mathcal{M}$, and it is 0. On the other hand, the DM can still be uncertain about the output of a TM, and of the "goodness" of the output. For example, if $M$ is a TM that halts after one step and outputs 0, the DM may be certain that $M$'s output is 0, but be uncertain as to the "goodness" of its output. Of course, such an algorithm might still be worth using: if the agent places a high prior probability on the input not being prime (which would be the case if the input was chosen uniformly at random among all numbers less than $2^{40}$), then the expected utility of answering 0 for all inputs is quite high. A yet better algorithm would be to use some naive test for primality, run it for $2^{20}$ steps, and return 0 unless the algorithm says that the number is prime. The DM can then ask what the value is of learning whether a specific TM $M$ is good (i.e., returns the correct answer for all inputs). This depends on the DM's prior probability that $M$ is good; but if it is low, then the value of information is also low. Finally, we can ask the value of being told a good algorithm (assume that the DM is certain that there is a good algorithm, which always returns the right answer in less than $2^{20}$ steps, but doesn't know which it is). This amounts to learning the value of a random variable $X$ whose range is a subset of $\mathcal{M}$, where $X = M$ only if $M$ is a good TM. Clearly, after learning this information, the DM's expected expected utility will be 10 (no matter what he learns, his expected utility will be 10). The value of this information depends on the expected utility of the DM's best current algorithm. Note that if the DM believes that the input is chosen uniformly at random, then the expected utility of even the simple algorithm that returns 0 no matter what is close to 10. On the other hand, if the DM believes that the input is chosen so that primes and non-primes are equally likely, the best algorithm is unlikely to have expected utility much higher than 1 (the best strategy is likely to involve testing whether the number is prime, outputting the answer if the tests reveal whether the number is prime within $2^{20}$ steps, and outputting 2 otherwise). In this case, the value of this information would be close to 9. ∎

**Example 3.3** Consider the number-in-the-safe example, viewed as a computational decision problem $\mathcal{D} = (S, T, A, \Pr, \mathcal{M}, \mathcal{C}, \mathcal{O}, u')$. Recall that the state space $S$ has the form $(s_1, s_2, s_3)$, where $s_1$ is the number in the safe, $s_2$ is the combination of the safe, and $s_3$ models the DM's uncertainty regarding the output of TMs and their running time. There is only a single type, so we can take $T = \{t_0\}$. We have the obvious uniform probability on the first two components of $S$. Again, we restrict $\mathcal{M}$ to algorithms that halt within $2^{20}$ steps. If it takes one time unit to test a particular combination, and the DM believes that

the best approach is to generate some sequence of $2^{20}$ combinations and test them, then it is clear that the DM believes that the expected utility of this approach is $2^{-20}(1,000,000)$. Learning the first 20 digits makes the problem feasible, and thus results in an expected expected utility of $1,000,000$ (no matter which 20 digits are the right ones, the expected utility is $1,000,000$), and so has a high value of information. ∎

## 3.3 Value of conversation

Recall that, for value of information, we consider how much it is worth for a DM to find out which cell (in some partition of the state space $S$) the true state $s$ is in. In other words, we consider the question of how much it is worth for the DM to learn the value of $f(s)$ of some function $f$ on input the true state $s$. A more general setting considers how much it is worth for a DM to interact with another TM $I$ (for informant) that is running on input the true state $s$.

**Example 3.4** Suppose a number between 1 and 100 is chosen uniformly at random. If the DM guesses the number correctly, he receives a utility of 100; otherwise, he receives a utility of 0. Without any further information, the DM clearly cannot get more than 1 in expected utility. But if he can sequentially ask 7 yes/no questions, he can learn the number by using binary search (i.e., first asking if the number is greater than 50; if so, asking if it is greater than 75; etc.), getting a utility of 100. Thus, the value of a conversation with a machine that answers 7 yes/no questions is 99. ∎

The *value of conversation with (a TM) $I$* for standard decision problem can be formalized in exactly the same way as value of information. Formalizing computational value of conversation requires extending the notion of computational decision problems to allow the DM to choose among *interactive* Turing machines $M$ (this was already done in [Halpern and Pass 2010]). We omit the formal definition of an interactive Turing machine (see, for example, [Goldreich 2001]); roughly speaking, the machines use a special tape where the message to be sent is placed and another tape where a message to be received is written. We assume that the DM chooses a TM $M$. $M$ then proceeds in two phases. First there is a communication phase, where $M$ converses with the informant $I$; then, after the communication phase is over, $M$ chooses an action for the underlying decision problem. Note that what an interactive TM does (that is, the message it sends or the action it takes after the communication phase is over) can depend on its input, the history of messages received, and the random coins it tosses (if it randomizes).

When considering an interactive TM $M$, we assume that the complexity function $\mathcal{C}$ depends not only on the machine $M$ and its type $t$, but also on the messages that the DM receives, and its random coin tosses. More precisely, we define the *view* of an interactive machine $M$ to be a string $t; h; r$ in $\{0,1\}^*; \{0,1\}^*; \{0,1\}^*$, where $t$ is the part of the type actually read by $M$, $r$ is a finite bitstring representing the string of random bits actually

used, and $h$ is a finite sequence of messages received and read. If $v = t; h; r$, we take $M(v)$ to be the output of $M$ given the view. (Note that $M(v)$ is either a message or an action in the underlying decision problem, if the conversation phase is over.) We now consider output functions $\mathcal{O} : \mathbf{M} \times S \times \{0,1\}^* \to I\!N$, where $\mathbf{M}$ denotes a set of (interactive) Turing Machines, and let $\mathcal{O}(M, s, v)$ describe what the DM thinks the output of the machine $M$ is, given the view $v$, if the state of nature is $s$. Analogously, we now consider complexity functions $\mathcal{C} : \mathbf{M} \times S \times \{0,1\}^* \to I\!N$, and let $\mathcal{C}(M, s, v)$ describe the complexity of the machine $M$ given the view $v$ if the state of nature is $s$.

When running with $M$, $I$ gets as input the actual state $s$ (we want to allow for the possibility that $I$ has access to some featuers of the world that $M$ does not). That means that the state $s$ is playing a double role here; it is used both to capture the fact that $M$ is interacting (in part) with nature, and may get some feedback from nature, and to model the DM's uncertainty about the world. To formalize the computational value of conversation with $I$, let the random variable $\mathbf{view}^{I,M}(s, t, r_I, r_M)$ denote the view of the DM in state $s$ at the end of the communication phase when communicating with $I$ (running on input $s$ with random tape $r_I$) if the DM uses the machine $M$ (running on input $t$ with random tape $r_M$). We assume that $\mathbf{view}^{I,M}(s, t, r_I, r_M)$ is generated by computing the messages sent by $M$ and $I$ at each step using $\mathcal{O}$; that is, $M$'s first message is $\mathcal{O}(M, s, v_0)$, where $v_0$ is $M$'s initial view $t; \langle\,\rangle; (r_M)_0$ (here $\langle\,\rangle$ denotes the empty history, and $r'_M$ is a prefix of $r_M$, $M$'s sequence of random bits (however much randomness $M$ used to determine its first message), $\mathcal{O}(I, s, v_1)$, where $v_1 = s; \langle m_0 \rangle; r'_I, r'_W$ is a finite prefix of $r_W$, and $m_0$ is the first message sent by $M$, and so on. This means that $M$'s beliefs about the sequence of messages sent is determined by his beliefs about the individual messages sent in all circumstances.[4]

Let $\Pr^+$ denote the distribution on $S \times T \times (\{0,1\}^\infty)^2$ that is the product of $\Pr$ and the uniform distribution on pairs of random strings. For each pair $(I, M)$ of interactive TMs, we consider the random variable $u'_{I,M}$ defined on $S \times T \times (\{0,1\}^\infty)^2)$ by taking $u'_{I,M}(s, t, r_I, r_M) = u'(s, t, \mathcal{O}(M, s, v), \mathcal{C}(M, s, v))$, where $v = \mathbf{view}^{I,M}(s, t, r_I, r_M)$. That is, $u'_{I,M}(s, t, r_I, r_M)$ describes the utility of the actions that result when $M$ converses with $I$ in state $s$ given input $t$ and random tape $r_I$ for $I$ and $r_M$ for $M$, taking the complexity of the interaction into account. The expected utility of $M$ when communicating with $I$ is $E_{\Pr^+}[u'_{I,M}]$.

The computational value of conversation with $I$ is now defined as

$$\sup_{M \in \mathcal{M}} E_{\Pr^+}\left[u'_{I,M}\right] - \sup_{M \in \mathcal{M}} E_{\Pr^+}\left[u'_{\perp,M}\right], \tag{2}$$

where $\perp$ is the "silent" machine that sends no messages. That is, we compare the expected utility of best machine communicating with $I$ and the expected utility of the best machine that runs in isolation (i.e., is communicating with $\perp$).

[4]We can allow for $M$'s beliefs about the sequence of messages sent to be independent of his beliefs about individual messages, at the price of complicating the framework.

There is a subtlety in this definition that is worth emphasizing. In general, when defining determining the best choice of TM, we must ask whether it is reasonable to assume that the TM knows it's input. That is, is the choice of TM being made before the DM knows the input, or after? For example, in the primality-testing problem of Example 2.1, does the DM choose a TM before knowing what number is or after. The answer to this question has no impact if we do not take complexity into account, but it has a major impact if we do consider complexity. Clearly, if we know what the input $n$ is, we can choose a TM that is likely to give the right answer for $M$. There is clearly a very efficient TM that gives the right answer for a specific input $n$; it is the constant-time TM that just says "yes" if $n$ is prime, or the constant-time TM that just says "no" if $n$ is not prime. Of course, if there is uncertainty as to the quality of the TM, the DM may be uncertain as to what utility he gets with each choice. But the complexity is guaranteed to be low. On the other hand, if the choice of TM must be made before the TM knows the input, even if the DM understands the quality of the TM chosen, there may be no efficient TM that does well for all possible inputs.

Whether it is appropriate to assume that the TM is chosen before or after the DM knows the input depends on the application. For the most part, in [Halpern and Pass 2010], we implicitly assumed that the choice was made before the DM knew the input; this seemed reasonable for the applications of that paper. Here, in the definition of value of computational information, we implicitly assumed that the DM chose the best TM *after* learning the cell $q$ (but before learning the input $t$). We could also have computed the value of computational information under the assumption that the TM had to be chosen before discovering $q$. This would have amounted to putting the sup outside the scope of the $E_{\Pr}$ in Equation (1); this would have given

$$\sup_{M \in \mathcal{M}} E_{\Pr}\left[E_{\Pr_{\mathbf{q}}}\left[u'_M\right]\right] - \sup_{M \in \mathcal{M}} E_{\Pr}\left[u'_M\right]. \tag{3}$$

Here we are implicitly assuming that the TM $M$ chosen takes the cell $\mathbf{q}(s,t)$ as an input; moreover, the TM "understands" that the "right" thing to do with $\mathbf{q}(s,t)$ is to condition (and thus, to compute the expectation using $\Pr_{\mathbf{q}}$). Again, it is possible to allow more generality—the TM does not have to condition; the definition of computational value of of conversation implicitly allows this. While (3) is a perfectly sensible definition, it seems less appropriate when considering value of information, where a DM might be willing and able to devote a great deal of computation to a problem after getting information (although there may well be cases where (3) is indeed more appropriate than (1)).

By way of contrast, in (2), we are implicitly assuming that the DM must choose the interactive TM *before* learning the conversation; he does not get to choose a different one for each conversation. We are evaluating the value of conversation with $I$, rather than the value of a particular conversation with $I$. This is why we do not consider the expected expected utility of the best algorithm after receiving the information, but rather consider

the expected utility of "communicating, interpreting, and finally acting". Intuitively, we are assuming that a DM must choose a TM to interpret and make use of the information gleaned from the conversation; we want to take the cost of doing this interpretation into account, by choosing a TM that is able to interpret all possible computations.

We could in principle define a notion of value of *particular conversations* with $I$, rather than the value of conversing with $I$, by assuming that the DM chooses one TM that decides how to converse with $I$, and then, after the conversation, chooses the best TM to take advantage of that particular conversation. Thus, at the second step, the TM chosen would depend on the conversation. Formally, this amounts to having another sup inside the scope of $E_{\Pr^+}$, but this seems less appropriate here.

If we do not take the cost of computation into account, whether we learn the conversation before or after making the choice of TM is irrelevant. Indeed, the value of conversation can be viewed as a special case of value of information: for each "conversation-strategy" $\sigma$ for the DM, simply consider the value of receiving a transcript of the conversation between $I(s)$ and $\sigma(t)$ (where $t$ is the type of the DM). The value of conversation with $I$ is then simply the maximum value of information over all conversation strategies $\sigma$. By way of contrast, we cannot reduce computational value of conversation to value of information. If there is a computational cost associated with computing the messages to send to $I$, the value of a conversation is no longer just the maximum value of information.

**Example 3.5** Consider the guess-the-number decision problem from Example 3.4 again. What is the value of a conversation with an informant $I$ that picks two large primes $p$ and $q$, and sends the product $N = pq$ to the DM? If the DM manages to factor $N$, $I$ sends the DM the number chosen; otherwise $I$ simply aborts. Clearly, the value of information in the "best" conversation is 99 (the DM learns the number and gets a utility of 100). However, to implement this conversation requires the DM to factor large number. If computation is costly and factoring is hard (as is widely believed), it might not be worth it for the DM to attempt to factor the numbers. Thus, the value of conversation with $I$ would be 0 (or close to 0). ∎

## 3.4 Value of conversation and zero knowledge

The notion of a *zero-knowledge proof* [Goldwasser, Micali, and Rackoff 1989] is one of the central notions in cryptography. Intuitively, a zero-knowledge proof allows an agent (called the *prover*) to convince another agent (called the *verifier*) of the validity of some statement $x$, without revealing any additional information. For instance, using a zero-knowledge proof, a prover can convince a verifier that a number $N$ is the product of 2 primes, without actually revealing the primes. The zero-knowledge requirement is formalized using the so-called *simulation paradigm*. Roughly speaking, a proof $(P, V)$ (consisting of a strategy $P$ for the prover, and a strategy $V$ for the verifier) is said to be *perfect zero knowledge* if, for

every verifier strategy $\tilde{V}$, there exists a simulator $S$ that can reconstruct the verifier's view of the interaction with the prover with only a polynomial overhead in runtime.[5] Note that the simulator is running in isolation and, in particular, is not allowed to interact with the prover. Thus, intuitively, in a zero-knowledge proof, the verifier receives only messages from the prover that it could have efficiently generated on its own by running the simulator $S$. The notion of *precise zero-knowledge* [Micali and Pass 2006] aims at more precisely quantifying the knowledge gained by the verifier. Intuitively, a zero-knowledge proof of a statement $x$ has precision $p$ if any view that the verifier receives in time $t$ after talking to the prover can be reconstructed by the simulator (i.e., without the help of the prover) in time $p(|x|, t)$. (There is nothing special about time here; we can also consider precision with respect more general complexity measures.)

As we now show, there is a tight connection between the value of conversation for computational decision problems and zero knowledge. To explain the ideas, we first need to introduce a new notion, which should be of independent interest: *value of computational speedup*.

Computers get faster and faster. How much is it worth for a DM to get a faster computer? To formalize this, we say that a complexity function $\mathcal{C}'$ is *at most a $p$-speedup* of the complexity function $\mathcal{C}$ if, for all machines $M$, types $t$, and states $s$, $\mathcal{C}'(M, s, t) \leq \mathcal{C}(M, s, t) \leq p(\mathcal{C}'(M, s, t))$. Intuitively, if $p$ is a constant, the value of a $p$-computational speedup for a DM measures how much it is worth for the DM to change to a machine that runs $p$ times faster than his current machine. More precisely, the *value of a $p$-speedup* in a computational decision problem $\mathcal{D} = (S, T, A, \Pr, \mathcal{M}, \mathcal{C}, \mathcal{O}, u')$ is the difference between the maximum expected utility of the DM in $\mathcal{D}$ and the maximum expected utility in any decision problem $\mathcal{D}'$ that is identical to $D$ except that the complexity function in $\mathcal{D}'$ is $\mathcal{C}'$, where $\mathcal{C}'$ is at most a $p$-speedup of $\mathcal{C}$.

We now present the connection between zero-knowledge and value of conversation. Given a language $L$, an *objective* complexity function $\mathcal{C} : \mathbf{M} \times T \to I\!N$ (one that does not depend on the state of nature), and length parameter $n$, let $\mathcal{D}^{\mathcal{C}}_{L,n}$ denote the class of computational decision problems $\mathcal{D} = (S, T, A, \Pr, \mathcal{C}', \mathcal{O}, \mathcal{M}, u)$, where $\mathcal{M}$ is the set of interactive Turing machines, $S \subseteq \{0,1\}^n$, types in $T$ have the form $x; t'$, where $x \in S$ and $t' \in \{0,1\}^*$, and $\Pr$ is such that $\Pr(s, t) > 0$ only if $s = x$, $t = x; t'$, and $x \in L$ (so that the DM knows $x$ and that $x \in L$). We also require that (1) the DM does not have any uncertainty about the output and the complexity functions: for all $M, s, t$, $\mathcal{O}(M, s, t) = M(t)$ (so the DM knows the correct outputs of all machines) and $\mathcal{C}'(M, s, t) = \mathcal{C}(M, t)$ (so the DM knows the complexities of all machines); and (2) $\mathcal{D}$ is *monotone in complexity*: for all types $t \in T$, actions $a \in A$, and complexities $c \leq c'$, $u(t, a, c) \geq u(t, a, c')$; that is, the DM never prefers to compute more. We prove the following theorem in Appendix A.

[5] Technically, what is reconstructed is a distribution over views, since both the prover and the verifier may randomize.

**Theorem 3.6** *If $(P, V)$ is a zero-knowledge proof system for the language $L$ with precision $p(\cdot, \cdot)$ with respect to the complexity function $\mathcal{C}$, then for all $n \in N$ and all computational decision problem $\mathcal{D} \in \mathcal{D}^{\mathcal{C}}_{L,n}$, the value of conversation with $P$ in $\mathcal{D}$ is no higher than the value of a $p(n, \cdot)$-computational speedup in $\mathcal{D}$.*

Thus, intuitively, if the DM is not uncertain about the complexities and the outputs of machines, the value of participating in a zero-knowledge proof is never higher than the value of (appropriately) upgrading computers.

# 4 Discussion and Related Work

We have introduced a formal framework for decision making that explicitly takes into account the cost of computation. Doing so requires taking into account the uncertainty that a DM may have about the running time of an algorithm, and the quality of its output. The framework allows us to provide formal decision-theoretic solutions to well-known observations such as the status-quo bias and belief polarization.

Of course, we are far from the first to recognize that decision making requires computation—computation for knowledge acquisition and for inference. Nor are we the first to suggest that the costs for such computation should be explicitly reflected in the utility function. Horvitz [1987] credits Good [1952] for being the first to explicitly integrate the costs of computation into a framework of normative rationality. For example, Good points out that "less good methods may therefore sometimes be preferred" (for computational reasons). In a sequence of papers (see, for example, [Horvitz 1987; Horvitz 2001] and the references therein), Horvitz continues this theme, investigating various policies that trade off deliberation and action, taking into account computation costs. The framework presented here could be used to provide formal underpinnings to Horvitz's work.

In terms of next steps, we have considered only one-shot decision problems here. It would be very interesting to extend this framework to sequential decision problems. Moreover, we have assumed that agents can compute the probability of (or, at least, are willing to assign a probability to) events like "TM $M$ will halt in 10,000 steps" or "the output of TM $M$ solves the problem I am interested in on this input". Of course, calculating such probabilities itself involves computation. Similarly, calculating utilities may involve computation; although the utility was easy to compute in the simple examples we gave, this is certainly not the case in general. It would be relatively straightforward to extend our framework so that the TMs computed probabilities and utilities, as well as actions. However, once we do this, we need to think about what counts as an "optimal" decision if the DM does not have a probability and utility, or has a probability only on a coarse space. An alternative approach might be to allow the set of TMs that the DM considers possible to increase (at some computational cost), but assume that DM has all the relevant probabilistic

information about the TMs that it can choose among. As this discussion should make clear, there is much fascinating research to be done in this area.

# References


Agrawal, M., N. Keyal, and N. Saxena (2004). Primes is in P. *Annals of Mathematics 160*, 781–793.

Bacchus, F. and A. J. Grove (1995). Graphical models for preference and utility. In *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence (UAI '95)*, pp. 3–11.

Boutilier, C., R. I. Brafman, C. Domshlak, H. Hoos, and D. Poole (2004). Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of A.I. Research 21*, 135–191.

Goldreich, O. (2001). *Foundations of Cryptography, Vol. 1*. Cambridge University Press.

Goldwasser, S., S. Micali, and C. Rackoff (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on Computing 18*(1), 186–208.

Good, I. J. (1952). Rational decisions. *Journal of the Royal Statistical Society, Series B 14*, 107–114.

Halpern, J. Y. and R. Pass (2010). Game theory with costly computation. In *Proc. First Symposium on Innovations in Computer Science*.

Hintikka, J. (1975). Impossible possibble worlds vindicated. *Journal of Philosophical Logic 4*, 475–484.

Horvitz, E. (1987). Reasoning about beliefs and actions under computational resource constraints. In *Proc. Third Workshop on Uncertainty in Artificial Intelligence (UAI '87)*, pp. 429–444.

Horvitz, E. (2001). Principlies and applications of continual computing. *Artificial Intelligence 126*, 159–196.

Kahneman, D., P. Slovic, and A. Tversky (Eds.) (1982). *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge/New York: Cambridge University Press.

Micali, S. and R. Pass (2006). Local zero knowledge. In *Proc. 38th ACM Symposium on Theory of Computing*, pp. 306–315.

Mullainathan, S. (2002). A memory-based model of bounded rationality. *Quaterly Journal of Economics 117*(3), 735–774.

Pass, R. (2006). *A Precise Computational Approach to Knowledge*. Ph. D. thesis, MIT. Available at http://www.cs.cornell.edu/~rafael.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. San Francisco: Morgan Kaufmann.

Rabin, M. (1998). Psychology and economics. *Journal of Economic Literature XXXVI*, 11–46.

Rabin, M. and J. Schrag (1999). First impressions matter: A model of confirmatory bias. *Quaterly Journal of Economics 114*(1), 37–82.

Rabin, M. O. (1980). Probabilistic algorithm for testing primality. *Journal of Number Theory 12*, 128–138.

Rantala, V. (1982). Impossible worlds semantics and logical omniscience. *Acta Philosophica Fennica 35*, 18–24.

Rubinstein, A. (1986). Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory 39*, 83–96.

Samuelson, W. and R. Zeckhauser (1998). Status quo bias in decision making. *Journal of Risk and Uncertainty 1*, 7–59.

Savage, L. J. (1954). *Foundations of Statistics*. New York: Wiley.

Solovay, R. and V. Strassen (1977). A fast Monte Carlo test for primality. *SIAM Journal on Computing 6*(1), 84–85.

Wilson, A. (2002). Bounded memory and biases in information processing. Manuscript.

# A Precise Zero Knowledge

We recall the definition of *precise zero knowledge* [Micali and Pass 2006; Pass 2006], which is an extension of the notion of zero knowledge [Goldwasser, Micali, and Rackoff 1989]. We start by informally recalling the notion of an interactive proof; see [Goldwasser, Micali, and Rackoff 1989; Goldreich 2001] for more details. An *interactive proof* is a pair $(P, V)$ of protocols, where $P$ is used by the *prover* and $V$ is used by the *verifier*. The pair $(P, V)$ is required to satisfy two properties for every input $x$: *completeness*—if $x \in L$, then after running $(P, V)$ with input $x$, the verifier is convinced that $x \in L$ with probability 1, and *soundness*—if $x \notin L$, then the verifier running $V$ should reject with overwhelming probability, no matter what strategy $P'$ the prover uses.

Roughly speaking, zero-knowledge proofs are interactive proofs where the verifier does not learn anything beyond the fact that $x \in L$ from the interaction with the prover. This is formalized by requiring that for every strategy $V'$ of the verifier, there exists a *simulator* strategy $S$ that, given a string $x$ and whatever auxiliary information $z$ the verifier may have, can reconstruct the view of the verifier when running $V'$ on input $(x, z)$ with the prover running $P$ on input $x$. (Note that in the definition of zero knowledge, the verifier may have

some information $z$ as well as the input string $x$. The information $z$ is often called the *auxiliary input*, and models any prior knowledge the verifier has about $x$.)

The traditional notion of zero knowledge requires only that the *worst-case* complexity (size and running-time) of the simulator $S$ is polynomially related to that of $V'$. Precise zero knowledge additionally requires that the complexity of the simulator $S$ "respects" the complexity of the verifier's protocol $V'$ in an "execution-by-execution" fashion: a zero-knowledge proof is said to have precision $p(n,t)$ if the running time of the simulator $S$ (on input an instance $x \in \{0,1\}^n$) be bounded by $p(n,t)$ whenever $S$ outputs a view in which the running time of $V'$ is $t$. We generalize the definition of [Micali and Pass 2006], which considers only the running time as a complexity measure by allowing arbitrary (objective) complexity measures $\mathcal{C} : \mathbf{M} \times S \times T \to I\!N$. In our generalization, instead of bounding only the complexity of $S$, we require that the complexity of the combined machine $V'(S(\cdot)$ is bounded by $p(n,t)$ whenever $S$ outputs a view in which the running time of $V'$ is $t$. (Note that if we take the complexity of $V'$ to be its running time, then this change is inconsequential: running $V'$ on the output of $S$ adds only $t$ computational steps, since $t$ is the running time of $V'$ on the view output by $S$.)

**Definition A.1 (Perfect Precise $\mathcal{ZK}$)** *Let $L$ be a language, $(P,V)$ an interactive proof system for $L$, $\mathcal{C}$ a complexity function, and $p : N \times N \times N \to N$ a monotonically increasing function. We say that $(P,V)$ is* perfect $\mathcal{ZK}$ *with $\mathcal{C}$-*precision $p$ *if, for every interactive Turing machine $V'$, there exists a probabilistic algorithm $S$ such that the following two conditions hold:*

1. *For all $x \in L$ and $z \in \{0,1\}^*$, the probability that the verifier receives the view $v$ in an interaction between $P$ on input $x$ and $V'$ on input $(x,z)$ is identical to the probability that $S(x,z)$ outputs $v$.*
2. *For all $x \in L$ and $z \in \{0,1\}^*$, and all sufficiently long $r \in \{0,1\}^*$, $\mathcal{C}(V'(S),(x,z,r)) \leq p(|x|,\mathcal{C}(V',S(x,z,r))$. (Here $r$ is a random string, which is needed to determine the output of $S$, since $S$ may randomize.)*

## A.1 The proof

**Theorem A.2 (Theorem 3.6 restated)** *If $(P,V)$ is a zero-knowledge proof system for the language $L$ with precision $p(\cdot,\cdot)$ with respect to the complexity function $\mathcal{C}$ then, for all $n \in N$ and all computational decision problems $\mathcal{D} \in \mathcal{D}^{\mathcal{C}}_{L,n}$, the value of conversation with $P$ in $\mathcal{D}$ is no higher than the value of a $p(n,\cdot)$-computational speedup in $\mathcal{D}$.*

**Proof:** Recall that, given a language $L$, complexity function $\mathcal{C}$, and length parameter $n$, $\mathcal{D}^{\mathcal{C}}_{L,n}$ denotes the class of computational decision problems $\mathcal{D} = (S,T,A,\Pr,\mathcal{M},\mathcal{C}',\mathcal{O},u')$, where $\mathcal{M}$ is the set of interactive Turing machines, $S \subseteq \{0,1\}^n$, types in $T$ have the form

$x; t'$, where $x \in S$ and $t' \in \{0,1\}^*$, and $\Pr$ is such that $\Pr(s,t) > 0$ only if $s = x$, $t = x; t'$, and $x \in L$, for all $M, s, t$, $\mathcal{O}(M, s, t) = M(t)$, $\mathcal{C}'(M, s, t) = \mathcal{C}(M, t)$, and $u$ is monotone in complexity. Consider some language $L$, $n \in N$, decision problem $\mathcal{D} \in \mathcal{D}^{\mathcal{C}}_{L,n}$, and an interactive TM $M$. Assume $M$ gets expected utility $d$ in $\mathcal{D}$ after interacting with $P$. We show that there exists a decision problem $\mathcal{D}'$ and a machine $M'$ such that $\mathcal{D}'$ is a $p(n,\cdot)$-computational speedup of $\mathcal{D}$ and $M'$ has expected utility $d' \geq d$ in $\mathcal{D}'$. This implies that the value of conversation with $P$ in $\mathcal{D}$ is no higher than the value of a $p(n,\cdot)$-computational speedup in $\mathcal{D}$.

Let $S$ be the zero-knowledge simulator for $M$, and let $M'(v) = M(S(v))$ (i.e., $M'$ first runs $S$ and then runs $M$ on the output of $S$). Let $\tilde{\mathcal{C}}$ be defined identically to $\mathcal{C}$ except that for every $v$, $\tilde{\mathcal{C}}(M(S), v) = \mathcal{C}(M, S(v))$. Consider the decision problem $\tilde{D}$ that is identical to $\mathcal{D}$ except that the complexity function is $\tilde{\mathcal{C}}$. Since, for every input $(x, z)$, the output of $S(x, z)$ is identically distributed to the view of $M(x, z)$ in an interaction with $P(x)$, and since $\tilde{\mathcal{C}}(M(S), v) = \mathcal{C}(M, S(v))$, it follows that $U_{\tilde{D}}(M') = d$. Let $\mathcal{C}'$ be defined identically to $\mathcal{C}$ except that for every $v$, $\mathcal{C}(M(S)), v) = \mathcal{C}(M, S(v))$ if $\mathcal{C}(M, S(v)) \leq \mathcal{C}(M(S(\cdot)), v)$ and $\mathcal{C}(M(S), v)$ otherwise. Since utility is monotone in complexity, we have that

$$U_{\mathcal{D}'}(M') \geq U_{\tilde{\mathcal{D}}} = d,$$

which concludes the proof. ∎