# Chapter 2

# Theory of finite goal-oriented communication

This chapter describes a formal theory of semantic communication in terms of "goals" for communication. In this chapter, we focus exclusively on *finite goals,* where an agent wishes to reach some desired state of being, in contrast to *infinite goals* where an agent wishes to maintain some desirable state over time, which we will introduce in Chapter 6. We will describe our model for communicating agents and state our main definitions; in particular, we will introduce the "basic universal setting" for finite goals, in which we aim to design an agent for a finite goal that achieves the goal of communication with a partner in polynomial time whenever some other polynomial time agent could communicate with that partner. We then prove some basic results about this model—in particular, theorems characterizing when universal communication is possible with a given class of partners.

Subsequent chapters will depart from this basic model in a variety of ways. For example, in Chapters 4 and 7, we will depart from this setting by respectively employing a more restricted setting to obtain a more efficient protocol or weakening the reliability requirements to obtain more powerful protocols. On the other hand, in Chapter 5, we will show that the theorems introduced in this chapter hold for classes of users with differing computational resources

The theory and results developed in this chapter first appeared in an early form in a technical report [84] extending previously published work [83] with Madhu Sudan; the vastly cleaner version of the framework presented here is derived from a later technical report with Oded Goldreich and Madhu Sudan [70], and generally owes its present clarity and simplicity to Oded Goldreich.

B. Juba, *Universal Semantic Communication*,
DOI 10.1007/978-3-642-23297-8_2,

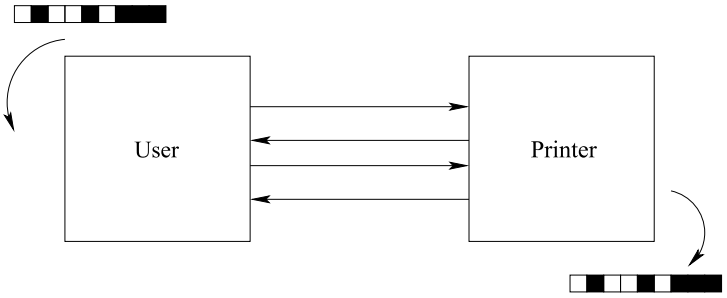


Figure 2.1: A user communicating with a printer

## 2.1 An informal overview of the theory

We begin by attempting to motivate the development of the basic notions in our theory from its foundations in modern philosophy. We will informally illustrate the considerations that lead to the specific choices we make in the formal development in Section 2.2, and we will give an overview of some of our main results – two from Section 2.3 and one from Chapter 4 – at the same intuitive, informal level.

### 2.1.1 Goals: a formal explication of meaning

Communication is a means to an end, and the "meaning" of a message is no more and no less than the conditions under which it is used. This view of meaning and language was proposed simultaneously by various authors in the 1920s, and was brought to prominence through Wittgenstein's "language games" [156, 157]. Following Wittgenstein, we can clarify what we mean by using a simple language game as an example; we have chosen one that, while simple enough to have stood among the original examples, we hope is also sufficiently familiar to impart our motivations for studying "semantic communication."

Consider the everyday case where we, situated at a PC, communicate with a printer (say, via a printer cable for simplicity). The players in this game are ourselves at the PC, which we will refer to together as the "user," and the "printer." At some point, the user has in mind a sequence of white and black dots that he or she would like printed on a page, and after our printer driver exchanges some signals with the printer – suppose it sends the message "010010111" across the cable – the printer produces a sheet of paper with our desired sequence of dots (see Figure 2.1 for a schematic). In our scenario, the message "010010111" has *meaning*: upon hearing it, the printer produces some particular sequence of white and black dots. The philosophical claim is that *there is no more to meaning in general*, nothing deeper happening in any act of communication.

More generally, the user's printer driver communicates successfully when the printer produces the same sequence of dots on its sheet of paper as we had envisioned (composed in our word processor, etc.). In particular, we could distinguish between cases in general where the printer produces the pattern we had in mind and cases where, as a result of using the wrong driver for example, the printer produces some other, garbage output. We take this distinction of cases where communication succeeds and cases where it fails as fundamental, and we say that the goal of communication is to enact successful communication. Thus, the goal of communication for the user is that the printer should produce the pattern of dots envisioned by the user.

From an engineering standpoint, this is well enough, since such goals are all we ever aim to facilitate with a particular implementation. Perhaps this talk about meaning even seems a little irrelevant—we recall that Shannon [135] explicitly disregarded everything occurring beyond the sender and receiver at the two ends of the communications channel. The philosophical claim is only relevant in that it asserts that by studying goals of communication in general, we cover *all possible* scenarios for communication, and we will never find occasion to consider any requirements more demanding than the correct usage of messages in our solutions to any communication problem.[1]

We will insist on baking these wider aspects of goals of communication in to our models since, unlike Shannon, we will attempt to confront the problems that arise when the sender and receiver have not agreed on a communication protocol in advance. For example, we might imagine that the cable serves as a binary channel, and that one printer, $P_0$ interprets a '0' as a white dot to be printed, and a '1' as a black dot, while another printer, $P_1$ interprets a '1' as a white dot and a '0' as a black dot. Such choices of encoding are arbitrary, after all, and ideally we would like for the user to be able to succeed at his or her goal regardless of whether the cable is attached to $P_0$ or $P_1$. One might hope that, like Shannon, we could dispense with talk about "meaning" and simply have the user figure out the printer's encoding.

Unfortunately, it turns out that without a protocol having been fixed *a priori*, the situation is more severe in some subtle ways, and this naïve approach is generally doomed to failure. Leaving the printers aside for a minute, suppose that Alice and Bob speak different natural languages and wish to have a discussion via some binary channel. We would expect that a third party who knows both languages could give finite encoding rules to Alice and Bob to facilitate this discussion, and we might be tempted to ask that a "universal protocol" translate Alice's statements into the same statements in Bob's language that the third party would have selected and vice-versa. In the absence of the third party, this is unreasonable to expect, though: Quine [121]

[1] Wittgenstein's claim was actually stronger than this, asserting that *human language* is described by no more and no less than a collection of similar but ultimately distinct language games. We won't need this stronger claim, since we only consider communications systems, in which case it is somewhat evident that the "language games" for different tasks may vary.

has suggested that mutually incompatible interpretations of a language may be entirely consistent with observation, and hence indistinguishable from "correct."

We can observe an example of Quine's "indeterminacy" in the setting where Alice and Bob are restricted to using the binary channel for communication: suppose that Alice and Bob were given encoding rules that were identical to those that a third party would have given them, except that some symmetric sets of words have been exchanged—say, Alice thinks "left" means "right," "clockwise" means "counter-clockwise," etc. (If we wished to be more formal, we could consider objects such as graphs with nontrivial automorphisms, as done in Example 2.20.) Unless they were given some additional means to tell that these basic concepts have been switched, observe that they would still have a conversation that is entirely sensible to each of them, and yet Bob would be interpreting Alice incorrectly. Thus, if we are to have any hope of constructing universal protocols at all, we must work with the broader model and be prepared to accept interactions that are indistinguishable from successes as "successes" as well.

Even the simple example of the two printers $P_0$ and $P_1$ immediately raises some important issues, since as we have set things up so far, this task may be *impossible*. We have in mind some dot we want printed first, perhaps a white dot, and if we send a '0' first, this is incorrect for $P_1$, whereas if we send a '1' first, this is incorrect for $P_0$. The obvious fix is that we should allow ourselves to print some "test pages" first, which amounts to relaxing our notion of "successful communication" in our language game. For simplicity, we will say that the user's goal is achieved if the desired pattern of dots appears as the suffix of the printer's output. A user who accomplishes this with both $P_0$ and $P_1$ is said to be universal with respect to the class $\{P_0, P_1\}$ and the relaxed goal of printing (henceforth $G_{\text{print}}$), or $\{P_0, P_1\}$-universal for $G_{\text{print}}$.

More broadly, we will only attempt to address goals of communication that are "forgiving" in the sense that for every state of the other players (e.g., the printers) it is still possible to achieve the goal, and we will likewise require our universal users to achieve the goal regardless of the initial states of the other players. Notice that this restriction to forgiving goals rules out some thornier scenarios that actually arise in the real world: for example, the printer might have some special string that allows us to "update" the firmware, which, if used incorrectly, could prevent the printer from either printing new pages or responding to new firmware updates. Ultimately, we would like to guarantee that this kind of catastrophic failure is extremely unlikely to happen, but addressing these issues is not our aim for the time being, and we will simply rule such issues out of bounds.

### 2.1.2 Sensing functions

From the definition of the relaxed goal of printing $G_{\text{print}}$, an approach to designing a $\{P_0, P_1\}$-universal user for $G_{\text{print}}$ springs immediately to mind: send a '0' to the printer; if a white dot is produced, we know we are connected to $P_0$, otherwise we know we are connected to $P_1$, and either way we can print accordingly. It seems almost too obvious to mention, but this is fine just so long as we are permitted to watch the dots leaving the printer.

If we cannot see the printer's output – perhaps it is in another room – then again, our goal is impossible: this time, supposing that the final dot should be white, again sending a '0' last is incorrect for $P_1$, and sending a '1' last is incorrect for $P_0$. Although we might be tempted to try to relax the goal further here – perhaps just require our desired pattern to appear as a substring – this snag only hints at a deeper problem.

Leaving the printer $P_1$ aside for a minute, we will describe a class of "password-protected printers," or more generally the password closure of a player which we will denote $\mathcal{PW}$, which will have further importance later: for a fixed player $P$, for each binary string (password) $\sigma$, $\mathcal{PW}(P)$ contains the player $P^\sigma$ that ignores messages unless they are prefixed by $\sigma$, and otherwise drops $\sigma$ and responds to the suffix as $P$ would. So for example, $\mathcal{PW}(P_0)$ contains the printer $P_0^{1101}$ that ignores messages that do not start with "1101" and upon receiving "1101010010111" prints the same pattern that $P_0$ would print upon receiving "010010111."

Without any feedback from the printer, a $\mathcal{PW}(P_0)$-universal user for $G_{\text{print}}$ (or even a further relaxed printing goal) can never stop trying—otherwise, the user will only have tried a finite number of passwords, whereas there are infinitely many distinct passwords, and thus certainly some corresponding printers which would have simply ignored every message sent by the user before he or she gave up. Needless to say, this is undesirable behavior either way: we would like to be able to succeed at our goal and move on to do something else.

We would be in good shape if only we could obtain a signal that our pattern had been printed successfully. More generally, what we are after is a binary-valued sensing function $V$ for our goal that the user can reasonably compute from his or her goal and history of messages (exchanged with the other players or their broader environment). By "reasonably compute," we mean that $V$ can be computed in probabilistic polynomial time (in terms of some reasonable parameter associated with the goal, such as the length of the pattern to be printed), and we will say that the sensing function is safe with a player $P$ if whenever it produces a '1' during an interaction with $P$, the user can "halt" (i.e., stop sending messages) and the goal will be achieved.

A safe sensing function specifies a *sufficient* condition for achieving the goal, but not a *necessary* one—as stated, it may be that the goal is achieved and we do not know about it. In particular, the function that always evaluates to '0' is quite safe, but also quite useless. Similarly, any safe sensing function

for our earlier notion of printing – i.e., one that only outputs '1' if the output of the printer is *precisely* the pattern we have in mind – is also safe for the weaker goal $G_{\text{print}}$, but as we noted, the strong goal is impossible to achieve universally with $\{P_0, P_1\}$.

The guarantee we are after turns out to be similar to the kind of "forgiving" conditions we demanded earlier when we relaxed the strong printing goal to $G_{\text{print}}$—we want a sensing function which, no matter the state of the other players, can still be made to produce a '1' by further interactions. In particular, we wish to constrain the kind of behavior required of us to behavior which we could be realistically expected to exhibit, that is to say, behavior that can be generated by probabilistic polynomial time computation. To be more precise, we now say that the sensing function $V$ is viable with some fixed player if there is some probabilistic polynomial time computable user behavior such that for any state of the other player, the interaction of the user behavior with the other player leads to a point where $V$ outputs '1' with high probability. The notion of a safe and viable sensing function leads us directly to our first theorem, which reassures us that we are on the right track:

**Theorem (Theorem 2.25, informal statement)** *For any class of players $\mathcal{P}\ell$ and goal $G$, there is a probabilistic polynomial time $\mathcal{P}\ell$-universal user for $G$ if and only if there is a sensing function for $G$ which is safe and viable with every member of $\mathcal{P}\ell$.*

The proof is simple: taking a probabilistic polynomial time $\mathcal{P}\ell$-universal user for $G$, we observe that the function that checks to see if the user would halt is a safe and viable sensing function. On the other hand, given such a sensing function, there is a standard trick (due to Levin [97]) by which we dovetail all (probabilistic) user behaviors, so that each runs with only a constant factor slowdown. We are guaranteed that $V$ will output a '1' by viability, and if we halt when this happens, safety guarantees that we have succeeded at the goal $G$. Moreover, because we only have a constant factor slowdown, viability guarantees that we run in probabilistic polynomial time when communicating with each fixed player in $\mathcal{P}\ell$. Thus, safe and viable sensing functions are precisely the functions that tell universal users when they should halt.

### 2.1.3 Capabilities and limits of universal users

Readers who are familiar with Levin's enumeration technique will recall that the "constant factor overhead" it incurs in the running time is actually exponential in the length of the corresponding program. For even moderately large programs, this "constant factor" may be enormous. By contrast, we expect that any reasonable design of a universal user for the two printers $\{P_0, P_1\}$ will not require this kind of overhead. We are left to wonder if

the enumeration technique is merely a crutch, whether or not the kind of overhead it exhibits is really essential in general.

For a sufficiently broad class of players, the overhead turns out to be essential. This is easiest to see by considering the password closure of some player; there are $2^\ell$ passwords of length $\ell$, so a user who sends fewer than $2^\ell$ messages cannot expect a response from every member of the password closure who uses a password of length $\ell$. Thus:

**Theorem (Theorem 4.3, informal statement)** *A $\mathcal{PW}(P)$-universal user for a goal $G$ that requires another player to act must send $\Omega(2^\ell)$ messages to $P^\sigma$ for some $\sigma$ of each length $\ell$.*

Since the passwords of length $\ell$ can be "hard coded" into programs at the cost of $\ell$ additional bits, this is the same qualitative relationship between program lengths and running times as shown by Levin's enumeration technique. So, the generic users constructed by this proof technique come at some substantial "cost" in their running time. We may well wonder what this high price buys us.

The price in running time turns out to buy us a remarkable kind of robustness, which we describe presently. Fix a goal of communication $G$, and consider the class $\mathcal{P}_G$ of $G$-helpful players $P$ for whom there is some corresponding probabilistic polynomial time user $U_P$ such that, no matter the state of the player $P$, $U_P$ succeeds at $G$ with $P$. A $\mathcal{P}_G$-universal user for a goal $G$ then *succeeds at $G$ whenever it is feasible for some user to reliably do so.* In particular, the class of $G$-helpful players encompasses players who we can think of as speaking in some foreign or alien language, so a universal user for this class would serve as a universal communicator. Of course, since passwords can be built in to user behaviors, $\mathcal{P}_G$ contains all of the password-protected versions of its players, and therefore we cannot expect such a powerful user to be too efficient.

The class of $G$-helpful players also contains wilder differences in behavior than mere password protection—since a sufficiently long password can map messages outside of any finite domain, *every possible* behavior on a finite set of finite histories is exhibited by some player in $\mathcal{P}_G$. This observation leads to the following important requirement on universal users' sensing functions:

**Theorem (Theorem 2.37, informal statement)** *If a sensing function is safe for every player in $\mathcal{P}_G$, then the sensing function must be safe for all players, even malicious and unhelpful ones.*

The proof of this theorem is somewhat subtle, and would demand more precision than we have been employing so far. We will need the definitions from Section 2.2, and the full proof will appear in Section 2.3. The main idea, though, is that when a malicious player is able to trick a user into halting prematurely, there is a finite subset of the histories in which the user is fooled that occur with an arbitrarily small loss in probability. Thus, the

representative of the finite malicious behavior in $\mathcal{P}_G$ would also trick the user into halting prematurely; turning this around, if the user doesn't halt prematurely with such players, then the user also must not halt prematurely with malicious players.

A corollary of this theorem (together with our first theorem) is that the existence of a $\mathcal{P}_G$-universal user for $G$ is equivalent to the existence of a sensing function for $G$ that is viable for $\mathcal{P}_G$ and always safe. So for example, the results described in Section 1.4.2 follow from our characterization by well-known facts and techniques—as we will demonstrate in Section 3.3. Thus, the characterization helps shed light on the capabilities and limits of "universal communication" in ways we will illustrate in Chapter 3.

## 2.2 Model of communication and goals

We begin by describing the basic model of communicating *agents* and their *goals* in a broader *environment*. We will distinguish between two kinds of agents in our language games: agents that represent ourselves, that we call *users*, and other agents that we interact with, who we call *servers*. We will then describe *universal* protocols that a user can employ to reliably achieve these goals, and minimal *helpfulness* conditions on servers which permit goals to be reliably attained with that server. In the basic "universal setting," we aim to construct a universal protocol that reliably succeeds under these minimal conditions, like the protocol described in our prior work [83]. Our model (if one removes the servers) and basic universal protocols respectively turn out to be very similar to the model of agents and "Asymptotic Bounded Optimal Agents" introduced by Russell and Subramanian [127], so our work could be thought of as a study of communication in a variant of their model, and we will take care to point out where the models diverge.

### 2.2.1 Agents: users, servers, and their environment

In familiar terminology from the Theory of Computation, we consider an interaction between several parties, (at least) the environment, and two agents, a user and a server, that proceeds in rounds. We will assert that, on each round, all parties are in one of countably many states,[2] and that the parties share communication channels, which likewise carry a finite sequence of symbols – elements of $\{0,1\}^*$ – and hence the overall system is in one of countably many states.[3] We will denote this countable state space of the system by $\Omega$. Thus, $\Omega$ is a product of the following sets: for each $i$th party

[2]This is without loss of generality—the environment's state could be a history of the entire interaction up to the present round.

[3]We note that the restriction that a channel only carry finitely many distinguishable symbols in one unit of time seems to be an unstated conclusion of Shannon's model. In particular even if a channel is capable of carrying an arbitrary real number in $[-1,1]$, but introduces Gaussian error, the capacity of the channel reduces to a finite amount.

in the interaction, there is an internal state space $\Omega^{(i)}$, and for each pair of parties $(i, j)$, there is a communications channel carrying a message from $i$ to $j$ with state space $\Omega^{(i,j)}$. We denote the index of the user by $\mathtt{u}$, the environment by $\mathtt{e}$, and the server (when there is precisely one) by $\mathtt{s}$.

The actions of all three parties will be dictated by a local function, the party's respective strategy:

**Definition 2.1** (Strategies)**.** The *strategy* of the $i$th party in a system is a function $P_i : \Omega^{(i)} \times \left(\bigtimes_{j \neq i} \Omega^{(j,i)}\right) \to \Omega^{(i)} \times \left(\bigtimes_{j \neq i} \Omega^{(i,j)}\right)$ representing the party's actions in the current round: $P_i$ takes as input the party's state and incoming messages at the beginning of the round, and computes the party's new state and outgoing messages for the following round.

Although the parties' strategies only operate on the specified "local" components of the global state, for convenience we may abuse notation by writing a strategy as a function mapping $\Omega$ to itself.

The communication channels corresponding to $\Omega^{(\mathtt{e},\mathtt{u})}$ and $\Omega^{(\mathtt{e},\mathtt{s})}$ model the agents' senses, whereas $\Omega^{(\mathtt{u},\mathtt{e})}$ and $\Omega^{(\mathtt{s},\mathtt{e})}$ model the agents' actions in the environment. This model naturally fits the same basic outline of agents described elsewhere (e.g., in the AI textbook of Russell and Norvig [126] and references described there). The channels joining the user and the server, on the other hand, are intended to model communication between the agents. Although less standard, this feature of the model is undoubtedly also employed elsewhere (e.g., in the study of multi-agent systems).

Likewise, in the following section, we will introduce a notion of *goals* for these agents, that resembles definitions appearing elsewhere—roughly, the agent's goal will be to have some effect on the environment. One difference in our work is that the goals, rather than the agents, come to play the central role in our study. In particular, a crucial difference between our setting and some settings considered elsewhere (e.g., the universal agents of Hutter [81]) is that we do not assume that "utilities" are computed for us by the environment—in our setting, in general it will be up to the agent to decide whether or not it is satisfied.

### 2.2.2 Goals of communication

Our objective is to study communication in general, particularly communication when the details of a protocol have not been fixed in advance. To determine when such robust communication is possible and particularly when it is impossible, we first need a formalization of communication problems in general, which we provide next, in terms of "goals." As described in Section 2.1, the goal of communication will specify a subset of states in which communication was used successfully.

We find it cleanest to formally describe a goal of communication in terms of the formal states of the environment. This is so because it will permit

us to freely vary the agents involved and the language they speak without worrying about how these variations affect the goal—in particular, such a formal model decouples the requirements on a user from its implementation, giving us the freedom to implement the same user strategy in a variety of ways.[4] Thus, goals will be formally represented by a *pair* of the environment's strategy (thus describing the semantics of its states) together with a subset of "successful" states which we define using a "referee" predicate:[5]

**Definition 2.2** (Finite referees and successful states)**.** A *finite referee* is a function from a state of the environment to a Boolean value, $R : \Omega^{(\mathtt{e})} \to \{0,1\}$. We say that $\{\sigma \in \Omega^{(\mathtt{e})} : R(\sigma) = 1\}$ is the set where communication is *successful*.

We specify that the referee is "finite" in contrast to the "infinite" goals that we will introduce in Chapter 6, where rather than aiming to reach a successful state, the user wishes to remain in a successful state during an infinite execution. When it is understood from context, we will drop the specification that the referee is "finite" or "infinite."

In particular now, and in contrast to the infinite goals we will consider later, we will only concern ourselves with settings where a user wishes to spend only a finite amount of time in pursuit of a goal. This certainly models the aims of common "client software" and other typical applications of PCs, such as the printing example described in Section 2.1, as well as the "subgoals" that an agent studied in AI might wish to pursue.[6] We model this by giving the user a distinguished "halting state" that signifies the end of the execution:

**Definition 2.3** (Halting)**.** The *halting state* $\sigma_F$ is a distinguished member of $\Omega^{(\mathtt{u})}$. When the user's strategy outputs $\sigma_F$, we say that the user *halts*.

As with standard Turing machines, we can think of entering the halting state in our model as "returning from a function call," and correspondingly, a desirable "post condition" for a user strategy is that the user halts in a successful state.

**Definition 2.4** (Achieving goals)**.** We say that the user *achieves* the goal specified by a referee $R$ and an environment with states $\Omega^{(\mathtt{e})}$ if the user halts and communication was successful: that is, when $\sigma^{(\mathtt{u})} = \sigma_F$ and $R(\sigma^{(\mathtt{e})}) = 1$.

[4] Russell and Subramanian [127] describe at length similar motivations for modeling their agents as abstract functions as they introduce their model.

[5] As such, our goals are directly analogous to the "task environments" of Russell and Subramanian [127], except that instead of their time-dependent real-valued utilities, we use a Boolean-valued referee. We prefer not to cast our model in terms of utilities since agents' limitations are more naturally stated in terms of goals.

[6] Although these goals are formally similar to the "episodic" task environments of Russell and Subramanian [127], we conceive of them differently—namely, Russell and Subramanian envision that each round corresponds to a constant number of steps of the execution and the episode is ended when the agent selects a single action, whereas we consider a longer execution in which each round corresponds to some polynomial number of steps and the agent may take many actions before electing to terminate the execution.

We reiterate that the goal is given by the pair of the environment's strategy and the referee. In a deterministic environment, there is only one set of reachable successful states, and hence, although there may be many ways of achieving the goal, in a sense there is only one aim for the user. Although most goals are not of this form, we can give at least one interesting example of such a goal.

**Example 2.5** (Turing test)**.** We recall Turing's famous test [149], which we can model in simplified form as follows. We consider two classes of "servers," computers and humans, which we denote as $\mathcal{S}_1$ and $\mathcal{S}_2$, containing agents that are assumed to report their respective class affiliations to the environment. We assume that in each round, the environment records the message "$i$" reported by the server as a member of $\mathcal{S}_i$, and the environment stores the last message sent from the user in its internal state (i.e., so $\sigma^{(\mathtt{e})}$ is a pair, $(\sigma^{(\mathtt{u},\mathtt{e})}, i)$). If the user's last message to the environment is "$i$" where $S \in \mathcal{S}_i$, then $R(\sigma^{(\mathtt{e})}) = 1$. Otherwise, $R(\sigma^{(\mathtt{e})}) = 0$.

To describe most interesting goals in our model, we need to consider richer kinds of environments, which we will introduce next.

### Non-deterministic and probabilistic environment strategies

It is evident that, in our model as described thus far, a deterministic environment cannot even capture the goal achieved by the *printer driver* in the story described in Section 2.1. The difficulty is that for the printer driver, success in printing the various "patterns the user has in mind" should correspond to disjoint sets of states of the environment, whereas with a deterministic environment, the referee predicate specifies only one such set. We can capture the goal of printing and much more by instead considering a *class* of environments, which we capture with non-deterministic strategies:

**Definition 2.6** (Non-deterministic strategies)**.** A *non-deterministic strategy* is a set of strategies; an *actual strategy* is a member of this set.

For simplicity, we will generally assume that the environment's strategy is formally non-deterministic (since this is almost always the case), preferring to consider even deterministic environments as being given by a singleton set of strategies. Operationally, we will think of the environment as non-deterministically (i.e., adversarially) choosing an actual strategy from its non-deterministic strategy. The significance of this from the user's perspective is that the user's strategy needs to achieve the goal with every actual strategy that the environment could choose from its set of strategies.

The various actual strategies of the environment are direct analogues of the "input" in classical computation, whereas the user's strategies are analogues of the various functions one might try to compute. To continue this analogy, and enable the quantitative study of the user's strategies, we associate a size parameter with the environment:

**Definition 2.7** (Size parameter)**.** For a given non-deterministic environment strategy $\mathcal{E}$, the *size parameter* is a function $n : \mathcal{E} \to \mathbb{N}$ taking the environment's actual strategy $E \in \mathcal{E}$ to its size $n = n(E)$.

The size parameter is used to determine, for example, the running time of a program implementing the user's strategy. It is the generalization of the "input lengths" from classical computational complexity theory to our model.

The basic use of non-deterministic strategies is best illustrated by demonstrating how it captures our printing goal (i.e., the goal achieved by a printer driver):

**Example 2.8** (Printing)**.** The goal of printing $G_{\text{print}}$ is given by a non-deterministic environment $\mathcal{E}$ such that for each string to be printed, $x \in \{0,1\}^*$, there corresponds an environment $E_x \in \mathcal{E}$ which on each round stores the last message sent by the server together with $x$ in its state, and forwards that message together with $x$ to the user. $R(\sigma^{(\mathtt{e})}) = 1$ if $x$ is the server's last message, and $R$ evaluates to 0 otherwise. The size parameter, $n(E_x)$, is the length of $x$.

The formalism of non-deterministic strategies can capture much more than simply specifying the "user's input." Although the environment's state space is countable, we expressly avoided specifying that the environment's actual strategy has a finite description; indeed, we permit this strategy to be given by an arbitrary function, and we permit the non-deterministic strategy to contain uncountably many actual strategies. In particular, if we define the environment's actual strategies so that the environment's state contains an index that is updated on each round, then since the environment never repeats states, this non-deterministic choice of actual strategy makes the infinitely many "non-deterministic choices" that may occur during execution. This use of non-determinism models changes to the environment by events that are independent of the interaction between the user and server; changes that only depend on the current state of the environment, and that are otherwise entirely arbitrary.

The reason we choose to formalize non-determinism as the environment making an "up-front" choice is that it will enable us to sanely introduce randomness to the model, by allowing the actual strategies to be randomized functions. Indeed, the advantage of this approach (as opposed to allowing the environment to make non-deterministic choices "on-line") is that it will allow us to talk sensibly about the probabilities of events concerning the execution of the actual strategy. We first formally specify the definition of probabilistic strategies:

**Definition 2.9** (Probabilistic strategies)**.** A *probabilistic strategy* of the $i$th party in a system is a random process $P_i$ representing the party's actions in the current round: for each state of the $i$th party and each sequence of incoming messages at the beginning of the round $P_i$ associates a distribution

over the $i$th party's states and outgoing messages for the following round. That is, for each element of $\Omega^{(i)} \times \left(\bigtimes_{j \neq i} \Omega^{(j,i)}\right)$, $P_i$ associates a distribution over $\Omega^{(i)} \times \left(\bigtimes_{j \neq i} \Omega^{(i,j)}\right)$.

As indicated by the definition, in general, we allow all parties to use probabilistic strategies; since probabilistic strategies can easily capture deterministic strategies by using a strategy $P_i$ for which each outcome in its distribution's support occurs with probability 1, we will in fact formally assume that *all* parties in the interaction use a probabilistic strategy as we proceed. Thus, our formal definition of a finite goal of communication is as follows:

**Definition 2.10** (Goals)**.** A *finite goal of communication* is given by a pair, consisting of the environment's non-deterministic probabilistic strategy and a finite referee.

We remind the reader that the referee determines its verdict by examining the state of the environment, where the meaning of these states (in the sense discussed in Section 2.1) is determined by the environment's strategy. In particular, since the environment can maintain the entire history of the interaction as reported by the agents in its state, we can set up a goal of communication in which the referee computes an arbitrary function of these interpretations of the interaction together with the rest of the environment's state.

Once the environment's non-deterministic choice of an actual probabilistic strategy is fixed, the resulting system under consideration is a (stationary) discrete time Markov process with countable state space $\Omega$, stopped when the user halts:[7]

**Definition 2.11** (Executions)**.** An *execution* of a system consisting of $m$ probabilistic strategies $P_1, \ldots, P_m$ is an $\Omega$-valued discrete time stochastic process $\{X_t\}_{t=1}^{\tau}$ such that $\tau \in \mathbb{N} \cup \{\infty\}$ is the random index of the first round in which the user halts and for each $i \in [m]$ and each $t < \tau$, $(X_{t+1}^{(i)}, (X_{t+1}^{(i,j)})_{j \neq i}) \leftarrow P_i(X_t^{(i)}, (X_t^{(j,i)})_{j \neq i})$ An execution *starting from global state* $\sigma_1 \in \Omega$ is defined similarly, except we fix $X_1$ to $\sigma_1$.

For a fixed system, we can thus speak freely of "the probability that the user achieves the goal" in an execution, since this is simply $\Pr[R(X_\tau^{(\mathsf{e})}) = 1]$ in familiar terminology.

### 2.2.3 Universal users

Now that we possess a formal model of a generic finite communication problem, we return to our original motivation, an investigation of the extent to

[7] The sequence of $\sigma$-algebras associated with each round is defined in the obvious way, and it is easy to verify that the "first time the user halts" is a stopping time due to our Markovian setup.

which communication can be made independent of the details of any particular protocol. To this end, we will begin to explicitly consider the role played by other entities in the interaction—entities which, in hopes of emphasizing the applications of our model to computer networks, we will refer to as "*servers*." Roughly, we wish to design user protocols that achieve a fixed goal with an arbitrary member of a large *class* of servers; such a user is said to be "universal" for the class of servers, and in this way, the user protocol is able to communicate effectively in spite of variations in the server protocols. Presently, we turn to refining this notion of a desirable universal user protocol.

Our first refinement is motivated by our observation in Section 2.1 that in order to have any hope of universality, we must restrict our attention to goals that are "forgiving" of an initial failure of the user protocol to communicate adequately with the server. Since we would like to describe such goals without reference to a particular user protocol, the cleanest and simplest definition of such a forgiving goal is stated as a guarantee that no matter what state the execution has entered, a good user protocol can still communicate successfully—so any prior failed attempts at communication are then "forgiven" by the referee. (We will refine this definition to refer only to states of the execution that can be *feasibly* reached in Chapter 4, which will substantially broaden the applicability of our notions.) Such a user protocol that communicates successfully from any state that the execution could have entered could surely be said to be "robust," motivating the following definition:

**Definition 2.12** (Robustly achieving goals)**.** A pair of user strategy $U$ and server strategy $S$ are said to *robustly achieve the goal* $G = (\mathcal{E}, R)$ *with probability* $p$ if for every $E \in \mathcal{E}$ and every state $\sigma$ in which the user strategy is started from its initial state, the probability that the user achieves the goal in the execution of $(E, U, S)$ started from $\sigma$ is at least $p$.

Naturally, a goal that is robustly achieved by some pair of user and server turns out to be forgiving of initial failures by the user to communicate:

**Proposition 2.13** (Robust achievement overcomes initial miscommunication)**.** *Let $U'$ be a user strategy that, except in an event of probability zero, begins executing according to user strategy $U_0$ from its initial state after some finite number of rounds. Let $S$ be any server and $G = (\mathcal{E}, R)$ be a goal. Then if $(U_0, S)$ robustly achieves the goal $G$ with probability $p$, then so does $(U', S)$.*

**Proof** Let $\tau_0 \in \mathbb{N} \cup \{\infty\}$ be the random index of the first round in which $U'$ begins executing $U_0$ from its initial state. Recalling the definition of robust achievement, we know that $U_0$ achieves the goal in the execution $(E, U_0, S)$ started from state $\sigma_{\tau_0}$ with probability $p$; that is, when $U_0$ halts, $R$ is satisfied with probability $p$. Since this execution is identical to the suffix of the execution $(E, U', S)$ starting from round $\tau_0$, when $U'$ halts, the

distribution over the states of $E$ is identical in both executions. Thus, $U'$ achieves the goal with probability $p$ as well. ■

With this refined notion of achieving goals in hand, we turn to providing a definition of a user protocol that operates with a large class of servers. Given that our attention is restricted to goals that may be robustly achieved, it is natural and desirable to ask for a user protocol that itself *robustly* achieves the goal; this way, we do without an assumption that we "know" that the environment (and server) start in some predetermined initial state. With this caveat in mind, the definition is natural.

**Definition 2.14** (Universal users)**.** We say that a user strategy $U$ is $(\mathcal{S}, p)$-*universal for a goal* $G$ if for every $S \in \mathcal{S}$, $(U, S)$ robustly achieves $G$ with probability $p$. If for every $S \in \mathcal{S}$ there is a negligible function $\epsilon : \mathbb{N} \to [0, 1]$ such that $U$ is $(\mathcal{S}, 1 - \epsilon(n(E)))$-universal for $G$, then we simply say that $U$ is $\mathcal{S}$*-universal for* $G$.

Although we are primarily interested in these robust universal users, we will find a few occasions where robust achievement is out of the question (or an open question) but we can show that a weaker kind of universal user exists, one that is sensitive to the initial state.

**Definition 2.15** (Weakly universal user)**.** Suppose that $\mathcal{S}^*$ is a set of pairs $(S, \sigma)$ where $S$ is a server strategy and $\sigma$ is a global state. We then say that a user strategy $U$ is *weakly* $(\mathcal{S}^*, p)$*-universal for a goal* $G = (\mathcal{E}, R)$ if for every $E \in \mathcal{E}$ and $(S, \sigma) \in \mathcal{S}^*$, $(U, S)$ achieves $G$ with probability $p$ in the execution $(E, U, S)$ started from global state $\sigma$.

### Time-bounded user strategies

We now further refine our notion of a desirable user protocol by focusing on the model of computation implementing the protocols. We motivate these considerations by observing that *in practice,* we would only be interested in protocols that are efficiently computable, which we naturally associate with "polynomial time computable." Of course, to make sense of this notion, we rely on a given parameterization of the *size* of the environment, as described in Definition 2.7. Thus:

**Definition 2.16** (Time-bounded user strategies)**.** Fix a universal interactive Turing machine $\phi$, and a non-deterministic environment strategy $\mathcal{E}$ with size parameter $n$. For a function $t : \mathbb{N} \to \mathbb{N}$, we say that a user strategy $U$ is *time-$t$ bounded* if there is a program on $\phi$ such that for every server strategy $S$ and every state of $E \in \mathcal{E}$ in the execution with $S$ and $E$, $\phi$ computes $U$ until it halts and runs for no more than $t(n)$ steps. When $U$ is time-$t$ bounded for a polynomial $t$, we say that $U$ is a *polynomial time protocol.*

From a *purely formal* perspective, there is nothing special about polynomial time, and thus we consider other classes of protocols explicitly in

Chapter 5, but surely the case of polynomial time protocols is the most intuitive and the one of greatest natural interest. Since fixing the class of algorithms will also lead to a simplification in the notation and presentation more generally, *henceforth we restrict our attention to polynomial time protocols.*

Once we have bounded the computational resources available to the user, it becomes interesting to consider the goal of solving computational problems (posed by the environment). Indeed, this is the goal considered in Section 1.4.2. We now show how this goal is formalized in the present terminology.

**Example 2.17** (Computation)**.** Fix a computational problem $\Pi$. The goal of solving $\Pi$, $G_\Pi$, is given by a non-deterministic environment $\mathcal{E}$ such that, for each instance $x$ of $\Pi$, there corresponds an environment $E_x \in \mathcal{E}$ which on each round sends $x$ to the user and stores the last message sent by the user together with $x$ in its state. $R(\sigma^{(\mathsf{e})}) = 1$ if the user's last message is $\Pi(x)$.[8] The size parameter $n(E_x)$, is simply the length of $x$.

For a problem $\Pi$ that can be solved in polynomial time, we remark that $G_\Pi$ can be achieved robustly without the server's assistance. This stands in contrast to the goal of printing from Example 2.8, which inherently depends on the server's assistance.

### 2.2.4 Helpful servers

We close this section with a description of an important class of server strategies, one that is maximal in a sense that we will describe shortly. Recall that our aim was to study protocols for communication when the details of a protocol have not been fixed in advance, and that we captured such protocols in Definition 2.14: these protocols robustly achieve a fixed goal with an arbitrary member of some (large) class of servers. It is natural to consider, in conjunction with such a definition, the *largest possible* class of servers. In this case, one constraint on such servers that would immediately follow from the existence of a universal user is that some user strategy exists that robustly achieves the goal. We call such servers, which permit *some* user strategy to robustly achieve the goal, "helpful" servers.

**Definition 2.18** (Helpful servers)**.** We say that a server strategy $S$ is *$p$-helpful* for a goal $G$ if there exists a polynomial time user protocol $U$ such that $(U, S)$ robustly achieves the goal with probability $p$. We denote the class of $p$-helpful servers for a goal $G$ by $\mathcal{S}_{G,p}$. If $S$ is $1 - \epsilon(n(E))$-helpful for a goal $G$ and some negligible function $\epsilon : \mathbb{N} \to [0, 1]$, then we simply say that $S$ is *helpful for* $G$, and we denote the class of such servers by $\mathcal{S}_G$.

[8] We take $\Pi$ to be a promise function problem, so not every string $x$ counts as an instance, but for those that do, we assume $\Pi(x)$ is a nonempty set. So, if no satisfactory pair $(x, y)$ exists, then we assume that some special string $\perp$ indicating this is in $\Pi(x)$.

In particular, a polynomial time $(\mathcal{S}_{G,p}, p)$-universal protocol for $G$ robustly achieves $G$ with probability $p$ with any server for which some (unknown) other polynomial time protocol robustly achieves $G$ with probability $p$. This is rather like the notion of (asymptotic) bounded optimality used by Russell and Subramanian [127]: the protocol achieves $G$ essentially as well as any other bounded protocol. We call the problem of achieving a goal with respect to the class of all helpful servers for the goal the basic "universal setting," and in Section 1.4.2, we asserted that a nontrivial goal (i.e., computation) can be achieved in the basic universal setting. In the present chapter, we will primarily examine generic goals in general settings. We will, however, give one result about the basic setting in this chapter and another one in Chapter 4, which motivate looking beyond it.

## 2.3 Sensing and universality

Not every goal can be reliably achieved. Consider the following goal:

**Example 2.19** (Guessing coins)**.** The environment's non-deterministic strategy is given by $\mathcal{E} = \{E_n\}$, where $E_n$ tosses $n$ fair coins and stores them, along with the user's last message in its state, and provides no other feedback. The referee declares success if the user's last message equals the coin tosses. Notice, since the coin tosses are independent of the user's strategy, no user strategy succeeds in $E_n$ with probability greater than $2^{-n}$.

Of course, goals such as the one described in Example 2.19 are immediately ruled out if we restrict our attention to goals with *helpful* servers. One might even hope that we could achieve any goal whenever a helpful server exists for that goal. Unfortunately, it turns out that the situation is more severe in some subtle ways for a user who wishes to communicate with the members of a broad class $\mathcal{S}$: goals which the user could hope to achieve "in principle" with the aid of a trusted third party (since the server is helpful) may turn out to be unattainable "in practice" due to the user misunderstanding the server.

We can observe an example of this by recalling our informal example when Alice and Bob speak different natural languages and are restricted to using the binary channel for communication. In this example, Alice spoke one of two languages that were identical except that the words for "left" and "right," "clockwise" and "counter-clockwise," etc. were switched. Since Alice and Bob are restricted to using the binary channel, unless they possess some prior shared knowledge of some physical objects or they are given some other means to fix an orientation, Bob has no way of distinguishing the two languages. In particular, if Bob's goal is to determine whether Alice has in mind "left" or "right," this is an example of a goal which cannot be achieved reliably with a collection of helpful Alices. For the sake of completeness, we also give an abstract, formal example that some readers may find more compelling:

**Example 2.20** (Identifying a vertex)**.** Fix a graph with a nontrivial automorphism $\pi$ and vertex set $V$. In the environment's non-deterministic strategy $\mathcal{E} = \{E_v\}_{v \in V}$, on each round, $E_v$ sends $v$ to the server, and stores $v$ with the user's last message in its state. The referee declares success if the user's last message is "$v$."

Suppose that the class of helpful servers contains $S_\iota$ and $S_\pi$, which are identical, except that $S_\pi$ applies $\pi$ to the environment's message (the corresponding user needs only to apply $\pi^{-1}$ to its last message). Since $S_\iota$ with $E_{\pi(v)}$ is indistinguishable from $S_\pi$ with $E_v$, regardless of whether the user sends $v$ or $\pi(v)$, the user is wrong with one pair, and hence this is an example of a goal which cannot be achieved reliably with a collection of helpful servers.

Example 2.20 is not so strange if we keep in mind that the channel connecting the server with the environment may model an entirely different kind of communications channel from the channel connecting the user and the server. For example, it may model some communication internal to the server or it may be the encoding of some kind of video input to the server, whereas the user-server channel may be across the internet. Since $\pi(v)$ is just as good a representation of the vertex $v$ as "the identity," there is no reason to expect that the server should use one or the other, and these kinds of issues fall entirely within our intended scope.

We claim that the difficulty in Examples 2.19 and 2.20 is that the environment and server provide the user with no means to tell whether or not communication is succeeding. In the remainder of this section, we introduce "sensing functions" that capture such "means to tell whether or not communication is succeeding," and prove that (at least for finite goals) the feedback from sensing is precisely what is required for the construction of universal protocols.

### 2.3.1 Sensing: safety and viability

Roughly, a sensing function for a goal is some locally computable function that allows the user to effectively guess the referee's verdict. Intuitively, a "locally computable function" is one that the user can compute from the "user's view" of the execution: a history of its states and messages exchanged with other parties in the execution. This is not quite sufficient for many nontrivial examples of sensing, though, because whenever sensing relies on a probabilistic test, we require that the user's messages were drawn from an appropriate distribution; the soundness of the test may not hold if the distribution is skewed. A simple, natural, and sufficient means to address this issue is to provide the sensing function with the outcomes of the (fair) coin tosses used during the execution of the user protocol. We will elect not to provide the sensing function with the user's states in the execution since these may be meaningless without a description of the user protocol (and when a

user protocol is fixed, the sensing function can just as well recover the states from the coins and messages). Thus, we have the following definition of the user's view.

**Definition 2.21** (User's view)**.** The *user's view* of an execution is given by the list of the user's incoming and outgoing messages and coin tosses on each round of the execution.

Now, a sensing function takes the user's view of the execution to a Boolean verdict, which we take as a prediction of the referee's verdict. A good sensing function should satisfy two complementary properties that we call "safety" and "viability." Roughly, a safe sensing function guesses conservatively—when it predicts that the referee is satisfied, then the referee is actually satisfied with high probability. A viable sensing function, on the other hand, is effectively nontrivial, in the sense that some feasible user strategy can lead the sensing function to guess that the referee is satisfied. Note that a sensing function must satisfy both safety and viability to be meaningful: a sensing function that predicts that the referee is never satisfied is trivially safe but unviable, whereas a sensing function that predicts that the referee is always satisfied is trivially viable but unsafe. We need both properties for the sensing function to be of any use.

**Definition 2.22** (Sensing, safety, and viability)**.** A *sensing function* is a Boolean function of the user's view of the execution.

- We say that the sensing function is *p-safe* for a goal $G = (\mathcal{E}, R)$ with a server $S$ if for any $t \in \mathbb{N}$, user strategy (as a probabilistic strategy in which the underlying sample space is given by coin tosses), and $E \in \mathcal{E}$, the probability that $R(X_t) = 1$ conditioned on the sensing function outputting 1 when run on the user's view up to round $t$ is at least $p$. If there exists some negligible function $\epsilon : \mathbb{N} \to [0, 1]$ such that the sensing function is $1 - \epsilon(n(E))$-safe for $G$ with $S$, then we say that the sensing function is *safe* for $G$ with $S$.

- We say that the sensing function is $(t, p)$-*viable* for a goal $G$ with a server $S$ if there exists a user strategy $U_S$ such that with probability at least $p$, when started from any state of the server and environment, $U_S$ runs for $t$ steps in expectation and the sensing function outputs 1 on the user's view of the execution $(E, U_S, S)$. If there exists some negligible function $\mu : \mathbb{N} \to [0, 1]$ such that the sensing function is $(t, 1-\mu(n(E)))$-viable for $G$ with $S$, then we say that the sensing function is $t$-viable for $G$ with $S$.

(When $G$ or $S$ is understood from context, we may drop the mention of them.)

To illustrate sensing, we describe a sensing function for our printing goal.

**Example 2.23** (Sensing for printing)**.** Recall the definition of our printing goal $G_{\text{print}}$ given in Example 2.8: the environment chooses a string $x$ that it sends to the user, along with the last message printed by the server on each round, and the referee is satisfied whenever these strings are the same. Thus, since both of these strings are present in the user's view, the function that checks that they are equal is 1-safe with *every* server strategy. Furthermore, if the server $S$ is $p$-helpful for $G_{\text{print}}$ with a user protocol running in time $t$, then this sensing function is $(t, p)$-viable for $G_{\text{print}}$ and $S$.

A nontrivial example of sensing is given by a sensing function for the goal of solving problems in PSPACE, considered in Section 1.4.2. It turns out that this construction, when combined with Theorem 2.35 (a strengthened variant of Theorem 2.25 below), immediately yields Theorem 1.4, the "positive" claim from Section 1.4.2.

**Example 2.24** (Sensing for computation)**.** Recall the goal of solving a computational problem $\Pi$, described in Example 2.17. A nontrivial safe sensing function computable in polynomial time can be constructed for $\Pi \in \text{PSPACE}$. Let $(P_\Pi, V_\Pi)$ be a public-coin interactive proof system with perfect completeness, soundness error $\epsilon$, and a binary prover (as essentially given by Shamir [103, 134]) for the set of pairs $(x, \Pi(x))$, which is clearly in PSPACE whenever $\Pi$ is. Then $V$ simulates $V_\Pi$ by providing it the user's coin tosses from each round,[9] and providing it a specially marked subsequence of the messages from the user to the environment as the messages from the prover; $V$ only outputs 1 if the user's last message is $y$ such that in the preceding rounds, $V_\Pi$ would accept on input $(x, y)$. Since $V_\Pi$ is a public-coin proof system with soundness error $\epsilon$, $V$ is $(1-\epsilon)$-safe for $G_\Pi$ and all server strategies.

If $\Pi$ is additionally PSPACE-complete then it turns out that $V$ is also $(1-\mu'_S, t_S)$-viable for $G_\Pi$ with any $1-\mu_S$-helpful server for some polynomial $t_S$ and negligible functions $\mu_S$ and $\mu'_S$. To see this, observe that given the polynomial time user $U_0$ such that $(U_0, S)$ achieves $G_\Pi$ with probability $1-\mu_S$, we can construct a user strategy $U^*$ that first uses $U_0$ with $S$ to obtain $\Pi(x)$, subsequently stores the entire history of coin tosses, and uses this history together with $(x, \Pi(x))$ to compute the polynomial time reduction from the optimal prover's next message to an instance of $\Pi$, which it solves by again using $U_0$ with $S$. Since $(P_\Pi, V_\Pi)$ has perfect completeness, the optimal prover's messages satisfy $V_\Pi$, and hence $V$. Since $U^*$ invokes $U_0$ at most a polynomial number of times and $\mu_S$ is negligible, the simulation satisfies $V$ with probability $1-\mu'_S$ for some other negligible function $\mu'_S$. Finally, since $U^*$ also runs in some polynomial time $t_S$ with $S$, $V$ is $(t_S, 1-\mu'_S)$-viable for $G$ with $S$ as claimed.

Both of these examples of sensing functions have the desirable property that they are viable with all servers that are helpful for their respective goals. We will see in the next section that this means that they yield universal users

[9] $V$ considers $V_\Pi$ to reject if insufficiently many coins are provided on any round.

for the basic universal setting, i.e., they achieve the goal in polynomial time whenever it is possible for some polynomial time user to do so.

### 2.3.2 Sensing is necessary and sufficient for finite goals

We now show that the feedback from a suitable sensing function is precisely what is required to successfully communicate with a broad class of servers. We prove the following theorem:

**Theorem 2.25** (Sensing is equivalent to universality)**.** *Let any goal $G$ and any class of servers $\mathcal{S}$ be given along with polynomials $t_S : \mathbb{N} \to \mathbb{N}$ for every $S \in \mathcal{S}$. Then there exists a sensing function that is safe and $O(t_S)$-viable for $G$ with every $S \in \mathcal{S}$ if and only if there is an expected $O(t_S)$-time bounded $\mathcal{S}$-universal user for $G$.*

More precisely, we prove Theorem 2.25 in two parts. We show first that given a universal user for a class of servers, we can construct a sensing function with essentially no loss in parameters, so in this sense, if it is possible to communicate with the class of servers, then sensing is (at least implicitly) available with that class of servers. We then show how to construct a universal user for the class of servers given a sensing function for that class of servers. Actually, the proof of Proposition 2.26 (and to a lesser extent, Proposition 2.27 below) is essentially immediate given the observation that *sensing functions are precisely the functions that tell a universal user when to halt.* Since this turns out to be a useful observation in its own right, we include it in the conclusion of the first proposition.

**Proposition 2.26** (Sensing is necessary for universality)**.** *Let $U$ be a universal user strategy for a goal $G$ and a class of servers $\mathcal{S}$ be given along with functions $t_S : \mathbb{N} \to \mathbb{N}$, and $\mu_S, \epsilon_S : \mathbb{N} \to [0,1]$ for every $S \in \mathcal{S}$ such that in the execution $(E, U, S)$ $U$ halts with probability $1 - \epsilon_S(n)$, and conditioned on it halting, runs in $t_S(n)$ steps in expectation and with probability $1 - \mu_S(n)$ achieves the goal. Then the function $V$ of the user's view that outputs 1 iff the user sent the same messages as $U$ on each round and on the final round $U$ would halt, is a sensing function that is $1 - \mu_S$-safe for $G$ with respect to every $S \in \mathcal{S}$, and $(t_S, 1 - \epsilon_S)$-viable for $G$ with respect to every $S \in \mathcal{S}$. Furthermore, $V$ is computable in time $t_S$ whenever $U$ runs in time $t_S$.*

**Proof** Given a universal user strategy $U$ for a goal $G$ and class of servers $\mathcal{S}$ that halts with probability $1 - \epsilon_S$ and conditioned on halting, achieves $G$ with probability $1 - \mu_S$, let $V$ be the function described in the statement of the proposition. Notice that $V$ is computable in time $t_S(n)$ given that $U$ runs in time $t_S(n)$.

Let any $E \in \mathcal{E}$ and $S \in \mathcal{S}$ be given, and consider the execution $(E, U, S)$. Observe that when $U$ halts, by construction $V$ outputs 1, which occurs with probability $1 - \epsilon_S(n)$ by assumption, and $U$ runs for $t_S(n)$ steps in expectation by assumption. Therefore, $V$ is $(t_S(n), 1 - \epsilon_S(n))$-viable for $G$ with respect

to every $S \in \mathcal{S}$. Similarly, consider any $E \in \mathcal{E}$, $S \in \mathcal{S}$, and user strategy $U^*$, and suppose that $V$ outputs 1 in some round $t$ of the execution $(E, U^*, S)$. If $(U^*, S)$ failed to achieve the goal in round $t$ with probability greater than $\mu_S$, notice that since $U$ would have produced the same interaction as $U^*$ on the corresponding views, in particular, there would exist a set of coin tosses and messages occurring with probability greater than $\mu_S$ on which $(U, S)$ would also have failed to achieve the goal. Therefore $(U^*, S)$ can only fail with probability $\mu_S$ when $V$ outputs 1, i.e., $V$ is $1-\mu_S$-safe for $G$ with every $S \in \mathcal{S}$. ■

**Proposition 2.27** (sensing is sufficient for universality)**.** *For a class of servers $\mathcal{S}$, let functions $t_S : \mathbb{N} \to \mathbb{N}$, $\epsilon_S : \mathbb{N} \to [0, 1/3]$, and $\mu_S : \mathbb{N} \to [0, 1]$ for each $S \in \mathcal{S}$, and a goal $G$ be given, and suppose there exists a sensing function $V$ for $G$ that is $1-\mu_S(n)$-safe with every $S \in \mathcal{S}$, $(t_S(n), 1-\epsilon_S(n))$-viable with every $S \in \mathcal{S}$, and expected $t_S(n)$-time computable with every $S \in \mathcal{S}$. Then there is an expected $O(t_S(n))$-time bounded $(\mathcal{S}, 1-O(t_S(n)\cdot\mu_S(n)))$-universal protocol for $G$.*

**Proof** Our argument borrows heavily from a classic result due to Levin [97].

**Construction.** Given a sensing function $V$, we enumerate protocols in stages, where in stage $i$ we enumerate protocols of length up to $i - 2\log i$: on each protocol of length $\ell$, we spend up to $t = \frac{2^i}{\ell^2 2^\ell}$ steps simulating the protocol until it halts, and then running $V$ on the resulting view. When $V$ outputs 1, we halt.

**Analysis.** When executing with a server $S \in \mathcal{S}$, since $V$ is given to be $(t_S(n), 1-\epsilon_S(n))$-viable for $G$ with $S$, there is some expected $t_S$-time bounded user strategy $U_S$ of length $\ell_S$ that, for any state of the execution, satisfies $V$ with probability $1 - \epsilon_S(n) \geq 2/3$. Since $V$ is furthermore assumed to be expected $t_S(n)$-time computable, when $t > 32t_S(n)$ (in stage $i^*$), we run $U_S$ until it halts in the first $16t_S(n)$ steps with probability at least $15/16$ and then see $V$ output 1 in the next $16t_S(n)$ steps with probability at least $\frac{2}{3} - \frac{1}{16}$. Thus, we run $U_S$ $r$ additional times with probability at most $\left(\frac{11}{24}\right)^r$.

Notice that if we run $r$ additional stages, we spend an additional

$$\sum_{i=i^*}^{i^*+r} \sum_{\ell \leq i - 2\log i} 2^\ell \frac{2^i}{\ell^2 2^\ell} \leq \sum_{\ell \leq \ell_S + r} \sum_{i=i^*}^{i^*+r} \frac{2^i}{\ell^2} \leq O(2^{i^*+r})$$

steps, which occurs with probability at most $\left(\frac{11}{24}\right)^r$. Thus, the overall expected running time is at most $O(\sum_{r=1}^{\infty} 2^{i^*} 2^r (11/24)^r) = O(2^{i^*})$ where $2^{i^*} = 2^{\ell_S + 2\log \ell_S + 1} t_S(n)$, so this is $O(t_S(n))$.

By the assumed safety of $V$, each time we run $V$, it outputs 1 when the referee is not satisfied with probability at most $\mu_S(n)$. By a union bound,

since we run $V$ at most $O(t_S(n))$ times in expectation, this occurs with probability at most $O(t_S(n)\mu_S(n))$. Since, from any state of the execution, we halt with probability 1 and never halt unless $V$ outputs 1, we only fail to achieve the goal when $V$ outputs 1 and the referee is unsatisfied. Thus, since $S \in \mathcal{S}$ and the starting state was arbitrary, our protocol robustly achieves the goal with every $S \in \mathcal{S}$ with probability $1 - O(t_S(n) \cdot \mu_S(n))$, as needed. ■

Thus, the search for universal user protocols for a finite goal can be cast as the search for safe and viable sensing functions for that goal. In many cases, the problem of constructing sensing functions is either simpler and more natural or can be addressed with existing techniques in a relatively straightforward way, as demonstrated by Examples 2.23 and 2.24, respectively. In the basic universal setting, we will see how this characterization can also be used to demonstrate limitations on universal users.

### 2.3.3 Extensions and variants of sensing: alternative constructions

In a sense, Theorem 2.25, on the equivalence of sensing functions and universal users in finite goals, is the central result of the present chapter. The notions of "sensing function" and "universal user" employed in that result, while natural enough and perhaps the simplest definitions, are surely not the *only* reasonable definitions one might state. In the present section, we will explore some main variants of these definitions that turn out to be useful in the construction of universal users in some situations. In particular, we will first consider some useful ways in which the kind of sensing functions required by Proposition 2.27 can be weakened while still obtaining the same conclusion—briefly, we can design sensing for a related goal featuring any number of "private outputs" and we can assume that the source code of the user protocol is given as input to the sensing function. We will then consider stronger variants of the definitions of sensing and universal users featuring "controllable safety," that is, where the safety of the sensing function and correspondingly, the probability that the universal user achieves the goal can be efficiently controlled to be bounded by some a priori known quantity, independent of the server. Such control over the error can be obtained in many natural situations, notably when the algorithm employed by the sensing function permits its success probability to be amplified by standard techniques, and this variant in particular will allow us to easily obtain the results claimed in Section 1.4.2.

#### Using some relaxed sensing functions

We first discuss some ways in which Proposition 2.27 can be strengthened, constructing a universal user from a relaxed notion of sensing that will be

convenient to use in Chapter 3 when we provide more examples of universal users for a variety of goals. Essentially, in Corollary 2.30 below, we will observe that the technique used to prove Proposition 2.27 is strong enough to obtain a universal user even when the sensing function requires "private outputs" that are taken to be ignored by the environment, and even when the sensing function depends on the code of the user strategy.

We begin by describing "private outputs" in more detail. As motivation, recall that, as we noticed in the construction of our sensing function for computation (in Example 2.24), it was useful for the user to send some messages to the environment that did not directly help achieve the goal. It turns out to be convenient in the design of sensing functions generally to further assume that the user's messages to the environment and the environment's states are structured as a tuple, in which some components of the user's current message are only stored in special components of the environment's state for a single round and then discarded. We will refer to these components as the user's *private outputs*, since the states of the execution are otherwise unaffected by these parts of the message. Formally:

**Definition 2.28** (Private outputs)**.** Suppose that the states of the environment and messages of the user are products of $k+1$ components, i.e., of the form $\Omega_0^{(\mathsf{e})} \times \Omega_1^{(\mathsf{e})} \times \cdots \times \Omega_k^{(\mathsf{e})}$ and $\Omega_0^{(\mathsf{u},\mathsf{e})} \times \Omega_1^{(\mathsf{u},\mathsf{e})} \times \cdots \times \Omega_k^{(\mathsf{u},\mathsf{e})}$ respectively, such that every actual strategy $E$ of the environment is given by $k+1$ independent functions $E_0, \ldots, E_k$ such that components $1, \ldots, k$ of $E$ only depend on the respective components of the user message, and component 0 is independent of these components of the user message. Then we say that components $1, \ldots, k$ of the user's message are *private outputs*.

In general, we allow the private outputs to affect the state of the environment for a single round, and thus influence the referee. In the design of sensing functions, however, it will be useful to assume the existence of additional private outputs that are not only discarded, but moreover also known to be ignored by the referee. We will show how to handle this assumption in Corollary 2.30, below.

Our second relaxation is to consider sensing functions that are assumed to have access to the program for the user strategy in addition to the user's view. We refer to such sensing functions as *grey-box* sensing functions (and, when we wish to make the distinction clear, we will refer to the sensing functions of Definition 2.22 as *black-box* sensing functions). Except for this additional input, the following definition is essentially identical to Definition 2.22.

**Definition 2.29** (Grey-box sensing)**.** Fix a reference universal interactive Turing machine $\phi$. A *grey-box sensing function* is a Boolean function of a program $U$ for $\phi$ and the user's view of an execution in which the user employs $U$ as its strategy.

- We say that the grey-box sensing function is *p-safe* for a goal $G = (\mathcal{E}, R)$ with a server $S$ if for any round $t \in \mathbb{N}$, program $U$ for $\phi$, and $E \in \mathcal{E}$,

whenever the sensing function outputs 1 in an execution with $S$ in round $t$, the referee also outputs 1 in round $t$ with probability at least $p$. If there exists some negligible function $\epsilon : \mathbb{N} \to [0,1]$ such that the grey-box sensing function is $1-\epsilon(n(E))$-safe for $G$ with $S$, then we say that the grey-box sensing function is *safe* for $G$ with $S$.

- We say that the grey-box sensing function is $(t,p)$-*viable* for a goal $G$ with a server $S$ if there exists a program $U_S$ for $\phi$ such that with probability at least $p$, when started from any state of the server and environment, $U_S$ runs for at most $t$ steps in expectation and the grey-box sensing function outputs 1 on $U_S$ and the user's view of the execution $(E, U_S, S)$. If there exists some negligible function $\mu : \mathbb{N} \to [0,1]$ such that the grey-box sensing function is $(t, 1-\mu(n(E)))$-viable for $G$ with $S$, then we say that the grey-box sensing function is $t$-viable for $G$ with $S$.

It is easy to see that a grey-box version of a black-box sensing function can be obtained by merely ignoring the code of the user strategy. Furthermore, grey-box sensing functions can equivalently be thought of as a (uniform) family of sensing functions, one for each user strategy.

We now turn to the statement and proof of Corollary 2.30.

**Corollary 2.30** (Grey-box sensing functions with private outputs are sufficient for universality)**.** *Let $G = (\mathcal{E}, R)$ be a finite goal, and let $G'$ be a version of $G$ with any number of additional private outputs that are ignored by the referee. For a class of servers $\mathcal{S}$, let functions $t_S : \mathbb{N} \to \mathbb{N}$, $\epsilon_S : \mathbb{N} \to [0, 1/3]$, and $\mu_S : \mathbb{N} \to [0,1]$ for each $S \in \mathcal{S}$ be given, and suppose there exists a grey-box sensing function $V$ for $G'$, that uses an efficient reference universal interactive Turing machine $\phi$, is $1-\mu_S(n)$-safe with every $S \in \mathcal{S}$, $(t_S(n), 1-\epsilon_S(n))$-viable with every $S \in \mathcal{S}$, and expected $t_S(n)$-time computable with every $S \in \mathcal{S}$. Then there is an expected $O(t_S(n))$-time bounded $(\mathcal{S}, 1-O(t_S(n)\cdot\mu_S(n)))$-universal protocol for $G$.*

**Proof** Consider the following modification of the construction used in the proof of Proposition 2.27:

Given a grey-box sensing function $V$, we enumerate protocols on $\phi$ in stages, where in stage $i$ we enumerate protocols of length up to $i - 2\log i$: on each protocol of length $\ell$, we spend up to $t = \frac{2^i}{\ell^2 2^\ell}$ steps simulating the protocol until it halts, only forwarding to the environment the components of the user's messages that exist in $G$. If the current protocol halts within $t$ steps, we then run $V$ on the current protocol and resulting simulated view, including the extra components of the user's messages in $G'$. When $V$ outputs 1, we halt.

Since the referee in $G'$ is assumed to ignore the components of the user's message not present in $G$, the referee in $G$ is satisfied on the projection of an execution $(E, U, S)$ of $G'$ iff the referee in $G'$ would also be satisfied with

the corresponding execution. Thus, the safety and viability of $V$ with $G'$ are sufficient for the analysis w.r.t. $G$, and the rest of the proof is identical to that of Proposition 2.27. ■

Thus, Corollary 2.30 places a handful of additional tools at our disposal for the design of universal users. We will see how these ease matters in Chapter 3. Of course, we stress that as a consequence of Proposition 2.26, strictly speaking we never *needed* either the extra power of grey-box sensing or additional private outputs for sensing—and in fact, these features actually provide no extra power. These features are merely a matter of convenience.

### Controlling the probability of errors

We now turn to consider stronger variants of both sensing and universal users that will turn out to be equivalent, yielding an incomparable variant of Theorem 2.25 that will be extremely useful when it is available to us. It will not always be available due to the following somewhat obvious yet counterintuitive point about goals (when viewed as a generalization of computational problems): *one cannot always amplify correctness.*

**Example 2.31** (Safety cannot always be amplified)**.** Consider the following goal: on each round, the environment flips a (private) fair coin to decide whether or not the referee is satisfied. If the referee is satisfied, the environment sends '1' to the user; otherwise, the environment sends '0' to the user with probability $1 - \epsilon$ and '1' with probability $\epsilon$. The function that reports this message from the environment is a $\frac{1}{1+\epsilon}$-safe sensing function for this goal (and $(O(1), 1)$-viable). It is not hard to see that no sensing function (and no universal user) can do better since the environment's choice of "false positives" versus "positives" is independent of the user.

Still, in many situations, particularly situations of natural interest, sensing is possible by means of probabilistic tests where the correctness of the test can be amplified by the usual means, e.g., repeating the test many times and taking a majority vote. We capture such sensing functions with the following definition.

**Definition 2.32** (Sensing with controllable safety)**.** We say that a sensing function has *controllable safety* for a goal $G$ with a server $S$ if it takes a rational number (in binary) as an additional input, and on input $\epsilon$ it is $(1-\epsilon)$-safe for $G$ with $S$. We will abuse notation and let $|\epsilon|$ denote the *length* (in binary) of this input.

When appropriate, we will describe the running time of the sensing function explicitly, but generally we will consider it "efficient" if it runs in time polynomial in the length of $\epsilon$ in bits, as well as the size parameter $n(E)$.

When such sensing functions are available, we can correct for one of the main deficiencies of the universal users constructed by Proposition 2.27, the

dependence on the (unknown) server in their success probabilities. In fact, the existence of such stronger sensing functions turns out to be (again) equivalent to the design of stronger universal user protocols, where the probability of failure can likewise be controlled to fall below any desired tolerance level, as is the case for probabilistic polynomial time algorithms. Before we prove this, we give a formal definition of such protocols, and a natural example of a class of algorithms captured by them.

**Definition 2.33** (Universal users with controllable error)**.** We say that a user strategy is $\mathcal{S}$*-universal with controllable error for a goal* $G$ if for every $S \in \mathcal{S}$ and rational $\epsilon > 0$, $U$ takes as auxiliary input $\epsilon$ given in binary, and $(U(\epsilon), S)$ robustly achieves $G$ with probability $1 - \epsilon$.

Again, we will explicitly comment on the running time of these universal users when appropriate, but we expect "efficiency" to mean a running time that is polynomial with respect to the length of $\epsilon$ in bits and the size parameter.

We briefly digress to show how we can capture Valiant's PAC-learning model [152] as a goal for communication in which the class of servers corresponds to the class of concepts. The usual $(\epsilon, \delta)$ definition of a PAC-learning algorithm will correspond to a user with controllable error for this goal, and we will furthermore note that in this setting, sensing with controllable safety is always available.

**Example 2.34** (PAC-learning)**.** We can define the goal of (improper, distribution-free) PAC-learning by a class of environments $\mathcal{E} = \{E_{D,n,\epsilon}\}$, parameterized by $n \in \mathbb{N}$, $\epsilon \in \mathbb{Q}$, and a distribution $D$ over $\{0,1\}^n$, and a referee that, in $E_{D,n,\epsilon}$ interprets the messages from the user and server as representations of circuits computing functions $C_U, C_S : \{0,1\}^n \to \{0,1\}$, and is satisfied iff $\Pr_{x \in D}[C_U(x) = C_S(x)] \geq 1 - \epsilon$. On each round, $E_{D,n,\epsilon}$ sends $x$ sampled from $D$ to both the user and the server, and sends $\epsilon$ to the user.

Naturally, a concept class $\mathcal{C}$ corresponds to a class of servers $\mathcal{S}(\mathcal{C})$ in the following way: for each function $C \in \mathcal{C}$, there is a server $S_C$ in $\mathcal{S}(\mathcal{C})$ that responds to a message $x \in \{0,1\}^n$ from the environment by sending $C(x)$ to the user, and sending a circuit computing $C$ on $\{0,1\}^n$ to the environment.

Thus, a user with controllable error who achieves this goal with every server in $\mathcal{S}(\mathcal{C})$ in $m + 1$ rounds outputs a circuit after receiving $m$ samples such that the circuit is $1 - \epsilon$ close to $C \in \mathcal{C}$ on inputs of length $n$ under the distribution $D$ with probability at least $1 - \delta$, for every $n$, $\epsilon$, and $D$ specified by the environment and every input to the user $\delta$, as needed for PAC-learning (with sample complexity $m$). In particular, if the user runs in time polynomial in $n$, $1/\epsilon$, and $\log 1/\delta$, then we capture precisely the usual notion of efficient PAC-learning.

Moreover, in this setting, we have a generic sensing function with controllable safety: $V(\delta)$ outputs 1 iff the circuit proposed by the user disagrees with at most $\epsilon m - \sqrt{\frac{m}{2} \log \frac{1}{\delta}}$ of the $m$ samples. Then, Hoeffding's inequality

tells us that if $V$ accepts, then with probability at least $1-\delta$, the proposed circuit agrees with $C$ with probability at least $1-\epsilon$ under $D$. It should be immediately clear that the sensing function is viable with any class of servers corresponding to a PAC-learnable concept class.

We are now ready to prove a variant of Theorem 2.25, showing an equivalence of sensing functions and universal users in this alternative, "controllable error" setting:

**Theorem 2.35** (Sensing functions with controllable safety are equivalent to universal users with controllable error)**.** *Let any goal $G$ and any class of servers $\mathcal{S}$ be given, along with a polynomial $t_S : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and function $\delta_S : \mathbb{N} \times \mathbb{Q} \to [0, 1/3]$ for every $S \in \mathcal{S}$. Then there exists a sensing function with controllable safety, running in expected time $\tilde{O}(t_S(n, |\epsilon|))$, and $(\tilde{O}(t_S(n, |\epsilon|)), 1 - \delta_S(n, \epsilon))$-viable for $G$ with every $S \in \mathcal{S}$ iff there is an expected $\tilde{O}(t_S(n, |\epsilon|))$-time bounded $\mathcal{S}$-universal user with controllable error for $G$. Furthermore, given a universal user with controllable error, the sensing function we obtain indicates whether or not the user would halt on a given view with input $\epsilon$.*

**Proof** The reverse direction is essentially immediate: for each fixed $\epsilon$, we have a $(\mathcal{S}, 1-\epsilon)$-universal user and we can apply Proposition 2.26 to find that the function described in the furthermore claim is a sensing function with controllable safety. Thus, all that remains to show is that we can obtain a universal user with controllable error from a sensing function with controllable safety. To do so, we will need to revisit the construction of a universal user from Proposition 2.27 in some detail.

**Construction.** We again enumerate protocols in stages $i = 1, 2, \ldots$, where in each $i$th stage we enumerate protocols of length up to $i - 2 \log i$. On each protocol of length $\ell$, we spend up to $t = \frac{2^i}{\ell^2 2^\ell}$ steps running the protocol until it halts and then (if the protocol halted) running our sensing function with safety parameter $\varepsilon = \epsilon 2^{-(i+2\ell+1)}$. If the sensing function accepts, then we halt, and otherwise we continue.

**Analysis.** Fix any $S \in \mathcal{S}$. Since the sensing function is $(\tilde{O}(t_S(n, |\varepsilon|)), 1 - \delta_S(n, |\varepsilon|))$-viable, some expected $\tilde{O}(t_S(n, |\varepsilon|))$-time bounded user strategy $U_S$ of length $\ell_S$ satisfies $V$ on any safety parameter $\varepsilon$ with probability $1-\delta_S \geq 2/3$ from any state of the execution. Likewise, $V$ is $\tilde{O}(t_S(n, |\varepsilon|))$-time computable, and so for some constants $C$, $k$, and $C'$, if

$$t = \frac{2^i}{\ell_S^2 2^{\ell_S}} > C t_S(n, |\varepsilon|) \log^k t_S(n, |\varepsilon|) \geq C' t_S(n, |\epsilon| + i) \log^k t_S(n, |\epsilon| + i)$$

then $U_S$ runs to completion and $V$ outputs 1 with probability at least 13/24. Observe that since $t_S$ is a polynomial, this inequality is satisfied for some

sufficiently large value of $i$, $i^*$. In particular, if $t_S$ has maximum degree $D$ in its second argument,

$$i^* \geq \log(C' t_S(n, |\epsilon|) \log^k t_S(n, |\epsilon|)) + (D+1) \log \log(C' t_S(n, |\epsilon|) \log^k t_S(n, |\epsilon|))$$

suffices, and when we run $r$ additional stages, we spend $O(2^{i^*+r})$ steps total, which occurs with probability at most $(11/24)^r$. Therefore, we find that the total expected running time of our universal user is at most

$$O\left(\sum_{r=1}^{\infty} 2^{i^*} 2^r (11/24)^r\right) = O(2^{i^*}).$$

Now, we see that using the minimum value for $i^*$, the expected running time is

$$O(t_S(n, |\epsilon|) \log^{D+k+1} t_S(n, |\epsilon|)) = \tilde{O}(t_S(n, |\epsilon|))$$

as claimed.

Now, note that in phase $i$, by our controlled safety guarantee, for each protocol of length $\ell$, we only risk halting without success with probability $\epsilon 2^{-(i+2\ell+1)}$. Thus, by a union bound, our total probability of halting without success is given by

$$\sum_{i=1}^{\infty} \sum_{\ell \leq i - 2\log i} 2^{\ell} \epsilon 2^{-(i+2\ell+1)} \leq \epsilon \sum_{i=1}^{\infty} 2^{-i} \sum_{\ell=1}^{\infty} 2^{-\ell} = \epsilon$$

as needed for a universal user with controllable error. ■

So, for example, since we noted in Example 2.34 that there is a sensing function that is safe and viable for every class of servers corresponding to a PAC-learnable concept class, the universal learning algorithm of Goldreich and Ron [72] can be obtained (up to logarithmic factors) as a corollary of Theorem 2.35.

Before we move on, we note that just as the argument used in Proposition 2.27 was strong enough to provide the same quality of universal user from a grey-box sensing function which required additional private outputs (Corollary 2.30), the argument in Theorem 2.35 also yields a universal user with controllable error from a grey-box sensing function which requires additional private outputs, so long as it also features controllable safety.

**Corollary 2.36** (Grey-box sensing functions with private outputs and controllable safety are sufficient for universal users with controllable error)**.** *Let $G$ be a finite goal and let $G'$ be a version of $G$ with any number of additional private outputs that are ignored by the referee. For a class of servers $\mathcal{S}$, let functions $t_S : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $\delta_S : \mathbb{N} \times \mathbb{Q} \to [0, 1/3]$ for each $S \in \mathcal{S}$ be given and suppose there exists a grey-box sensing function $V$ for $G'$ with controllable error, that uses an efficient reference universal interactive Turing machine $\phi$,*

*is* $(t_S(n,|\epsilon|), 1-\delta_S(n,\epsilon))$*-viable with every* $S \in \mathcal{S}$*, and expected* $t_S(n,|\epsilon|)$*-time computable with every* $S \in \mathcal{S}$*. Then there is an expected* $\tilde{O}(t_S(n,|\epsilon|))$*-time bounded* $\mathcal{S}$*-universal protocol for* $G$ *with controllable error.*

The proof is no more than a concatenation of the arguments of Corollary 2.30 and Theorem 2.35.

### 2.3.4 Safety requirements in the basic universal setting

We now present one of two limitation results that motivate considering classes of servers other than the class of all helpful servers for a goal (the other result is postponed until Chapter 4). We show that whenever a sensing function is safe with all servers that are helpful for a given goal (i.e., in the basic universal setting) then the sensing function is actually safe with *all* server strategies, even malicious and unhelpful ones. This explains why our only examples of sensing functions that are safe (and viable) for all helpful servers for a goal (e.g., given in Examples 2.23 and 2.24 so far; more examples are given in Chapter 3) are actually safe with all servers.

In particular, this result implies that the sensing functions we obtain from Proposition 2.26 for universal users in the basic universal setting are also actually safe with all servers. It is easiest to see why we consider this to be a limitation of the basic universal setting by considering computational goals: the result says that computational goals with universal users can only exist for problems with interactive proof systems. Thus, Theorem 1.5 from Section 1.4.2, stating that we can only hope to give universal users for computational goals corresponding to problems in PSPACE, follows from the special case of this result for computational goals. The present result says, furthermore, that any goal that can be achieved in the basic universal setting for finite executions has a cryptographic-strength test of its achievement. Therefore, when a goal permits no such test, we must either leave the basic universal setting and consider a smaller class of servers, or else we must switch to the infinite executions setting and find a means to achieve the goal without sensing (we prove an analogous theorem for sensing in infinite executions in Chapter 6).

**Theorem 2.37** (Safety with all helpful servers implies safety with all servers)**.** *Let* $G$ *be a goal and suppose that* $V$ *is a sensing function that is* $p$*-safe for* $G$ *with respect to every* $S \in \mathcal{S}_{G,p'}$ *for some nonempty* $\mathcal{S}_{G,p'}$*. Then* $V$ *is also* $p$*-safe for* $G$ *with respect to every server strategy.*

**Proof** Suppose that $V$ is not $p$-safe for $G$ with respect to some server strategy $S$. Then, for some $E \in \mathcal{E}$, some user strategy $U$, and some initial state $\sigma$, there are finite executions $(E,U,S)$ starting from $\sigma$ occurring with probability $p+\delta > p$ for which $V$ outputs 1 on the user's view, but the referee is not satisfied in the corresponding state of the execution; we will call such an execution a violation of safety.

The total probability of such a violation can be written by summing over triples consisting of round numbers, finite sequences of coin tosses, and lengths of the longest message sent by the user: a triple contributes to the sum the probability of a first violation in the execution occurring at the round number using precisely that sequence of coin tosses in which the user's longest message is the given length. Since every execution with a violation is witnessed by one such triple, the sum equals $p + \delta$; in particular, since this is a countable sum that may be written as a limit, there is some finite subset of these triples for which the sum is at least $p + \delta/2 > p$. Let $M$ be the maximum length of the longest user message over this set of triples.

Let $S' \in \mathcal{S}_{G,p'}$ be given, and consider the following server strategy $\tilde{s}$: $\tilde{s}$ has states corresponding to states of $S$ and states of $S'$. It behaves identically to $S$ until it receives a message from the user consisting of $0^{M+1}$, whereupon it enters some state of $S'$ and behaves identically to $S'$.

We first argue that $\tilde{s} \in \mathcal{S}_{G,p'}$. Given a polynomial time user protocol $U'$ such that $(U', S')$ robustly achieves $G$ with probability $p'$, let $\tilde{u}$ be the user strategy that first sends $0^{M+1}$ to the server, and then follows $U'$, and notice that $\tilde{u}$ is also a polynomial time user protocol. Let any state of the environment be given. Now, if $\tilde{s}$ starts in any state of $S$, $\tilde{u}$ sends $0^{M+1}$ in the first round, so $\tilde{s}$ enters a state of $S'$ and the execution $(E, \tilde{u}, \tilde{s})$ starting from the second round is identical to an execution of $(E, U', S')$. Thus, since $(U', S')$ robustly achieves $G$ with probability $p'$, $(\tilde{u}, \tilde{s})$ achieve $G$ in these executions with probability $p'$. If, on the other hand, $\tilde{s}$ starts in any state of $S'$, again, the execution $(E, \tilde{u}, \tilde{s})$ starting from the second round is identical to an execution of $(E, U', S')$, so $(\tilde{u}, \tilde{s})$ also achieves $G$ in these executions with probability $p'$, and we see $(\tilde{u}, \tilde{s})$ robustly achieves the goal with probability $p'$. Therefore, $\tilde{s} \in \mathcal{S}_{G,p'}$.

We claim that despite this, executions of $(E, U, \tilde{s})$ violate safety with probability at least $p + \delta/2 > p$. To see this, notice that in executions where $\tilde{s}$ starts in the state $\sigma^{(\mathbf{s})}$ of $S$, $(E, U, \tilde{s})$ is identical to $(E, U, S)$ started from $\sigma$ until $U$ sends a message of length greater than $M$. $M$ was chosen so that our finite set of triples describes an event of $(E, U, S)$ occurring with probability at least $p + \delta/2$ in which $U$ never sends a message of length greater than $M$ and a violation of safety occurs; the corresponding executions of $(E, U, \tilde{s})$ are also violations of safety since the user's view and the distribution over states of the environment are identical. Therefore, $V$ is not $p$-safe with $\tilde{s} \in \mathcal{S}_{G,p'}$. ■

We remark further that Theorem 2.37 also holds in the controlled error setting, since we can apply the theorem to the sensing function for each fixed value of the safety parameter $\epsilon$. Thus, we find analogously:

**Corollary 2.38** (Controllable safety with all helpful servers implies controllable safety with all servers)**.** *Let $G$ be a goal and suppose that $V$ is a sensing function with controlled safety for $G$ with respect to every $S \in \mathcal{S}_{G,p}$ for some*

*nonempty $\mathcal{S}_{G,p}$. Then $V$ is also a sensing function with controlled safety for $G$ with respect to every server strategy.*

http://www.springer.com/978-3-642-23296-1

Universal Semantic Communication
Juba, B.
2011, XX, 400 p., Hardcover
ISBN: 978-3-642-23296-1