The Advice Taker 2.0

Loizos Michael
Open University of Cyprus
loizos@ouc.ac.cy

Abstract

Starting from McCarthy's seminal proposal for the construction of an advice-taking machine, we show how tools from abstract argumentation and computational learning can help disambiguate and formalize that goal. Our focus is on establishing provable guarantees on the realizability of the advice taker, and on implementing an actual prototype system.

1 Introduction

However, certain elementary verbal reasoning processes so simple that they can be carried out by any non-feeble minded human have yet to be simulated by machine programs. — McCarthy, 1959

The construction of the *advice taker* as a means towards the endowment of machines with intelligent behavior was put forward by McCarthy [1959] both as a complement to the use of programming — the then-prevalent mode of human-machine interaction — and as an intermediate milestone to the eventual use of learning from experiences. Programming has certainly stuck around as a key way of interacting with machines, while the recent seemingly uncanny ability of learning-based machines to outperform human experts across tasks and domains might suggest that McCarthy's agenda is now dated.

This might have been the case had it not been for a new impetus on the viability of modern machine-learning techniques (such as deep learning) to produce human-like intelligence: their lack of operational transparency[1], and their brittleness — which, somewhat ironically, was the issue plaguing systems based on classical logic (like McCarthy's advice taker) that machine learning was supposed to address — as a result of their subtle dependency on biases in their training data.

We do not intend to debate whether the currently-trending machine-learning paradigm will eventually succeed in overcoming this impetus. We do contend, however, that McCarthy's proposal for the development of an advice taker still remains, today, relevant as *one* way of thinking about these issues, and perhaps making progress towards addressing them.

In this work we revisit McCarthy's [1959] proposal seeking to make three contributions: *(i)* to show that certain key, but admittedly under-specified, aspects of the original proposal can be made concrete through formal argumentation and learning theory, both of which have dealt with the problem of symbolic knowledge manipulation; *(ii)* to establish formal results on the realizability of constructing the advice taker; and *(iii)* to present an explicit example of such a construction.

[1]Randall Munroe's depiction: `https://xkcd.com/1838/`.

1.1 Overview

We introduce in Sections 2–4 a view of the advice taker as an online learning machine that seeks to approximate a target theory known to an advice giver (e.g., a human) interacting with the advice taker, in a manner that offers PAC-like guarantees [Valiant, 1984]. Both the target theory and the advice taker's hypothesis are formalized in Sections 6–7 through a novel structured argumentation framework. The framework accounts for the representation of arguments that entail not only inferences, but also actions that the advice taker is assumed to execute. Unlike work in reinforcement learning, however, the learning of these action-entailing arguments does not presuppose the execution of the actions and the receipt of a reward indicating their appropriateness in a given context. Instead, a supervised learning approach is used to learn all arguments, both inference-entailing and action-entailing ones.

Facilitated by the underlying structured argumentation framework, and unlike what a straightforward application of the PAC semantics would suggest: the advice taker does not only predict the label of an example — which would correspond to an extension of the advice taker's hypothesis — but also provides an explanation on how the prediction is reached; correspondingly, the advice giver does not return the correct label — which would correspond to an extension of the target theory — but instead offers advice, which generally corresponds to only a part of an argument, and is both smaller than the full correct label, and cognitively easier for a human to construct.

Our approach, thus, takes a step back from a pure learning-based view of the advice taker, and a step towards programming, by treating each piece of advice as a "code snippet" that the advice taker *integrates* into its current hypothesis. We quantify the necessary information content of these pieces of advice in Section 5, and show in Section 8 that PAC-like guarantees can indeed be established for a natural advice giver and a simple integration policy. An advice taker that adopts this particular integration policy is fully implemented in SWI-Prolog, while a visual web-based interface is under continual development, with the current version being accessible online from: `http://cognition.ouc.ac.cy/prudens/`.

2 The Advice Taker and the Advice Giver

[...] its behavior will be improvable merely by making statements to it [that] will require little if any knowledge of the program or the previous knowledge of the advice taker. — McCarthy, 1959

Unlike most contemporary work on Knowledge Representation and Reasoning, which tacitly assumes that knowledge is available *a priori*, we consider the involvement of an explicit *advice giver* in an online knowledge acquisition process.

We start with the supposition that the advice taker follows an elaborate-integrate cycle. Each cycle is initiated by an external or internal stimulus, such as an incoming query by a user, an event occurrence, the lapse of time, etc. The advice taker continues to sense the current context, which it uses as input to the processes of that cycle. By context we mean anything visible to the advice giver that might be relevant for the advice taker to consider during that cycle, including data that the advice taker may produce through its internal processes. As this need not be specified more concretely for the purposes of the current section, we shall simply assume, for now, that a ***context*** $x \in \mathcal{X}$ is a finitely-representable piece of data.

The advice taker proceeds to elaborate the information in context x by drawing inferences that follow from its current knowledge and x. Some of the elaborations may correspond to actions that the advice taker executes, including the action of simply reporting to a user some or all of the inferences that were drawn. Each elaboration is accompanied by an explanation on how it was reached, which is made available to the advice giver. With access to those explanations, the advice giver proceeds to offer *advice* to the advice taker, and the latter integrates that advice into its current knowledge, which is later used to elaborate contexts in subsequent cycles.

As is the case for context, a high-level abstract view of the data involved during elaboration and integration suffices for now, and a more concrete proposal will be made in the sequel. We shall assume, therefore, that the advice taker and the advice giver share a set $\mathcal{D}$ of finitely-representable ***derivations***, which can be thought to be determined by a previously agreed-upon syntax, such as a fragment of first-order logic.

During the elaboration step, the advice taker maps the current context x into a set of derivations, through an ***elaboration theory*** $t : \mathcal{X} \to 2^{\mathcal{D}}$ that it maintains and revises across cycles. The derivations in $t(x)$ determine the inferences and actions that follow from x and their corresponding explanations.

During the integration step, the advice giver computes and offers a finitely-representable ***advice*** $\alpha \in \mathcal{A}$ to the advice taker, by applying a ***feedback function*** $f : \mathcal{X} \times 2^{\mathcal{D}} \to \mathcal{A}$ on the current context x and the advice taker's elaboration $t(x)$. In turn, the advice taker integrates advice α into its current elaboration theory t to produce a revised version. Although we shall not specify, for the time being, how each piece of advice is integrated, we shall make provisions for $\mathcal{A}$ to include the symbol '$\varnothing$' indicating that the advice giver has "no advice" to give during that cycle, or, equivalently, that it finds the elaboration $t(x)$ of the advice taker ***acceptable***, meaning that the advice taker is not expected to revise its current elaboration theory. We shall further assume that for each context x, there exists at least one elaboration $e(x) \in 2^{\mathcal{D}}$ that is acceptable.

Necessarily, the interaction between the advice taker and the advice giver ends up endowing the former with subjective knowledge that reflects the beliefs and biases of the latter. We will, indeed, embrace this subjectivity of the knowledge for most of this paper, and we will briefly revisit at the end the question of how *commonsense* knowledge can be acquired by the advice taker through some form of autonomous learning.

3 How Should the Advice Taker Elaborate?

One will be able to assume that the advice taker will have available to it a fairly wide class of immediate logical consequences of anything it is told and its previous knowledge. — McCarthy, 1959

The advice taker elaborates a context by producing inferences and actions, supported by a set of derivations that are returned by its elaboration theory. Whether an elaboration is meaningful boils down to how the elaboration theory is revised after a piece of advice is received by the advice taker. We shall require, then, that the advice taker works towards ensuring that its elaboration theory *eventually* is such that $t(x)$ is acceptable to the advice giver (i.e., $f(x, t(x)) = \varnothing$) *sufficiently frequently*. To make this intuition concrete, we proceed to quantify the terms "frequently", "sufficiently", and"eventually".

We do not interpret "frequently" narrowly in terms of some fraction of contexts that needs to be elaborated in an acceptable fashion. Indeed, it might be the case that certain contexts occur rarely, and hence are practically inconsequential, while a small fraction of contexts might occur frequently enough so that their acceptable elaboration would be critical. To quantify this situation, we shall assume that contexts are drawn independently at random from some fixed, but unknown to the advice taker, probability distribution $\mathcal{P}$ over contexts in $\mathcal{X}$. We acknowledge that this independence assumption might not always hold in certain realistic scenarios, where contexts across different cycles of interaction might end up being correlated. Nonetheless, we adopt this simplifying assumption if only for the purposes of formal analysis of the interaction.

Given the above notion of frequency, we say that an elaboration theory $t : \mathcal{X} \to 2^{\mathcal{D}}$ is $(1 - \varepsilon)$***-approximately acceptable*** under $\mathcal{P}$ against an advice giver with feedback function f if $f(x, t(x)) = \varnothing$ for a context x drawn from $\mathcal{P}$, except with probability at most ε over the randomness of sampling from $\mathcal{P}$. Instead of fixing "sufficiently" to correspond to a particular value of ε, we shall let the advice taker receive an arbitrary value for $\varepsilon \in (0, 1]$ as a parameter when it is initialized, and we will expect the advice taker to eventually produce a $(1 - \varepsilon)$-approximately acceptable elaboration theory.

This leads to the question of how long "eventually" is; how long are we willing to allow the advice taker to be in a training mode where its elaborations are not sufficiently frequently accepted, before producing a desirable elaboration theory. We are not suggesting that the advice taker should ever stop accepting advice and integrating it in its elaboration theory; indeed, we view the elaborate-integrate cycle as a perpetual process. Rather, we are looking only to quantify the length of an initial period during which the advice taker's elaborations should still be under scrutiny by the advice giver, much like how the actions of a novice human assistant are closely monitored during an initial training period of their employment.

On the one hand, the advice taker should be given sufficient training time to compensate for the burden of its knowledge acquisition task, which grows with certain task parameters, such as: the representation size n of the drawn contexts, the representation size s of the advice giver's feedback function, the value of $1/\varepsilon$ (since the task becomes harder as ε becomes smaller). We shall assume, therefore, that the training time is some function $g(1/\varepsilon, n, s)$. On the other hand, allowing $g(1/\varepsilon, n, s)$ to grow arbitrarily (e.g., exponentially) in its parameters would entail a computationally overly-powerful advice taker, and would impose unrealistic demands on the advice giver. A theoretically-reasonable compromise is to let $g(1/\varepsilon, n, s)$ grow polynomially in its parameters. Pragmatically, the degree of the polynomial needs to be a small value.

Despite our effort to allocate a fair amount of resources to the advice taker, we have not accounted for the possibility of the advice taker simply being unlucky! Consider, for example, a probability distribution $\mathcal{P}$ that assigns probability $1/3$ to context x_1 and probability $2/3$ to context x_2. Suppose, further, that we set $\varepsilon := 1/2$, and expect the advice taker to produce a $1/2$-approximately acceptable elaboration theory t under $\mathcal{P}$ against some advice giver. It follows that $t(x_2)$ should eventually be acceptable by the advice giver. But what if the advice taker never gets the opportunity to interact with the advice giver in context x_2 to get advice on what might be acceptable in that context? Despite x_2 being the most probable context under $\mathcal{P}$, the probability of never drawing it is non-zero, irrespectively of any finitely-long training period.

Consequently, we slightly downgrade our expectations, so that the advice taker may fail to eventually produce a desirable elaboration theory, but only with a "sufficiently small" probability δ. As in our treatment of ε, we let the advice taker receive an arbitrary value for $\delta \in (0, 1]$ as a parameter when it is initialized, and we allow it to fail with at most probability δ. Correspondingly, we also allow the advice taker's training time to depend on (and grow polynomially in) $1/\delta$, so that the advice taker's training time is a function $g(1/\delta, 1/\varepsilon, n, s)$.

4 An Advice Taker with Learning Guarantees

Our ultimate objective is to make programs that learn from their experience as effectively as humans do. — McCarthy, 1959

In the preceding section we had quantified the type of guarantees that we expect the advice taker to offer on its operation. As we have already discussed, these guarantees are necessarily tied to the advice giver. In a sense, there is no objectively correct way for the advice taker to draw inferences and act, since, in particular, there is no objectively correct elaboration theory (or knowledge) that the advice taker can have. Instead, the inferences and actions of an elaboration theory (or knowledge) can, at best, conform to the advice giver's feedback.

Even with this shift from objective truth to subjective conformance, the requirements that we have imposed on the advice taker — to probably and tractably produce an approximately acceptable elaboration theory — might seem overly arduous. In this section we seek to bring these requirements together, and establish results on their realizability. Our formalization adopts and extends the celebrated *probably approximately correct* semantics for concept learning [Valiant, 1984], emphasizing tractable operation (through the restrictions on the training time g), provable guarantees on performance (through the parameters δ, ε that quantify the quality of the returned elaboration theory), and a crisp specification of the interaction between the advice taker and the advice giver.

Definition 4.1 (PAC Advice Takers)**.** *An advice taker is **probably approximately conformant (PAC)** to an advice giver with a feedback function in a class $\mathcal{F}$ (with fixed sets $\mathcal{X}$, $\mathcal{D}$, $\mathcal{A}$) if for every real values $\delta, \varepsilon \in (0, 1]$, every probability distribution $\mathcal{P}$ over contexts in $\mathcal{X}$ that have a finite representation size of at most n, and every feedback function $f \in \mathcal{F}$ with a finite representation size of s, the advice taker is given access to the parameters $\delta, \varepsilon, n, \mathcal{F}$, and proceeds as follows:*

a context x is drawn independently at random from $\mathcal{P}$; the advice taker uses its current (possibly stochastic) elaboration theory $t : \mathcal{X} \to 2^{\mathcal{D}}$ to return a set $t(x)$ of derivations; the advice giver offers the piece of advice $f(x, t(x))$; the advice taker integrates the advice to revise the elaboration theory t.

*After time at most $g(1/\delta, 1/\varepsilon, n, s)$ the advice taker terminates and returns, except with probability at most δ, an elaboration theory t that is $(1-\varepsilon)$-approximately acceptable under $\mathcal{P}$ against f. If the running-time function g grows polynomially in its parameters, then the advice taker is **efficient**.*

Definition 4.1 captures concisely the requirements on the interaction and on the elaboration theory of the advice taker up until the end of the training phase; as we have already discussed, the interaction and revision of the elaboration theory might continue after that. For the purposes of our subsequent formal analysis, we have allowed the elaboration theory to be a stochastic function, so that the advice taker can choose to produce random sets of derivations during the training phase, if this happens to be beneficial towards its goal of being PAC. In a later section we will offer an explicit construction of a PAC advice taker without appealing to this stochasticity.

Although the advice taker interacts with a single predetermined advice giver, the latter's feedback function is generally unknown to the former; had this not been the case, the advice taker could simply be assumed to have access to all the advice the advice giver could provide. On the flip side of things, the advice taker can, and does, have some information about certain aspects of the advice giver's feedback function (such as the set $\mathcal{A}$ of all pieces of advice it can possibly return), depending on the particular scenario in which the interaction takes place. This partial knowledge of the advice giver's feedback function f is captured in Definition 4.1 by assuming that f belongs in a class $\mathcal{F}$ that is known to the advice taker, while leaving open the possibility that f could be any member of $\mathcal{F}$, and insisting that the advice taker delivers in all cases.

With the aforementioned clarifications, we can now establish formal results on the realizability of a PAC advice taker.

Our first result shows that a (possibly non-efficient) PAC advice taker always exists, demonstrating, on the one hand, the non-vacuousness of Definition 4.1, and highlighting, on the other hand, the importance of insisting on efficiency.

Theorem 4.1 (Existence of a Universal PAC Advice Taker)**.** *For every class $\mathcal{F}$, there exists an advice taker that is PAC to an advice giver with any feedback function that belongs in $\mathcal{F}$.*

Proof (sketch). Memorize the first m contexts, and map each through t to the next set of derivations in $2^{\mathcal{D}}$ that have not yet been paired with it, until all m elaborations are acceptable. Do not execute any of the actions in the considered elaborations. The result follows for an appropriate choice of m. $\square$

The advice taker constructed in the proof of Theorem 4.1 does not really utilize any advice from the advice giver other than for the purpose of checking whether its elaboration of each given context is acceptable. This minimal reliance on the advice is compensated by the long running time of the advice taker, which effectively performs a blind systematic search of an appropriately selected subspace of all elaboration theories.

5 How Should the Advice Giver Advise?

In order for a program to be capable of learning something it must first be capable of being told it. In fact, in the early versions we shall concentrate entirely on this point [...] — McCarthy, 1959

In view of Theorem 4.1, a natural next step is to examine how the advice offered by the advice giver can be utilized more effectively by the advice taker towards constructing its elaboration theory. Since we are still approaching the interaction of the advice taker and the advice giver at an abstract level, we cannot yet discuss the structure of this advice, but we can, however, examine the amount of information that it conveys.

To quantify the amount of such information that might be necessary, we first introduce an auxiliary definition. Consider a partition u of $2^{\mathcal{D}}$ of size $|u|$, and denote by $u[i]$ its i-th element in some order. We shall say that u is ***shattered*** by a class $\mathcal{F}$ of feedback functions if there exists a context $x \in \mathcal{X}$ such that for every $i \in \{1, 2, \ldots, |u|\}$, there exists a feedback function $f \in \mathcal{F}$ for which $\{e \mid f(x, e) = \varnothing\} \subseteq u[i]$. The ***(information) sparsity*** $\texttt{spr}(\mathcal{F})$ of a class $\mathcal{F}$ is the maximum value $|u|$ for a chosen partition u of $2^{\mathcal{D}}$ that is shattered by $\mathcal{F}$.

Intuitively, the sparsity of a class $\mathcal{F}$ captures how little information is conveyed to the advice taker, in a worst-case context x, towards identifying what is acceptable to the advice giver, after the advice taker has successfully eliminated a feedback function in $\mathcal{F}$ from being the one actually used by the advice giver. Consequently, the higher the sparsity of $\mathcal{F}$, the less information the advice taker receives after each elimination, and the more feedback functions it needs to eliminate before gathering enough information to meet its goal.[2]

Theorem 5.1 (Necessary Feedback for Efficient PAC Advice Takers)**.** *If there exists an advice taker that is PAC to an advice giver with any feedback function* $f : \mathcal{X} \times 2^{\mathcal{D}} \to \mathcal{A}$ *in class* $\mathcal{F}$*, and runs in time* $g(\cdot)$*, then* $|\mathcal{A}|^{g(\cdot)}$ *is* $\Omega(\texttt{spr}(\mathcal{F}))$*.*

Proof (sketch). An advice taker receives at most $g(\cdot)$ pieces of advice and, therefore, at most $\log |\mathcal{A}| \cdot g(\cdot)$ bits of information, each eliminating at most half of the remaining parts of u from including the acceptable elaborations according to f. If $|\mathcal{A}|^{g(\cdot)} \leq (1/2) \cdot \texttt{spr}(\mathcal{F})$, then at least two parts of u are not eliminated, giving a t with an expected error of at least 1/2. The result follows for appropriate choices of δ and ε. $\square$

[2]This measure of the lack of information in a *worst-case example* differs from VC-dimension [Kearns and Vazirani, 1994], which aims to lower-bound the *number of distinct examples* needed to learn.

Depending, then, on the paucity of initial information that the advice taker has on which elaborations the advice giver might find acceptable (to the extent that such information follows from the advice taker's knowledge of $\mathcal{F}$), the constraint on the efficiency of the advice taker has an effect on the richness of the advice that the advice giver needs to offer.

6 Knowledge Representation and Reasoning

[...] probably have a single rule of inference which will combine substitution for variables with modus ponens. — McCarthy, 1959

Our emphasis up to this point was on formalizing and quantifying certain aspects of the interaction between the advice taker and the advice giver. Before giving an explicit construction of the two actors, we must make concrete the language in which the advice taker represents the knowledge that it reasons with and the advice giver represents the advice it offers.

Recall that the role of the advice taker's reasoning module is not to identify certain objective truths, nor to deduce what is logically entailed by a given context and the available knowledge. Instead, the role of reasoning is to produce elaborations that are acceptable by the advice giver. This subjectivity lessens the burden on the consideration of any particular type of reasoning semantics for the advice taker, since the interaction with the advice giver can finely-tune the produced elaborations by customizing the former's knowledge. This is decidedly demonstrated by Theorem 4.1, which establishes the existence of a PAC advice taker without reliance on the reasoning semantics. The primary role of the reasoning semantics, then, is to support the flexible revision of the knowledge. Our proposal below should, therefore, be read in this spirit, as a *reasonable candidate* for the representational syntax and the reasoning semantics, but by no means as the only option.

We consider a fixed language $\mathcal{L}$ that assigns finite names to the members of the two non-overlapping sets of ***constants*** and ***variables***, and defines the following constructs: An ***atom*** is a constant or a variable. A ***predicate*** is a constant followed by a finite parentheses-surrounded comma-separated list of atoms; these atoms are the predicate's ***arguments***, and their multitude is the predicate's ***arity***. A ***literal*** is a predicate or a predicate prefixed by the symbol '$\neg$'. A ***query*** is a predicate prefixed by the symbol '?'. An ***action*** is a predicate prefixed by the symbol '!'. A ***term*** is a literal, a query, or an action. Two variable-free terms are ***conflicting*** if they are the literals λ and $\neg\lambda$ for a predicate λ. Following Prolog convention, we capitalize the first letter of variables but not of constants.

A ***context*** in $\mathcal{X}$ is a collection of variable-free literals. A ***rule*** in $\mathcal{R}$ is an expression of the form $\varphi \rightsquigarrow \tau$, where the ***body*** φ of the rule is a finite comma-separated list of literals or queries, and the ***head*** τ of the rule is a literal or an action whose variables all appear in some of the terms of φ. An ***inferential*** rule is a rule with a literal head, while a ***production*** rule is a rule with an action head. An ***instance*** of a rule is an expression obtained from the rule after all variables in the rule's terms are replaced with constants, so that all appearances of the same variable across the rule map to the same constant. Two (inferential) rule instances are ***conflicting*** if their (variable-free) head terms are conflicting. Two (inferential) rules are ***conflicting*** if they have conflicting instances.

A ***knowledge base*** $\kappa = \langle \varrho, \succ \rangle$ comprises a finite collection $\varrho \subseteq \mathcal{R}$ of rules, and an irreflexive antisymmetric ***priority relation*** $\succ$ that is a subset of $\varrho \times \varrho$ and includes pairs of conflicting (inferential) rules only. For simplicity of exposition, we may occasionally assume that rules in ϱ are named, and define $\succ$ over their names. The priority relation is lifted to apply on rule instances also, in the natural manner. A knowledge base is ***totally-prioritized*** if for every pair of conflicting (inferential) rules $r_1, r_2 \in \varrho$ either $r_1 \succ r_2$ or $r_2 \succ r_1$ holds. For a totally-prioritized knowledge base, the priority relation need not be specified explicitly, as it can be determined by the order in which rules are represented in the knowledge base.

The advice taker reasons with such a knowledge base with the aim to elaborate a context. At a high level, reasoning proceeds by applying a forward chaining policy, starting with an initial set of variable-free predicates specified by the given context, and repeatedly applying modus ponens by interpreting rules as classical implications. Whenever a query $?\lambda$ is encountered in the body of a rule, this is treated as a call to an external oracle to decide if the predicate λ is true on its given arguments. If not all of the arguments of λ happen to be constant at the time of this call, then the call unifies, if possible, the variables of λ with constants such that the instantiated predicate is true. If multiple such instantiations are possible, this gives multiple instances of the rule, the evaluation of each of which proceeds independently. Whenever a (variable-free) action is encountered in the head of a rule, this is treated as an execution of a correspondingly-named external procedure with inputs determined by the action's (constant) arguments.

To make our reference to an external oracle and to external procedures concrete, we will stipulate the existence, alongside the knowledge base κ, of a Prolog program π, and shall, henceforth, write $\kappa(\pi)$ when we wish to make explicit the Prolog program π that is linked to the knowledge base κ. The encountering of a query $?\lambda$ or an action $!\lambda$ in a rule of κ corresponds to the posing of the associated predicate λ as a goal to π. In the case of an action, the Prolog call is made for its side-effects, not for determining whether it succeeds. This unified treatment allows one to include seamlessly arbitrary pieces of imperative code in an otherwise declarative knowledge base. Such imperative code allows the advice taker to gain access to the sensors and actuators of any device on which the advice taker is operating, and, in particular, to perceive signals beyond those that are available in a context. A special kind of these sensors, and we expect the most oft-used ones, are those that check for the axiomatic truth of statements involving, most typically, equality, ordering, or set membership.

Definition 6.1 (Derivation)**.** *A **derivation** for a term τ_0 in a context x under a knowledge base $\kappa(\pi)$ is a singly-rooted directed acyclic hypergraph such that: each vertex is a distinct variable-free term; each leaf term is either a literal that belongs in x or a query that is true according to π; the root term is τ_0; each directed hyperedge between term sets B and H is labeled by an instance $\varphi_i \rightsquigarrow \tau_i$ of a rule in κ such that $H = \{\tau_i\}$ and B comprises the terms in φ_i; no term appears in the head of more than one labeling rule instance. The **crown** rule instance of a derivation, if it exists, is the labeling rule instance with head τ_0, and is the only possible use of an instance of a production rule in the derivation.*

We conclude this section with an example knowledge base.

Example 6.1. *A phone assistant implementing a user's policy, with rules in increasing order of priority. Queries refer to built-in or easy-to-implement Prolog predicates, and actions refer to Prolog predicates that control phone functions.*

Policy: Decline calls when busy; e.g., when in a meeting at work. Calls from family members are important. Send an SMS when declining important calls. Repeated calls are urgent; unless coming from "serial" repeated callers. Repeated important calls are urgent. Answer urgent calls. Answer calls from colleagues that are expected.

my_pos(P1), work_pos(P2), ?dist(P1,P2,D), ?(D<50) ⇝ at_work

calendar(From,To), time(T), ?(From<T), ?(T<To) ⇝ in_meeting

in_meeting, at_work ⇝ busy

call_from(C) ⇝ will_answer(C)

call_from(C), busy ⇝ ¬will_answer(C)

will_answer(C) ⇝ !answer(C)

¬will_answer(C) ⇝ !decline(C)

contact(C,Info), ?member(in_group(family),Info) ⇝ important(C)

call_from(C), important(C), ¬will_answer(C) ⇝ !sms(C,'Busy...')

contact(C,Info), ?member(log(L),Info), ?last_entry(E,L), time(T), ?member(when(W),E), ?diff(T,W,D), ?(D<3) ⇝ recent_caller(C)

call_from(C), recent_caller(C) ⇝ urgent(C)

"... left as an exercise to the reader!" ⇝ serial_repeated_caller(C)

call_from(C), serial_repeated_caller(C) ⇝ ¬urgent(C)

call_from(C), recent_caller(C), important(C) ⇝ urgent(C)

call_from(C), urgent(C) ⇝ will_answer(C)

contact(C,Info), ?member(in_group(work),Info) ⇝ colleague(C)

call_from(C), colleague(C), is_expected(C) ⇝ will_answer(C)

The explicit treatment of time, in domains where this is beneficial, can proceed analogously to our work on story understanding [Diakidoy *et al.*, 2014; 2015], with extra knowledge used to address the specific problems associated with temporal reasoning (e.g., the persistence of observed information).

7 An Argumentative Elaboration Theory

In our opinion, a system which is to evolve intelligence of human order should [at least be such that:] Interesting changes in behavior must be expressible in a simple way. — McCarthy, 1959

The advice taker elaborates a context x by mapping it through some elaboration theory t to a set $t(x)$ of derivations. Definition 6.1 already accounts for several of the features we require for this elaboration process, including its dual role to specify drawn inferences (which come through crown inferential rule instances) and actions to be taken (which come through crown production rule instances). To complete our proposal on how a knowledge base gives rise to elaborations, it remains to specify *which* derivations should appear in $t(x)$.

Without loss of generality, we interpret any set of actions for which derivations exist in $t(x)$ as being intended to be executed together in an arbitrary order. Whenever a particular order is required, then a different action corresponding to an external procedure that orders things appropriately should be used. In effect, then, derivations can be incompatible only because of a conflict between their respective instances of inferential rules. To resolve which derivations appear together in $t(x)$, we appeal to abstract argumentation [Dung, 1995].

Priorities in a knowledge base indicate a preference on how conflicts between rule instance pairs are to be resolved, effectively stating that if the bodies of both rule instances are known to hold, then the evidence favors the head of the more preferred rule instance to be drawn as an inference. In the absence of a priority between two conflicting rule instances, then neither of their heads is to be drawn as an inference. This idea can be lifted to handle conflicts between derivations.

Definition 7.1 (Rebuttal)**.** *Consider two derivations, the first one for term τ_1 and the second one for some unspecified term, both in a context x under a knowledge base $\kappa(\pi)$ with $\kappa = \langle \varrho, \succ \rangle$. The first derivation **rebuts** the second derivation on rule instance r_2 if there exists a non-leaf term τ_2 in the second derivation pointed to by a hyperedge labeled by rule instance r_2, such that τ_2 is conflicting with τ_1, while term τ_1 is either:*

- *a leaf in the first derivation; the rebuttal is **exogenous**,*
- *the head of the first derivation's crown rule instance r_1, and it holds that $r_2 \not\succ r_1$; the rebuttal is **endogenous**.*

The derivation being rebutted is always attacked on a weak link: a rule instance r_2 in the derivation that either conflicts with a leaf of the attacking derivation, or is less preferred than the crown rule instance r_1 of the attacking derivation. The former type of rebuttal is equivalent to saying that r_2 is attacked because the negation of its head appears in context x. Indeed, the attacking derivation in the former type of rebuttal necessarily comprises only that single leaf, since this is the only way that the leaf can also be the derivation's root τ_1.

Each derivation naturally corresponds to an argument, and the rebuttal relation between derivations corresponds to an attacking relation between arguments. The argumentation framework that is ***induced*** in the stated manner by a knowledge base $\kappa(\pi)$ and a context x can be seen to adopt the following choices under the general ASPIC+ categorization [Prakken, 2010]: axiomatic premises (which correspond to the context x), defeasible rules, rebutting attacks, and application of rule preferences on the last link. We, thus, conclude the specification of the advice taker's reasoning semantics by fixing the set $t(x)$ of derivations that it uses to elaborate x to be the unique *grounded extension* of the induced argumentation framework.

The grounded extension-based semantics is chosen, in part, because of the tractability of computing the grounded extension. This tractability is known to hold only when it is measured against the size of the argumentation framework. The induced argumentation framework, in our case, can be exponentially larger than the size of the knowledge base (even for a "propositional" knowledge base). Fortuitously, we are able to prove the following, essentially optimal, tractability result.

Theorem 7.1 (Efficient Computation of the Grounded Extension)**.** *Consider a knowledge base $\kappa(\pi)$ and a context x. Let $||\kappa(\pi)||$ be the number of distinct crown rule instances in all derivations in x under $\kappa(\pi)$, and let $|x|$ be the size of x. Assume, further, that every call to π takes unit time. Then, the grounded extension of the argumentation framework induced by $\kappa(\pi)$ and x can be computed in time polynomial in $||\kappa(\pi)||$ and $|x|$. Furthermore, there exists no algorithm that can compute the grounded extension of the argumentation framework induced by $\kappa(\pi)$ and x in time sub-linear in $||\kappa(\pi)||$ and $|x|$.*

Proof (sketch). Starting from x, repeatedly apply modus ponens to construct a ***derivation network*** N that concisely represents all derivations in x under $\kappa(\pi)$ [S1]. Mark literals in x [S2] and repeat the following until convergence: remove literals that conflict with marked literals [S3]; remove hyperedges that are not labeled by the crown rule instance of a derivation in N [S4]; mark hyperedges that are labeled by a rule instance r_1 whose body literals are all currently marked, and for which any other hyperedge that is labeled by a conflicting rule instance r_2 is such that $r_1 \succ r_2$ [S5]; mark the head term of every hyperedge that is currently marked [S6]. Return the marked part of N [S7], which concisely represents the grounded extension. The Appendix shows graphically the execution of the process on an example knowledge base. □

The argumentation-based semantics gives the advice taker considerable flexibility in revising its elaboration theory. The addition or removal of rules or priorities, as might be suggested by the advice giver, can be done in a local and revision-tolerant[3] manner [McCarthy, 1998], while still having a significant impact on the operating behavior of the advice taker.

8 Explicit Construction of an Advice Taker

A machine is instructed mainly in the form of a sequence of imperative sentences; while a human [...] in declarative sentences describing the situation in which action is required together with a few imperatives that say what is wanted. — McCarthy, 1959

We conclude the development of our framework for PAC advice taking by providing explicit, but domain-independent, constructions of an advice taker and an advice giver.

The advice taker maintains during each interaction cycle i a knowledge base $\kappa_i(\pi)$, and elaborates a given context x using the elaboration theory t_i induced by $\kappa_i(\pi)$, as described in the preceding section. The advice giver can be thought to hold its own fixed knowledge base $\kappa(\pi)$ also, which it uses to identify whether it finds the inferences drawn and / or actions taken by the advice taker acceptable. The argumentative nature of reasoning presents the advice giver with a natural strategy for offering advice to the advice taker: if the latter's elaboration $t_i(x)$ is not acceptable by the former in context x, then it must be because $t_i(x)$ does not coincide with the grounded extension of the argumentation framework induced by $\kappa(\pi)$ and x; the former's advice, then, seeks to reduce this discrepancy, towards making $\kappa_i(\pi)$ and $\kappa(\pi)$ have induced argumentation frameworks that share a common grounded extension.

Consider, therefore, some context x during the i-th cycle of interaction. The advice taker proposes an elaboration $t_i(x)$ based on its current knowledge base $\kappa_i(\pi)$. In response, the advice giver offers the advice $f^{\kappa(\pi)}(x, t_i(x))$ through a feedback function $f^{\kappa(\pi)}$ that depends on the grounded extension $e(x)$ of the argumentation framework induced by $\kappa(\pi)$ and x. The advice taker concludes the cycle by revising $\kappa_i(\pi)$ to

[3]McCarthy [1998] actually calls this property "elaboration tolerance" — not to be confused with our use of the term "elaboration".

If: A rule r, an instance of which is in $t_i(x)$, is not in $\kappa(\pi)$.

Advise: "Rule r is invalid and should not be used!"

Revise: The advice taker removes rule r from its knowledge base, along with all its dependent priorities.

Else if: One of rule r's instances that is in $e(x)$, is applicable (i.e., its body is satisfied) in $t_i(x)$, but is not in $t_i(x)$.

Advise: "Rule r is relevant and should be applied!"

Revise: The advice taker adds rule r to its knowledge base, with priority over all conflicting rules.

Else if: A derivation d rebuts, under $\kappa(\pi)$, a derivation in $t_i(x)$, and it is not rebutted by a derivation in $e(x)$.

Advise: "Derivation d's rebuttal should be recognized!"

Revise: The advice taker adds rules with instances in d to its knowledge base, with priority over all conflicting rules.

Else: (None of the preceding conditions holds.)

Advise: "The elaboration of the context is acceptable!"

Revise: The advice taker retains its knowledge base.

Figure 1: Example Construction of an Advise-Revise Cycle.

$\kappa_{i+1}(\pi)$. In all cases, $\kappa = \langle \varrho, \succ \rangle$ and $\kappa_i = \langle \varrho_i, \succ_i \rangle$ for each cycle i, with κ_0 chosen arbitrarily as long as extra time polynomial in $||\kappa_0(\pi)||$ is allowed. We can show the following:

Theorem 8.1 (Existence of an Efficient PAC Advice Taker)**.** *Assume unit-time access to an oracle that computes grounded extensions of argumentation frameworks as discussed in Theorem 7.1. For every fixed Prolog program π, and every class $\mathcal{F} \subseteq \{f^{\kappa(\pi)} \mid \kappa$ is any totally-prioritized knowledge base$\}$ of feedback functions that offer advice as described in Figure 1, there exists an efficient advice taker that is PAC to an advice giver with any feedback function that belongs in $\mathcal{F}$.*

Proof (sketch). The advice taker and the advice giver interact as in Figure 1, until m consecutive elaborations are acceptable. The result follows for an appropriate choice of m. □

Thinking of the advice giver as a human wishing to get an advice-taking machine to operate in a certain manner, we call this mode of human-machine interaction ***machine coaching***. Lying between machine learning and machine programming, in this mode the human coach interacts rather actively, but still naturally, with the machine, as summarized below.

machine learning:
- the interaction is *one-sided* and can be online or batch
- the human *labels* inputs with outputs of the target theory
- the machine *generalizes* to create a hypothesis theory

machine programming:
- the interaction is *one-sided* and happens at the beginning
- the human *generates* explicit parts of the target theory
- the machine *blindly adds* parts to the hypothesis theory

machine coaching:
- the interaction is *dialectical* and is necessarily online
- the human *recognizes* mistakes in the hypothesis theory
- the machine appropriately *revises* the hypothesis theory

To highlight a key aspect of machine coaching, note that our example advice giver can, in principle, either tackle by itself the advice taker's task of elaborating contexts using the grounded extension of the former's induced argumentation framework, or directly program the advice taker with the former's knowledge. Pragmatically, however, machine coaching allows the advice giver to *offload* equitably its computational and cognitive burdens, exploiting the computational superiority of machines, and acknowledging the lighter human cognitive load to recognize mistakes in a dialectical setting rather than to generate arguments [Mercier and Sperber, 2011].

9 Conclusions

[...] a system which can be told to make a specific improvement in its behavior [...] Once this is achieved, we may be able to tell the advice taker how to learn from experience. — McCarthy, 1959

McCarthy [1959] had envisioned much more that the advice taker could do, most intriguing of which is perhaps the specification of its elaboration and integration procedures through machine coaching. We believe that our proposal offers a general enough basis to investigate such possible extensions, by considering, for example, the case of knowledge bases with production rules whose associated actions operate not on the advice taker's environment, but on the advice taker itself.

With an eye towards developing argumentation-based systems compatible with human cognitive capabilities and limitations [Michael *et al.*, 2015; Kakas and Michael, 2016], machine coaching is most usefully viewed not as a primary mode of human-machine interaction — except for simple everyday tasks (cf. Example 6.1) — but as a debugging and personalization mode subsidiary to programming and learning, which currently serve better the goal of acquiring imperative and general (commonsense) declarative knowledge, respectively.

On the matter of declarative knowledge, our earlier work shows that it can be learned autodidactically [Michael, 2008] without human interaction, and just by reading text [Michael, 2009] as found on the Web. Noting that learning cannot proceed independently from the manner in which knowledge is used to reason [Michael, 2014], our recently-proposed `NERD` algorithm [Michael, 2016] for learning in a certain argumentative setting seems adaptable to this work's reasoning semantics. The role of machine coaching in learning-driven knowledge acquisition is exemplified by observing that text on the Web encodes what can be called *websense* [Michael, 2013], with whatever biases and mistakes this entails. Its inevitable discrepancies with commonsense or user-specific knowledge can be ironed out by using machine coaching alongside machine learning. Whether deep learning can likewise be integrated with machine coaching remains an intriguing prospect.

References

[Diakidoy *et al.*, 2014] Irene-Anna Diakidoy, Antonis Kakas, Loizos Michael, and Rob Miller. Story Comprehension through Argumentation. In *Proceedings of the 5th International Conference on Computational Models of Argument*, pages 31–42, Scottish Highlands, U.K., 2014.

[Diakidoy *et al.*, 2015] Irene-Anna Diakidoy, Antonis Kakas, Loizos Michael, and Rob Miller. STAR: A System of Argumentation for Story Comprehension and Beyond. In *Proceedings of the 12th International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 64–70, Palo Alto, CA, U.S.A., 2015.

[Dung, 1995] Phan M. Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming, and n-Person Games. *Artificial Intelligence*, 77(2):321–357, 1995.

[Kakas and Michael, 2016] Antonis Kakas and Loizos Michael. Cognitive Systems: Argument and Cognition. *IEEE Intelligent Informatics Bulletin*, 17(1):14–20, 2016.

[Kearns and Vazirani, 1994] Michael Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, MA, U.S.A., 1994.

[McCarthy, 1959] John McCarthy. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, England, U.K., 1959.

[McCarthy, 1998] John McCarthy. Elaboration Tolerance. In *Proceedings of the 4th International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 198–216, London, England, U.K., 1998.

[Mercier and Sperber, 2011] Hugo Mercier and Dan Sperber. Why Do Humans Reason? Arguments for an Argumentative Theory. *Behavioral and Brain Sciences*, 34(2):57–74, 2011.

[Michael *et al.*, 2015] Loizos Michael, Antonis Kakas, Rob Miller, and György Turán. Cognitive Programming. In *Proceedings of the 3rd International Workshop on Artificial Intelligence and Cognition*, pages 3–18, Turin, Italy, 2015.

[Michael, 2008] Loizos Michael. *Autodidactic Learning and Reasoning*. Doctoral Dissertation, Harvard University, Cambridge, MA, U.S.A., 2008.

[Michael, 2009] Loizos Michael. Reading Between the Lines. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1525–1530, Pasadena, CA, U.S.A., 2009.

[Michael, 2013] Loizos Michael. Machines with WebSense. In *Proceedings of the 11th International Symposium on Logical Formalizations of Commonsense Reasoning*, Ayia Napa, Cyprus, 2013.

[Michael, 2014] Loizos Michael. Simultaneous Learning and Prediction. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning*, pages 348–357, Vienna, Austria, 2014.

[Michael, 2016] Loizos Michael. Cognitive Reasoning and Learning Mechanisms. In *Proceedings of the 4th International Workshop on Artificial Intelligence and Cognition*, pages 2–23, New York City, NY, U.S.A., 2016.

[Prakken, 2010] Henry Prakken. An Abstract Framework for Argumentation with Structured Arguments. *Argument and Computation*, 1(2):93–124, 2010.

[Valiant, 1984] Leslie G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

A Example Construction from Theorem 7.1

Figure 2 illustrates key steps of the construction used in the proof of Theorem 7.1. Capital letters are used to denote arbitrary *variable-free* literals, which implies that rules coincide with their instances, as is the case in a "propositional" knowledge base. The construction is applied on a knowledge base κ with rules $r_{01}, \ldots r_{19}$ and priorities $r_{01} \succ r_{02}$, $r_{07} \succ r_{08}$, $r_{17} \succ r_{09}$, $r_{09} \succ r_{06}$. The first panel depicts the rules of κ, whereas subsequent panels show the processing of the derivation network resulting from κ and a context $x = \{A, \neg Z\}$.

Priority $r_{01} \succ r_{02}$ eventually leads to the removal of r_{02}, and to the consequent breaking of the cycle between P and Q during iteration 2. Priority $r_{07} \succ r_{08}$ eventually leads to the marking of r_{07} and L, and to the consequent removal of r_{08} during iteration 3. Priorities $r_{17} \succ r_{09}$ and $r_{09} \succ r_{06}$ initially keep r_{06}, r_{16} from being marked, until after the marking of r_{17}, and the removal of $\neg R$ and r_{09} during iteration 4. The grounded extension is the marked part in the last panel.

Rule instances $r_{12}, \ldots, r_{17}$ demonstrate a scenario of how exponentially-many derivations can be succinctly encoded in a derivation network, which is a central feature towards achieving efficient computation of the grounded extension.

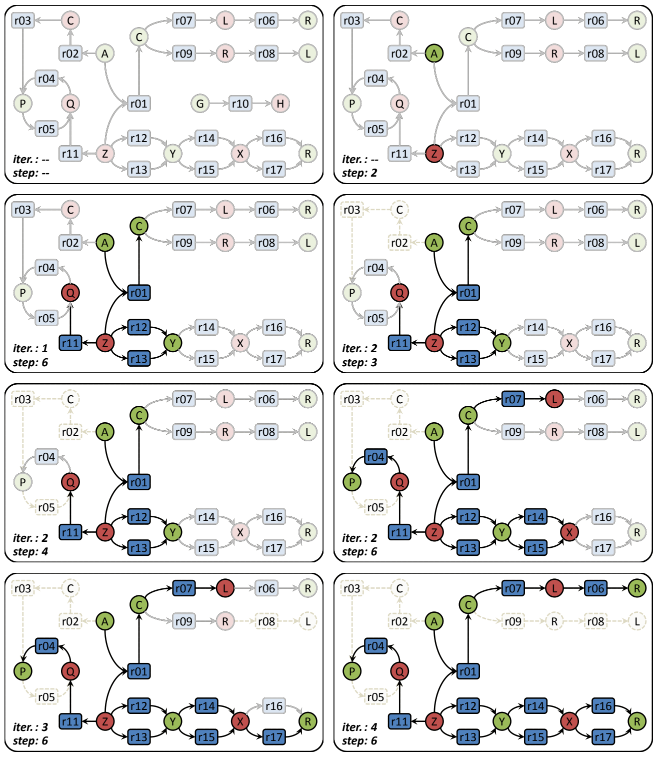

Figure 2: Key steps of the construction used in the proof of Theorem 7.1. Light colors / dark colors / dashed lines show existing / marked / removed parts of the derivation network. Green / red circles show positive / negative literals. Blue rectangles show hyperedges with arrows indicating directions.