What's Missing in AI: The Interface Layer

Pedro Domingos
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
pedrod@cs.washington.edu

1 The Interface Layer

AI has made tremendous progress in its first 50 years. However, it is still very far from reaching and surpassing human intelligence. At the current rate of progress, the crossover point will not be reached for hundreds of years. We need innovations that will permanently increase the rate of progress. Most research is inevitably incremental, but the size of those increments depends on the paradigms and tools that researchers have available. Improving these provides more than a one-time gain; it enables researchers to consistently produce larger increments of progress at a higher rate. If we do it enough times, we may be able to shorten those hundreds of years to decades.

If we look at other subfields of computer science, we see that in most cases progress has been enabled above all by the creation of an interface layer that separates innovation above and below it, while allowing each to benefit from the other. Below the layer, research improves the foundations (or, more pragmatically, the infrastructure); above it, research improves existing applications and invents new ones. Table 1 shows examples of interface layers from various subfields of computer science. In each of these fields, the development of the interface layer triggered a period of rapid progress above and below it. In most cases, this progress continues today. For example, new applications enabled by the Internet continue to appear, and protocol extensions continue to be proposed. In many cases, the progress sparked by the new layer resulted in new industries, or in sizable expansions of existing ones.

Table 1: Examples of interface layers.

Field	Interface Layer	Below the Layer	Above the Layer
Hardware	VLSI design	VLSI modules	Computer-aided chip design
Architecture	Microprocessors	ALUs, buses	Compilers, operating systems
Operating systems	Virtual machines	Hardware	Software
Programming systems	High-level languages	Compilers, code optimizers	Programming
Databases	Relational model	Query optimization, transaction mgmt.	Enterprise applications
Networking	Internet	Protocols, routers	Web, email
HCI	Graphical user interface	Widget toolkits	Productivity suites
AI	???	Inference, learning	Planning, NLP, vision, robotics

The interface layer allows each innovation below it to automatically become available to all the applications above it, without the “infrastructure” researchers having to know the details of the applications, or even what all the existing or possible applications are. Conversely, new applications (and improvements to existing ones) can be developed with little or no knowledge of the infrastructure below the layer. Without it, each innovation needs to be separately combined with each other, an $O(n^2)$ problem that in practice is too costly to solve, leaving only a few of the connections made. When the layer is available, each innovation needs to be combined only with the layer itself, and we obtain $O(n^2)$ benefits with $O(n)$ work.

In each case, the separation between applications above the layer and infrastructure below it is not perfect. If we care enough about a particular application, we can usually improve it by developing infrastructure specifically for it. However, most applications are served well enough by the general-purpose machinery, and in many cases would not be economically feasible without it. For example, ASICs (application-specific integrated circuits) are far more efficient for their specific applications than general-purpose microprocessors; but the vast majority of applications are still run on the latter. Interface layers are an instance of the 80/20 rule: they allow us to obtain 80% of the benefits for 20% of the cost.

The essential feature of an interface layer is that it provides a language of operations that is all the infrastructure needs to support, and all that the applications need to know about. Designing it is a difficult balancing act between providing what the applications need and staying within what the infrastructure can do. A good interface layer exposes important distinctions and hides unimportant ones. How to do this is often far from obvious. Creating a successful new interface layer is thus not easy. Typically, it initially faces skepticism, because it is less efficient than the existing alternatives, and appears too ambitious, being beset by difficulties that are only resolved by later research. But once the layer becomes established, it enables innovations that were previously unthinkable.

2 What Is the Interface Layer for AI?

In AI, the interface layer has been conspicuously absent, and this, perhaps more than any other factor, has limited the rate of progress. An early candidate for this role was first-order logic. However, it quickly became apparent that first-order logic has many shortcomings as a basis for AI, leading to a long series of efforts to extend it. Unfortunately, none of these extensions have achieved wide acceptance. Perhaps the closest first-order logic has come to providing an interface layer is the Prolog language. However, it remains a niche language even within AI. This is largely because, being essentially a subset of first-order logic, it shares its limitations, and is thus insufficient to support most applications at the 80/20 level.

Many (if not most) of the shortcomings of logic can be overcome by the use of probability. Here, graphical models (i.e., Bayesian and Markov networks) have to some extent played the part of an interface layer, but one with a limited range. Although they provide a unifying language for many different probabilistic models, graphical models can only represent distributions over propositional universes, and are thus insufficiently expressive for general AI. In practice, this limitation is often circumvented by manually transforming the rich relational domain of interest into a simplified propositional one, but the cost, brittleness, and lack of reusability of this manual work is precisely what a good interface layer should avoid. Also, to the extent that graphical models can provide an interface layer, they have done so mostly at the conceptual level. No widely accepted languages or standards for representing and using graphical models exist today. Many toolkits with specialized functionality have been developed, but none that could be used as widely as (say) SQL engines are in databases. Perhaps the most widely used such toolkit is BUGS, but it is quite limited in the learning and inference infrastructure it provides. Although very popular with Bayesian statisticians, it fails the 80/20 test for AI.

It is clear that the AI interface layer needs to integrate first-order logic and graphical models. One or the other by itself cannot provide the minimum functionality needed to support the full range of AI applications.

Table 2: Examples of logical and statistical AI.

Field	Logical Approach	Statistical Approach
Knowledge representation	First-order logic	Graphical models
Automated reasoning	Satisfiability testing	Markov chain Monte Carlo
Machine learning	Inductive logic programming	Neural networks
Planning	Classical planning	Markov decision processes
Natural language processing	Definite clause grammars	Probabilistic context-free grammars

Further, the two need to be fully integrated, and not simply provided alongside each other. Most applications require simultaneously the expressiveness of first-order logic and the robustness of probability, not just one or the other. Unfortunately, the split between logical and statistical AI runs very deep. It dates to the earliest days of the field, and continues to be highly visible today. It takes a different form in each subfield of AI, but is omnipresent. Table 2 shows examples of this. In each case, both the logical and the statistical approach contribute something important. This justifies the abundant research on each of them, but also implies that ultimately a combination of the two is required.

In recent years, we have begun to see increasingly frequent attempts to achieve such a combination in each subfield. In knowledge representation, knowledge-based model construction combines logic programming and Bayesian networks, and substantial theoretical work on combining logic and probability has appeared. In automated reasoning, researchers have identified common schemas in satisfiability testing, constraint processing and probabilistic inference. In machine learning, statistical relational learning combines inductive logic programming and statistical learning. In planning, relational MDPs add aspects of classical planning to MDPs. In natural language processing, work on recognizing textual entailment and on learning to map sentences to logical form combines logical and statistical components. However, for the most part these developments have been pursued independently, and have not benefited from each other. This is largely attributable to the $O(n^2)$ problem: researchers in one subfield can connect their work to perhaps one or two others, but connecting it to all of them is not practically feasible.

My collaborators and I have recently introduced Markov logic, a language that combines first-order logic and Markov networks (Domingos *et al.*, 2006). A knowledge base (KB) in Markov logic is a set of first-order formulas with weights. Given a set of constants representing objects in the domain of interest, it defines a probability distribution over possible worlds, each world being an assignment of truth values to all possible ground atoms. The distribution is in the form of a log-linear model: a normalized exponentiated weighted combination of features of the world.[1] Each feature is a grounding of a formula in the KB, with the corresponding weight. In first-order logic, formulas are hard constraints: a world that violates even a single formula is impossible. In Markov logic, formulas are soft constraints: a world that violates a formula is less probable than one that satisfies it, other things being equal, but not impossible. The weight of a formula represents its strength as a constraint. Finite first-order logic is the limit of Markov logic when all weights tend to infinity. Markov logic allows an existing first-order KB to be transformed into a probabilistic model simply by assigning weights to the formulas, manually or learning them from data. It allows multiple KBs to be merged without resolving their inconsistencies, and obviates the need to exhaustively specify the conditions under which a formula can be applied. On the statistical side, it allows very complex models to be represented very compactly; in particular, it provides an elegant language for expressing non-i.i.d. models (i.e., models where data points are not assumed independent and identically distributed). It also facilitates the incorporation of rich domain knowledge, reducing reliance on purely empirical learning.

[1]Log-linear models are also known as, or closely related to, Markov networks, Markov random fields, maximum entropy models, Gibbs distributions, and exponential models; and they have Bayesian networks, Boltzmann machines, conditional random fields, and logistic regression as special cases.

Table 3: A comparison of Alchemy, Prolog and BUGS.

Aspect	Alchemy	Prolog	BUGS
Representation	First-order logic + Markov networks	Horn clauses	Bayesian networks
Inference	Model checking, MCMC	Theorem proving	MCMC
Learning	Parameters and structure	No	Parameters
Uncertainty	Yes	No	Yes
Relational	Yes	Yes	No

Markov logic builds on previous developments in knowledge-based model construction and statistical relational learning, but goes beyond them in combining finite first-order logic and graphical models without restrictions. Unlike previous representations, it is supported by a full range of learning and inference algorithms, in each case combining logical and statistical elements. Because of its generality, it provides a natural framework for integrating the logical and statistical approaches in each field. For example, DCGs and PCFGs are both special cases of Markov logic, and classical planning and MDPs can both be elegantly formulated using Markov logic and decision theory. With Markov logic, combining classical planning and MDPs, or DCGs and PCFGs, does not require new algorithms; the existing general-purpose inference and learning facilities can be directly applied.

AI can be roughly divided into "foundational" and "application" areas. Foundational areas include knowledge representation, automated reasoning, probabilistic models, and machine learning. Application areas include planning, vision, robotics, speech, natural language processing, and multi-agent systems. An interface layer for AI must provide the former, and serve the latter. We have developed the Alchemy system as an open-source embodiment of Markov logic and implementation of algorithms for it (Kok *et al.*, 2006). Alchemy seamlessly combines first-order knowledge representation, model checking, probabilistic inference, inductive logic programming, and generative/discriminative parameter learning. Table 3 compares Alchemy with Prolog and BUGS, and shows that it provides a critical mass of capabilities not previously available.

3 Research Below the Interface

Each algorithm included in Alchemy is state-of-the-art, and in most cases the product of new research that is directly relevant to the original field. For example, we have developed:

- A satisfiability tester that scales to much larger relational domains that was previously possible.
- An MCMC algorithm that can correctly and efficiently handle deterministic dependencies.
- A generalization to relational domains of the voted perceptron, a discriminative parameter learning algorithm previously developed for hidden Markov models.
- An inductive logic programming algorithm that uses a likelihood measure as the objective function, and is thus able to accurately learn relational probability distributions.

However, these algorithms are only the first steps in making Alchemy robust and scalable enough to be routinely used by application researchers. The goal of Alchemy as an open-source system is to allow easy integration of new algorithms, and to benefit from continuing progress in areas like probabilistic inference, satisfiability testing, statistical learning, and inductive logic programming. By providing their algorithms as Alchemy modules, researchers in the foundational areas of AI allow them to be automatically combined with algorithms from other foundational areas, and used by application researchers with little or no ramping up. Thus Alchemy potentially allows researchers to have $O(n)$ impact with $O(1)$ effort.

The confluence of learning, logic and probability also suggests many fascinating new research directions: probability distributions over infinite relational domains; higher-order Markov logics; representing uncertainty over all elements of a first-order language; extending Markov logic to encompass numeric features; lifted first-order probabilistic inference; first-order decision theory; learning the structure of relational distributions from incomplete data; statistical predicate invention; ontology induction; learning at multiple levels of abstraction; etc.

4 Research Above the Interface

For researchers and practitioners in application areas, Alchemy offers a large reduction in the effort required to assemble a state-of-the-art solution, and to extend it beyond the state of the art. The representation, inference and learning components required for each subtask, both logical and probabilistic, no longer need to be built or patched together piece by piece; Alchemy provides them, and the solution is built simply by writing formulas in Markov logic. A few lines of Alchemy suffice to build state-of-the-art systems for applications like collective classification, link prediction, entity resolution, information extraction, ontology mapping, and others. Because each of these pieces is now simple to implement, combining them into larger systems becomes straightforward, and is no longer a major engineering challenge. For example, we are currently beginning to build a complete natural language processing system in Alchemy, which aims to provide the functionality of current systems in one to two orders of magnitude fewer lines of code. Most significantly, Alchemy facilitates extending NLP systems beyond the current state of the art, for example by integrating probabilities into semantic analysis.

One of our goals with Alchemy is to support the growth of a repository of reusable knowledge bases in Markov logic, akin to the shareware repositories available today, and building on the traditional knowledge bases already available. Given such a repository, the first step of an application project becomes the selection of relevant knowledge. This may be used as is or manually refined. A new knowledge base is initiated by writing down plausible hypotheses about the new domain. This is followed by induction from data of new knowledge for the task. The formulas and weights of the supporting knowledge bases may also be adjusted based on data from the new task. New knowledge is added by noting and correcting the failures of the induced and refined KBs, and the process repeats. Over time, new knowledge is gradually accumulated, and existing knowledge is refined and specialized to different (sub)domains. Experience shows that neither knowledge engineering nor machine learning by itself is sufficient to reach human-level AI, and a simple two-stage solution of knowledge engineering followed by machine learning is also insufficient. What is needed is a fine-grained combination of the two, where each one bootstraps from the other, and at the end of each loop of boostrapping the AI system's state of knowledge is more advanced than at the beginning. Alchemy supports this.

More broadly, a tool like Alchemy can help the focus of research shift from very specialized goals to higher-level ones. This is essential to speed progress in AI. As the field has grown, it has become atomized, but ultimately the pieces need to be brought back together. However, attempting to do this without an interface layer, by gluing together a large number of disparate pieces, rapidly turns into an engineering quagmire; systems become increasingly hard to build on for further progress, and eventually sheer complexity slows progress to a crawl. By keeping the pieces simpler and providing a uniform language for representing and combining them, even if at some cost in performance, an interface layer enables us to reach much farther before hitting the complexity wall. At that point, we have hopefully acquired the knowledge and insights to design the next higher-level interface layer, and in this way we can continue to make rapid progress.

Highly focused research is essential for progress, and often provides immediate real-world benefits in its own right. But these benefits will be dwarfed by those obtained if AI reaches and surpasses human intelligence, and to contribute toward this, improvements in performance in the subtasks need to translate into improvements in the larger tasks. When the subtasks are pursued in isolation, there is no guarantee that this will happen, and in fact experience suggests that the tendency will be for the subtask solutions to evolve into local optima, which are best in isolation but not in combination. By increasing the granularity of the tasks that can be routinely attempted, platforms like Alchemy make this less likely, and help us reach human-level AI sooner.

References

Domingos, P., Kok, S., Poon, H., Richardson, M., & Singla, P. (2006). Unifying logical and statistical AI. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. Boston, MA: AAAI Press. http://www.cs.washington.edu/homes/pedrod/papers/aaai06c.pdf.

Kok, S., Singla, P., Richardson, M., and Domingos, P. (2005). The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://www.cs.washington.edu/ai/alchemy.