COSC410 Lecture 6
Automated Reasoning Algorithms

Willem Labuschagne
University of Otago

2016

Rules of Inference

Logic-based AI began in 1956 with a program called Logic Theorist (by Simon, Newall and Shaw) which was able to prove a number of mathematical theorems with the aid of syntactic rules of inference. How does this sort of program work?

The first important point to make is that such programs only work for classical logics, i.e. for logics in which the consequence relation is $\models$. In such a classical logic, the syntactic shape of a sentence (called its *logical form*) determines what the models of the sentence are, and this allows us to write programs that compute whether $\alpha \models \beta$ by examining the shapes of α and β. Without writing any code, let's see how this might be done.

Suppose we have a set X of sentences that express the information we have accumulated so far. Call this our database. Now we want a program that can be applied to our database in order to generate sentences β such that $X \models \beta$.

Such a program is built around one or more rules of inference. A rule of inference has two parts, corresponding to the premiss and the conclusion of an inference. If the program can match sentences of the database to the premisses of the rule, then the program adds the conclusion to the database.

It's important to bear in mind that the program transforms premiss-strings into conclusion-strings without understanding what the strings mean. In other words, the algorithm would work syntactically, by pattern-matching, not semantically. It is easier and more efficient to tell the algorithm what to do according to the shapes of symbols than according to their meanings.

Note that I am not saying it is impossible to design algorithms that will calculate the models of a sentence and proceed semantically to work out whether $\alpha \models \beta$. The drawback is that such semantic algorithms tend to be very inefficient. Consider that a propositional language with n atomic sentences will have 2^n truth assignments, and to find the models of a sentence the semantic algorithm might have to check each of the 2^n truth assignments individually. Algorithms having an exponential complexity cannot be used except for very small values of n. Hence the interest in syntactic algorithms — there is the hope that if all the algorithm has to do is to trundle along a string of symbols and determine what to do next by seeing what symbols occur, then maybe efficiencies tending towards $O(n)$ might be achievable if one is very lucky or very clever.

What sorts of rules might be used to develop an automated reasoning algorithm that works syntactically (using the shapes of symbols) rather than semantically (using the meanings of symbols)?

Five main approaches, or *deductive architectures*, have been developed. In historical order:

- Hilbert-style architectures
- Gentzen-style sequent calculus
- Prawitz's natural deduction system
- Beth's tableaux system (popularised by Raymond Smullyan)
- JA Robinson's resolution rule.

Although resolution has been by far the most popular approach used by the computer science community, and you can get a free automated reasoner based on resolution called Prover9 (see http://www.cs.unm.edu/ mccune/prover9/), we choose to give the flavour of natural deduction, as its rules of inference are easier to grasp.

A natural deduction architecture has two inference rules for every connective (i.e. for each of the symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$). One rule *introduces* the connective, i.e. tells us how a sentence with that symbol as its main connective might be inferred as a conclusion. The other rule *eliminates* the connective, i.e. tells us what conclusions may be inferred from a premiss with that symbol as its main connective.

By way of an example, consider the rule of
And-introduction: from premisses α and β, infer conclusion $\alpha \wedge \beta$.

So if the sentences α and β are already in our database X, then the rule allows us to add $\alpha \wedge \beta$ to the database.

This makes sense: if we already know α and we already know β then we already have all the information conveyed by the sentence $\alpha \wedge \beta$. The sentences α and β are the two premisses of the rule, and $\alpha \wedge \beta$ is the conclusion that the rule allows us to infer. In the sentence $\alpha \wedge \beta$ the connective $\wedge$ is the main connective. Other connectives may have been used to build α and β and may thus occur inside the string $\alpha \wedge \beta$, but the *main* connective of $\alpha \wedge \beta$ is the one used in the final step of building the sentence $\alpha \wedge \beta$. The inference rule of And-introduction introduces the connective $\wedge$ to build the conclusion $\alpha \wedge \beta$ from the premisses α and β.

Conversely, we have the self-explanatory rule of
And-elimination: from a premiss of the form $\alpha \wedge \beta$ we may infer α and we may infer β.

As a second example, consider the connective $\rightarrow$. In one direction we have the rule
Conditional-elimination: from premisses α and $\alpha \rightarrow \beta$, infer conclusion β.

So if the sentences α and $\alpha \rightarrow \beta$ are already in our database X, then we may add β to our database.

This is an ancient and well-known rule, traditionally known as *Modus Ponens*, which is Latin for 'the method of affirming'. The idea is that the gap from α to β can be bridged with the help of $\alpha \rightarrow \beta$, provided we can 'affirm' α, in other words provided we have α in our database.

Another name for the rule of Conditional-elimination is the rule of detachment, and it's easy to see why some people use this name, as the rule tells us what we need in order to to detach β from $\alpha \rightarrow \beta$.

In the converse direction we have the slightly more complicated rule
Conditional-introduction: If from α we can deduce β by a sequence of one or more applications of our inference rules, then we may infer $\alpha \to \beta$.

For example, perhaps α might be the sentence $\varphi \wedge (\beta \wedge \psi)$, so that by And-elimination we get the conclusion $\beta \wedge \psi$ and then by And-elimination again we get the conclusion β. Now the rule of Conditional-introduction tells us we can infer $\alpha \to \beta$.

Each of the other connectives also has introduction and elimination rules. Given such a set of inference rules, an automated reasoner could begin with an initial database X of sentences expressing what the agent had learnt so far, perform an inference step with the help of a rule whose premisses are in the database already, and produce a new database by adding the conclusion of the rule to the previous database. The procedure may be iterated.

To illustrate, imagine that we have a language with $A = \{p, q, r, s\}$. Suppose the agent's initial information is expressed by the following database of sentences:

$$\{\ p,\ p \to q,\ r \to p,\ q \to s\ \}.$$

The automated reasoner can apply the rule of Conditional-elimination (Modus Ponens) by matching the first two sentences in the database to the premisses of the rule, and the premisses p and $p \to q$ then give the conclusion q, which the automated reasoner adds to the database. The new database is

$$\{\ p,\ p \to q,\ r \to p,\ q \to s,\ q\ \}.$$

Similarly, applying Modus Ponens to the last two sentences gives the new sentence s which is added to the database:

$$\{\ p,\ p \to q,\ r \to p,\ q \to s,\ q,\ s\ \}.$$

And of course we could apply And-introduction to the first and last sentences to deliver

$$\{\ p,\ p \to q,\ r \to p,\ q \to s,\ q,\ s,\ p \wedge s\ \}.$$

And so on.

Properties of Automated Reasoning Algorithms

Suppose we plan to use a syntactic reasoning algorithm. Since it is the entailment relation $\models$ that characterises which inferences from premiss to conclusion are legitimate in classical logic, we are always interested in the question: *How well does the reasoning algorithm mimic* $\models$*?*

Each of the five deductive architectures, including natural deduction, allows the design of an automated reasoning algorithm that is a pretty good approximation of the classical entailment relation $\models$. The question is, what do we mean by 'pretty good approximation'?

There are three key properties that relate directly to the adequacy with which the algorithm simulates $\models$. These are the properties of *soundness*, *completeness*, and *decidability*.

Suppose we write $X \vdash \beta$ to say that our algorithm is able to use premisses from X to infer the conclusion β with the help of the allowed set of inference rules, i.e. is able to *deduce* β from X. (The symbol $\vdash$ used for deduction is called the single turnstile, and should not be confused with the double turnstile symbol we are using for entailment, $\models$.)

By *soundness* we mean that if $X \vdash \beta$ then $X \models \beta$, in other words our reasoning algorithm deduces only sentences that are classically entailed by X. (Soundness is about safety. If a sound automated reasoner deduces β from the database, the inference is safe because $X \models \beta$.)

For example, the rule of And-elimination is $\alpha \wedge \beta \vdash \alpha$, and this rule is sound because it is indeed always the case that $\alpha \wedge \beta \models \alpha$.

We would normally insist that our reasoning algorithm be built up from sound rules of inference, but there are some contexts in which we might allow the use of unsound rules such as *negation by failure.*

Example. *The unsound rule negation by failure.*
Here is a real-life situation in which we might want to use this rule. Suppose our database X has information about the airline flights between cities, and we are confident that all existing flights have been recorded in the database. Then we could adopt the closed world assumption, namely the assumption that if it isn't in our database, at least implicitly, then it doesn't exist. If someone were now to ask whether a particular flight existed, we would consult our automated reasoner which would try to infer that flight from the database. Should the inference fail, then the unsound rule of negation as failure would allow us to infer that the flight does not exist.

Here is an even simpler illustration. Suppose we start with the database

$$\{\ p,\ p \to q,\ r \to p,\ q \to s\ \}.$$

And suppose further that we want to know whether r is the case.

Since r is an atomic sentence, we know that it can't be inferred by the rule of And-introduction, or indeed the introduction of any other connective. And since the only connective occurring in our database is $\to$, we could limit the algorithm to using Modus Ponens, which eliminates $\to$. After a couple of steps, the algorithm would produce

$$\{\ p,\ p \to q,\ r \to p,\ q \to s,\ q,\ s\ \}.$$

After this, the rule Modus Ponens has no way to infer any additional sentences, so if we can somehow design the algorithm to recognise that it has exhausted the available premisses, it can in effect recognise that it has failed to infer r. The rule negation as failure would then justify inferring $\neg r$.

Notice that this rule is unsound. It is not the case that $X \models \neg r$ because if we have a language with $A = \{p, q, r, s\}$ then the truth assignment corresponding to the string 1111 *would satisfy all the sentences in X and would also satisfy r, i.e. would fail to satisfy $\neg r$. Thus* 1111 *is a model of X that is not a model of $\neg r$. Thus $X \not\models \neg r$.*

The next property to consider is called *adequacy* by some logicians, and *completeness* by most. By (adequacy or) completeness we mean that if $X \models \beta$ then $X \vdash \beta$, in other words our reasoning algorithm is strong enough to deduce every sentence classically entailed by X. (Completeness is about power. Is the automated reasoner powerful enough to simulate $\models$?)

One generally doesn't care about completeness unless one also has soundness, because completeness without soundness could be quite dangerous. Imagine a silly reasoning algorithm that deduces every sentence β from the database X, so that $X \vdash \beta$ for all $\beta \in L_A$. This silly reasoner is complete, because every sentence β such that $X \models \beta$ is certainly going to be deduced from X by the reasoner. The problem is that lots of sentences that shouldn't be deduced from X will also be deduced. Soundness would prevent things from going wrong in this way.

The logic programming language Prolog is based on a reasoning algorithm known as SLDNF-resolution, where the NF stands for negation as failure. So Prolog makes inferences that are unsound and thus unsafe unless we have the closed world assumption. The SLDNF-resolution algorithm is also incomplete. Again, this is something we can sometimes live with fairly happily, say when we are working with databases, but it does mean the algorithm is not in general a very good simulation of $\models$.

The soundness and completeness of an automated reasoning algorithm together guarantee that

$$X \vdash \beta \text{ if and only if } X \models \beta.$$

Think of such a reasoner as a reasonably good question-answering device. We may ask it whether $X \vDash \beta$, and if it is indeed the case that $X \vDash \beta$ then our sound and complete reasoner will confirm this by outputting $X \vdash \beta$ or words to that effect.

Reasonably good means there is good news but also some bad news.

The good news is that such a reasoner won't tell us lies. Suppose we ask whether $X \vDash \beta$. And suppose it is in fact the case that $X \nvDash \beta$. Then our sound and complete reasoner will not try to fool us by answering 'yes' (i.e. by outputting the lie $X \vdash \beta$).

The bad news is that sometimes the reasoner will be silent when we want an answer. In cases where $X \nvDash \beta$, the requirement of soundness and completeness does not oblige the reasoning algorithm to tell us 'no' (i.e. it doesn't have to output that $X \nvdash \beta$). Won't we be able to tell that $X \nvdash \beta$ by having the reasoning program stop without giving us the confirmation $X \vdash \beta$? No, because the reasoning algorithm might not terminate at all, i.e. may loop. And the difficulty is, we won't know whether it's going to terminate. (If you did COSC 341, this should remind you of the Halting Problem.)

We see that a sound and complete reasoning algorithm is halfway perfect — it guarantees to confirm that $X \models \beta$ by telling us that $X \vdash \beta$, but it doesn't guarantee to confirm that $X \nvDash \beta$ by telling us anything.

What would be even better than a sound and complete reasoning algorithm would be a reasoning algorithm that is a *decision procedure*. Such an algorithm would make the question "Does $X \models \beta$?" decidable, i.e. would guarantee to say **yes** or **no** as appropriate. A decision procedure is thus a sound and complete reasoning algorithm that can also cope with the case in which $X \nvDash \beta$. A decision procedure will tell us that $X \vdash \beta$ if and only if it is the case that $X \models \beta$, and in addition the decision procedure will tell us that $X \nvdash \beta$ if and only if $X \nvDash \beta$.

For some logics, the entailment relation is decidable, and for others it is not. That is, for some logics, such as classical propositional logic, we can design a syntactic reasoning algorithm that is a decision procedure for $\models$. For other logics (e.g. first-order logic with infinitely many predicates of every arity), it has been shown to be impossible to design a decision procedure, although one can still design a sound and complete reasoning algorithm. And finally, there are known to be logics for which it is impossible to design a sound and complete reasoning algorithm, and thus also impossible to design a decision procedure.

From Rules of Inference to Semantic Algorithms

In 1956 an important conference was held at Dartmouth University in the USA. Here John McCarthy coined the term 'artificial intelligence'. When Herbert Simon (a prominent researcher

in AI who later earned a Nobel Prize for Economics) demonstrated the automated reasoning program they called the Logic Theorist, a rosy glow of optimism descended upon the assembled throng, who were mightily impressed that this program could actually prove non-trivial mathematical theorems. In this moment the idea of Symbolic AI was born.

Symbolic AI is based on the notion that one should be able to design an intelligent agent by representing a lot of knowledge in sentences of L_A and then turning loose a syntactic reasoning algorithm like Logic Theorist on that knowledge.

This makes AI seem simple, and puts classical logic firmly at the centre. All you would need to do is represent your information *declaratively*, in other words in sentences of some logic language, and then hook up a sufficiently strong syntactic reasoning algorithm to get an intelligent agent. It didn't seem to occur to anyone that this agent wouldn't actually be able to do anything, like serve tea or fetch a parcel from the Post Office. Proving mathematical theorems is very different from processing the inputs derived from sensors and planning movement.

When researchers in AI tried to design an agent to have sensors (e.g. for vision) and effectors with which to grasp or move, they discovered that things were much more complicated than they had seemed. In the mid-1960s Marvin Minsky and Seymour Papert at MIT conceived a "summer vision project", setting a graduate student the task of solving the object recognition problem in computer vision over the summer. As an example of cheerful optimism, it would be hard to outdo this. It took decades of research by hundreds of researchers before we reached the point of having digital cameras programmed to recognise which part of an image is a face.

Generally speaking, the great discovery that came from trying to build robots capable of doing everyday things (instead of just proving mathematical theorems) was that common sense reasoning was needed, and that such common sense involved the use of default rules (rules of thumb). According to this insight, classical logic is a good way to represent mathematical reasoning but not a good way to represent everyday common sense reasoning. This eventually led to the development of nonmonotonic logic and the gradual collapse of the Symbolic AI dream.

We can now summarise both the strengths and the weaknesses of syntactic reasoning algorithms. For situations in which an agent reasons with sufficient information and wants to infer only conclusions that are absolutely guaranteed, e.g. in mathematics, classical entailment works well and a syntactic reasoning algorithm simulating it is useful. Problems remain in respect of efficiency. It has been found to be very difficult to direct the computations of such an algorithm towards the goal and to avoid costly digressions into irrelevant avenues. Some of the most effective of these algorithms (e.g. the OTTER program developed at the Argonne National Laboratory) are therefore designed to function in partnership with a human who steers the program.

Perhaps the main limitation is the requirement that sufficient information be provided to the syntactic reasoning algorithm. In doing mathematics, one has complete information in the form of definitions and one merely wishes to unravel the classical consequences of the definitions. In contrast, for everyday decision-making one has to somehow represent default rules because much of what we learn from the environment is heuristic information of this sort. We used a semantic representation of default rules in our exposition of nonmonotonic logic. There have also been attempts to develop syntactic representations by means of special rules of inference, for example in the so-called *default logic* developed by Raymond Reiter. These have not been remarkable for their success.

Not only are default rules difficult to represent by syntactic rules of inference, but the Ineffability Theorem suggests that for any sufficiently complex system, there will be factual information

that can be described semantically but not expressed in the representation language for that system (i.e. not described declaratively). So if we want to treat complex systems for which the corresponding language is required to have infinitely many atomic sentences, then the whole idea of stating all the relevant facts declaratively is misguided.

Now, if syntactic reasoning algorithms are useful when classical entailment is what we need, how can we try to achieve greater efficiency?

One approach is to improve efficiency by changing the question "Does $\alpha \models \beta$?" into the apparently simpler question "Is $\alpha \wedge \neg\beta$ satisfiable?".

If $\alpha \wedge \neg\beta$ is satisfiable, then there is at least one model of α that is also a model of $\neg\beta$, and so we know that $X \not\models \beta$. On the other hand, if $\alpha \wedge \neg\beta$ is unsatisfiable, then every model of α must be a model of β, and so $\alpha \models \beta$.

Algorithms to solve the satisfiability problem are called SAT-solvers. Notice something interesting. Satisfiability is about the existence of a model. So our attempt to improve the efficiency of syntactic reasoning algorithms has led us to develop an essentially semantic algorithm – typically one that tries to build a model of the given sentence.

There are other reasons to move in the direction of incorporating semantics. If the agent is to be capable of processing information like a human, then we must question whether this can be done by transforming strings without paying any attention to what the strings mean, because humans do pay attention to meanings. There are famous arguments in AI pointing out the need to take semantics into account, for example the Chinese Room argument of John Searle and the Symbol Grounding Problem described by Stevan Harnad.

In humans, reasoning is guided and shaped as much by semantics (non-verbal images) as by language. Indeed, the way we understand abstract concepts is usually by analogy with some concrete perceptual experience stored as non-verbal images. A great deal of recent research on conceptual metaphor (see for example the many books by George Lakoff) and embodiment (see Ali Knott's book on 'Sensorimotor Cognition and Natural Language Syntax') explores this connection.

In summary then, it is an interesting fact about classical logic and the entailment relation $\models$ that one can use pattern-matching on the shapes of the symbols to design sound and complete syntactic reasoning algorithms that simulate $\models$, and even to get a decision procedure in some cases (such as classical propositional logic, or the restricted versions of first-order logic known as description logics).

This interesting fact used to be regarded by philosophers as tremendously important, and almost as if it were a defining characteristic of logic, until a few decades ago when some people began to realise that pattern-matching on symbol-shapes only works because $\models$ is so restrictive, i.e. allows so few inferences, namely only those forced to exist by the nature of the language and not by the nature of the agent's environment.

Our environments are continually changing, so that much of what we (or the agents we may design) learn from the environment is in the form not of facts (e.g. The ball is round) but of default rules (If a ball strikes you in the face then normally it hurts like a bugger).

We represent default rules semantically. The moment one wishes to formalise reasoning that relies on information which is represented semantically rather than being encoded in the symbols of the language, syntactic (pattern-matching) algorithms are too limited.

An important kind of semantically-oriented algorithm is *model-checking*. The 2007 Turing Award of the ACM was given to three computer scientists, Clarke, Sifakis, and Emerson, who developed model-checking algorithms involving sentences of temporal logic. Such algorithms have been applied mainly to show that hardware designs have various desired properties.

Exercises

Quiz: For the quiz in lecture 7, look back to exercise 2 in lecture 2 and the properties of rational consequence relations in the exercises of lecture 4. You will be given two properties and asked whether they hold for $\models$ and for $\mid\!\sim$.

Now, what about reasoning algorithms.

Every COSC410 exam has an essay question. Here are 3 essay questions you may work out in advance – I'd be happy to give feedback on your attempts so that you can tune them nicely. One of these essays is guaranteed to be in the exam.

1. Write a well-planned essay on the expressiveness of propositional languages.
2. Write a well-planned essay on proof by induction and its use to prove metatheorems about logic.
3. Write a well-planned essay on reasoning algorithms, giving examples of rules of inference and discussing the adequacy with which such algorithms simulate $\models$.