COSC462 Lecture 4
More Metatheory

Willem Labuschagne
University of Otago

Abstract

We look at some more nice properties of propositional languages and classical entailment, including *compactness*, and describe a very crude automated reasoning algorithm.

1 Model-theoretic properties

As in the previous lecture, we avoid complications by assuming that the ontology for every language L_A is (S, V) with $S = W_A$ and V the identity function on W_A.

The Ineffability Theorem showed us a way in which propositional languages L_A with infinite A are different from those with finite A. However, there are more similarities than differences. For example, regardless of the size of A, the set of models of a sentence α can be constructed from the models of the atoms in α by set-theoretic operations like union, intersection, and complementation.

Theorem 1 *For all $\varphi, \psi \in L_A$ it is the case that*

- $\mathcal{M}(\neg\varphi) = \overline{\mathcal{M}(\varphi)}$
- $\mathcal{M}(\varphi \wedge \psi) = \mathcal{M}(\varphi) \cap \mathcal{M}(\psi)$
- $\mathcal{M}(\varphi \vee \psi) = \mathcal{M}(\varphi) \cup \mathcal{M}(\psi)$

Proof. *To see that $\mathcal{M}(\neg\varphi) = \overline{\mathcal{M}(\varphi)}$ it suffices to note that s satisfies $\neg\varphi$ iff s fails to satisfy φ.*

To see that $\mathcal{M}(\varphi \wedge \psi) = \mathcal{M}(\varphi) \cap \mathcal{M}(\psi)$, it is sufficient to note that s satisfies $\varphi \wedge \psi$ iff s satisfies φ and s satisfies ψ.

To see that $\mathcal{M}(\varphi \vee \psi) = \mathcal{M}(\varphi) \cup \mathcal{M}(\psi)$, note that s satisfies $\varphi \vee \psi$ iff s satisfies φ or s satisfies ψ. ■

There is another way in which all propositional languages behave very similarly, regardless of whether A is finite or infinite. This is the *compactness* property, which is a kind of finiteness property having to do with satisfiability.

Definition 2 *(**Satisfiability**) A set $\Gamma \subseteq L_A$ of sentences is satisfiable if there is at least one state $s \in S$ that satisfies every $\gamma \in \Gamma$.*

So a set of sentences Γ is satisfiable if its set of models $\mathcal{M}(\Gamma)$ is nonempty. (In the case of a single sentence γ, γ is satisfiable if $\mathcal{M}(\gamma) \neq \varnothing$). A contradiction like $p_0 \wedge \neg p_0$ is an unsatisfiable sentence, while a set like $\Gamma = \{p_0, p_1, (p_1 \rightarrow \neg p_0)\}$ is an example of an unsatisfiable set of sentences (as you are invited to verify).

Theorem 3 *(**Compactness**) A set Γ of sentences is satisfiable iff every finite subset of Γ is satisfiable.*

Proof. *One direction of the proof is easy. If Γ is satisfiable, then there is a state s satisfying all the sentences in Γ, and so every finite subset of Γ is satisfied by s.*

The converse direction is more of a challenge. Suppose every finite subset of Γ is satisfiable. We shall build a bigger set Δ, such that $\Gamma \subseteq \Delta$, and show that Δ is satisfiable. The reason we build Δ instead of just working with Γ is that Δ is going to be big enough to help us define a way to allocate truth values to all sentences, in other words we use Δ to build a state (valuation) which turns out to satisfy the whole Γ.

First of all, let us imagine the sentences of L_A written down in order as $\alpha_1, \alpha_2, \ldots$ It's not hard to dream up an algorithm to do this — we might use a grammar for the language to generate longer and longer strings, and write them down in the order in which they are generated.

Now let us define a sequence of sets that gradually add sentences to Γ. Let $\Delta_0 = \Gamma$, and for every n let $\Delta_{n+1} = \Delta_n \cup \{\alpha_{n+1}\}$ if every finite subset of $\Delta_n \cup \{\alpha_{n+1}\}$ is satisfiable, or let $\Delta_{n+1} = \Delta_n \cup \{\neg\alpha_{n+1}\}$ otherwise.

There are a couple of things to notice. The construction of the sequence is something we can imagine being carried out, but we may not ourselves be able to do it in practice, because it not only goes on forever but it may be very hard work to decide whether every finite subset of $\Delta_n \cup \{\alpha_{n+1}\}$ is satisfiable. Nevertheless, if we had unlimited time we could use truth tables and do it. So it is possible in principle.

The second thing to notice is that every Δ_i has the property, by virtue of our construction, that all its finite subsets are satisfiable. The reason is this. The initial set Δ_0 has the property, since $\Delta_0 = \Gamma$. Suppose Δ_n has the property that all its finite subsets are satisfiable, but neither $\Delta_n \cup \{\alpha_{n+1}\}$ nor $\Delta_n \cup \{\neg\alpha_{n+1}\}$ has the property. Then there must be

finite subsets X and Y of Δ_n such that $X \cup \{\alpha_{n+1}\}$ and $Y \cup \{\neg\alpha_{n+1}\}$ are both unsatisfiable. But what about $X \cup Y$? Any valuation satisfying $X \cup Y$ must satisfy either α_{n+1} or $\neg\alpha_{n+1}$. So either we contradict the unsatisfiability of $X \cup \{\alpha_{n+1}\}$ and $Y \cup \{\neg\alpha_{n+1}\}$, or else the finite subset $X \cup Y$ of Δ_n is unsatisfiable, which is also a contradiction, since we assumed all finite subsets of Δ_n to be satisfiable.

Let's return to the construction. Take Δ to be the union of all the Δ_n, in other words take Δ to be the set of all sentences which belong to at least one of the Δ_n. This set Δ has three interesting properties. Obviously $\Gamma \subseteq \Delta$, since $\Gamma = \Delta_0$. Also, for every sentence $\beta \in L_A$, either $\beta \in \Delta$ or else $\neg\beta \in \Delta$. And finally, every finite subset of Δ is satisfiable. Why? Well, because that finite subset is, for some n, a finite subset of Δ_n and we know that the finite subsets of every Δ_n are all satisfiable.

Now consider the valuation v such that, for every $p \in A$, $v(p) = 1$ iff $p \in \Delta$. We claim that for each sentence $\beta \in L_A$, the state v satisfies β iff $\beta \in \Delta$.

The proof uses induction. The shortest sentences are the atoms, and if β is an atom then by construction v satisfies β iff $\beta \in \Delta$. Assume that this holds for all sentences shorter than k (the Induction Hypothesis). Let β be of length k. There are various cases:

If $\beta = \neg\varphi$ then φ is shorter than k and so we may argue that if v satisfies β then v does not satisfy φ, so $\varphi \notin \Delta$ by the Induction Hypothesis, and so $\beta \in \Delta$. Conversely, if $\beta \in \Delta$ then $\varphi \notin \Delta$, so by the Induction Hypothesis v does not satisfy φ, and so v satisfies β.

If $\beta = \varphi \wedge \mu$, then both φ and μ are shorter than k and so we may argue that if v satisfies β then v satisfies both φ and μ, so by the Induction Hypothesis $\varphi \in \Delta$ and $\mu \in \Delta$, and now we are left with two possibilities: either $\varphi \wedge \mu \in \Delta$ or $\varphi \wedge \mu \notin \Delta$. We can eliminate the latter, for if $\varphi \wedge \mu \notin \Delta$ then $\neg(\varphi \wedge \mu) \in \Delta$, and now the finite subset $\{\varphi, \mu, \neg(\varphi \wedge \mu)\}$ of Δ is unsatisfiable. So $\varphi \wedge \mu \in \Delta$. Conversely, suppose $\beta \in \Delta$. Now both $\varphi \in \Delta$ and $\mu \in \Delta$, for if not we again get a finite subset of Δ that is unsatisfiable. For example, if $\varphi \notin \Delta$, then $\neg\varphi \in \Delta$ and so $\{\neg\varphi, \varphi \wedge \mu\}$ is an unsatisfiable finite subset of Δ. But we know that the finite subsets of Δ are all satisfiable. And so v satisfies both φ and μ, whence v satisfies β.

The cases $\beta = \varphi \vee \mu$, $\beta = \varphi \rightarrow \mu$, and $\beta = \varphi \leftrightarrow \mu$ are similar and left for the exercises.

The induction now follows, so that for sentences of all lengths it is the case that they are satisfied by v iff they belong to Δ. But since v satisfies all the sentences in Δ, v certainly satisfies all the sentences in $\Gamma \subseteq \Delta$. So Γ is satisfiable. ■

The Compactness Theorem illustrates a technique that logicians of-

ten use to prove results about logic, namely the idea of a maximal satisfiable set Δ. Versions of the Compactness Theorem could be proved for different kinds of logic, but since our concern is with applied logic rather than logic as a part of mathematics, we shall not do so.

In order to fully appreciate the importance of the Compactness Theorem we shall pause to examine some of its consequences. Recall that $\Gamma \models \beta$ iff $\mathcal{M}(\Gamma) \subseteq \mathcal{M}(\beta)$.For example, $\{\alpha, \beta\} \models \alpha$ because every valuation satisfying α as well as β is of course a valuation satisfying α.

Corollary 4 *If $\Gamma \models \beta$ then there is some finite subset $\Gamma' \subseteq \Gamma$ such that $\Gamma' \models \beta$.*

Proof. *First we establish a connection between entailment and unsatisfiability, namely that $\Gamma \models \beta$ iff $\Gamma \cup \{\neg\beta\}$ is unsatisfiable.*

Suppose $\Gamma \models \beta$. Then every valuation satisfying all the sentences in Γ also satisfies β. So no valuation can satisfy all the sentences in Γ as well as satisfying $\neg\beta$. So the set $\Gamma \cup \{\neg\beta\}$ is unsatisfiable.

Conversely, if $\Gamma \cup \{\neg\beta\}$ is unsatisfiable, then every valuation satisfying all the sentences of Γ must also satisfy β, so that $\Gamma \models \beta$.

Now we use the connection between entailment and unsatisfiability.

Suppose $\Gamma \models \beta$.

Then $\Gamma \cup \{\neg\beta\}$ is unsatisfiable.

So $\Gamma' \cup \{\neg\beta\}$ is unsatisfiable for some finite $\Gamma' \subseteq \Gamma$ (otherwise by compactness $\Gamma \cup \{\neg\beta\}$ would have to be satisfiable).

So $\Gamma' \models \beta$ for some finite $\Gamma' \subseteq \Gamma$. ■

Isn't this remarkable? Use as many sentences as you like to build a set Γ. Look at the set of models of Γ. Pick any sentence β which is true in all those models. Then there is a finite subset of Γ, which is really just another way of saying that there is a *single sentence* α (since we could take the conjunction of all the sentences in the finite subset) such that $\alpha \models \beta$.

On the other hand, maybe it's not so remarkable. The set Γ expresses some information (has some nonmodels). If $\Gamma \models \beta$ then this just means β expresses some of the information in Γ. And β is a finitely long string. So we would expect to find some finite part of Γ that expresses all the information in β (and perhaps even more). So the corollary to the Compactness Theorem is a vindication of our intuition about information, not a surprise!

My treatment of the Compactness Theorem is loosely based on that in Enderton H: *A Mathematical Introduction to Logic* (2nd edition), Harcourt/Academic Press 2001, which I would describe as one of the better logic textbooks out there.

Exercise 5 *1. For all sets X and Y, we define the complement of Y relative to X to be the set $X - Y = \{x \in X \mid x \notin Y\}$. Thus for example $\overline{\mathcal{M}(\varphi)} = S - \mathcal{M}(\varphi)$.*

Describe how to construct $\mathcal{M}(\varphi \to \psi)$ and $\mathcal{M}(\varphi \leftrightarrow \psi)$ from S, $\mathcal{M}(\varphi)$, and $\mathcal{M}(\psi)$. You may use diagrams if you wish.

2. Finish the proof of the Compactness Theorem by completing the remaining cases of the inductive argument to show that $v \in \mathcal{M}(\beta)$ iff $\beta \in \Delta$.

3. To get a feel for the construction of Δ used in the Compactness Theorem, try the following.

Consider the language L_A with $A = \{p, q\}$, and let us restrict our attention to the following 16 non-equivalent sentences, which we give in order of decreasing number of models:

$p \vee \neg p$

$p \vee q$, $q \to p$, $p \to q$, $\neg p \vee \neg q$

p, q, $p \leftrightarrow q$, $\neg(p \leftrightarrow q)$, $\neg p$, $\neg q$

$p \wedge q$, $p \wedge \neg q$, $\neg p \wedge q$, $\neg p \wedge \neg q$

$p \wedge \neg p$.

Take $\Gamma = \{p\}$. Construct a maximal satisfiable set Δ such that $\Gamma \subseteq \Delta$.

(Hint: It's easiest if you are systematic. Write down, for each of the 16 sentences, its set of models from $\{11, 10, 01, 00\}$. Go through the 16 sentences in turn, building sets $\Delta_0, \Delta_1, \ldots$. At every stage, check for satisfiability by keeping track of the models of Δ_n and checking that $\mathcal{M}(\Delta_n) \cap \mathcal{M}(\alpha_{n+1}) \neq \varnothing$.)

Do you think it would be possible to build a different maximal satisfiable set Δ' such that $\{p\} \subseteq \Delta'$ but $\Delta \neq \Delta'$? Justify your answer.

2 Reasoning algorithms

Suppose we have a set Γ of sentences. Call this our database. Now consider the problem of writing a program that can be applied to our database in order to generate sentences β such that $\Gamma \vDash \beta$.

Such a program would be an automated reasoner. Since we are not insanely ambitious, we won't insist that the program be an intelligent agent able to do everything a human can. All the program has to be able to do is to transform strings in Γ into strings β entailed by Γ, without understanding what the strings mean. In other words, we are satisfied

with an algorithm that works syntactically, not semantically, because it is easier to tell the algorithm what to do according to the shapes of symbols than according to their meanings.

Later, we shall be taking a more in-depth look at automated reasoning. For the present, we just try to convey the general idea by looking at a very crude approach. The transformations done to strings by automated reasoners are called *inferences*.

The most ancient and well-known inference rule is called *Modus Ponens*, which is Latin for 'the method of bridging the gap', and it works like this: given copies of two strings α and $\alpha \rightarrow \beta$ as input, the automated reasoner bites off, and spits out, the string β. So we may visualise an automated reasoner equipped with *Modus Ponens* being applied to a database Γ. The reasoner would look inside Γ for a pair of strings of the form α and $\alpha \rightarrow \beta$, and as soon as it finds such a pair the reasoner would add β to the database. Then the reasoner does the same to the new database. Thus the database grows step by step.

There is a second very natural inference rule, which we shall call *And*. It works like this: given copies of two strings α and β as input, the automated reasoner squashes them together and spits out the conjunction $\alpha \wedge \beta$. When the reasoner is applied to a database Γ, it looks for any two sentences α and β inside the database and adds $\alpha \wedge \beta$ to the database. In practice we would want to control the application of this inference rule, because it can consume the reasoner and turn it into a monomaniac that churns out conjunctions without stopping. We will see later what sort of strategies one might use to control the application of inference rules (in lectures 16-20).

2.1 The fabric example

We now show that our simple reasoning algorithm can be useful. We shall apply the algorithm to an example taken from Maier and Warren (1988): *Computing with Logic*, Addison-Wesley, pp 8-16.

The database Γ consists of information that can be used to classify different kinds of cloth.

In woven fabric, two sets of threads are interlaced at right angles. *Warp* threads run the length of the piece of fabric. During the weaving process, the warps are raised or lowered in different patterns and *fill* threads are passed back and forth between them. The three basic groups of weaves are *plain* weaves, *twill* weaves, and *satin* weaves.

In a plain weave, the warp threads cross over and under successive fill threads, with adjacent warp threads going in alternate directions around the fill threads, so that the warp pattern repeats every two threads. (See figure 1.)

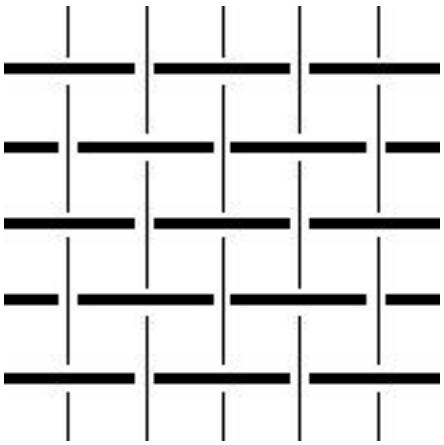

Figure 1: Plain weave with warps left to right

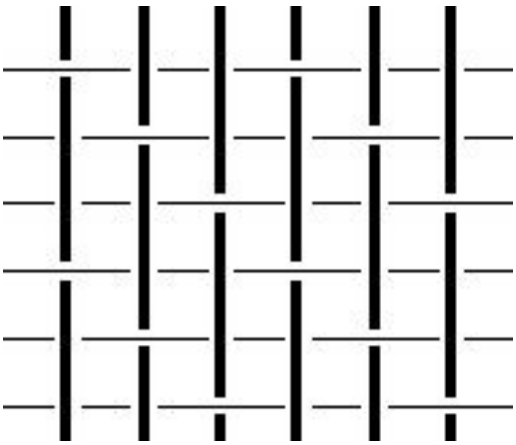

Figure 2: Twill weave with warps up and down

In a twill weave, a warp thread passes over several fill threads called *floats*, and passes under one or possibly more fill threads called *sinks*. Adjacent warp threads are offset by one fill thread so that the warp pattern gives the twill fabric a diagonal texture. (See figure 2.)

Satin weave is characterised by long floats or sinks, usually involving four or more fill threads, which gives the satin fabric a lustrous look. Adjacent warp threads in a satin fabric are offset by more than one fill thread, to avoid giving a diagonal texture.

The following sentences in Γ summarise the properties we can use to classify a fabric into one of the basic categories of weaves. For readability, we do not use atoms like p_0 or p_1 but strings that remind us of English.

- `alternatingWarp` → `plainWeave`
- `diagonalTexture` → `twillWeave`
- (`hasFloats` ∧ `warpOffset=1`) → `twillWeave`
- (`hasFloats` ∧ `warpOffset>1`) → `satinWeave`

Different fabrics with plain weaves can be distinguished by the colours of the threads, the spacing of the threads, the texture, and the fibre (the substance from which the threads are made). A *balanced* fabric has as many warp threads per inch as fill threads per inch. A *sheer* fabric has fine threads spaced far enough apart to let light through. *Percale* is cotton fabric with a balanced weave and a smooth texture, *organdy* is a sheer cotton fabric, and *organza* is a similar fabric of silk. All these fabrics have threads of a single colour, so may be grouped together in a larger category of *solid* plain weaves.

- (`plainWeave` ∧ `oneColour`) → `solidPlain`
- (`solidPlain` ∧ `cotton` ∧ `balanced` ∧ `smooth`) → `percale`
- (`solidPlain` ∧ `cotton` ∧ `sheer`) → `organdy`
- (`solidPlain` ∧ `silk` ∧ `sheer`) → `organza`

To get a pattern into a fabric one must have groups of warp or fill threads of different colours. *Plaids* have groups of different coloured threads in both warp and fill. *Gingham* is a cotton plaid in which the different groups have the same width.

- (`plainWeave` ∧ `colourGroups`) → `patternPlain`
- (`patternPlain` ∧ `warpStripe` ∧ `fillStripe`) → `plaid`
- (`plaid` ∧ `equalStripe`) → `gingham`

Basketweave is a variation on plain weave in which groups of two or more warp threads function as a unit. One kind of basketweave is *oxford cloth*, in which warp threads are grouped in twos while fill threads are single (called type 2/1). Also, in oxford cloth the fill threads are thicker than the warp threads. For *monk's cloth*, the groups of warp threads and fill threads are the same size, and groups usually consist either of two threads or four (giving types 2/2 and 4/4). In monk's cloth the warp and fill threads have the same thickness. *Hopsacking* has a rough texture and an *open* weave, i.e. adjacent threads do not touch.

- (plainWeave ∧ groupedWarps) → basketWeave
- (basketWeave ∧ type2To1 ∧ thickerFill) → oxford
- (basketWeave ∧ type2To2 ∧ sameThickness) → monksCloth
- (basketWeave ∧ type4To4 ∧ sameThickness) → monksCloth
- (basketWeave ∧ rough ∧ open) → hopSacking

Ribbed weave fabrics are a variation on plain weave with some threads thicker than others. There are several ribbed fabrics in which the fill threads are thicker than the warp threads, and the varieties are distinguished by the size and shape of the ribs — *faille* has small flat ribs, while *grosgrain*, *bengaline*, and *ottoman* have rounded ribs that are small, medium, and heavy respectively.

- (plainWeave ∧ someThicker) → ribbedWeave
- (ribbedWeave ∧ thickerFill) → crossRibbed
- (crossRibbed ∧ smallRib ∧ flatRib) → faille
- (crossRibbed ∧ smallRib ∧ roundRib) → grosgrain
- (crossRibbed ∧ mediumRib ∧ roundRib) → bengaline
- (crossRib ∧ heavyRib ∧ roundRib) → ottoman

Napped fabrics are finished by brushing with wire brushes to give a very soft texture. *Flannel* is the most common napped fabric, and may be either plain or twill weave in either cotton or wool.

- (plainWeave ∧ cotton ∧ napped) → flannel
- (twillWeave ∧ cotton ∧ napped) → flannel
- (plainWeave ∧ wool ∧ napped) → flannel
- (twillWeave ∧ wool ∧ napped) → flannel

In *leno* weave, a special attachment to the loom, called a doup, crosses and uncrosses pairs of warp threads between fill threads. *Marquisette* is a fabric with an open leno weave.

- (plainWeave ∧ crossedWarps) → lenoWeave
- (lenoWeave ∧ open) → marquisette

Pile fabrics have an extra set of loosely woven threads that produce loops on one or both sides of the fabric, and may be categorised as *fill pile* or *warp pile* depending on whether the extra threads are parallel to the fill threads or warp threads. *Velvet* has a warp pile with cut loops, wehreas *terry cloth* is a fill pile fabric with uncut loops on both sides. *Corduroy* and *velveteen* are also fill pile fabrics, but have the loops cut like velvet. In corduroy the loops are aligned to give ridges, but in velveteen the loops are staggered to give a solid effect.

- (plainWeave $\wedge$ extraFill) $\rightarrow$ fillPile
- (plainWeave $\wedge$ extraWarp) $\rightarrow$ warpPile
- (warpPile $\wedge$ cut) $\rightarrow$ velvet
- (fillPile $\wedge$ uncut $\wedge$ reversible) $\rightarrow$ terry
- (fillPile $\wedge$ cut $\wedge$ alignedPile) $\rightarrow$ corduroy
- (fillPile $\wedge$ cut $\wedge$ staggeredPile) $\rightarrow$ velveteen

Twills vary according to the relative lengths of the floats and sinks. If the lengths are the same, the fabric is an *even* twill. If not, the fabric is *faced*, with *filling-faced* having longer sinks and *warp-faced* having longer floats. *Drill* and *denim* are warp-faced twills, but denim has white fill threads whereas drill has fill threads of the same colour as the warp threads. *Serge* is an even twill with a heavy rib.

- (twillWeave $\wedge$ float=sink) $\rightarrow$ evenTwill
- (twillWeave $\wedge$ float$<$sink) $\rightarrow$ fillingFaced
- (twillWeave $\wedge$ float$>$sink) $\rightarrow$ warpFaced
- (warpFaced $\wedge$ colouredWarp $\wedge$ whiteFill) $\rightarrow$ denim
- (warpFaced $\wedge$ oneColour) $\rightarrow$ drill
- (evenTwill $\wedge$ heavyRib) $\rightarrow$ serge

For satin weaves, we get *satin* if the floats are in the warp and *sateen* if they are in the fill. Both have a smooth finish. *Moleskin* is a napped satin weave cotton fabric.

- (satinWeave $\wedge$ warpFloats $\wedge$ smooth) $\rightarrow$ satin
- (satinWeave $\wedge$ fillFloats $\wedge$ smooth) $\rightarrow$ sateen

- `(satinWeave ∧ cotton ∧ napped) → moleskin`

Now imagine that the agent wants to find out what she is wearing, and looks down at her clothes. She notices that there is a diagonal pattern to the fabric, that the warp threads dominate on the visible side of the fabric, and that the warp threads are blue while the fill threads are white. Thus she adds to her database the following atoms:

- `diagonalTexture`
- `float>sink`
- `colouredWarp`
- `whiteFill`

Next the agent moves from perception to reasoning. Applying *Modus Ponens* to the obvious atom and the sentence

`diagonalTexture → twillWeave`

she infers

`twillWeave.`

Applying *And* she infers

`twillWeave ∧ float>sink`

so that *Modus Ponens* and

`(twillWeave ∧ float>sink) → warpFaced`

deliver

`warpFaced.`

From the obvious atoms, *And* infers (in two steps)

`warpFaced ∧ colouredWarp ∧ whiteFill`

so that *Modus Ponens* can use

`(warpFaced ∧ colouredWarp ∧ whiteFill) → denim`

to infer that she is wearing

`denim`.

2.2 Properties of reasoning algorithms

First let us give a name to a sequence of inferences from the database.

Definition 6 *A baby deduction of β from Γ is a finite sequence of sentences $\langle \alpha_0, \ldots, \alpha_n \rangle$ such that $\alpha_n = \beta$ and for every $k \leq n$:*

- $\alpha_k \in \Gamma$ *or*
- *there exist $i, j < k$ with $\alpha_j = (\alpha_i \rightarrow \alpha_k)$*
- *or there exist $i, j < k$ with $\alpha_k = (\alpha_i \wedge \alpha_j)$.*

To check that $\langle p_0, (p_0 \rightarrow p_1), p_1 \rangle$ is a baby deduction of $\beta = p_1$ from the set $\Gamma = \{p_0, p_0 \rightarrow p_1\}$ is entirely mechanical. By inspection, p_0 is in the set Γ, as is $(p_0 \rightarrow p_1)$. By virtue of the fact that the sequence has earlier members $\alpha_i = p_0$ and $\alpha_j = (p_0 \rightarrow p_1)$, it now follows that $\alpha_k = p_1$ is entitled to its place in the sequence also.

Similarly, the reasoning performed in the fabric example corresponds to the deduction $\langle \alpha_0, \ldots, \alpha_{12} \rangle$ where

$\alpha_0 =$ `diagonalTexture`
$\alpha_1 =$ `diagonalTexture` $\rightarrow$ `twillWeave`
$\alpha_2 =$ `twillWeave`
$\alpha_3 =$ `float>sink`
$\alpha_4 =$ `twillWeave` $\wedge$ `float>sink`
$\alpha_5 =$ `(twillWeave` $\wedge$ `float>sink)` $\rightarrow$ `warpFaced`
$\alpha_6 =$ `warpFaced`
$\alpha_7 =$ `colouredWarp`
$\alpha_8 =$ `warpFaced` $\wedge$ `colouredWarp`
$\alpha_9 =$ `whiteFill`
$\alpha_{10} =$ `warpFaced` $\wedge$ `colouredWarp` $\wedge$ `whiteFill`
$\alpha_{11} =$ `(warpFaced` $\wedge$ `colouredWarp` $\wedge$ `whiteFill)` $\rightarrow$ `denim`
$\alpha_{12} =$ `denim.`

Definition 7 *Write $\Gamma \vdash \beta$ to say that there exists a baby deduction of β from Γ.*

One way to think of it is that our automated reasoner can output the metalanguage string $\Gamma \vdash \beta$ in order to tell us that it has added the object-level string β to the database Γ.

How do we judge whether our automated reasoner is a good one? There are various questions we could ask.

Is the automated reasoner *sound*? A sound reasoner would add β to the database Γ only if $\Gamma \models \beta$. So of course an unsound reasoner is one that may occasionally add to the database Γ some sentence γ for which it is not the case that Γ entails γ. Since the aim of an automated reasoner is to mimic the entailment relation $\models$, we would be reluctant to use an unsound reasoner, and would tend to do so only if the reasoner had other properties that were overwhelmingly important for the context of use.

Our crude baby reasoner is sound.

Theorem 8 *(Soundness) If $\Gamma \vdash \beta$ then $\Gamma \models \beta$.*

Proof. *Suppose $\Gamma \vdash \beta$. Thus there exists a baby deduction $\langle \alpha_0, \ldots, \alpha_n \rangle$ of β from Γ.*

Now we use induction on the length of the deduction to show that $\Gamma \vDash \beta$.

A deduction of length 1 *consists of the single sentence* β*, and it must be the case that* $\beta \in \Gamma$*, whence it follows that* $\Gamma \vDash \beta$.

Suppose the result holds for all baby deductions of length $\leq k$*. Consider any baby deduction of* β *from* Γ *having length* $k+1$*, say* $\langle \alpha_0, \ldots, \alpha_k \rangle$.

There are three possibilities for $\alpha_k = \beta$.

If β *is a member of* Γ*, then* $\Gamma \vDash \beta$.

So consider the second case, in which β *belongs by Modus Ponens, i.e. there are* $i, j < k$ *with* $\alpha_j = (\alpha_i \rightarrow \alpha_k)$.

The sequence up to α_i *is a deduction of* α_i *and is shorter than* $k+1$*, so by the induction hypothesis* $\Gamma \vDash \alpha_i$.

Similarly the sequence up to α_j *is a deduction of* α_j *and is shorter than* $k+1$*, so by the induction hypothesis* $\Gamma \vDash \alpha_i \rightarrow \alpha_k$.

But if a model v *of* Γ *satisfies* α_i *and also satisfies* $\alpha_i \rightarrow \alpha_k$*, then* v *has to satisfy* α_k.

Thus $\Gamma \vDash \alpha_k$.

The third case, in which β *belongs by And, is similar.* ■

There is a second question we might ask about our automated reasoner.

Is the reasoner *complete*? A reasoner is complete if its deductions go far enough. In other words, completeness is the property:

If $\Gamma \vDash \beta$ then $\Gamma \vdash \beta$.

Our baby reasoner is not complete, sadly. Its inference rules can only transform input strings provided one of them has the syntactic form α and another has the form $\alpha \rightarrow \beta$, or else in order to produce a string of the form $\alpha \wedge \beta$. Suppose we start with $\Gamma = \{p\}$. We know that $\Gamma \vDash \neg\neg p$. However, no baby deduction starting from this Γ can ever produce the string $\neg\neg p$. Thus we have a counter-example to completeness.

Although we would generally like our automated reasoner to be complete, we may accept an incomplete reasoner if it has some compensating advantage such as being very efficient. For example, Prolog is based on a reasoning algorithm called SLDNF-resolution which is incomplete relative to the particular form of entailment used in that context, but SLDNF-resolution has the advantage that in its limited context of use it is efficient. We'll say more about this later.

We see that a sound and complete reasoning algorithm is one which will output $\Gamma \vdash \beta$ if, and only if, $\Gamma \vDash \beta$. We can think of such a reasoner as a question-answering device. We may ask it whether $\Gamma \vDash \beta$, and if it is indeed the case that $\Gamma \vDash \beta$ then our sound and complete reasoner will confirm this by outputting $\Gamma \vdash \beta$ or words to that effect.

This is, however, not all we might aspire to. Suppose we ask whether $\Gamma \models \beta$ when it is in fact the case that $\Gamma \not\models \beta$. Then our sound and complete reasoner will not try to fool us by answering 'yes' (i.e. by outputting $\Gamma \vdash \beta$), which is good, but on the other hand our sound and complete reasoner doesn't have to tell us 'no' either (i.e. doesn't have to output that $\Gamma \not\vdash \beta$). Won't we be able to tell that $\Gamma \not\vdash \beta$ by having the reasoning program stop without giving us the confirmation $\Gamma \vdash \beta$? No, because quite possibly the reasoning algorithm won't terminate at all. And what's more, we won't know that it's not going to terminate.

To illustrate this situation, consider the database $\Gamma = \{p_0\} \cup \{p_0 \rightarrow p_1, p_1 \rightarrow p_2, \ldots\}$. Take $\beta = p_{113}$. Our crude reasoner will perform a sequence of inferences until it has constructed a deduction sequence terminating with p_{113}, whereupon the reasoner will halt and output $\Gamma \vdash \beta$. But now suppose that we take $\Gamma' = \{p_0 \rightarrow p_1, p_1 \rightarrow p_2, \ldots\}$. The reasoner will search through Γ forever, because it will never find a pair of sentences in Γ that will allow it to make an MP inference. The problem is that while the reasoner is chugging away, we won't know whether it is getting closer and closer to finding what it needs or whether the search is hopeless. So we would be very interested in upgrading our reasoner to a *decision procedure*.

A decision procedure for a language L_A is an algorithm that, given any subset Γ and any sentence β, will terminate after a finite number of steps and answer either 'yes' or 'no' to our question 'Does $\Gamma \models \beta$?'. Sometimes we are able to devise an automated reasoner that is a decision procedure, sometimes not. For example, if we knew that Γ will be finite, we could use a decision procedure that involves examining truth tables. However, even if a decision procedure exists, it may not be useful. A truth table for a set Γ and sentence β in which a total of n atoms occur will need to have 2^n rows, and so a truth table algorithm is exponential, which is not efficient enough for realistic applications. In fact, the decision problem for propositional logic is an NP-complete problem (which will mean something to those of you who did complexity theory in COSC341). One way to see that the problem is NP-complete is to recall that the most famous of all NP-complete problems is the Satisfiablity Problem: given a sentence α, is α satisfiable? As we saw earlier in this lecture, $\Gamma \models \beta$ iff $\Gamma \cup \{\neg\beta\}$ is unsatisfiable, and so the question 'Does $\Gamma \models \beta$?' can be rephrased as the Satisfiability Problem.

3 Glossary

- **compactness** — a property that may be possessed by an object language, and which represents a sort of finiteness; formally, a compact language is one in which, for any set Γ of sentences, we can

check whether Γ is satisfiable by examining just the finite subsets of Γ.

- **completeness** — the property possessed by a reasoning algorithm which, for all subsets Γ of a language L_A and all sentences $\beta \in L_A$, will confirm that $\Gamma \models \beta$ by halting with suitable output (e.g. $\Gamma \vdash \beta$). If it is not the case that $\Gamma \models \beta$, the the reasoner may fail to halt.

- **decision procedure** — a reasoning algorithm which is not only sound and complete but is also guaranteed to terminate when $\Gamma \not\models \beta$.

- **soundness** — the property possessed by a reasoning algorithm which will halt with output $\Gamma \vdash \beta$ only in cases where indeed $\Gamma \models \beta$.