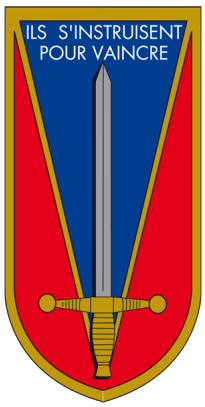

ECOLE SPÉCIALE MILITAIRE DE SAINT-CYR
PROMOTION CES DE NEUCHÈZE - Années 2014-2017

Scuola di Applicazione dell'Esercito / Università degli Studi di Torino - Dipartimento Informatica

Implementation of a theorem-prover for Lewis conditional logics

Author:
SLT VITALIS Quentin

Supervisor:
M. POZZATO Gian-Luca
Tutor:
M. BUISSON Jérémy

11th September 2016 - 2nd December 2016

FICHE DE SYNTHESE

Sujet du mémoire : Implémentation d'un prouveur pour la logique des conditionnels de Lewis **N° Projet :**
Auteur : Sous-Lieutenant Vitalis Quentin

Organisme d'acceuil : Scuola di Applicazione dell'Esercito / Università di Torino - Dippartimento Informatica
Directeur de projet : Mr. Pozzato **Tuteur :** Mr. Buisson

Jury : Informatique

Date de soutenance : Mardi 3 Janvier 2017

Mots clés : logique des conditionnels, programmation déclarative, Prolog.

Etude :

PRESENTATION :

L'informatique a permis l'automatisation de différents processus et l'accomplissements de tâches considérées impossible pour l'homme. C'est également le cas en mathématiques: l'informatique a permis de prouver des théorèmes que personne n'a jamais pu prouver, comme le théorème des quatre couleurs. Parallèlement à cela, les mathématiques, et plus particulièrement la logique, a une part importante dans le développement de l'informatique. En effet, la logique des conditionnels, modélisation mathématiques des raisonnements logiques, est à la base de l'intelligence artificielle.
Notre but ici est de développer un programme permettant de prouver automatiquement des théorèmes pour la logique des contionnels de Lewis, une des plus récentes, et une des plus proches des raisonnements logiques formés dans le langage commun.

DEMARCHE :

Dans un premier temps, nous nous intéressons à l'évolution de la logique des conditionnels à travers l'histoire pour essayer de comprendre ce qui a ammené Lewis à ce modèle et pour essayer de comprendre le modèle en lui-même. Puis nous étudions le langage Prolog qui va nous servir à programmer notre logiciel. Enfin, nous nous attaquons à l'implémentation en elle-même.
Nous avons commencé par rédiger les règles et axiomes qui constituent la logique de Lewis. Puis nous avons étudié les boucles créées par le logiciel au cas par cas, en essayant de démontrer des formules simples, dans un premier temps, puis moins simples par la suite.

LIMITES :

Cependant, ce programme possède quelques limites : premièrement, le fait que ce programme n'est pas "automatisé" dans le sens où il faut connaître le fonctionnement de Swipl pour l'exécuter. Il faudrait compléter l'implémentation par un script permettant l'exécution du programme via un formulaire.
La seconde limite est que nous avons créé ce programme en se focalisant sur son bon fonctionnement, par conséquent, il pourrait être amélioré pour être encore plus efficient.

CONCLUSION :

A travers ce logiciel nous permettons l'analyse de la logique des conditionnels de Lewis, et espérons avoir fait progresser, à notre échelle, le domaine de l'intelligence artificielle. C'est néanmoins dans tous les cas le premier prouveur de théorème pour la logique des conditionnels des Lewis.
Ce programme peut cependant être encore amélioré, notamment par l'implémentation d'un script (par exemple en PHP), qui permettrait de créer une interface sous forme de formulaire, et rendrait l'utilisation du programme plus simple.

ABSTRACT

Thesis subject : Implementation of a theorem-prover for Lewis conditional logics **Project N° :**
Author : Second Lieutenant Vitalis Quentin

Organism : Scuola di Applicazione dell'Esercito / Università di Torino - Dippartimento Informatica

Project director : Mr. Pozzato **Tutor :** Mr. Buisson

Jury : Computer Science

Presentation date : Tuesday 3rd January 2017

Key words : conditional logics, declarative programming, Prolog.

Thesis :

PRESENTATION :

Computer sciences allowed the automatization of many process aswell as the accomplishment of considered impossible untill then. It is also the case in mathematics: computer sciences made it possible to prove theorem no-one could, as the Four-Color theorem for instance. In parallel of this, mathematics, and more particularly the logics, had an important part in the development of computer sciences. Indeed, the conditional logics, mathematical modelisation of logical reasonings, is at the heart of artificial intelligence.
Our goal here is to develop a program which will allow us to prove theorems automatically in the particular case of Lewis conditional logics, one of the most recent, and one of the closest to the reasonings we express in natural language.

APPROACH :

First, we studied the evolution of conditional logics through history in order to try to understand what brought Lewis to this model, and to try to understand the model itself. Then we studied the Prolog language which we will use for our implementation. Finally, we get interested in the implementation itself.
We started writing the rules and axioms of Lewis logics. Then we studied the infinite loops created by the program, through some attempts to prove some simple formulae, at the eginning, and more difficult ones afterward.

LIMITS :

Nevertheless, this program has a few limits: first, the fact that this program is not entirely "automatized". Indeed, the user has to know the way of functionning of Swipl to execute it. It should be complemented by a script allowing its execution through a form.
The second limit is that we implemented this program focusing on its well functionning, therefore, it could be improved to be a bit more efficient than it already is.

CONCLUSION :

Through this program we allow the analysis of the Lewis conditional logics, and hope to have make a progress, at our scale, in the artificial intelligence field. It is however the first theorem-prover for the Lewis conditional logics.
Yet, this programm can be improved, especially by the implementation of a script (for instance in PHP), that would allow to create an interface, a form, and would make its utilization a lot easier for everyone.

Acknowledgments

I would like to thank M. Pozzato for always guiding me in the right direction through this thesis, and M. Buisson for all his always pertinent advices and corrections. I would also like to thank the Magg. Formichetti and the Magg. Campanale aswell as all the staff of the Scuola di Applicazione dell'Esercito for giving me the opportunity to work in the best conditions possible.

Notification

"This report is the result of a Cadet Officer's work.
On the occasion of filing and possible publication, the Saint-Cyr Coëtquidan Schools attract your attention on the fact that this report's version is not a proofread version. Thus this report may contain spelling or syntax mistakes as well as imprecision."

Table of contents

Introduction

Thanks to computer science, we made great improvements in the mathematical field. Many demonstrations were made possible only thanks to processors, as the proof of the Four Color theorem [19, 9] for which no "non-computer involving" proofs were found up to now.

On the other hand, logics, demonstrations and proofs are an essential part of computer science and define a type of programing in itself. That led to the creation of many programming languages such as Coq or Prolog, but it also gave birth to high-level programming languages used nowadays in development, as C, C++,..

In fact, the development of any program can be divided in two parts:

- What the program will have to do. In this step you will need to *declare* the rules it will follow, and the facts it has to care about
- How the program will do it. We *impose* step by step the execution

We just described the two main paradigms of programming: *declarative programming* and *imperative programming*. Declarative programming, the paradigm in which we operate, is therefore crucial in the creation of any computer program.

Moreover, the recent researches have placed logics and particularly conditional logics in the center of the Artificial Intelligence field, giving a new interest to logic programming.

We can consider that the goal of A.I. is to create a human-like reasoning through a machine, and conditional logics are precisely the translation of the logical reasoning. It was always studied and improved to match human reasoning in the common language. Besides, the study of conditionals

is closely linked to the evolution of logic through history.
It is an entire part of the semantic analysis in language studies and it highlights the problem of ambiguity in the common language and, therefore, the problem it represents for artificial intelligence.

We will however be interested through this paper in a particular aspect of conditional logics, the Lewis' approach. His formalization of conditional logics was used to model hypothetical reasoning, beliefs and more particularly conditional beliefs, something impossible with the previous conditional operators we used such as the material conditional. For our computation, we will be interested in the internal calculi of Lewis logics. We will try to implement a theorem prover for these logics, using the logic programming language *Prolog*, in order to make proof trees of assessments and logical formulae, and thus, interest ourselves in the reasoning in itself.

First, we will present the evolution of conditional logics (Chapter 1, p.12), in order to have a better understanding on what Lewis logics consist in, how did he get to create this logic, and why it is interesting for us. We will then introduce the language and the software we used to make this implementation (Chapter 2, p.31), and finally, we will discuss the implementation and its outcomes (Chapter 3, p.39).

Chapter 1

Conditional logics

1.1 Introduction to conditional logics

A conditional sentence is a sentence of the form "if A, then B", in which we can identify an if-clause A called the *antecedent* and a then-clause B, the *consequent*. This kind of sentences, expressing hypothetical judgments, and the conditional form in itself, is regularly used in the common language, even if it does not contain every time the if-then typology:

1. Bring me to the party and we'll have fun. → If you bring me to the party, then we'll have fun.

2. No Hitler, no A-bomb. → If there had been no Hitler, there would have been no A-bomb.[1]

Typologically we can distinguish at least two different kinds of conditional sentences, the *indicative* conditionals and the *counterfactual* ones. The difference between the two typologies can be expressed through the examples above, the first represents the indicative conditionals, and the second represent the counterfactual type. Besides, the study of this topic has deep roots, starting with the Stoic school [18], and many mathematicians proposed their own analysis of conditional sentences.

One of the first of these analysis is the **material conditional**, also called material implication, is provided by Philo the Dialectician[2], and

[1] D. Lewis, 1973.

[2] Greek dialectic philosopher of the Megarian school, fl. 300 BCE.

was brought up to date for the modern logic by Frege [5] and Russel [17]. The material conditional is a logical connective often symbolized as $\rightarrow$, translated in English by "if..then.. ". Yet, in fact the material conditional has little to do with this translation. Indeed, the statement "P $\rightarrow$ Q "does not imply any causal relationship between P and Q. A more accurate translation would have been "if P is true, then Q is also true", which would mean that it is false when P is true but Q is false. But if P is true and so does the conditional, then Q is true. In fact, the conditional does not imply any causal relationship between the antecedent and the consequent. Philo the Dialectician precises: "the conditional is true when it does not start with the true to end with the false; therefore, there are for this conditional three ways of being true, and one of being false". [4] Which can be translated in the truth table below:

P	**Q**	**P $\rightarrow$ Q**
T	**T**	T
T	**F**	F
F	**T**	T
F	**F**	T

Table 1.1: Truth table for material conditional

Thus, the material conditional is logically equivalent to $\neg(P \wedge \neg Q)$, which can be interpreted in classical logic as $\neg P \vee Q$ thanks to the De Morgan's Laws.

Nevertheless, some material conditional properties were seen as problematic when expressed in natural language. Clarence Irving Lewis[3], known as one of the founders of modal logic, noticed in 1912 what is acknowledged as the "paradoxes of material implication": following the material conditional analysis, any material conditional with a true consequent will be true, and in the same way, any false sentence can be entailed by any other and be proven true. That is, one can be confronted with these kinds of statements, considered as true in this analysis:

- If vampires exist then Paris is in France. (true consequent)

[3]C.I. Lewis, 1883 - 1964

- If 2 is odd then 2 is even. (false antecedent)

We need to introduce the model symbol $\models$ to transcript these two problematical properties. The formula $\Gamma \models \delta$ means that every model that makes Γ true makes δ true.
We then can write, with A the antecedent and C the consequent:

$$\neg A \models A \rightarrow C \text{ (false antecedent)}$$

$$C \models (A \rightarrow C) \text{ (truth consequent)}$$

Therefore, trying to find a conditional that can be more accurately expressed in natural language, C. I. Lewis introduced the **strict conditional**. Unlike the material conditional, the strict conditional depends on a modal operator, the *necessity*, from modal logic. The strict implication for two propositions P and Q is synthesized by $\Box(P \rightarrow Q)$, and means, in every world where P is true, Q is true.
It seemed to have solved the problem of the false antecedent and the true consequent. Nevertheless, we can show that some of the material conditional properties can be applied to the strict conditional:

- Transitivity:
$$(P \rightarrow Q), (Q \rightarrow R) \models (P \rightarrow R)$$
$$\Box(P \rightarrow Q), \Box(Q \rightarrow R) \models \Box(P \rightarrow R)$$
- Monotony:
$$(P \rightarrow R) \models (P \wedge Q) \rightarrow R$$
$$\Box(P \rightarrow R) \models \Box((P \wedge Q) \rightarrow R)$$
- Contraposition:
$$(P \rightarrow Q) \models (\neg Q \rightarrow \neg P)$$
$$\Box(P \rightarrow Q) \models \Box(\neg Q \rightarrow \neg P)$$

Yet, having these similar properties implies that the strict conditional would reproduce some of the "paradoxes" of the material implication. In fact, all of the properties above will be proven problematic when expressed in any natural language [2]:

- Transitivity
  - If J. Edgar Hoover were today a communist, then he would be a traitor. If J. Edgar Hoover had been born a Russian, then he would today be a communist.
  - So, if J. Edgar Hoover had been born a Russian, he would be a traitor. [Stalnaker, 1968]
- Monotony (also known as antecedent strengthening)
  - If I had struck this match, it would have lit.
  - If I had struck this match and done so in a room without oxygen, it would have lit. [Goodman, 1947]
- Contraposition
  - If the US had halted the bombing, North Vietnam would not have agreed to negotiate.
  - If North Vietnam had agreed to negotiate, the US would not had halted the bombing. [Stalnaker, 1968]

Thus, many logicians agreed, stating that strict conditionals still cannot describe the reality adequately.

Afterward, much of the last century work on this topic can be related to a footnote written by Ramsey in 1929 in his essay *On a problem in formal logic* [15, 16], that will be later acknowledged as the **Ramsey Test**:

Ramsey Test. *If two people are arguing "If A will C ?" and are both in doubt as to A, they are adding A hypothetically to their stock of knowledge and arguing on that basis about C... We can say they are fixing their degrees of belief in C given A.*

This footnote had been interpreted in many ways and led to a wave of different analysis from the years 1940s to the early 1960s, referred to as **cotenability theories of conditionals** [1]. The idea at the base of these theories is that a conditional $\Phi > \Psi$[4] is true if Φ, together with a suitable set of laws, entails Ψ.

[4]Here, we arbitrary use the symbol $>$ instead of $\rightarrow$ to avoid any confusion with the material conditional

These theories had been interesting recently for some researchers who wanted to focus on an analysis on the causal and temporal structure of events to give an independent characterization of the set of laws we mentioned before. However, these theories did not weigh much on the topic of conditional logics.

1.2 Stalnaker theory of conditionals

In 1968, Stalnaker [20] provided a new approach on the Ramsey Test. In his essay he interprets the Ramsey Test as follows, according to him this is how to evaluate a conditional:

> *First, add the antecedent (hypothetically) to your stock of beliefs; second, make whatever adjustments are required to maintain consistency (without modifying the hypothetical belief in the antecedent); finally, consider whether or not the consequent is then true.*

That is, Stalnaker is reconstructing the Ramsey Test for a conditional "If P then Q" following 3 steps:

1. Add P to the beliefs.
2. Adjust the beliefs to make them consistent.
3. Verify whether Q is true or not.

The problem now for Stalnaker is the transition from beliefs conditions to truth conditions. He develops here the concept of **possible worlds** to answer this question. He explains the possible world as "The ontological analogue of a stock of hypothetical beliefs".
According to Stalnaker, one can evaluate a conditional $A \rightarrow B$ in this logic, following his 3 steps: Given ω the actual world, we consider a possible world ω' in which A is true and which differs minimally from the actual world (which corresponds to add A to the beliefs and adjust them). If B is true in ω' then the conditional "If A then B" is true in ω.

1.2.1 Language

The language used by Stalnaker is his conditional logics is $\mathcal{L}_{>}$, that is the language of propositional logics, in which he adds his conditional symbol $>$. In his language, if A is an atom then A is a formula. We can write the rest of the definition of his language in the following way:
For A and B formulae,

$$A, B ::= A|\neg A|A \wedge B|A \vee B|A \rightarrow B|A > B$$

1.2.2 Model structure

Stalnaker Model. *Let M be a Stalnaker model, M is an ordered triple $\langle W, R, \lambda \rangle$, W is the set of all possible worlds, R is the reflexive relation of relative possibility between two worlds, λ is a member of W understood as the* ***the absurd world*** *- where contradictions and all their consequences are true.*

To illustrate the relation of relative possibility R, if α and β are members of W, then $\alpha R \beta$ means β is possible with respect to α. Moreover, every world is therefore possible with respect to itself.
λ is the only element that is not part of the standard modal semantics. It is not accessible from any other world with the relation R. Its goal is to allow for an interpretation of a conditional where the antecedent is impossible, to do so we need an absurd world.
To complete this structure, Stalnaker defines a selection fonction $f : p(W) \times W \rightarrow W$ which associates to a proposition and a possible world, a possible world and which satisfies five conditions:

1. $f(A, \omega) \models A$
2. $f(A, \omega) = \lambda$ only if no ω' such that $\omega R \omega$' and $\omega' \models A$
3. if $\omega \models A$ then $f(A, \omega) = \omega$
4. if $f(A, \omega) \models C$ and $f(C, \omega) \models A$, then $f(A, \omega) = f(C, \omega)$
5. if $f(A, \omega) \neq \lambda$, then $\omega R f(A, \omega)$

The first clause means that the antecedent is true in the closest world selected by the function. The second together with the fifth clause, added by Nute [12] in 1980, make the selected world the absurd world when it's

not possible for the antecedent to be true in any possible world. The third clause is here to ensure that if the actual world satisfies the antecedent, then the closest world is the actual world itself. Finally, the fourth clause is a clause of "consistency"on the distance between possible worlds, it means that if the consequent is true in the closest world ω_1 for the antecedent, and if there exists a closest world ω_2 for the consequent in which the antecedent is true, then it is likely that $\omega_1 = \omega_2$.

Therefore, if we define V a valuation for atomic sentences in worlds of W, a Stalnaker model structure M is often written $\langle W, R, V, \lambda, f \rangle$.
Let's now define what is the syntax Stalnaker uses in his model.

1.2.3 The formal system

Stalnaker defines an axiomatic system to match the model structure seen before. He calls this system **C2**. As he wrote in his article [20], he defined his conditional connective > with this system:
$\Box A =_{df} \neg A > A$
$\Diamond A =_{df} \neg(A > \neg A)$
$A \gtrless B =_{df} (A > B) \wedge (B > A)$
(Modus Ponens) If A and (A $\rightarrow$ B) are theorems, then B is a theorem
(Gödel rule of necessitation) If A is a theorem $\Box A$ is a theorem

(a1) Tautology instanciation: Any tautologous (well-formed) formula is an axiom

(a2) $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

(a3) $\Box(A \rightarrow B) \rightarrow (A > B)$

(a4) $\Diamond A \rightarrow ((A > B) \rightarrow \neg(A > \neg B))$

(a5) $(A > (B \vee C)) \rightarrow ((A > B) \vee (A > C))$

(a6) $(A > B) \rightarrow (A \rightarrow B)$

(a7) $(A \gtrless B) \rightarrow ((A > C) \rightarrow (B > C))$

1.2.4 Stalnaker's semantics properties

Thanks to his semantics and his theory about possible worlds, Stalnaker not only invalidates the paradoxes of material implication, as the strict conditional already did, but also invalidates the problems of the strict conditional analysis, the transitivity, the strengthening of the antecedent and the contraposition. He wrote in his article [20], about these three properties:

Transitivity:

> From $A > B$ and $B > C$, one cannot infer $A > C$

To take the Hoover example, calling A the proposition "Hoover is Russian", B the proposition "Hoover is communist", C the proposition "Hoover is a traitor", we can build a Stalnaker model M such as, $M, \omega \models (A > B), (B > C)$ but $M, \omega \not\models (A > C)$. That is, there can exists a possible world ω_1 for A in which B is true, there can exists a possible world ω_2 for B in which C is true, but, unless $\omega_1 = \omega_2$, nothing says C is true in ω_1, the closest world from the actual world where Hoover is Russian is not necessary the closest world where he is communist, it can be a world where he is American, but communist for instance,..

Strenghtening of the antecedent:

> The strenghtening of the antecedent follows from the transitivity rule; But it is obvious that the former rule is invalid; we cannot always strengthen the antecedent of a true conditional and have it remain true.

We can take here the example of the match. If the closest world in which we struck the match is the world in which it lights ($f(struck, \omega) \models light$). Then it is possible that the closest world where we struck the match in a room without oxygen is not the world in which it lights ($f(struck \wedge \neg oxygen, \omega) \models \neg light$). This would mean that the closest world where we struck the match is not a world where we do it in a room without oxygen.

Contraposition:

> Contraposition, valid for both the material and the strict implication is false for the conditional corner. $A > B$ may be true while $\neg B > \neg A$ is false.

Retaking the example of the US in Vietnam: In the closest world in which the US halts the bombing, the North Vietnam would not agree to negotiate is possible, if it wants the complete withdrawal of the troops. But in the closest world where North Vietnam agrees to negotiate, the fact US will not have halted the bombing is not possible, as it would have been a prerogative for them to come to an agreement.
The closest world in which the US halts the bombing is not necessarily the same as the one where the North Vietnam agrees to negotiate.

Stalnaker's analysis also introduced two properties, the *conditional negation* and the *conditional excluded middle*.

The conditional negation can be expressed as follows:

$$\neg(A > C) \models A > \neg C$$

This means the following sentence entails the second:

- It's false that Napoleon would not be surprised if he saw the France right now.
- If Napoleon saw the France right now, he would be surprised.

The conditional excluded middle is:

$$\models (A > C) \vee (A > \neg C)$$

Wheter one or the other of the following sentences is true, but one must be true:

- If the bee goes extinct, the world would be destroyed.
- If the bee goes extinct, the world wouldn't be destroyed.

We will see later that this is a consequence of the unique character of Stalnaker's closest world.

1.3 Lewis conditional logics

Five years later, David Lewis [11] proposed a new theory on conditional logics. He agreed with most of Stalnaker's theory and accepted the idea of possible worlds and similarity relations, yet, he clearly rejected Stalnaker's selection function and particularly drops two aspects of this approach, the *uniqueness assumption* and the *limit assumption*.

1.3.1 Uniqueness assumption

Stalnaker's uniqueness assumption is that for the actual world ω and an antecedent A there is at most one closest worlds, $f(A, \omega)$ (For a reminder, there can also be no possible world for A and ω, this is the purpose of λ, the absurd world).
Lewis rejected this assumption wondering why there couldn't exist other possible worlds, as similar as the actual world than the "closest" one and where the antecedent is also true.

He took the same example as Quine [14] took before him:

1. If Bizet and Verdi were compatriots, they would be Italians.
2. If Bizet and Verdi were compatriots, they would be French.

According to Stalnaker there exists only one closest world, thus only one of the two consequents is true. On the other hand, Lewis affirms that in this situation there will be one possible world closer to the actual world than the other.

Thus, according to Lewis, the selection function should not select necessarily only one possible world.
Therefore,

$$M, \omega \models (A \mathrel{\Box\!\!\rightarrow} C) \Leftrightarrow M, f(A, \omega) \models C$$

Should become

$$M, \omega \models (A \mathrel{\Box\!\!\rightarrow} C) \Leftrightarrow \forall \omega' \in f(A, \omega),\ M, \omega' \models C$$

Here we have $A \mathrel{\Box\!\!\rightarrow} C$ true in ω if, and only if, C is true in all the $\Box\!\!\rightarrow$-closest worlds for A from ω. It is as we had add a necessity operator to the

conditional, linking it to the antecedent: $A \boxrightarrow C$ is true if C is true in all the closest worlds where A is true.

With this reconsideration, Lewis also invalidates Stalnaker's conditional excluded middle. Indeed, if we take the example of Bizet and Verdi again, let ω be the actual world, ω_F be the world where the two are French, and ω_I the world where the two are Italian.
According to Lewis, the selection function would be $f(A, \omega) = \omega_F, \omega_I$. Therefore, if they were compatriots, they wouldn't have been French (in ω_I), but they also wouldn't have not been French (in ω_F). If we note F the condition of the two being French, we would have:

$$M, \omega \nvDash (A \boxrightarrow F) \vee (A \boxrightarrow \neg F)$$

This is more explicit in natural language: if Bizet and Verdi were compatriots, it's not sure they would have been French, but it's not sure they wouldn't have been French either.

1.3.2 Limit assumption

If we take a relation order $\leq_\omega$ to express that one possible world is closer to the actual world ω than another. That is, $u \leq_\omega v$ would mean u is closer to ω than v. Without the uniqueness assumption, we consider that there exists more than one possible world minimal in the sense of $\leq_\omega$. The Lewis limit assumption supposes that, however, there exists $\leq_\omega$-minimal worlds, there is a minimal limit in this relation order.

Lewis took the example of a line measuring under one inch in the actual world ω.
If it were measuring more than one inch, which possible world would be the minimal according to $\leq_\omega$?
If we take a possible world u where it is measuring one inch and ϵ thou, there always will be a possible world v where it is one inch and $\epsilon/2$ thou long,..
If we consider there is no minimal and that all worlds where the line is longer than one inch are all most similar, then we would consider that a world where the line is a thousand miles long is as close to our world as one where it is 1,1 inch long.

Stalnaker and Lewis agreed that we cannot give up on this assumption, if we did, we could also give up on the concept of similarities between world and look for another semantics.

The Lewis theory takes into account the possibility of having this kind of problems, including an infinity of possible worlds closer to the actual world one to another. He thus modified the criteria of similarity between worlds: let A be the antecedent and C the consequent, instead of wanting closest A-possible worlds to be C-worlds, we will be looking for a possible world u both A-world and C-world such as there are no possible worlds v closer to ω that is an A-world and a $\neg$C-world.

$(A \boxdot\!\!\rightarrow C)$ is true on ω if, and only if, there is a $(A \wedge C)$-possible world closer to ω than any other $(A \wedge \neg C)$-possible world.[5]

1.3.3 Sphere semantics

Lewis then introduced a new semantics he called the **Sphere semantics**. In this semantics, spheres have to be understood as sets of worlds. Each world is, in this approach, equipped with a nested system of spheres. For a given world ω, the other worlds in the inner sets (starting from the one containing directly the world we consider) are the most plausible ones, as contrary to the outer sets which represents the less plausible.
S is a sphere to ω if its worlds are accessibles from ω and more plausibles than these which are out of S.

Sphere Model. *Let M be a sphere model, M is a triple $\langle W, [\![.]\!], \mathcal{S}\rangle$ for the language $\mathcal{L}_{\boxdot\!\!\rightarrow}$, W is the set of all possible worlds, $[\![.]\!]$ an atomic valuation, assigning to an atom p the set of worlds where p is true, $\mathcal{S}$ a system of sphere.*

Modeling relation. *Let $M = \langle W, I, \mathcal{S}\rangle$ be a sphere model, for every world $\omega \in W$, we define the modeling relation as follows:*

1. *$M, \omega \models p$ if, and only if, $\omega \in [\![p]\!]$*
2. *$M, \omega \models \neg A$ if, and only if, $M, \omega \not\models A$*
3. - *$M, \omega \models (A \wedge B)$ if, and only if, $M, \omega \models A$ and $M, \omega \models B$*

[5] If there exists an A-possible world

- $M, \omega \models (A \vee B)$ *if, and only if,* $M, \omega \models A$ *or* $M, \omega \models B$
- $M, \omega \models (A \rightarrow B)$ *if, and only if, if* $M, \omega \models A$ *then* $M, \omega \models B$

4. $M, \omega \models (A \boxrightarrow B)$ *if, and only if, if there exists* $S \in \mathcal{S}_\omega$ *such as* $[\![A]\!] \cap S \neq \emptyset$*, then there exists* $T \in \mathcal{S}_\omega$ *such as* $[\![A]\!] \cap T \neq \emptyset$ *and* $\forall \omega \in T$*,* $M, \omega \models (A \rightarrow B)$

In this logic, Lewis takes as primitive the comparative plausibility $\preccurlyeq$, that we can define as follows:
$M, \omega \models (A \preccurlyeq B)$ if, and only if, $\forall \alpha \in \mathcal{S}_\omega$ if $[\![B]\!] \cap \alpha \neq \emptyset$, then $[\![A]\!] \cap \alpha \neq \emptyset$.
We could translate it as "A is at least as plausible as B".

If we take the definition of $\boxrightarrow$ we established before[6], we can translate it as:
$\omega \in [\![A \boxrightarrow B]\!]$ if, and only if, if $\forall \alpha \in \mathcal{S}_\omega$, $\alpha \cap [\![A]\!] \neq \emptyset$ then $\exists \alpha \in \mathcal{S}_\omega$, such as $\alpha \cap [\![A \wedge B]\!] \neq \emptyset$ and $\alpha \cap [\![A \wedge \neg B]\!] = \emptyset$.

Saying that $[\![\bot]\!] = \emptyset$ by definition, we can then express $\boxrightarrow$ with $\preccurlyeq$:

$$A \boxrightarrow B \equiv (\bot \preccurlyeq A) \rightarrow ((A \wedge \neg B) \preccurlyeq (A \wedge B))$$

Indeed, saying $\alpha \cap A \neq \emptyset$ is the same as saying that A "is more plausible than" the false assumption, and $\exists \alpha \in \mathcal{S}_\omega$, such as $\alpha \cap [\![A \wedge B]\!] \neq \emptyset$ and $\alpha \cap [\![A \wedge \neg B]\!] = \emptyset$ simply means that $A \wedge B$ "is more plausible than" $A \wedge \neg B$.

It is worth noticing that we can also define $\preccurlyeq$ with $\boxrightarrow$:

$$A \preccurlyeq B \equiv \neg((A \vee B) \boxrightarrow \bot) \rightarrow ((A \vee B) \boxrightarrow \neg A)$$

1.3.4 Syntax: The system $\mathbb{V}$

$\mathbb{V}$ is the set of formulae valid in all spheres.
For this set, we define a sphere model M=$\langle W, [\![.]\!], \mathcal{S} \rangle$
The set of conditional formulae for this model is given by the following grammar, with p a propositional variable, $\top$ the symbol for *true* and $\bot$ the symbol for *false*: $A, B ::= p|\bot|\top|A \wedge B|A \vee B|A \rightarrow B|A \preccurlyeq B$.

[6] See p.21

Extensions of $\mathbb{V}$ can be semantically defined by specifying additional conditions to the class of sphere models:

- $\mathbb{N}$ normality : $\forall \omega \in W$ we have $\mathcal{S}_\omega \neq \emptyset$
- $\mathbb{T}$ total reflexitivity : $\forall \omega \in W$ we have $\omega \in \bigcup \mathcal{S}_\omega$
- $\mathbb{W}$ weak centering : normality holds and $\forall \alpha \in \mathcal{S}_\omega$ we have $\omega \in \alpha$
- $\mathbb{C}$ centering : $\forall \omega \in W$ we have $\{\omega\} \in \mathcal{S}_\omega$
- $\mathbb{A}$ absoluteness : $\forall \omega, v \in W$ we have $\mathcal{S}_\omega = \mathcal{S}_v$

We denote the extensions by the concatenation of two or more letters (Ex: $\mathbb{VW}$, $\mathbb{VNA}$,..) The axiomatization of all these rules if the language we have defined earlier gives us the following table:

- Tautology instanciation: Any tautologous (well-formed) formula is an axiom.
- Modus ponens: If A is a theorem, and $A \rightarrow B$ is a theorem, then B is a theorem.

CPR	$(A \wedge \neg B) \rightarrow (A \preccurlyeq B)$	**CPA**	$(A \preccurlyeq A \vee B) \vee (B \preccurlyeq A \vee B)$
TR	$(A \preccurlyeq B \preccurlyeq C) \rightarrow (A \preccurlyeq C)$	**CO**	$(A \preccurlyeq B) \vee (B \preccurlyeq A)$
N	$\neg(\bot \preccurlyeq \top)$	**W**	$A \rightarrow (A \preccurlyeq \top)$
T	$(\bot \preccurlyeq \neg A) \rightarrow A$	**A1**	$(A \preccurlyeq B) \rightarrow (\bot \preccurlyeq \neg(A \preccurlyeq B))$
C	$(A \preccurlyeq \top) \rightarrow A$	**A2**	$\neg(A \preccurlyeq B) \rightarrow (\bot \preccurlyeq (A \preccurlyeq B))$

$$\mathcal{A}_{\mathbb{V}} := \{CPR, CPA, TR, CO\}$$

$$\mathcal{A}_{\mathbb{VN}} := \mathcal{A}_{\mathbb{V}} \bigcup \{N\} \quad \mathcal{A}_{\mathbb{VT}} := \mathcal{A}_{\mathbb{V}} \bigcup \{N, T\} \quad \mathcal{A}_{\mathbb{VW}} := \mathcal{A}_{\mathbb{V}} \bigcup \{N, T, W\}$$

$$\mathcal{A}_{\mathbb{VC}} := \mathcal{A}_{\mathbb{V}} \bigcup \{N, T, W, C\} \quad \mathcal{A}_{\mathbb{VA}} := \mathcal{A}_{\mathbb{V}} \bigcup \{A1, A2\} \quad \mathcal{A}_{\mathbb{VNA}} := \mathcal{A}_{\mathbb{V}} \bigcup \{N, A1, A2\}$$

Table 1.2: Axioms for Lewis logics

1.3.5 Sequent Calculus

The **Sequent calculus** brought to us by Gerhard Gentzen [6, 7] is a formal system, generalizing the form natural deduction applied to mathematical objects called the *sequents*. It's actually a tool for proof search in mathematical logic, organizing formulae in the form $A_1, A_2, .. \vdash C_1, C_2, ..$ where

A_i's and C_i's are formulae. The left-hand side formulae are called the *antecedents* and the right-hand side the *consequents*. In this system, proofs are established as *reduction trees*, applying rules to a set of formula Γ in order to obtain more simple formulas and eventually get axioms, finishing the proof. For instance, rules are applied as follows:

$$\frac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B} \to_R$$

Table 1.3: Example of application of a rule: $(\to_R)$

And ends up in the so-called reduction trees:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\bot \Rightarrow [B \lhd A]}\,\bot_L \qquad \dfrac{\dfrac{\dfrac{}{A \Rightarrow B, A}\,init}{[B, A \lhd A]}\,jump \quad \dfrac{\dfrac{}{B \Rightarrow B, A}\,init}{[B, A \lhd B]}\,jump}{[B \lhd A], [A \lhd B]}\,com}{[B \lhd A], A \preccurlyeq B}\,\preccurlyeq_R}{(A \preccurlyeq B) \to \bot \Rightarrow [B \lhd A]}\,\to_L}{(A \preccurlyeq B) \to \bot \Rightarrow (B \preccurlyeq A)}\,\preccurlyeq_R}{((A \preccurlyeq B) \to \bot) \to (B \preccurlyeq A)}\,\to_R$$

Table 1.4: Example of a reduction tree: derivation of **CO** (proof given by the LeThProve program)

We say a set of formulae Γ is derivable if it admits a derivation, of which we can see an example above. In fact, a derivation is a tree in which nodes are sequents we obtain applying the rules of the calculus as we've seen before. A branch is thus a sequence of nodes Γ_i, themselves obtained by applying rules bottom-up. A branch will close if one of its nodes is a standard axiom, in the example before, axioms were the false antecedent and the initiation axiom[7]. A tree will then be closed if all its branches are

[7] See Table 2.6

closed, Γ will be said to have a derivation if it is the root of a closed tree.

To complete this definition we have to add a modification to the sequent calculus used in [13]. To express the disjunction of $\preccurlyeq$-formulae, as formulae of the type $(A_1 \preccurlyeq B) \vee (A_2 \preccurlyeq B) \vee (A_3 \preccurlyeq B) \cdots \vee (A_n \preccurlyeq B)$ were impossible to represent in derivation trees, we have chosen as [8] to regroup them in *blocks*.

Blocks. *A* block *is an object composed by a set Σ of formulae A_i and a formula B, we note it $[\Sigma \lhd B]$. It is the conjonction of $\preccurlyeq$-formulae, then we have $(A_1 \preccurlyeq B) \vee (A_2 \preccurlyeq B) \vee (A_3 \preccurlyeq B) \cdots \vee (A_n \preccurlyeq B) := [\Sigma \lhd B]$, with $\Sigma = \bigvee_{1 \leq i \leq n} (A_i)$.*

However, sequent calculus can be separated into two different categories, *external calculus* and *internal calculus*. External calculus, which goal is to import the semantics into the syntax, and internal calculus, which stay within the language, so that we're working directly with formulae of the language.
For the rest of this paper, we will stick to internal calculi as this is the aspect we are working on with the theorem prover.

1.3.6 Internal Calculus for Lewis's logics

We will take the language $\mathcal{L}$ for the logics we will study afterward. In that way, we will note $\mathcal{I}_{\mathcal{L}}$ the internal calculus, we will use $\Rightarrow$ instead of $\vdash$ between antecedents and consequents of a sequent. Given the sequent $\Gamma \Rightarrow \Delta$ we will denote its derivation $\mathcal{I}_{\mathcal{L}} \vdash \Gamma \Rightarrow \Delta$, that is, the derivation tree with $\Gamma \Rightarrow \Delta$ as root.

Non-invertible calculus

Non-invertible calculus is interesting in that it is used to introduce the cut-elimination proof. Yet, as we will prove it later, invertible and non-invertible are equivalent. Considering this, as the invertible calculus provides termination and completion for the derivation, it is actually the one we are working on. However, the non-invertible calculus gives us the basis for the rules of the invertible one, so here are the rules for derivation in the non-invertible calculus.

$$\frac{}{\Gamma, \bot \Rightarrow \Delta}\ \bot_L \qquad \frac{}{\Gamma, p \Rightarrow \Delta, p}\ init \qquad \frac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B}\ \to_R \qquad \frac{\Gamma \Rightarrow \Delta, [A \lhd B]}{\Gamma \Rightarrow \Delta, A \preccurlyeq B}\ \preccurlyeq_R$$

$$\frac{\Gamma, B \Rightarrow \Delta \quad \Gamma \Rightarrow \Delta, A}{\Gamma, A \to B \Rightarrow \Delta}\ \to_L \qquad \frac{\Gamma \Rightarrow \Delta, [D, \Sigma \lhd A] \quad \Gamma \Rightarrow \Delta, [\Sigma \lhd C]}{\Gamma, C \preccurlyeq D \Rightarrow \Delta, [\Sigma \lhd A]}\ \preccurlyeq_L$$

$$\frac{\Gamma \Rightarrow \Delta, [\Sigma_1, \Sigma_2 \lhd A] \quad \Gamma \Rightarrow \Delta, [\Sigma_1, \Sigma_2 \lhd B]}{\Gamma \Rightarrow \Delta, [\Sigma_1 \lhd A], [\Sigma_2 \lhd B]}\ com \qquad \frac{A \Rightarrow \Sigma}{\Gamma \Rightarrow \Delta, [\Sigma \lhd A]}\ jump$$

$$\frac{\Gamma, A, A \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta}\ Con_L \qquad \frac{\Gamma \Rightarrow \Delta, A, A}{\Gamma \Rightarrow \Delta, A}\ Con_R \qquad \frac{\Gamma \Rightarrow \Delta, [\Sigma \lhd A], [\Sigma \lhd A]}{\Gamma \Rightarrow \Delta, [\Sigma \lhd A]}\ Con_S \qquad \frac{\Gamma \Rightarrow \Delta, [\Sigma, A, A \lhd B]}{\Gamma \Rightarrow \Delta, [\Sigma, A \lhd B]}\ Con_B$$

$$\frac{\Gamma \Rightarrow \Delta, [\bot \lhd \top]}{\Gamma \Rightarrow \Delta}\ N \qquad \frac{\Gamma \Rightarrow \Delta, \Sigma}{\Gamma \Rightarrow \Delta, [\Sigma \lhd A]}\ W \qquad \frac{\Gamma \Rightarrow \Delta, B \quad \Gamma \Rightarrow \Delta, [\bot \lhd A]}{\Gamma, A \preccurlyeq B \Rightarrow \Delta}\ T$$

$$\frac{\Gamma, C \Rightarrow \Delta \quad \Gamma \Rightarrow \Delta, D}{\Gamma, C \preccurlyeq D \Rightarrow \Delta}\ C \qquad \frac{\Gamma^{\preccurlyeq}, B \Rightarrow \Delta^{\preccurlyeq}, \Sigma}{\Gamma \Rightarrow \Delta, [\Sigma \lhd B]}\ A$$

$\Gamma^{\preccurlyeq}$ and $\Delta^{\preccurlyeq}$ are Γ and Δ restricted to $\preccurlyeq$-formulae and blocks

$$\mathcal{I}_{\mathbb{V}} := \{\bot_L, init, \to_L, \to_R, \preccurlyeq_L, \preccurlyeq_R, com, jump, Con_R, Con_L, Con_S, Con_B\}$$
$$\mathcal{I}_{\mathbb{VN}} := \mathcal{I}_{\mathbb{V}} \bigcup \{N\} \qquad \mathcal{I}_{\mathbb{VT}} := \mathcal{I}_{\mathbb{V}} \bigcup \{N, T\} \qquad \mathcal{I}_{\mathbb{VW}} := \mathcal{I}_{\mathbb{V}} \bigcup \{N, T, W\}$$
$$\mathcal{I}_{\mathbb{VC}} := \mathcal{I}_{\mathbb{V}} \bigcup \{N, T, W, C\} \qquad \mathcal{I}_{\mathbb{VA}} := \mathcal{I}_{\mathbb{V}} \bigcup \{A\} \qquad \mathcal{I}_{\mathbb{VNA}} := \mathcal{I}_{\mathbb{V}} \bigcup \{N, A\}$$

Table 1.6: Rules for Lewis' non-invertible calculus: the calculus $\mathcal{I}_{\mathbb{V}}$ and its extensions

Invertible calculus

Equivalence between non-invertible calculus $\mathcal{I}_{\mathcal{L}}$ and invertible calculus $\mathcal{I}^{\mathrm{i}}_{\mathcal{L}}$ is proved via the admissibility of the contraction rules.
Indeed, to prove $\Gamma \Rightarrow \Delta, F, F$ is the same as to prove $\Gamma \Rightarrow \Delta, F$. If $\Gamma \Rightarrow \Delta, F$ is derivable then $\Gamma \Rightarrow \Delta, F, F$ will come to the same termination, both *init* and $\bot_L$ will have the possibility to be applied. Having twice the same formula does not add any information on the proof. We can then admit the weakening and contraction rules. However, while the calculus we present does

not contain those rules, implicitly, every time a formula or a block comes duplicated in the sequent, it is deleted in the next step using the weakening rule. This way of functioning allows the derivation of a duplicate-free sequent and thus a loop-free derivation, ensuring the termination of the derivation.
Moreover, admiting these contraction rules, we can prove that both invertible and non-invertible calculi are equivalent:

Theorem (of equivalence)**.** *Let A be a formula,*

$$\mathcal{I}_{\mathcal{L}} \vdash A \Leftrightarrow \mathcal{I}^{i}_{\mathcal{L}} \vdash A$$

Proof. $\Rightarrow\lrcorner$ This direction is justified by the admissibility of the weakening rule for $\mathcal{I}^{\mathrm{i}}_{\mathcal{L}}$ and their application.

$\Leftarrow\lrcorner$ This direction is trivial since we can legitimely applicate contraction and weakening rules at each step since these are rules for $\mathcal{I}_{\mathcal{L}}$. □

So, here are the rules for the invertible calculus, which we will use with our program.

$$\frac{}{\Gamma, \bot \Rightarrow \Delta}\ \bot_L \qquad \frac{}{\Gamma, p \Rightarrow \Delta, p}\ init \qquad \frac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B}\ \to_R \qquad \frac{\Gamma \Rightarrow \Delta, [A \lhd B]}{\Gamma \Rightarrow \Delta, A \preccurlyeq B}\ \preccurlyeq_R$$

$$\frac{\Gamma, B \Rightarrow \Delta \quad \Gamma \Rightarrow \Delta, A}{\Gamma, A \to B \Rightarrow \Delta}\ \to_L \qquad \frac{\Gamma, C \preccurlyeq D \Rightarrow \Delta, [D, \Sigma \lhd A] \quad \Gamma, C \preccurlyeq D \Rightarrow \Delta, [\Sigma \lhd C], [\Sigma \lhd A]}{\Gamma, C \preccurlyeq D \Rightarrow \Delta, [\Sigma \lhd A]}\ \preccurlyeq^{\mathrm{i}}_L$$

$$\frac{\Gamma \Rightarrow \Delta, [\Sigma_1, \Sigma_2 \lhd A], [\Sigma_2 \lhd B] \quad \Gamma \Rightarrow \Delta, [\Sigma_1, \Sigma_2 \lhd B], [\Sigma_1 \lhd A]}{\Gamma \Rightarrow \Delta, [\Sigma_1 \lhd A], [\Sigma_2 \lhd B]}\ com^{\mathrm{i}} \qquad \frac{A \Rightarrow \Sigma}{\Gamma \Rightarrow \Delta, [\Sigma \lhd A]}\ jump$$

$$\frac{\Gamma \Rightarrow \Delta, [\bot \lhd \top]}{\Gamma \Rightarrow \Delta}\ N \qquad \frac{\Gamma \Rightarrow \Delta, \Sigma, [\Sigma \lhd A]}{\Gamma \Rightarrow \Delta, [\Sigma \lhd A]}\ W^{\mathrm{i}} \qquad \frac{\Gamma, A \preccurlyeq B \Rightarrow \Delta, B \quad \Gamma, A \preccurlyeq B \Rightarrow \Delta, [\bot \lhd A]}{\Gamma, A \preccurlyeq B \Rightarrow \Delta}\ T^{\mathrm{i}}$$

$$\frac{\Gamma, C, C \preccurlyeq D \Rightarrow \Delta \quad \Gamma, C \preccurlyeq D \Rightarrow \Delta, D}{\Gamma, C \preccurlyeq D \Rightarrow \Delta}\ C^{\mathrm{i}} \qquad \frac{\Gamma^{\preccurlyeq}, B \Rightarrow \Delta^{\preccurlyeq}, \Sigma, [\Sigma \lhd B]}{\Gamma \Rightarrow \Delta, [\Sigma \lhd B]}\ A^{\mathrm{i}}$$

$\Gamma^{\preccurlyeq}$ and $\Delta^{\preccurlyeq}$ are Γ and Δ restricted to $\preccurlyeq$-formulae and blocks

$$\mathcal{I}^{\mathrm{i}}_{\mathbb{V}} := \{\bot_L, init, \to_L, \to_R, \preccurlyeq^{\mathrm{i}}_L, \preccurlyeq_R, com^{\mathrm{i}}, jump\}$$

$$\mathcal{I}^{\mathrm{i}}_{\mathbb{VN}} := \mathcal{I}^{\mathrm{i}}_{\mathbb{V}} \bigcup \{N\} \qquad \mathcal{I}^{\mathrm{i}}_{\mathbb{VT}} := \mathcal{I}^{\mathrm{i}}_{\mathbb{V}} \bigcup \{N, T^{\mathrm{i}}\} \qquad \mathcal{I}^{\mathrm{i}}_{\mathbb{VW}} := \mathcal{I}^{\mathrm{i}}_{\mathbb{V}} \bigcup \{N, T^{\mathrm{i}}, W^{\mathrm{i}}\}$$

$$\mathcal{I}^{\mathrm{i}}_{\mathbb{VC}} := \mathcal{I}^{\mathrm{i}}_{\mathbb{V}} \bigcup \{N, T^{\mathrm{i}}, W^{\mathrm{i}}, C^{\mathrm{i}}\} \qquad \mathcal{I}^{\mathrm{i}}_{\mathbb{VA}} := \mathcal{I}^{\mathrm{i}}_{\mathbb{V}} \bigcup \{A^{\mathrm{i}}\} \qquad \mathcal{I}^{\mathrm{i}}_{\mathbb{VNA}} := \mathcal{I}^{\mathrm{i}}_{\mathbb{V}} \bigcup \{N, A^{\mathrm{i}}\}$$

Table 1.8: Rules for Lewis' invertible calculus: the calculus $\mathcal{I}^{\mathrm{i}}_{\mathbb{V}}$ and its extensions

Chapter 2

Prolog

2.1 Introduction to Prolog

Prolog [3] is a programing language used for logics, created in 1972 by Alain Colmeraurer and Philippe Roussel, and is named from PROgraming in LOGics. It was really used in the years 1980, thanks to the contribution of Bob Kowalski [10] and the implementation on the Warren machine [21] This is a programing language where logical rules are defined and the compiler does the rest.

Used in artificial intelligence, the goal of Prolog is to verify first-order logic predicates, using backtracking, recursivity and unification. We need to precise what we mean by unification, which is a term a bit more unusual than backtracking or recursivity. Unification is, basically, the process of finding a substition of the variables in the formulae, allowing to make two terms structurally equals.
In fact, when you are doing a query, Prolog evaluates the predicates considering a database you define before. This specific database is composed of *clauses*. These clauses are actually rules and facts. We can see an example in the Figure 2.2.
Rules are in the form "*head:- tail.*", where head is a *term*, and tail is a *list of terms*. On the other hand, facts are simply rules without tail. Every rule and fact ends with a dot '.'.
Terms can be declined in 5 families, *numbers*, which is a trivial type, *atoms*, *variables*, *compounds* and *lists*.

In Prolog, *atoms* are written with lowercase letters, and can be interpreted as non-numerical constants (ex: a,b,..). *Variables*, written with uppercase letters, are used in goals to ask the Prolog interpreter on which term it could be replaced with (ex: X,Y,..). We can also define the character '_' to represent the *anonymous variable*. We use it when we are not interested into obtaining the term it could replace, an example will be provided in the Figure 2.1. *Compounds* are composed terms, they represent functions. *Lists* are a specific type of compounds, they are written in this way: [*Head*|*Tail*] in which Head can be whichever type of term, and Tail represents another list.

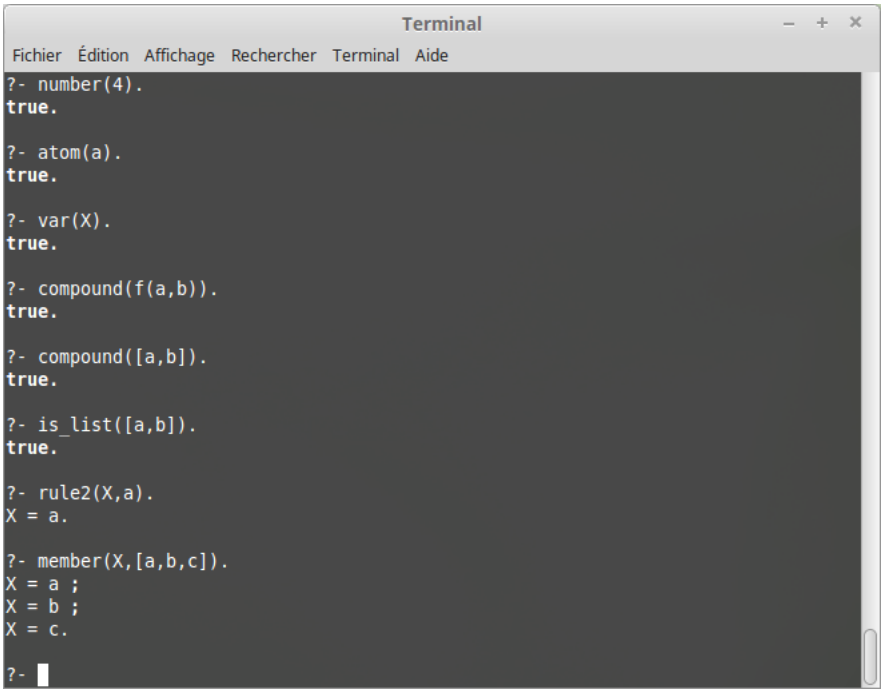

Figure 2.1: Example of terms and utilization of a variable

Let *member* be the following predicate:

```
1 member(X,[X|_]).
2 member(X,[_|Tail]):-member(X,Tail).
```

The type of programming we experience through logic programming, and by extension through Prolog, is called *Declarative programming*. As opposed to *Imperative programing*, declarative programming is much more

into describing *what* a program should do than *how* it should do it. In declarative programming, a program, following specifications, describes a result without listing the steps it should follow. Logical programing is a specific type of declarative programming, a *subparadigm*, in which the programs are composed of logical statements, and execute for proof searching.

Prolog thus allows to construct goals then submited to a software, according to facts and rules defined before, following an algorithm based on *SLD-resolution*.

There exists different software interpreting the language, in this paper we will be using **SWIPL**[1].
SWIPL is a software which examines goals written in Prolog. SWIPL actually examined the query according to clauses written into a file, by convention using the extension *'.pl'*, and consult them thanks to the command *consult*, as we may see in the figure below in the EMACS'[2] interface:

[1] www.swi-prolog.org
[2] www.gnu.org/software/emacs

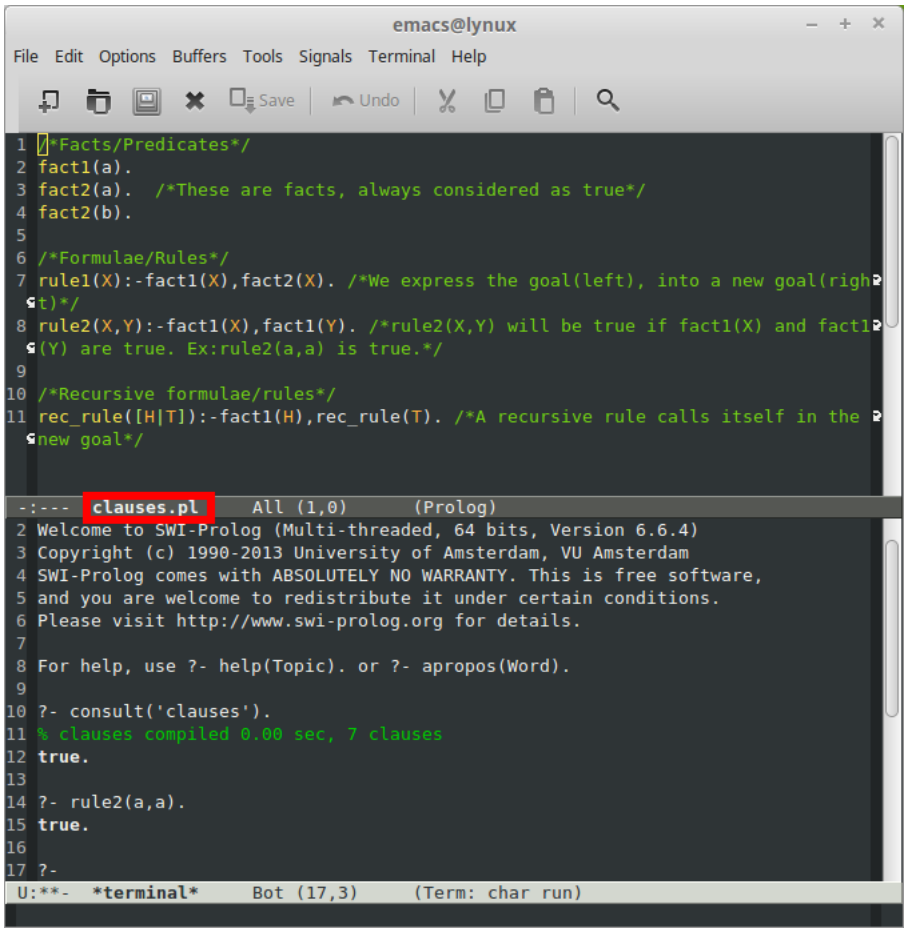

Figure 2.2: Example of Prolog language using EMACS' interface: Top screen, example of clauses in prolog; Bottom screen, execution of SWIPL

2.2 SLD-resolution

Selected, Linear and Definite resolution is an algorithm used to give proofs of a first logic formula. Let G $\leftarrow A_1, A_2, .., A_n$ be a goal for the algorithm. It examines the goal, selects a term A_i, looks for a clause C in the program that has A_i as head. It then formulates a new goal considering what it has to do to prove A_i, once it is proved, it returns and selects the A_{i+1} term. For instance, for the following program:

```
1 p:-q,r.
2 q:-u.
3 u.
```

If we submit the goal **?-p.** to the algorithm, it will select a term from the goal, here the only choice is p. It then looks for clauses containing p in the program, it finds p:-q,r. The new goal of the algorithm is then **q,r.** It selects the first term q and looks for a clause in the program, etc..
In fact, to use the "*head:-tail.*" form we defined before, when the head is selected as a goal, tail is a list of intermediate goals to examine, in order to fulfill the goal represented by *head*. With this point of view, you can consider facts, such as *u.* in the example above, as rules without tail, that is, without intermediate goals.
The processing of a SLD-resolution is called a *SLD-derivation* and is described by *SLD-trees*.

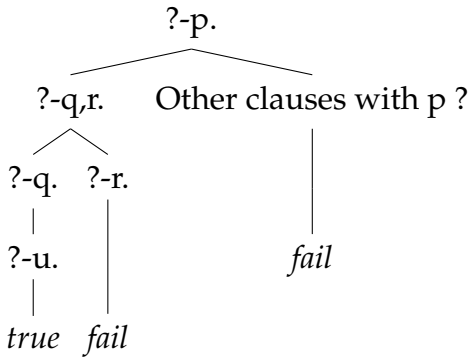

Table 2.1: Example of a SLD-tree with the program above

We can see here that the derivation eventually fails because there is no fact declaring r as true.

In fact, Prolog builds the SLD-tree following a *search rule*, which is a search strategy. Prolog uses the *depth-first* rule, in addition with a leftmost selection rule[3].

[3]We have seen it before: Prolog examines the leftmost term first

Which means that it will explore the first branch, the first possibility, before considering any other possibility of an answer. It returns the first answer it finds and ends here, unless the user wishes the contrary and asks for other results with ';'. Then Prolog continues to search for answers.
We can take as example the goal *?-member(X,[a,b,c]).* in the Figure 2.1.

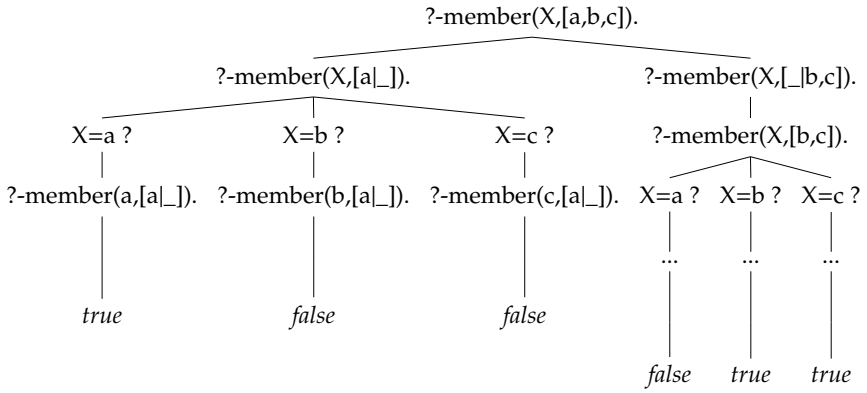

Table 2.2: Example of SLD-tree for *?-member(X,[a,b,c]).*

Here, for instance, the program will return *X=a* unless the user press ';', it then goes back to the last point where it had a choice to make: the node *?-member(X,[a|_]).*, it tests the atom *b*, etc..

To complement this depth-first rule and interact with the backtracking, we can introduce the cut '!'. The cut is an always true atom, which will block the backtracking and force the interpreter to stick to only one possibility, the one it already has chosen, with the variables fixed. For instance, with the following program:

```
1 p(X):-q(X),r(X),!,s(X).
2 p(X):-q(X).
3 q(a).
4 q(b).
5 r(a).
6 r(b).
7 s(b).
```

This gives us the following tree:

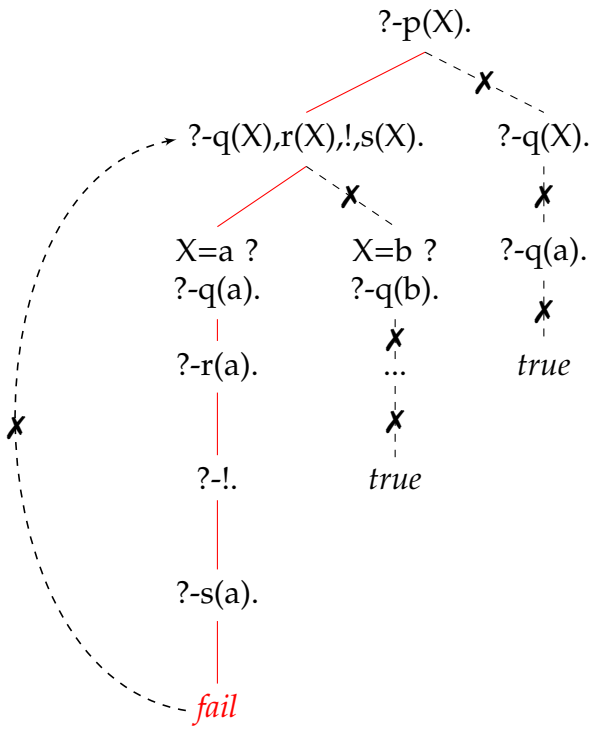

Table 2.3: Example of a tree for the cut (!) predicate

The dashed lines in the previous tree are possibilities that could have been explored without the cut (!). Here in fact the program only follows the red path and thus only returns *fail*. Indeed, after the ?-!. instruction, the backtracking is blocked, the variable X is fixed to 'a'.
This process can save much computational time avoiding branches prone to fail, but in the same way, it denies many possibilities that could have led to a different result.
There exists a functionality in SWIPL called *trace.* that writes down step by step what the program is doing, it helps a lot to understand this cut predicate, and how a program works in itself.

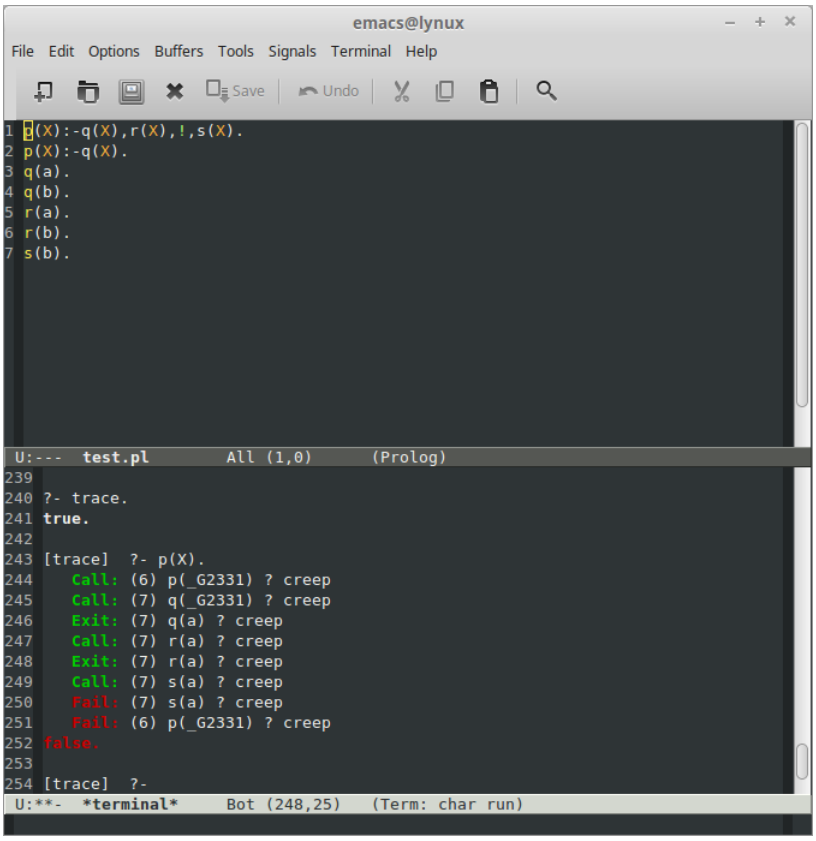

Figure 2.3: Trace for the goal *?-p(X).*

However, the process of "cutting" is often used to attempt to avoid infinite loops during the computation. This can also be used with the predicate "fail" to negate entire branches of SLD-trees. A cut will block all the backtracking for the rule examined, followed by "fail" it returns false for the present goal, and then cuts down an entire branch of a SLD-tree.

The alternative to the depth-first rule is the *breadth-first* one, which can be theoretically implemented in Prolog. It would mean that instead of ending a branch before going to explore another one, it would explore every possibility on the same level before continuing to the next step. Even if it could eventually give all the solutions for a given goal, it would be a lot less efficient if the goal is to find a solution, a proof, to a theorem.

Chapter 3

Implementation of a theorem-prover

3.1 The implementation

The first step was to decide on the syntax we will use to describe our rules, for more simplicity, we stuck with the binary operator $\preccurlyeq$ more than the $\Box\!\!\rightarrow$ to write the rules of the system. In a practical aspect, we chose to represent the different binary operators and terms in the following way:

- -> for the material implication $\rightarrow$
- < for the plausible comparative $\preccurlyeq$
- << for the blocks operator $\lhd$
- a,b,c,.. (lowercases) for the formulae instead of A,B,C,..[1]

With this syntax in hand, we had to find the function we will use to examine the theorem we want to prove, in the light of the rules of Lewis' conditional logics. We needed at least **three parameters**[2]:

- One to express the *antecedents*, it would be a **list** of formulae
- One to express the *consequents*, it would also be a **list** of formulae

[1] We had to do this because uppercases are considered by Prolog as variables.
[2] We later on created a fourth parameter to make an additional check

- Another to keep a trace of the rules used step by step, to be able to follow the demonstration and represent it later, it would be a **tree** of the rules used

Our function was then of the form:

```
1 prove(Gamma, Delta, Tree).
```

We then tried to express the logics' rules with this syntax.

3.1.1 The non-invertible calculus

We soon enough realized that the non-invertible calculus will represent several problems due to the contraction rules.
Adding the extensions created also a few problems, the logic $\mathbb{N}$ combined with the jump rule was prone to create infinite loops also.
Regarding the fact that both calculi are equivalent, we decided to focus more on the invertible calculus for the implementation.

3.1.2 The invertible calculus

The invertible calculus presents no contractions rules, but it also came with its own problems. First, the fact that we keep formulae through the application of rules, but there is also the fact that the logic $\mathbb{N}$ combined with the jump rule infinite loops were still there for instance.
We solved these problems thanks to multiple checks that we will introduce in the next sections.
Let's now focus on the implementation of the logic $\mathbb{V}$.

3.1.3 The logic $\mathbb{V}$

The first step was to implement the logic $\mathbb{V}$, base of Lewis' logics. Here is an example of a rule with one premise for the $\rightarrow_R$ rule (the Tree will only have one Sub-tree):

```
1 prove(Gamma, Delta, impR(Sub)):-select(A -> B, Delta, Delta1),!,
    prove([A,Gamma],[B,Delta1],Sub).
```

And with two premises for the $\preccurlyeq_L$ rule (two Sub-trees in the third parameter):

```
1 prove(Gamma,Delta,plL(Sub1,Sub2)):-member(A < B,Gamma),select(
    Sigma << C,Delta,Delta1),prove(Gamma,[[B,Sigma] << C,Delta1],
    Sub1),prove(Gamma,[Sigma << A,Sigma << C,Delta1],Sub2).
```

In this case, SWIPL will examine the first occurrence of prove and thus the first Sub-tree before examining the second.
In the first example, we can see the utilization of the cut rule. Indeed, we assume that if there are other expressions of the form $A \rightarrow B$ it will be examined in the rest of the branch. Therefore, we only need to select one of these expressions and fix the variables with it. We lose no information doing so, as all the possibilities of application of this rule are examined in the resulting branch. For instance, for the following goal:

```
1 prove([],[a->b,a->c,d->e],X).
```

The program will first examine $a->b$, this gives the goal:

```
1 prove([a],[b,a->c,d->e],X).
```

It then examines $a->c$, and then $d->e$, and finally gives the goal:

```
1 prove([d,a,a],[b,c,e],X).
```

And it will find no proof to terminate the derivation, so it will go back to the last choice-point using the backtracking method.
Without the cut, we would came back to the second goal *prove([a],[b,a->c,d->e],X).* examine $d->e$ before $a->c$, and then to the first goal *prove([],[a->b,a->c,d->e],X).* examine first $a->c$, etc.. But there is no need to do so as it was already examined and we saw it led to nothing. While, with the cut, we give up on the rule if the first branch fails, saving a lot of computing time, that is why it is used here in this rule.

On the other hand, for the rule $\preccurlyeq_L$, the cut will make us lose informations. For instance, the goal:

```
1 prove([],[(bot<(a->bot))->a],X,[]).
```

is not derivable with the cut in the $\preccurlyeq_L$ rule, but it is derivable without it, and its derivation is:

```
1 X = impR( n( plL(
2              w( impR( axini)))),
3              jump( axbot))).
```

We have to clarify a last point on these two exemples: the use of *select* and *member*.
These two predicates have a similar spirit and yet a different action. They both explore a list but, while *member* just verifies that its first parameter is in the list, *select* verifies that its first parameter is in the list and removes it from the list.
That is:

```
1 member(a,L).
```

returns *true* or *false* whether a is in L or not, and:

```
1 select(a,L1,L2).
```

will return true if a is in L1, and if it is the case, L2 is the list L1 without "a".
The context being the invertible calculus, some rules (as $\preccurlyeq_L$) allow elements to stay in the sequent despite the application of the rule. For instance, in :

$$\frac{\Gamma, C \preccurlyeq D \Rightarrow \Delta, [D, \Sigma \lhd A] \quad \Gamma, C \preccurlyeq D \Rightarrow \Delta, [\Sigma \lhd C], [\Sigma \lhd A]}{\Gamma, C \preccurlyeq D \Rightarrow \Delta, [\Sigma \lhd A]} \preccurlyeq_L^{\mathrm{i}}$$

$C \preccurlyeq D$ stays in the left part of the sequent through the application of the rule, as opposed to $[\Sigma \lhd A]$, which becomes $[D, \Sigma \lhd A]$ in the first sub-tree.
For a matter of simplicity, we used *select* on the rightmost part, to isolate and re-use easily the Δ.
For a complete understanding the predicates *member* and *select* are constructed as follow:

```
1 member(X, [X|_]).
2 member(X, [_|Tail]) :- member(X, Tail).
3 select(A, [A|B], B).
4 select(A, [B, C|D], [B|E]) :- select(A, [C|D], E).
```

3.1.4 The extensions

The second part consisted in writing down the rules for the extensions. Nothing particular except for the $\mathbb{A}$ rule which needed an extra predicate to remove all non-$\preccurlyeq$ rules:

```
1 prove(Gamma,Delta,a(Sub)):-select(Sigma << B,Delta,Delta1),
      extract(Gamma,GammaRestricted),extract(Delta1,DeltaRestricted
      ),prove([B,GammaRestricted],[Sigma << B,Sigma,DeltaRestricted
      ],Sub).
```

With extract the following predicate:

```
1 extract([],[]).
2 extract([A < B|Tail],[A < B|ResTail]):-!,extract(Tail,ResTail).
3 extract([Sigma << A|Tail],[Sigma << A|ResTail]):-!,extract(Tail,
      ResTail).
4 extract([_|Tail],ResTail):-extract(Tail,ResTail).
```

3.1.5 The checks

However, we had to make several additional checks to eliminate the infinite loops we mentioned above.
First, to avoid having too many lists included in lists and have a better analysis of duplications in the sequents, we had, first, to eliminate pointless lists. In order to do so, we flattened both the antecedent and the consequent:

```
1 prove(Gamma,Delta,X):-flatten(Gamma,GammaFlat),flatten(Delta,
      DeltaFlat),!,prove2(GammaFlat,DeltaFlat,X).
```

Then our initial goal would be of the form *prove(Gamma,Delta,X,L)*, but all the predicates of our program except the one above will have the head *prove**2**(Gamma,Delta,X,L):-..*

Afterward, we added a fourth parameter to the function to make sure the same sequent does not show itself more than once in the same branch:

```
1 prove2(Gamma,Delta,_,L):-member((Gamma,Delta),L),!,fail.
```

We then had to make sure that the same rule does not appear twice in the same sequent or in the same block:

```
1 check(A):-atom(A).
2 check([]).
3 check([A|GammaTail]):-not(member(A,GammaTail)),check_sup(A),
      check(GammaTail).
4 check_sup(A):-atom(A),!.
5 check_sup(A):-compound(A),not(is_list(A)),functor(A,<<,_),arg(1,
      A,X),!,check_spec(X).
6 check_sup(A):-compound(A),not(is_list(A)).
7 check_sup(A):-compound(A),is_list(A),check(A).
8 check_spec(X):-atom(X).
9 check_spec(L):-flatten(L,L1),check(L1).
```

We make this check individually on each part of the sequent. Basically, these few lines defines a recurisve function that verifies first that there is not twice the same formula in the sequent. Then, it checks wether the formula inspected is a block or not. If it is, it verifies each component (if it is a list, if there are repetitions, etc..), if it is not, it validates the check for this formula.
This check is only done one time per node in order to save compilation time. It is located in the program just before the invertible rules (the rules that are represented with the exposant "i"). In fact this kind of verification is only usefull for these rules because they are the only one that actually need it. Indeed, these are the only rules enclined to dupplicate formulae through their application. Besides, we also needed to execute those verifications after the axioms because it may cause some complications with the *Init axiom*.

Finally we added an inner check to the *init* atom to discriminate the empty list [ ]:

```
1 verif(P):-atom(P),not(is_list(P)),!.
2 verif(P):-compound(P),!.
3 prove2(Gamma,Delta,axini,_):-member(P,Gamma),verif(P),member(P,
    Delta),!.
```

3.1.6 Final output

We finally get an output in the form of a tree, of which each node is a rule applied on the formula.
For instance, for the axiom CPA,

```
1  ?- prove([],[((a<((a->bot)->b))->bot)->(b<((a->bot)->b))],X,[]).
2  X = impR( plR( impL(
3                 axbot,
4                 plR( com(
5                         jump( impL(
6                                 axini,
7                                 impR( axini))),
8                         jump( impL(
9                                 axini,
10                                impR( axini)))))))
```

On the first line we can see the goal entered, with the four arguments, and we ask for the third thanks to the variable X. Which is what the program

answers us on the second line.
As for the extensions, with the axiom T we obtain:

```
?- prove([],[(bot<(a->bot))->a],X,[]).
X = impR( c(
            axbot,
            impR( axini))).
```

We can see we only need the logic $\mathbb{C}$ to prove it but we can also do it with the logics $\mathbb{N}$ and $\mathbb{W}$ instead:

```
?- prove([],[(bot<(a->bot))->a],X,[]).
X = impR( n( plL(
              n( w( w( impR( axini)))),
              jump( axbot)))).
```

We observe here that several useless steps are added due to the rule N. However, the computation comes to an end, the result is logical, and not far from the more efficient computation we could be looking for.

3.2 Automatization

The first problem is that we want a simpler way to use this theorem prover, comment and uncomment the extensions we want to use or not is way too complicated for a customer.
The simplest way would be to use a different file for each logic, and each combination of logics ($\mathbb{VNA}$,$\mathbb{VTA}$,..).

Next, the output we get through SWIPL is not explicit at all, it would be a lot better and understandable to have the SLD-tree already printed.
To do this, we not only need the rules applied during the derivation, but also the sequents on which it were applied, and we can get them thanks to our fourth argument, a list of the sequents.
We therefore needed a script to adapt the results of this program to something readable. To make something adapted to an online implementation, accessible for all, the language PHP would for instance be a good programming linguage for this kind of script. Besides, SWIPL also proposes its own web server, programmable in Prolog.

Conclusion

We presented in short terms the evolution of the conditional logics through history. Which lead us to be able to understand the Lewis model we overviewed through a short synthesis. Afterward, we introduced the Prolog language, used in logic programming, subparadigm of the declarative programming. And, finally, we presented the implementation of our theorem-prover for Lewis' conditional logics using this Prolog language. For this implementation we made the choice of focusing on the invertible calculus, as it is equivalent to the non-invertible one. We introduced the particular syntax of the blocks in order to make the model implementable. Starting to implement the logic $\mathbb{V}$, base of Lewis logics, we then added the extensions and, through multiple verifications, we finally obtained the implementation as it is now. We developed this program with the goal of having it well functionning, and therefore did not focus particularly on its efficientness, letting it open for future improvements. Besides, we get interesting results and moreover, this is the first theorem prover for Lewis' logics.
We aimed, through this project, to provide some help to the progress in artificial intelligence, as Lewis' logics is one of the accurest modelisations of logics you may find in the common language, and thus could be adaptable for any modelisation of reasoning in the artificial intelligence field.

We want this prover to be accessible and improvable by whoever is interested in it, therefore, we created a GitHub to get the Prolog files and the project: https://github.com/vitalisquentin/LeThProve

List of Tables

List of Figures

Bibliography

[1] *Handbook of Philosophical Logic, vol. 4*. Springer, 2002.

[2] *Handbook of Philosophical Logic, vol. 14*. Springer, 2007.

[3] Agostino Dovier and Andrea Formisano. *Programmazione Dichiaritiva in Prolog, CLP e ASP*. PhD thesis, Università di Perugia, 2007.

[4] Sextus Empiricus. *Adversus Mathematicos VIII, Pros logikous, Section 113*.

[5] G. Frege. Sense and reference. *The Philosophical Review*, pages 209–230, 1948.

[6] Gerhard Karl Erich Gentzen. Untersuchungen über das logische schließen. i. *Mathematische Zeitschrift*, pages 176–210, 1934.

[7] Gerhard Karl Erich Gentzen. Untersuchungen über das logische schließen. ii. *Mathematische Zeitschrift*, pages 405–431, 1935.

[8] M Girlando, B. Lellmann, N. Olivetti, and G.L. Pozzato. Standard sequent calculi for lewis' logics of counterfactuals. 2016.

[9] Georges Gonthier. Formal proof - the four-color theorem. *Notices of the Amercian Mathematical Society*, pages 1382–1393, 2008.

[10] Robert A. Kowalski. Predicate logic as a computational formalism. *Proceedings of IFIP 74*, pages 569–574, 1974.

[11] David K. Lewis. *Counterfactuals*. Blackwell and Harvard University Press, 1973.

[12] Donald Nute. *Topics in Conditional Logic*. Springer, 1980.

[13] N. Olivetti and G.L. Pozzato. A standard and internal calculus for lewis counterfactual logics. *Proceedings of the 22nd Conference on Automated Reasoning with Analytic Tableaux and Related Methods - Lecture Notes in Artificial Intelligence LNAI, vol. 9323*, pages 270–286, sep 2015.

[14] Willard Van Orman Quine. *Methods of Logic*. Harvard University Press, 1950.

[15] Frank P. Ramsey. On a problem in formal logic. *Proceedings of the London Mathematical Society*, pages 264–286, 1929.

[16] Frank P. Ramsey. *Foundations - Essays in Philosophy, Logic, Mathematics and Economics, Law and Causality*. Humanities Press, 1931.

[17] B. Russell. On denoting. *Mind*, pages 479–493, 1905.

[18] A.J. Sanford and S. Garrod. What, when and how: Questions of immediacy in anaphoric reference resolution. *Language and Cognitive Processes*, pages 235–263, 1989.

[19] George Spencer-Brown. *Laws of Form, Appendix 5*. Lübeck, 1997.

[20] Robert C. Stalnaker. A theory of conditionals. *American Philosophical Quarterly*, pages 98–112, 1968.

[21] David H. Warren. Logic programming and compiler writing. *Software - Practice and Experience*, pages 97–125, feb 1980.