Mathematical Logic

** I don't take any credit for the material in this file. This is just for my own reference.*

Contents

1 Propositional Logic

A disjunction of literals $L_1 \vee \ldots \vee L_m$ is valid iff there are $1 \leq i, j \leq m$ such that L_i is $\neg L_j$.

Let ϕ be a formula of propositional logic. Then ϕ is satisfiable iff $\neg\phi$ is not valid.

A formula is in **negation normal form (NNF)** if negation is only applied to variables and $\neg, \wedge, \vee$ are the only connectives.

A clause is a **Horn clause** if it contains at most one positive literal.
A **Horn formula** is a conjunction of horn clauses (thus a special case of CNF).
Horn clauses can always be rewritten with exactly one positive literal (and some negative literals), using the constants $\bot, \top$.
Horn clauses can be represented as $(p_1 \wedge \ldots \wedge p_n) \to p_{n+1}$.

The algorithm for finding the unique minimal satisying truth assignment of a Horn formula ϕ:

- Mark all occurences of T true.

- If $p_1 \wedge \ldots \wedge p_k$ are all marked true and $(p_1 \wedge \ldots \wedge p_k) \rightarrow p$ is one of the conjuncts in ϕ, then mark p true. Repeat until a fixpoint is reached (i.e. until there is no more change).
- Make the remaining atoms false.

  ϕ is satisfiable iff $\bot$ is not marked true. If ϕ is satisfiable, the algorithm returns the minimal satisfying truth assignment (in terms of the number of atoms that are set true).

$\Gamma \nvDash \psi$ is easier to prove than $\Gamma \vDash \psi$ because in the former case we need only one model, while in the latter we potentially have to consider infinitely many.
$\Gamma \vdash \psi$ is easier to prove than $\Gamma \nvdash \psi$ because in the former case we need only a proof of ψ from Γ, while in the latter we potentially have to consider all proofs using Γ.

1.1 Propositional Modal Logic

Modal logic is a formalism that extends propositional and first order logic by adding operators to the language to express modalities (epistemic, doxastic, temporal, ...). In this report we will focus on propositional modal logic.
Modal logic is more expressive than first-order logic and corresponds to a fragment of second-order logic. For example the so-called McKinsey schema $M: \Box\Diamond A \rightarrow \Diamond\Box A$ is not defined by any set of first order sentences.

1.1.1 Syntax

Given a set of modality symbols MOD, the language of (propositional) modal logic ($\mathcal{L}_M$) is inductively defined using the following BNF ($m \in$ MOD)[Blackburn and Benthem, 2007]:

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \Box_m\varphi$$

Denotation of $\Box_m\varphi$ depends on the context. For example in context of reasoning about beliefs, $\Box_m\varphi$ denotes "agent m believes that φ". The symbol $\Diamond$ is defined as $\Diamond\varphi \equiv \neg\Box\neg\varphi$.

1.1.2 Semantics

The semantics of modal logic is based on the concept of possible worlds. These worlds comprise of the actual world and alternative worlds representing different ways the world could have been. The idea of possible worlds dates back to the German philosopher Gottfried Leibniz but its usage in modal logic and development of possible world semantics (described below) is attributed to Saul Kripke.
Given a set W of possible worlds and a set $\{R^m\}_{m\in\text{MOD}}$ of the accessibility relations between worlds, a *frame* is a tuple $\langle W, \{R^m\}_{m\in\text{MOD}}\rangle$. A *(Kripke) model* M is a triple $\langle W, \{R^m\}_{m\in\text{MOD}}, V\rangle$ where V is a valuation function mapping each propositional variable to a subset of W of worlds in which p is true. If $|\text{MOD}| = 1$, we write R instead of $\{R^m\}_{m\in\text{MOD}}$ and $\Box$ instead of $\Box_m$. [Blackburn and Benthem, 2007]

Given a model $M = \langle W, \{R^m\}_{m\in\text{MOD}}, V\rangle$ and $w \in W$, definition of φ being *true* (or *satisfied*) in M at w is as follows [Blackburn and Benthem, 2007]:

$$\begin{array}{lll}
M, w \vDash p & \text{iff} & w \in V_p \\
M, w \vDash \neg p & \text{iff} & M, w \nvDash p \\
M, w \vDash \varphi \wedge \psi & \text{iff} & M, w \vDash \varphi \text{ and } M, w \vDash \psi \\
M, w \vDash \Box_m\varphi & \text{iff} & \text{for all } v \in W \text{ such that } wR^m v \text{ we have } M, v \vDash \varphi
\end{array}$$

A formula φ is *true* in M ($M \vDash \varphi$), if $M, w \vDash \varphi$ for all $w \in W$. A formula is *valid* ($\vDash \varphi$) if it is true at every model. [Chellas, 1980]

1.1.3 Proof Theory of Modal Logic

Consider the set V of formulas that are valid in all Kripke models. Many of these formulas are "related" to each other. For example the formulas $p \vee \neg p$ and $\Box(p \vee \neg p)$ are both in V and are related in the sense that based on the fact that $p \vee \neg p$ is in V and based on the semantics of $\Box$, we expect that $\Box(p \vee \neg p)$ must be in V as well. So the question is "can we come up with a set of formulas $F \subset V$ and a set of rules R of producing new formulas such that F and R together produce V?" It turns out the answer is yes, and the following F and R do the job. Define F as the set of all the tautologies of propositional logic plus the following formula (axiom K)

$K.\quad \Box(\varphi \supset \psi) \supset (\Box\varphi \supset \Box\psi)$

And let R consist of the following rules

Modus Ponens. From φ and $\varphi \supset \psi$ infer ψ

Necessitation. From φ infer $\Box\varphi$

F and R together are called "the basic modal system K" [van Ditmarsch et al., 2007].

There are some formulas that are not valid in all Kripke models but we expect them to be among valid formulas in specific modal systems. For example if we use $\Box\varphi$ to denote "φ is known", we want the formula $\Box\varphi \supset \varphi$ always hold. Or if $\Box\varphi$ denotes "φ is believed" then we want the formula $\neg\Box\bot$ always hold. The question is "if we add these formulas (axioms) to the set F defined above (call the resulting set F'), can we apply some restrictions on Kripke model so that the set V' of valid formulas in the restricted Kripke models is the same as the set of formulas produced by F' and R?" It turns out this is possible. Consider the following formulas/axioms (these are the axioms that are related to the subject of this report):

$T.\ \Box\varphi \supset \varphi$

4. $\Box\varphi \supset \Box\Box\varphi$

5. $\neg\Box\varphi \supset \Box\neg\Box\varphi$

$D.\ \neg\Box\bot$

Adding axiom T to F corresponds to setting the restriction on Kripke models that their accessibility relation be reflexive. 4 corresponds to transitive relations, 5 corresponds to euclidean relations and D corresponds to serial relations.
Two modal axiom systems which are relevant to the subject of this report are $KT45$[1] which is called S_5 and is regarded as logic of knowledge (epistemic logic), and $KD45$ which is regarded as logic of belief. S_5 captures the class of Kripke models with equivalence relations (reflexive/euclidean/transitive or equivalently reflexive/symmetric/transitive) and $KD45$ captures the class of serial transitive Euclidean models. [van Ditmarsch et al., 2007]
An *epistemic model* is a (Kripke) model where the relation is an equivalence relation. Also it is common to use S (set of states) instead of W (set of worlds). So an epistemic model is represented as $M = \langle S, \{\sim_a\}_{a\in \mathrm{A}}, V\rangle$, where A is the set of agents. It is common to use the operator K_m instead of $\Box_m$ and $\hat{K}_m$ for $\Diamond_m$. The formula $K_a\varphi$ denotes "agent a knows that φ" and $\hat{K}_a\varphi$ denotes "φ is consistent with a's knowledge". Similarly for logic of belief we use $B_a\varphi$ and $\hat{B}_a\varphi$ with natural denotations.

[1]This is the basic modal system K extended by adding the axioms T, 4, and 5.

1.1.4 Temporal Logic of Concurrency

Given a language based on a countable set Φ of atomic formulae, a **logic** is defined to be any set $\Lambda \subseteq Fma(\Phi)^2$ such that

- Λ includes all tautologies
- Λ is closed under the rule of detachmennt, i.e, if $A, A \to B \in \Lambda$ then $B \in \Lambda$.

A logic is **normal** if it contains the schema $K: \ \Box(A \to B) \to (\Box A \to \Box B)$, and is closed under the rule of Necessitation, i.e, if $\vdash_\Lambda A$ then $\vdash_\Lambda \Box A$.[3]

Consider a propositional language with two modal connectives $[F]$ and $[P]$. A **frame** for this language has the form (S, R_F, R_P) with the modelling:

$$\begin{array}{llll} \mathcal{M} \models_s [F]A & \text{iff} & sR_Ft & \text{implies} \quad \mathcal{M} \models_t A \\ \mathcal{M} \models_s [P]A & \text{iff} & sR_Pt & \text{implies} \quad \mathcal{M} \models_t A \end{array}$$

We want sR_Pt iff tR_Fs so that $[F]$ and $[P]$ express properties of the same time-orderig. We can capture this property by the following schemata:

$$C_P : A \to [P]\langle F\rangle A$$
$$C_F : A \to [F]\langle P\rangle A$$

With this property R_F and R_P are interdefinable, so we may take one relation as primitive, and use frames $\mathcal{F} = (S, R)$, where $R \subseteq S \times S$, with the modelling:

$$\begin{array}{llll} \mathcal{M} \models_s [F]A & \text{iff} & sRt & \text{implies} \quad \mathcal{M} \models_t A \\ \mathcal{M} \models_s [P]A & \text{iff} & tRs & \text{implies} \quad \mathcal{M} \models_t A \end{array}$$

It is also natural to require a temporal ordering to be transitive. This property can be captured by the following schemata:

$$4_P : [P]A \to [P][P]A$$
$$4_F : [F]A \to [F][F]A$$

So we define a **time-frame** to be any structure $\mathcal{F} = (S, R)$ with R a transitive relation on S and with the modelling just given.

A **temporal logic** is any normal logic in the language of $[F]$ and $[P]$ that contains the schemata: $C_P, C_F, 4_P, 4_F$.

The smallest temporal logic which is $K_{\{P,F\}}C_PC_F4_P4_F$ is commonly known as K_t in the literature (only one of 4_P and 4_F is required in the definition of K_t; the other one is derivable from the remaining axioms).

It is useful to introduce the connective $\Box$ by

$$\Box A := [P]A \wedge A \wedge [F]A$$

The connective $\Box$ behaves like a S_5 modality in connected frames[4].

[2] The set of all formulae generated from Φ.

[3] $\vdash_\Lambda A$ iff $A \in \Lambda$

[4] A frame is **connected** if it satisfies $\forall s \forall t(sRt \vee s = t \vee tRs)$. There is no schema that is valid in precisely the connected frames.

A **linear temporal logic** is any normal logic in the language of $[F]-[P]$ that contains the smallest temporal logic K_t, and also the schemata

$$\Box A \to [P][F]A$$
$$\Box A \to [F][P]A$$

Temporal Logic of Concurrency

Syntax:

$$A ::= p \mid \bot \mid A_1 \to A_2 \mid \Box A \mid \bigcirc A \mid A_1 \mathcal{U} A_2$$

A *state sequence* is a pair $\mathcal{F} = (S, \sigma)$, where σ is a surjective function $\omega \to S$ enumerating S as a sequence $\sigma_0, \sigma_1, \ldots, \sigma_n, \ldots$ possibly with repitition (for example when S is finite).

A *model* $\mathcal{M} = (S, \sigma, V)$ on a state sequence is defined in the usual way.

The relation $\mathcal{M} \models_j A$ meaning "A is true at the jth state σ_j in $\mathcal{M}$" is defined by

$$\begin{array}{lll} \mathcal{M} \models_j p & \text{iff} & \sigma_j \in V(p) \\ \mathcal{M} \models_j \bigcirc A & \text{iff} & \mathcal{M} \models_{j+1} A \\ \mathcal{M} \models_j \Box A & \text{iff} & \text{for all } k \geq j,\ \mathcal{M} \models_k A \end{array}$$

The definitions of $\mathcal{M} \models A$ and $\mathcal{F} \models A$ are as usual.

1.1.5 Linear-time Temporal Logic (LTL)

Syntax:

$$\phi :: \top \mid \bot \mid \mathit{Classical} \mid X\phi \mid F\phi \mid G\phi \mid \phi U \phi$$

A **transition system** $\mathcal{M} = (S, \to, L)$ is a set of states S endowed with a transition relation $\to$ (a binary relation on S), such that every $s \in S$ has some $s' \in S$ with $s \to s'$, and a labelling function $L : S \to \mathcal{P}(\mathit{Atoms})$. Transition systems are also called **models**.

A **path** π in a model $\mathcal{M} = (S, \to, L)$ is an infinite sequence of states $s_1, s_2, \ldots$ in S such that for all $i \geq 1, s_i \to s_{i+1}$. We write the path as $s_1 \to s_2 \to \ldots$. We write π^i for the path starting at s_i.

Let $\mathcal{M} = (S, \to, L)$ be a model and $\pi = s_1, s_2, \ldots$ be a path in $\mathcal{M}$. Whether π satisfies an LTL formula is defined by the satisfaction relation as follows:

$$\begin{array}{lll} \pi \models \top & & \\ \pi \not\models \bot & & \\ \pi \models p & \text{iff} & p \in L(s_1) \\ \vdots \text{ (classical)} & & \\ \pi \models X\phi & \text{iff} & \pi^2 \models \phi \\ \pi \models G\phi & \text{iff} & \text{for all } i \geq 1\ \ \pi^i \models \phi \\ \pi \models F\phi & \text{iff} & \text{there exists } i \geq 1 \text{ s.t. } \pi^i \models \phi\ (F\phi \equiv \top U \phi) \\ \pi \models \phi U \psi & \text{iff} & \text{there exists } i \geq 1 \text{ s.t. } \pi^i \models \psi \text{ and for all } j \in \{1, \ldots, i-1\}\ \ \pi^j \models \phi \end{array}$$

Suppose $\mathcal{M} = (S, \to, L)$ is a model, $s \in S$, and ϕ an LTL formula. We write $\mathcal{M}, s \models \phi$ if for any execution path π of $\mathcal{M}$ starting at s, we have $\pi \models \phi$.

1.1.6 Computation Tree Logic(CTL)

Syntax:

$$\phi :: Classical \mid AX\phi \mid EX\phi \mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \mid A[\phi U \phi] \mid E[\phi U \phi]$$

Safety: Only one process is in its critical section at any time.
LTL: $G(\neg(c_1 \wedge c_2))$
CTL: $AG(\neg(c_1 \wedge c_2))$

Liveness: Whenever a process requests to enter its critical section, it will eventually be permitted to do so.
LTL: $G(t_1 \to Fc_1) \wedge G(t_2 \to Fc_2)$
CTL: $AG(t_1 \to AFc_1) \wedge AG(t_2 \to AFc_2)$

Non-blocking A process can always request to enter its critical section.
LTL: —
CTL: $AG(n_1 \to EXt_1) \wedge AG(n_2 \to EXt_2)$

No strict sequencing: Processes need not enter their critical section in strict sequence.
LTL: $\neg[G(c_1 \to c_1 W(\neg c_1 \wedge \neg c_1 W c_2))]$
CTL: $EF(c_1 \wedge E[c_1 U(\neg c_1 \wedge E[\neg c_2 U c_1])])$

Fairness constraint: Under certain conditions an event will occur (or fail to occur) infinitely often.

1.1.7 CTL*

The LTL formula ϕ equal to the CTL* formula $A[\phi]$.

$E[\phi \vee \psi] = E[\phi] \vee E[\psi]$
$A[\phi \wedge \psi] = A[\phi] \wedge A[\psi]$

1.1.8 Epistemic Logic

Given a countable set of atomic propositions P and a finite set of agents A,
an **epistemic model** is a structure $M = \langle S, \{\sim_a\}_{a \in \mathrm{A}}, V \rangle$, where

- S is a set of states.
- $\sim_a$ is a function, yielding for every $a \in A$ an equivalence relation $\sim_a \subseteq S \times S$.
- $V : P \to 2^S$ is a valuation function that for every $p \in P$ yields the set $V(p) \subseteq S$ of states in which p is true.

The language of epistemic logic (L_K) is inductively defined by the BNF

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a\varphi$$

The semantics of epistemic logic is as follows:

$$\begin{array}{lll} M, s \vDash p & \text{iff} & s \in V_p \\ M, s \vDash \neg\varphi & \text{iff} & M, s \nvDash \varphi \\ M, s \vDash \varphi \wedge \psi & \text{iff} & M, s \vDash \varphi \text{ and } M, s \vDash \psi \\ M, s \vDash K_a\varphi & \text{iff} & \text{for all } t \in S : s \sim_a t \text{ implies } M, t \vDash \varphi \end{array}$$

We say that φ is true in M (written $M \vDash \varphi$), if $M, s \vDash \varphi$ for all $s \in S$.
Formula $K_a\varphi$ stands for 'agent a know that φ'.

1.1.9 Public Announcement Logic

Given an epistemic model $M = \langle S, \sim, V \rangle$,

The language of the public announcement logic ($L_{K[]}$) is inductively defined by the BNF

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a\varphi \mid [\varphi]\varphi$$

The semantics of the public announcement logic is as follows:

$$\begin{array}{lll} M, s \vDash p & \text{iff} & s \in V_p \\ M, s \vDash \neg\varphi & \text{iff} & M, s \nvDash \varphi \\ M, s \vDash \varphi \wedge \psi & \text{iff} & M, s \vDash \varphi \text{ and } M, s \vDash \psi \\ M, s \vDash K_a\varphi & \text{iff} & \text{for all } t \in S : s \sim_a t \text{ implies } M, t \vDash \varphi \\ M, s \vDash [\varphi]\psi & \text{iff} & M, s \vDash \varphi \text{ implies } M|\varphi, s \vDash \psi \end{array}$$

Where $M|\varphi = \langle S', \sim', V' \rangle$ is defined as follows: ($[\![\varphi]\!]_M = \{s \in S \mid M, s \vDash \varphi\}$)

$$\begin{aligned} S' &= [\![\varphi]\!]_M \\ \sim'_a &= \sim_a \cap ([\![\varphi]\!]_M \times [\![\varphi]\!]_M) \\ V'_p &= V_p \cap [\![\varphi]\!]_M \end{aligned}$$

That is to say, those states which announcement formula is false in them are excluded from model as a result of announcement, because after announcement it is commonly known that these states are no longer possible.

Formula $[\varphi]\psi$ stands for 'after announcement of φ, it holds that ψ'. It is assumed that announcements are *truthful* and *public*. That the announcement is public, means that all agents can hear the announcement, that they know that the other agent can hear the announcement, that they know that the other agents know that the other agents can hear the announcement, etc., ad infinitum.

1.2 Natural Deduction

1.2.1 Natural Deduction in Sequent Form

1.3 Sequent Calculus

2 First Order Logic

2.1 Syntax

A first-order language with equality is specified by two disjoint sets of symbols, called the **vocabulary** of the language:

1. Logical symbols: The interpretation of these is fixed by the rules of first-order logic.
   - Parentheses of all shapes and sizes
   - Logical connectives: $\supset, \neg$
   - Variables (infinitely many): $x, y, z, x_1, y_1, z_1, \ldots$
   - Equality symbol: $=$
2. Non-logical symbols (parameters/**signature**): These vary with the interpretation.
   - Quantifier symbol: $\forall$.

- Predicate symbols: For each $n \geq 0$, a set (possibly empty) of symbols, called n-place or n-ary predicate symbols.
- Function symbols: For each $n \geq 0$, a set (possibly empty) of symbols, called n-place or n-ary function symbols. 0-ary function symbols are called constant symbols.

There are two types of legal syntactic expressions in FOL: terms and formulas. Intuitively, a term will be used to refer to something in the world, and a formula will be used to express a proposition. The set of **terms** of FOL is the least set satisfying these conditions:

- Every variable is a term;
- If $t_1, \ldots, t_n$ are terms, and f is a function symbol of arity n, then $f(t_1, \ldots, t_n)$ is a term.

The set of **formulas** of FOL is the least set satisfying these constraints:

- If $t_1, \ldots, t_n$ are terms, and P is a predicate symbol of arity n, then $P(t_1, \ldots, t_n)$ is a formula;
- If t_1 and t_2 are terms, then $t_1 = t_2$ is a formula;
- If α and β are formulas, and x is a variable, then $\neg\alpha, (\alpha \wedge \beta)$ and $\forall x.\alpha$ are formulas.

2.2 Semantics

a structure (or interpretation) $\mathcal{S}$ for a given first-order language is a function whose domain is the set of parameters of the language, and is defined by:

- $\forall^{\mathcal{S}}$ is a nonempty set, called the universe or the domain of the structure $\mathcal{S}$. The universe is usually written $|\mathcal{S}|$.
- For each n-ary predicate symbol P of the language, $P^{\mathcal{S}} \subseteq |\mathcal{S}|^n$. This is called the extension of P in the structure S.
- For each n-ary function symbol f of the language, $f^{\mathcal{S}}$ is an n-ary function on $|\mathcal{S}|$, i.e. $f^{\mathcal{S}} : |\mathcal{S}|^n \rightarrow |\mathcal{S}|$. In particular, when $n = 0$, so that f is a constant symbol, $f^{\mathcal{S}}$ is simply some element of the universe.

It is sometimes useful to think of the interpretation of predicates in terms of their characteristic functions. In this case, when P is a predicate of arity n, we view $P^{\mathcal{S}}$ as an n-ary function to $\{0, 1\}$. This characteristic function allows us to see more clearly how predicates of arity 0 (i.e., the propositional symbols) are handled. In this case, $P^{\mathcal{S}}$ will be either 0 or 1. We can think of the first one as meaning "false" and the second "true". For the propositional subset of FOL, then, we can ignore D completely, and think of an interpretation as simply being a mapping, from the propositional symbols to either 0 or 1.

Truth in a Structure

Suppose $s : V \rightarrow |\mathcal{S}|$ called a **variable assignment**, is defined on the set V of variables.
Define an extension $\bar{s} : T \rightarrow |\mathcal{S}|$ of the function s, on the set T of all terms as follows

- For each variable v, $\bar{s}(v) = s(v)$
- If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, then
  $\bar{s}(f(t_1, \ldots, t_n)) = f^{\mathcal{S}}(\bar{s}(t_1), \ldots, \bar{s}(t_n))$.

We can now define $\models_{\mathcal{S}} \phi[s]$, meaning that the formula ϕ is true in the structure $\mathcal{S}$ when its variables are given the values specified by s:

$$\begin{array}{lll}
\models_{\mathcal{S}} t_1 = t_2[s] & \text{iff} & \bar{s}(t_1) = \bar{s}(t_2) \\
\models_{\mathcal{S}} P(t_1, \ldots, t_n)[s] & \text{iff} & \langle \bar{s}(t_1), \ldots, \bar{s}(t_n) \rangle \in P^{\mathcal{S}} \\
\models_{\mathcal{S}} \neg\phi[s] & \text{iff} & \text{not } \models_{\mathcal{S}} \phi[s] \\
\models_{\mathcal{S}} (\phi \supset \psi)[s] & \text{iff} & \models_{\mathcal{S}} \neg\phi[s] \text{ or } \models_{\mathcal{S}} \psi[s] \\
\models_{\mathcal{S}} (\forall x)\phi[s] & \text{iff} & \text{for every } d \in |\mathcal{S}|,\ \models_{\mathcal{S}} \phi[s(x|d)]
\end{array}$$

Here, $s(x|d)$ is the function that is exactly like s except that for the variable x it assigns the value d.

2.3 Many-Sorted Logic

3 Second Order Logic

- Informally, a relation R will be second-order (but not first-order) definable whenever, it is defined inductively using the pattern:
"R is the smallest set such that ..." or
"R is the intersection of all sets such that ..."

4 Special Topics

4.1 Negation as Failure[5]

A relation instance is false if we *fail* to prove that it is true. To show that P is false we do an exhaustive search for a proof of P. If every possible proof fails[6], $\sim P$ is 'infered'. This is the way both PLANNAR (Hewwitt[1972]) and PROLOG (Roussel[1975], Warren et al.[1977]) handle negation.

So it is basically a proof rule:

$$\vdash (\sim \vdash P) \quad \text{infer} \quad \vdash \sim P$$

where the proof that P is not provable is always the exhaustive but unsuccessful (i.e. failed) search for a proof of P.

Note that to assume that a relation instance is false if it is not implied, is to assume that the database gives complete information about the true instances of its relations. This is the closed world assumption reffered to by Reiter[1978] and Nicolas and Gallaire[1978].

The Query Evaluation Process is essentially a linear resolution proof procedure with negated literals 'evaluated' by a failure proof. However we shall view the alternate derivations of the search space as different paths of a non-deterministic evaluation which can SUCCEED, FAIL or not terminate. A path terminates with SUCCESS if its terminal query is the empty query. A path terminates with FAILURE if the selected literal of its terminal query does not unify with the consequent literal of the selected database clause. The subsequent selection of a database clause, and the attempted unification is a non-deterministic step of the evaluation. Finally a non-terminating evaluation path comprises an infinite sequence of queries each of which is derived from the initial query as described below.

[5]Keith C. Clark, *Negation as Failure*, Logic and Databases, 1978
[6]This has a special meaning defined later

Evaluation Algorithm: Until an empty query is derived, which the evaluation succeeds, proceed as follows: Select a literal L_i from the current query $\leftarrow L_1\& \dots \& L_n$. The selection rule is constrained so that a negative literal is only selected if it contains no variables.

- Case 1: L_i is a positive literal $R(\vec{t})$.
  Non-deterministically choose a database clause about R

  $$R(\vec{t'}) \leftarrow L'_1\& \dots \& L'_m$$

  and try to unify L_i with $R(\vec{t'})$. If L_i doesn't unify with $R(\vec{t'})$, FAIL (this path). If L_i does unify, replace the current query with the derived query

  $$\leftarrow \{L_1\& \dots \& L_{i-1} \;\&\; L'_1 \dots \& L'_m \;\&\; L_{i+1}\& \dots \& L_n\}\theta$$

  If there is no database clause about the relation of the selected clause, FAIL.

- Case 2: L_i is a negative ground literal $\sim P$.
  Attempt to discover whether $\sim P$ can be assumed as a lemma, i.e. recursively enter the query evaluation with $\leftarrow P$ as a query. If the evaluation of $\leftarrow P$ SUCCEEDS, FAIL. If the evaluation of $\leftarrow P$ FAILS for every path of its non-deterministic evaluation, assume $\sim P$ as a lemma. Hence replace the current query by

  $$\leftarrow L_1\& \dots \& L_{i-1} \;\&\; L_{i+1}\& \dots \& L_n$$

Examples (propositional for simplicity):
$KB : \{a \leftarrow b\}$
$Query : a$
We resolve $\leftarrow a$ with $a \leftarrow b$, and we get $\leftarrow b$ (new query after replacement). There is no rule in KB with b in the head so FAIL this path. There is no other path so, FAIL.

$KB : \{a \leftarrow b, b \leftarrow a\}$
$Query : a$
We resolve $\leftarrow a$ with $a \leftarrow b$, and we get $\leftarrow b$. Then we resolve $\leftarrow b$ with $b \leftarrow a$, and we get $\leftarrow a$. Again resolve with $a \leftarrow b$. This is a non-terminating path. So we cannot decide whether a is inferred.

$KB : \{a \leftarrow \sim b\}$
$Query : a$
We resolve $\leftarrow a$ with $a \leftarrow b$, and we get $\leftarrow \sim b$. Now the query is $\sim b$ (second case in the algorithm). We continue with $\leftarrow b$. There is no rule in KB with b in the head so this path FAILS. There is no other path so, FAIL for every path. Hence we get empty query, so SUCCEED.

4.2 Answer Set Programming

- Inference-based approach (e.g. resolution in logic, top-down rule-based reasoning, Prolog): Provide a specification of the problem. A solution is given by a derivation of an appropriate query.
- Model-based approach (e.g. ASP, also SAT):
  Provide a specification of the problem. A solution is given by a model of the specification.
- Rules represent constraints on the program.
- A **(normal) rule**, r , is of the form

  $$a_0 \leftarrow a_1, \dots, a_m, not\ a_{m+1}, \dots, not\ a_n.$$

- A **(normal) logic program** is a finite set of rules.
- Given a set of atoms X from Π, Π^X is obtained from Π by deleting each rule having a *notA* in its body with $A \in X$ and then all negative atoms of the form *notA* in the bodies of the remaining rules. Then X is an **answer set** of Π just if Π^X "generates" X, i.e. $Cn(\Pi^X) = X$. An answer set is a minimal set of atoms satisfying the rules.

- The program $\{p \leftarrow \mathit{not}\ p\}$ has no answer sets.
- An **integrity constraint** is of the form

$$\leftarrow a_1, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n.$$

  It can be translated into a normal rule by mapping to to (x a new symbol)

$$x \leftarrow a_1, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n, \mathit{not}\ x.$$

- A **choice rule** is of the form

$$\{a_1, \ldots, a_m\} \leftarrow a_{m+1}, \ldots, a_n, \mathit{not}\ a_{n+1}, \ldots, \mathit{not}\ a_o$$

  The idea of a choice rule is to express choices over subsets of atoms. Any subset of its head atoms can be included in a stable model, provided the body literals are satisfied. Thus, for instance, the program $P = \{a \leftarrow, \{b\} \leftarrow a\}$ has two stable models, $\{a\}$ and $\{a, b\}$. For another example, at a grocery store you may or may not buy pizza, wine, or corn.
  `{buy(pizza), buy(wine), buy(corn)} :- at(grocery).`
- A **cardinality rule** is of the form

$$a_0 \leftarrow l\{a_1, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n\}u.$$

  Cardinality rules allow for controlling the cardinality of subsets of atoms via the lower bound l and upper bound u.
- A **conditional literal** is of the form

$$l : l_1 : \ldots : l_n$$

  The purpose of this construct is to govern the instantiation of the head literal l through the literals $l_1, \ldots, l_n$. In this respect, a conditional literal $l : l_1 : \ldots : l_n$ can be regarded as the list of elements in the set $\{l | l_1, \ldots, l_n\}$.
  For example, given three facts `color(red)`, `color(green)`, and `color(blue)`, the integrity constraint

```
:- color(v42,C) : color(C).
```

  results in

```
:- color(v42,red), color(v42,green), color(v42,blue).
```

- Classical negation: We can add classical negation to ASP (for atoms only) but it does not add to its expressive power.
  we extend our set of atoms $\mathcal{A}$ by $\overline{\mathcal{A}} = \{\neg a | a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \varnothing$. That is, $\neg a$ is the classical negation of a and vice versa. The semantics of classical negation is enforced by the addition of the following set of rules

$$P^{\neg} = \{\leftarrow b, \neg b \mid b \in \mathcal{A}\}$$

References

[Blackburn and Benthem, 2007] Blackburn, P. and Benthem, J. V. (2007). Moda logic: A semantic perspective. In Patrick Blackburn, J. V. B. and Wolter, F., editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*. Elsevier.

[Chellas, 1980] Chellas, B. (1980). *Modal Logic*. Cambridge University Press.

[van Ditmarsch et al., 2007] van Ditmarsch, H., van der Hoek, W., and Kooi, B. (2007). *Dynamic Epistemic Logic*. Springer.

Vahid Vaezian (vvaezian [at] sfu.ca), February 3, 2018