DM de Logique

Aloïs R., Romain R., Paul A.

April 2023

1 Relèvement au premier ordre

Question 1.A. Soit E une clause universelle non satisfiable (de logique du premier ordre). Montrons que l'on a $E \vdash_{RFI} \bot$.

D'après le théorème de Herbrand, E est satisfiable si et seulement si E est Herbrand-satisfiable, au sens satisfiable à l'aide d'un modèle de Herbrand. Notons $H(E)$ l'ensemble de clauses par instanciation de E, c'est à dire l'ensemble des clauses closes atteignable par une utilisation de la règle I. On montre que $H(E)$ et E sont équisatisfiables au sens de Herbrand et donc que $H(E)$ et E sont équisatisfiables au sens classique du terme.

Ainsi on a $H(E) \models \bot$ et $E \vdash_I H(E)$. On remarque que $H(E)$ est un ensemble de formules qui ne contiennent pas de quantificateurs et on peut donc considérer $H(E)$ comme un ensemble de formules de logique propositionnelle en assimilant chaque atome à une variable. De plus la satisfiabilité de $H(E)$ pour la logique du premier ordre et pour la logique propositionnelle sont équivalent en utilisant la structure triviale.

Or on sait que le système RF est réfutationnellement complet pour la logique propositionnelle donc on a : $H(E) \vdash_{RF} \bot$. On a donc bien par composition des deux dérivations, on a le résultat voulu : $E \vdash_{RFI} \bot$.

On a donc bien montré la complétude réfutationnelle des règles R, F et I pour les clauses universelles de la logique du premier ordre.

Question 1.B. Posons les règles R' et F' de la façon suivante :

— Soient C_1, C_2 deux clauses telles que $C_1 = C \vee L$ et $C_2 = C' \vee L'$. Dans le cas où il existe θ, θ' telles que $L\theta = \overline{L'\theta'}$, on définit la règle R' :

$$\frac{C \vee L \qquad C' \vee L'}{C\theta \vee C'\theta'}\mathrm{R}' \quad \Longleftrightarrow \quad \frac{\dfrac{C \vee L}{C\theta \vee L\theta}\mathrm{I} \qquad \dfrac{C' \vee L'}{C'\theta' \vee L'\theta'}\mathrm{I}}{C\theta \vee C'\theta'}\mathrm{R}$$

— De même, soit C_1 une clause de la forme $C_1 = C \vee L \vee L'$. Dans le cas ou il existe une substitution θ telle que $L\theta = L'\theta$, on définit la règle F' :

$$\frac{C \vee L \vee L'}{C\theta \vee L\theta}\mathrm{F}' \quad \Longleftrightarrow \quad \frac{\dfrac{C \vee L \vee L'}{C\theta \vee L\theta \vee L'\theta}\mathrm{I}}{C\theta \vee L\theta}\mathrm{F}$$

Montrons maintenant que les règles R' et F' sont réfutationnellement complètes pour les clauses universelles dans la logique du premier ordre. Soit C une clause universelle insatisfiable.

C possède un arbre de dérivation dans les règles R, F et I. Dans un premier temps, il est possible de combiner l'utilisation de plusieurs règles I successives en une seule, par composition des substitutions. Ainsi toute utilisation d'une règle I est suivie d'une règle R ou une règle F à l'exception d'une hypothétique règle I en fin de dérivation que l'on traitera à la fin.

En appliquant cette étape de combinaison des règles I à chaque modification de l'arbre de dérivation, on peut ainsi supposer que toute règle I qui n'est pas en fin de dérivation est suivie d'une règle R ou une règle F.

Prenons la règle I la plus profonde dans l'arbre :

— Si elle est suivie d'une règle F, on peut ainsi combiner les deux en une règle F'. En effet prenons la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{C_1}{C_1\theta}\mathrm{I}}{C_2}\mathrm{F}}{} \quad (1)$$

On sait par l'utilisation de la règle F qu'il existe une clause C et un littéral L tels que

$$C_1\theta = C \vee L \vee L \text{ et } \mathrm{C}_2 = \mathrm{C} \vee \mathrm{L}$$

Or la substitution ne rajoute pas de littéraux dans la clause donc on sait qu'il existe deux littéraux L_1 et L_2 et une clause C_3 tels que $L_1\theta = L_2\theta = L$, $C_3\theta = C$ et $C_1 = C_3 \vee L_1 \vee L_2$. Ainsi la dérivation (1) s'écrit de la façon suivante :

$$\dfrac{\dfrac{\dfrac{C_3 \vee L_1 \vee L_2}{C_3\theta \vee L_1\theta \vee L_2\theta}\mathrm{I}}{C_3\theta \vee L_1\theta}\mathrm{F}}{} \quad \Longleftrightarrow \quad \dfrac{\dfrac{C_3 \vee L_1 \vee L_2}{C_3\theta \vee L_1\theta}\mathrm{F}'}{}$$

— Si elle est suivie d'une règle R, alors on peut supposer, quitte à rajouter une règle I triviale sur la deuxième branche, que l'on a :

$$\dfrac{\dfrac{C_1}{C_1\theta}\mathrm{I} \qquad \dfrac{C_2}{C_2\theta'}\mathrm{I}}{C_3}\mathrm{R} \quad (2)$$

De même que précédemment, on sait, par l'utilisation de la règle R, qu'il existe deux clauses C_1' et C_2' et un littéral L tels que

$$C_1\theta = C_1' \vee L,\ C_2\theta' = C_2' \vee \overline{L} \text{ et } \mathrm{C}_3 = \mathrm{C}_1' \vee \mathrm{C}_2'$$

De même, il existe deux littéraux L_1 et L_2 et deux clauses C_1'' et C_2'' tels que $L_1\theta = \overline{L_2\theta'} = L$, $C_1''\theta = C_1'$, $C_2''\theta' = C_2'$ et $C_3 = C_1''\theta \vee C_2''\theta'$. Ainsi la dérivation (2) s'écrit de la façon suivante :

$$\dfrac{\dfrac{C_1'' \vee L_1}{C_1''\theta \vee L_1\theta}\mathrm{I} \qquad \dfrac{C_2'' \vee L_2}{C_2''\theta' \vee L_2\theta'}\mathrm{I}}{C_1''\theta \vee C_2''\theta'}\mathrm{R} \quad \Longleftrightarrow \quad \dfrac{C_1'' \vee L_1 \qquad C_2'' \vee L_2}{C_1''\theta \vee C_2''\theta'}\mathrm{R}'$$

En itérant cela jusqu'à ce que l'arbre de dérivation atteigne une forme stable, on obtient un arbre qui contient des règles R, F, R' et F' ainsi qu'une possible règle I à la racine. On transforme les règles R et F en R' et F' en introduisant la substitution triviale. Pour finir, si la racine est une règle I, alors il est possible de la supprimer en composant la substitution utilisée avec celles de la règle immédiatement au-dessus.

On a donc modifié l'arbre de dérivation de la clause C pour n'utiliser que des règles R' et F' sans changer la conclusion, on a donc $C \vdash_{\mathrm{R}'\mathrm{F}'} \bot$. On a ainsi montré la complétude réfutationnelle des règles R' et F' sur les clauses universelles.

2 Résolution avec sélection

Question 2.A.

Notre contre exemple est le suivant :

$$\{P \vee Q, \neg P \vee \neg Q, P \vee \neg Q, \neg P \vee Q, P \vee \neg P, Q \vee \neg Q\}$$

On selectionne de cette façon les litteraux pour les formules :

$$f(P \vee Q) = Q$$
$$f(\neg P \vee \neg Q) = \neg Q$$
$$f(P \vee \neg Q) = P$$
$$f(\neg P \vee Q) = \neg P$$
$$f(P \vee \neg P) = P$$
$$f(Q \vee \neg Q) = Q$$

Cette ensemble de formules propositionnelles est clos par résolution par selection. Il y a assez peu de factorisation à vérifier et celà fonctionne.

Aucune factorisation ne s'applique à aucune de ces formules.

Question 2.B. (Paul)

Soit S un ensemble de clauses de Horn définies positives et A un atome clos.

$$\text{Montrons que, } S \models A \iff S \vdash_C A$$

$\boxed{\Longrightarrow}$

Supposons que, $S \models A$

On va construire un modèle minimal(minimal de variable posé à vrai) de S qui va donc être modèle de A. En construisant ce modèle on va construire la dérivation avec la règle C de S à A.

Sans perte de généralité on suppose que S est un ensemble fini de clauses. En effet si $S \models A$ alors il existe $S^{'}$ un sous ensemble fini de S tel que $S^{'} \models A$.

Quand on dit que l'on pose un littéral à Vraie ou Faux, c'est que l'on est en train de construire le modèle.

De plus on va voir que pour tout littéral fixé à Vraie, on a une dérivation de S à ce litérral.

On étudie toutes les clauses de S.
(1) Si $n = 0$, alors on pose à Vraie ce littéral A. On a bien $S \vdash A$ car $A \in S$
(2) Si $n >= 0$, on a une formule de cette forme : $A_1 \wedge ... \wedge A_n \Rightarrow A^{'}$
- (3) Soit $\forall i, A_i$ est fixé à Vraie alors on fixe $A^{'}$ à Vraie et $\forall i, S \models A_i$.

$$A^{'}\text{a cette dérivation :} \frac{A_1 \wedge ... \wedge A_n \Rightarrow A^{'} \quad A_1 \; ... \; A_n}{A^{'}} \text{C}$$

- Soit $\exists i, A_i$ n'est pas fixé à Vraie, alors on ne fixe rien de plus à Vraie. Cette clause est bien vraie et ne contredit pas le fait que I est un modèle de S car $A_1 \wedge ... \wedge A_n$ est faux.

On réitère l'étape (2) jusqu'à ne plus avoir aucune clause qui vérifie (3)

Ce modèle est un modèle de A, car $S \models A$.
Or A est un literral donc est vraie.
Or toutes les variables fixés à vraies par ce modèle, sont dérivables par S.
Donc $S \vdash A$

On va maintenant relever cette preuve au 1er ordre avec le théorème de Herbrand.

$\boxed{\Longleftarrow}$

Montrons la correction de la conséquence : $\dfrac{A_1 \wedge ... \wedge A_n \Rightarrow A^{'} \quad A_1\theta \; ... \; A_n\theta}{A^{'}\theta}$C

On a la correction de ces trois règles dérivation :

$$\frac{C \quad C^{'}}{C \wedge C^{'}} \wedge_I \qquad \frac{C \Rightarrow \text{C}^{'} \quad C}{C^{'}} \Rightarrow_E \qquad \frac{C}{C\theta}\text{I}$$

$$\frac{A_1 \wedge ... \wedge A_n \Rightarrow A^{'} \quad A_1\theta \; ... \; A_n\theta}{A^{'}\theta}\text{C} \quad \longrightarrow \quad \frac{\dfrac{A_1 \wedge ... \wedge A_n \Rightarrow A^{'}}{A_1\theta \wedge ... \wedge A_n\theta \Rightarrow A^{'}\theta}\text{I} \quad \dfrac{A_1\theta \; ... \; A_n\theta}{A_1\theta \wedge ... \wedge A_n\theta}\wedge_I}{A^{'}\theta} \Rightarrow_E$$

On a réécrit la règle avec ces trois règles, on a donc la correction de C.

Question 2.C.

Question 2.D.

Question pas traitée.

3 Modélisation et vérification de protocoles

3.1 Modélisation en clauses de Horn

3.2 Vérification automatique

Question 3.A.

Supposons que $S^{'}$ n'a aucune clause avec pour conclusion $att(s)$.

Donc, $S' \nvdash_C att(s)$

Donc, $S' \nvdash_C att(s)$,d'après 2.C car S et S' sont bien ceux de la question 2.C.

Donc, $S \not\models att(s)$ $\square$, d'après 2.B

Question 3.B.

Question 3.C.

Nous avons implémenter tout en Rust. La structure du projet est la suivante
- Cargo.toml :
fichier de configuration, dépendances des lexers / parsers / affichage en arbre jolie

- src/ast.rs :

- src/derivation-tree.rs : Arbre de dérivation, uniquement utile pour l'affichage des arbres de dérivation.

- src/identifier.rs

- src/lexer.rs : le lexer.

- src/lib.rs : implémentation de beaucoup de fonctions, la plus intéressante est celle de saturation.

- src/main.rs : la boucle principale, qui gère entre autre les commandes que l'on peut entrer dans le repl(sniffer »).

- src/parser.rs : le parser.

- src/resolution.rs : implémentation de la fonction de résolution et de sélection.

- src/unify.rs : fonction unify de termes et atomes.

- src/union-find.rs : Optimisation pour unify, implémentation de la structure de donnée union-find.

Simplification pour permettre de saturer :
On vient d'obtenir par résolution la clause de Horn suivante : $C = A_1 \vee ... \vee A_n \Rightarrow A$

1. (Implémenté) Si $\exists i \mid A_i = \text{Var(X)}$, avec X une variable et $X \notin A$.
   Alors on ajoute la clause $A_1 \vee ... \vee A_{i-1} \vee ... \vee A_{i+1} \vee A_n \Rightarrow A$
2. (Implémenté) Si $C = A_1 \Rightarrow A \mid A_1 = A = Var(X)$X est une variable.
   Alors on ajoute par la règles.
3. (Implémenté) Les prémisses d'une règle sont implémentées par un ensemble. On n'a donc jamais de problèmes de dériver deux fois une règle sémantiquement la même. (dans notre implem : $A_1 \vee A_2 \Rightarrow A$ sont la même règle $A_2 \vee A_1 \Rightarrow A$. Jamais aucun doublons inutile.

$\rightarrow$ Pour examples/example.pif, on arrive à la saturation. De plus on obtient cette dérivation pour arriver à att(leak). C'est un arbre de dérivation, qui n'utilise que des résolutions. Les atomes en rouge sont ceux selectionnées.

```
sniffer >> load examples/example.pif
sniffer >> query att(leak).
att(leak)
├─ att(kleak) => att(leak)
│  ├─ att(senc(X, Y)) /\ att(Y) => att(X)
│  └─ att(senc(leak, kleak))
└─ att(kleak)
```

→ Pour examples/nspk-ab.pif, on arrive à la saturation. Voici toutes les clauses que l'on obtient avec à chaque fois un arbre de dérivation qui explique comment on l'obtient.

```
att(aenc(pair(pub(ska), na), pub(skb)))
att(ska) /\ att(VAR85) => att(pair(VAR85, nb))
att(VAR28) /\ att(skb) => att(aenc(pair(VAR28, nb), pub(ska)))
att(pub(skb)) /\ att(pub(ska)) /\ att(VAR14) => att(aenc(pair(VAR14, nb), pub(ska)))
att(pub(ska)) /\ att(VAR47) => att(aenc(pair(VAR47, nb), pub(ska)))
att(pub(skb)) /\ att(pub(ska)) /\ att(VAR33) => att(aenc(pair(VAR33, nb), pub(ska)))
att(VAR75) /\ att(skb) => att(aenc(pair(VAR75, nb), pub(ska)))
att(VAR76) /\ att(skb) /\ att(ska) => att(aenc(pair(VAR76, nb), pub(ska)))
att(VAR72) /\ att(skb) /\ att(ska) => att(aenc(pair(VAR72, nb), pub(ska)))
att(pub(ska)) /\ att(VAR36) /\ att(skb) => att(aenc(pair(VAR36, nb), pub(ska)))
att(pair(VAR5, VAR4)) => att(VAR4)
att(ska) /\ att(VAR21) => att(pair(VAR21, nb))
att(pub(skb))
att(pub(ska)) /\ att(VAR15) => att(aenc(pair(VAR15, nb), pub(ska)))
att(VAR67) /\ att(ska) => att(aenc(pair(VAR67, nb), pub(ska)))
att(pub(ska)) /\ att(VAR16) /\ att(skb) => att(aenc(pair(VAR16, nb), pub(ska)))
att(VAR74) /\ att(skb) /\ att(ska) => att(aenc(pair(VAR74, nb), pub(ska)))
att(aenc(nb, pub(skb)))
├─ att(aenc(pair(na, nb), pub(ska)))
│  ├─ att(aenc(pair(pub(ska), VAR12), pub(skb))) => att(aenc(pair(VAR12, nb), pub(ska)))
│  └─ att(aenc(pair(pub(ska), na), pub(skb)))
└─ att(aenc(pair(na, VAR11), pub(ska))) => att(aenc(VAR11, pub(skb)))
att(VAR0) /\ att(VAR1) => att(pair(VAR0, VAR1))
att(VAR26) => att(aenc(pair(VAR26, nb), pub(ska)))
att(pub(ska))
att(VAR70) /\ att(ska) => att(aenc(pair(VAR70, nb), pub(ska)))
att(aenc(pair(pub(ska), VAR12), pub(skb))) => att(aenc(pair(VAR12, nb), pub(ska)))
att(VAR86) /\ att(pub(VAR87)) => att(VAR86)
sniffer >> load examples/nspk-ab.pif
sniffer >> query att(secret).
no result for query
sniffer >> query att(na).
no result for query
sniffer >> query att(nb).
no result for query
```

Voici l'attaque sur le protocole nspk. On voit de nouveau l'attaque qui répond à cette commande "query att(na)."

4 Extensions

— Proposer une correction du protocole, et vérifier qu'elle assure les secrets attendus.

— Analyser d'autres protocoles : je vous proposerai des références.

— En cas d'attaque, fournir la dérivation du secret à partir des clauses de départ.

— En cas de non-terminaison, donner un moyen plus ou moins convivial pour comprendre pourquoi des clauses de plus en plus complexes sont générées malgré la stratégie adoptée.

— Ecrire un compilateur produisant les clauses de Horn à partir d'une spécification des processus des participants du protocole.

— Proposer une extension de la technique de vérification qui permette d'assurer des propriétés de correspondance : on souhaite par exemple s'assurer qu'une certaine action d'un participant ne puisse être effectuée qu'après une autre action d'un autre participant.