TP5
Décider de l'équivalence de politiques de sécurité d'un pare-feu

Fichiers du projet : `TP5_ACF.zip`, `tp5.thy`.

Le TP est à rendre **sur le MOODLE du cours**. Déposez sur MOODLE, une archive au format **ZIP** contenant :

1. Le répertoire de **projet TP5 Scala** complet
2. **La théorie Isabelle** complétée : `tp5.thy` et, si nécessaire, les autres fichiers Isabelle/HOL que vous utilisez.

---

1 Préliminaires

Décompressez l'archive `TP5_ACF.zip` dans votre répertoire ACF. Dans votre éditeur Scala, ouvrez le projet `TP5_ACF`. Dans Isabelle/HOL, ouvrez le fichier `tp5.thy`.

2 Contexte

Un pare-feu ("firewall" en anglais) est un mécanisme protégeant un réseau ou une machine particulière en lui appliquant une politique de sécurité. Par exemple, on peut protéger une machine connectée à Internet de connexions réseau malveillantes à l'aide d'un pare-feu et d'une politique de sécurité excluant certains paquets réseau en fonction de leur adresse d'origine ou de leur contenu. Une politique de sécurité (nommée "chain" en anglais) consiste en une **liste ordonnée de règles** filtrant ou redirigeant les paquets. Dans ce TP, on se focalisera sur les règles qui filtrent les paquets sur la base de leur adresse IP d'origine. Pour simplifier, on considère deux types de règles : `(Drop l)`, et `(Accept l)`, où `l` est une liste d'adresses IP. [1] L'effet de la règle `(Drop l)` sera d'écarter toutes les adresses présentes dans `l`. L'effet de la règle `(Accept l)` sera d'accepter toutes les adresses présentes dans `l`. Une politique de sécurité est une liste de telles règles. Pour une adresse `a` donnée, les règles sont appliquées dans leur ordre d'apparition dans la liste jusqu'à ce que l'une d'elle s'applique sur `a`. Si aucune d'elle ne s'applique l'adresse `a` est écartée.

Par exemple, prenons la politique `[(Drop [0.0.0.0, 1.1.1.1]),(Accept [1.1.1.1, 2.2.2.2])]`. Même si l'adresse `1.1.1.1` figure dans la règle `(Accept [1.1.1.1, 2.2.2.2])`, cette politique rejettera l'adresse `1.1.1.1` car la première règle rencontrée est `(Drop [0.0.0.0, 1.1.1.1])` et `1.1.1.1` appartient bien à la liste des adresses écartées. Cette même politique acceptera l'adresse `2.2.2.2` et refusera, par exemple, l'adresse `3.3.3.3`. En fait, cette politique refusera toute adresse en dehors de l'adresse `2.2.2.2`. L'acceptation des adresses est défini par la fonction `filter` dans `tp5.thy`. **Il est vivement recommandé de lire le code de cette fonction et de la tester sur quelques valeurs avant de démarrer le TP !** Vous ne devez pas modifier ces fonctions car elles font partie de la base de confiance du TP.

1. On retrouve les politiques de sécurité à base de règles `Drop` et `Accept` dans l'outil `iptables`. Le pare-feu `netfilter` combine ces deux types de règles... et bien d'autres.

3 Objectif

Dans le domaine de la sécurité réseau, la mise à jour de telles politiques de sécurité est connue pour être une tâche complexe et risquée. [2] En effet, toute transformation apportée à une politique de sécurité (comme un simple ré-ordonnancement de règles) peut changer totalement son effet : les adresses acceptées peuvent différer. On propose donc de définir une fonction testant l'équivalence de deux politiques de sécurité. Deux politiques de sécurité seront **équivalentes** si elles acceptent les mêmes adresses. Cet outil permettra de s'assurer qu'une mise à jour d'une politique de sécurité ne changera pas la protection d'un réseau.

4 Marche à suivre

Dans le fichier `tp5.thy` vous trouverez les définitions Isabelle/HOL pour les types adresses (`address`), règles (`rule`), et politiques de sécurité (`chain`). Ce fichier défini également la fonction `filter` qui permet de savoir si une adresse est acceptée par une politique de sécurité.

Travail attendu (le barême est donné à titre indicatif)

1. Programmez en Isabelle/HOL une fonction `equal` :: `chain` ⇒ `chain` ⇒ `bool` qui dit si deux politiques sont équivalentes (5pts) ;
2. Proposez un (ou des) lemmes assurant la correction de votre fonction `equal` par rapport à la fonction `filter`. Vérifiez qu'il n'a pas de contre-exemple simple (5pts) ;
3. Générez le code Scala correspondant à `equal` et intégrez le dans le projet `TP5_ACF`. Le code généré est à placer dans le fichier `tp5.scala` situé dans `src/main/scala`. Ensuite, dans la fonction `equal` de l'objet `EqualObject` remplacez `false` par un appel à **votre** fonction `equal` qui est maintenant dans `tp5.scala`. Avec `sbt run` vous pourrez exécuter l'objet `Main` qui vous donnera une idée de la correction de votre fonction en réalisant un peu plus de 28 millions de tests. Si nécessaire écrivez vos propres tests dans le fichier `TestEqual.scala` situé dans `src/test/scala` et exécutez les avec `sbt test`. (5pts) ;
4. Prouvez l'intégralité des lemmes du point 2. en Isabelle/HOL (5pts).

2. `https://www.researchgate.net/publication/289619483_Mignis_A_Semantic_Based_Tool_for_Firewall_Configuration`