Fiche de TP no. 1

Dans ces TPs nous allons apprendre comment utiliser les programmes `Prover9` et `Mace4`. La page web de `Prover9` et `Mace4`, avec les manuels, se trouve à l'url

http://www.cs.unm.edu/~mccune/mace4/
http://www.cs.unm.edu/~mccune/mace4/manual/2009-11A/

Vous pouvez aussi trouver d'autres outils pour la visualisation des preuves et des structures produites par `Prover9` et `Mace4` depuis la page web du cours :

http://pageperso.lif.univ-mrs.fr/~luigi.santocanale/teaching/LC/

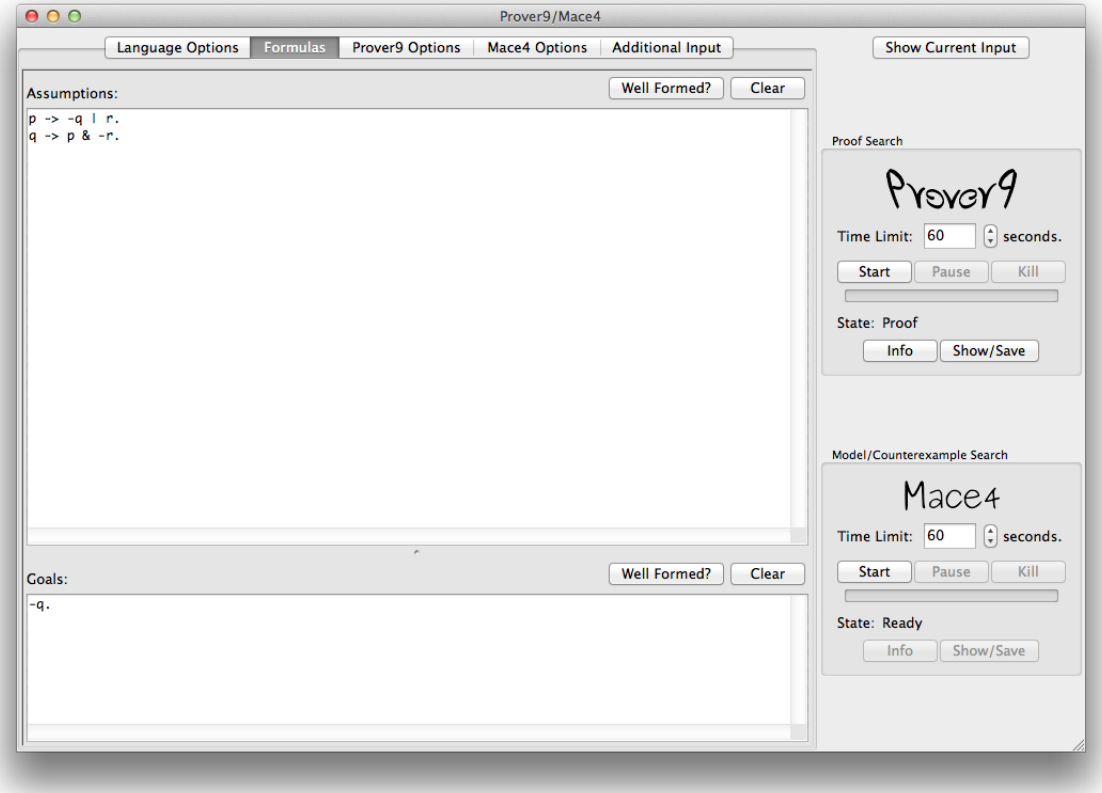
`Prover9` essaye de montrer que $\Gamma \models \phi$, c'est-à-dire que ϕ est une conséquence logique de Γ. A ce fin, il utilise la *résolution*—la méthode de la coupure pour la logique du premier ordre ; il utilise aussi un autre calcul, la *paramodulation*, plus adapté aux raisonnements par égalités.
`Mace4` essaye de montrer que $\Gamma \not\models \phi$, donc que ϕ n'est pas une conséquence logique de Γ. Il essaye donc de construireun modèle de Γ (une structure qui valide toute formule dans Γ) qui rend ϕ fausse.
On peut se servir des deux programmes via un interface graphique jointe. Depuis un terminal[1], tapez la commande suivante :

```
prover9-mace4-gui
```

Exemple. On souhaite utiliser la méthode de la coupure via l'outil `Prover9` pour montrer que

$$\{\, p \Rightarrow (\neg q \vee r), q \Rightarrow (p \wedge \neg r) \,\} \models \neg q \,,$$

c'est-à-dire que $\neg q$ est une consequence logique de l'ensemble $\{\, p \Rightarrow (\neg q \vee r), q \Rightarrow (p \wedge \neg r) \,\}$.
Pour ce faire, on recopie (en utilisant a syntaxe de `Prover9`) les formules de Γ dans la fenêtre des assomptions, et la formulas $\neg p$ dans la fenêtre des buts (goals) :

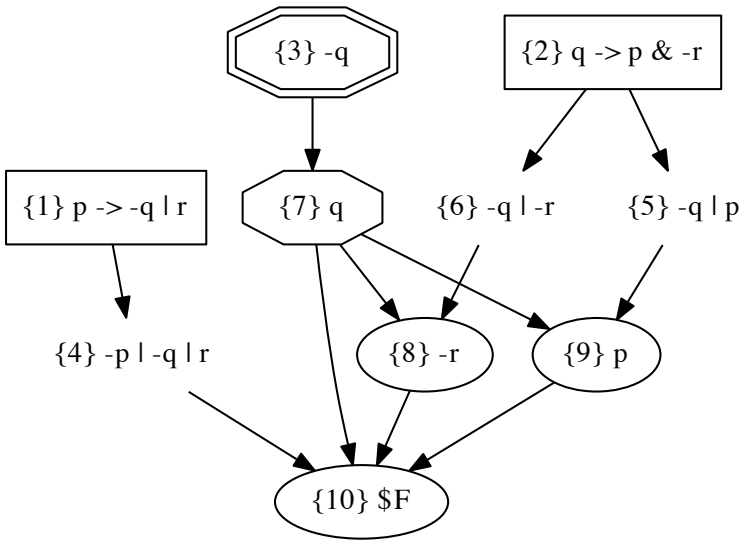


1. Sous Linux, ouvrez l'application `terminal` en faisant une recherche sur le mot clèf « `terminal` ».

Attention : il faut ajouter un point après chaque formule.
Voici la preuve construite par `Prover9` :

```
1 p -> -q | r # label(non_clause).  [assumption].
2 q -> p & -r # label(non_clause).  [assumption].
3 -q # label(non_clause) # label(goal).  [goal].
4 -p | -q | r.  [clausify(1)].
5 -q | p.  [clausify(2)].
6 -q | -r.  [clausify(2)].
7 q.  [deny(3)].
8 -r.  [back_unit_del(6),unit_del(a,7)].
9 p.  [back_unit_del(5),unit_del(a,7)].
10 $F.  [back_unit_del(4),unit_del(a,9),unit_del(b,7),unit_del(c,8)].
```

En utilisant l'outil `gvzify`, on peut transformer cette preuve de façon graphique comme suit :



Exercice 1. En consultant le manuel de `Prover9`, découvrez la syntaxe spécifique à `Prover9` et à `Mace4` des opérateurs logiques ($\wedge, \vee, \Rightarrow, \neg, \forall, \exists$) et des formules du calcul propositionnel et de la logique du premier ordre. Écrivez les formules

1. $(p \Rightarrow q) \Rightarrow ((p \wedge \neg q) \vee (p \vee q))$;
2. $\forall x(P(x) \Rightarrow \exists y(x < y \wedge P(y)))$.

(avec la syntaxe appropriée) dans une des fenêtres de `Prover9`. Vérifiez ensuite la correction de ces formules (button "Well formed"). Pouvez vous utiliser ces outils pour montrer que la première formule est une tautologie ? Comment ? Que faut-il écrire dans chaque fenêtre ?

Logique propositionnelle

Exercice 2. Utilisez `Prover9` (et/ou `Mace4`) pour résoudre cet exercice du TD 5 : *prouvez ou infirmez les affirmations suivantes* :

1. $\models p \Rightarrow p$
2. $\models ((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$
3. $\models ((s \Rightarrow r) \wedge p \wedge \neg r) \Rightarrow \neg r \wedge \neg s \wedge p$

4. $\models [(p \wedge q) \vee (r \wedge q)] \Rightarrow (p \vee r)$
5. $\{q \Rightarrow (\neg q \vee r), q \Rightarrow (p \wedge \neg r)\} \models q \Rightarrow r$
6. $\{q \Rightarrow (\neg q \vee r), q \Rightarrow (p \wedge \neg r)\} \models q \wedge r$
7. $\models (p \wedge (q \vee r)) \Leftrightarrow ((\neg p \Rightarrow r) \wedge (p \wedge q))$.
8. $\models (p \vee (q \wedge r)) \Leftrightarrow ((p \Rightarrow r) \vee (p \wedge q))$.
9. $\{p \Rightarrow q, q \Rightarrow r, p \vee \neg r\} \models p \wedge q \wedge r$.
10. $\{p \Rightarrow q, q \Rightarrow r, p \vee \neg r\} \models (p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)$.

Étapes à suivre, pour chaque problème à résoudre de l'exercice qui demande de montrer que $\Gamma \models \phi$:

1. ajoutez dans la fenêtre des assomptions les formules dans Γ ;
2. ajoutez la formule ϕ dans la fenêtre des buts ;
3. démarrez `Prover9` (et/ou `Mace4`) ;
4. inspectez ensuite la preuve produite (ou le modèle produit) ;
5. utilisez l'outil `viewproofs.py` pour transformer chaque preuve produite par `Prover9` en forme graphique.

Exercice 3. Le lieutenant Colombo enquête sur le crime commis la nuit du 11 octobre. Il dispose des informations suivantes :

1. Jacques ou Martin est coupable.
2. Si Martin est coupable, alors le crime a eu lieu avant minuit.
3. Si le crime a eu lieu après minuit, alors Jacques est coupable.
4. Le crime a eu lieu avant minuit.

Que peut-il en déduire sur l'identité du (ou des) coupable(s) ? Même question s'il dispose de l'information supplémentaire :

5. Si Jacques est coupable, alors le crime a eu lieu après minuit.

Modélisez ces informations en logique propositionnelle et utilisez `Mace4` et/ou `Prover9` pour tester et (in)valider vos hypothèses.

Logique du premier ordre

Exercice 4. Utilisez `Mace4` pour compter combien de relations d'ordre il y a sur un ensemble de 3 éléments. Pouse ce faire, on utilisera un langage du premier ordre sans symboles de fonctions, et avec les deux symboles de prédicat $(less, 2)$ et $(=, 2)$.
Étapes à suivre :

1. écrivez, sur papier, ce que veut dire qu'une relation binaire *less* est une relation d'ordre (réflexivité, transitivité, antisymétricité) ; écrivez ces conditions comme des formules de la logique du premier ordre sur le langage donné ;
2. sélectionnez, dans la fenêtre des options de `Mace4`, les options indiquant que l'on cherche tous les modèles sur un ensemble de taille 3 : `max_models=-1` et `domain_size=3` ;
3. ajoutez dans la fenetre des assomptions les formules logiques (du premier ordre) qui décrivent l'« être un relation d'ordre » ;
4. démarrez `Mace4` et inspectez ensuite le résultat ; dessinez le diagramme de Hasse de chacun des ordres ; sauvez ce résultat dans un fichier nommé `ordre.out` ;
5. affichez les résultats via l'outil `viewmodels.py` qu'on peut télécharger de la page du cours ; pour ce faire, posez vous dans un répertoire conteant les fichier les fichiers `viewmodels.py` et `ordre.out` et, tapez depuis un terminal,

```
./viewmodels.py ordre.out
```

À defaut, tapez

```
python ./viewmodels.py ordre.out
```

Répétez tout pour compter combien de relations d'ordre il y a sur un ensemble de 4 éléments.