# 3.7 Unification

## 3.7.1 Substitions et MGUs

Dans la suite, soient $\mathcal{S}_\mathtt{f}$ une signature composée de symboles de fonctions et $X$ un ensemble de variables. Nous nous intéresserons pas, dans cette section, aux symboles de relation.

**Définition 3.56.** Une *substitution* est une fonction $\sigma : X \to \mathcal{T}_{\mathcal{S}_\mathtt{f}}(X)$, telle que l'ensemble $\{ x \in X \mid \sigma(x) \neq x \}$ est fini.

**Exemple 3.57.** Supposons $\mathcal{S}_\mathtt{r} = \{ (f,2), (g,1) \}$ et $X = \{ x_0, \ldots, x_n, \ldots \}$. La fonction $\sigma$ telle que $\sigma(x_0) = f(g(x_0), x_1)$, $\sigma(x_1) = x_2$, et $\sigma(x_i) = x_i$ pour $i \geq 2$, est une substitution.

On dénote (et souvent on implémente sur ordinateur) les substitutions par des listes de couples (listes associatives), notées

$$\{ x_1 \to \sigma(x_1), \ldots, x_n \to \sigma(x_n) \}, \quad \text{ou} \qquad [\sigma(x_1)/x_1, \ldots, \sigma(x_n)/x_n] .$$

Si une variable n'apparaît pas dans une liste, alors la convention est qu'elle est fixée par $\sigma$ (c'est-à-dire $\sigma(x) = x$). Ainsi, la substitution de l'exemple 3.57, sera notée par

$$[f(g(x_0), x_1)/x_0, x_2/x_1] .$$

**Définition 3.58.** Pour tout terme $t$, on définit l'action de $\sigma$ sur $t$ comme suit :

$$x\sigma = \sigma(x) ,$$
$$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma) .$$

**Exemple 3.59.** On a

$$f(g(x), y)[z/x, g(y)/y] = f(g(z), g(y)) , \qquad g(f(x, f(y, x)))[g(w)/x] = g(f(g(w), f(y, g(w)))) .$$

**Définition 3.60.** La substitution *identité* (ou vide) est celle qui fixe toutes les variables. Elle est donc notée par []. La *composition* de deux substitutions $\sigma$ et $\tau$, notée $\tau \circ \sigma$, est la substitution définie par :

$$(\tau \circ \sigma)(x) = (\sigma(x))_\tau .$$

**Exercice 3.61.** Prouvez que la composition de substitutions est associative, et que la substitution identité est son un élément neutre. Prouvez les relations suivantes :

$$t[] = t , \qquad\qquad t(\tau \circ \sigma) = (t\sigma)\tau .$$

**Définition 3.62.** Soient $\sigma$ et $\tau$ deux substitutions. On dit que $\sigma$ est *plus générale* que $\tau$ (et on écrit $\sigma \leq \tau$), s'il existe une substitution $\rho$ telle que $\tau = \rho \circ \sigma$.

**Exemple 3.63.** Soit

$$\sigma = [f(w, x)/x, z/y] , \qquad\qquad \tau = [f(g(y), x)/x, c/y, c/z, g(y)/w] .$$

On a alors $\sigma \leq \tau$, à cause de

$$\rho = [c/z, g(y)/w] .$$

En fait, le calcul de la composition donne :

| $\sigma$ | | | $\rho$ | |
|---|---|---|---|---|
| $x$ | $\mapsto$ | $f(w, x)$ | $\mapsto$ | $f(g(y), x)$ |
| $y$ | $\mapsto$ | $z$ | $\mapsto$ | $c$ |
| $z$ | $\mapsto$ | $z$ | $\mapsto$ | $c$ |
| $w$ | $\mapsto$ | $w$ | $\mapsto$ | $g(y)$ |

**Exercice 3.64.** Proposez un algorithme qui calcule le composition de deux substitutions $\sigma$ et $\tau$ passées en paramètre. Les substitutions, en entrée et en sortie, seront représentés par des listes associatives.

**Définition 3.65.** Un *problème d'unification* est une liste $(s_1, t_1), \ldots, (s_n, t_n)$ avec $s_i, t_i \in \mathcal{T}_{\mathcal{S}_t}(X)$. Une solution de ce problème—appelé *unificateur* de $(s_1, t_1), \ldots, (s_n, t_n)$—est une substitution $\sigma$ telle que $s_i\sigma = t_i\sigma$, pour $i = 1, \ldots, n$. On notera $\mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,]$ l'ensemble des unificateurs $(s_1, t_1), \ldots, (s_n, t_n)$.

**Exemple 3.66.**

1. La substitution
$$\sigma = [g(z)/x, g(z)/y]\,.$$
est un unificateur de $(f(x, g(z)), f(g(z), y))$, car
$$f(x, g(z))[g(z)/x, g(z)/y] = f(g(z), g(z)) = f(g(z), y)[g(z)/x, g(z)/y]\,.$$
2. Nous avons $\mathsf{Unif}[\,(f(x, y), g(z))\,] = \varnothing$. De même pour $\mathsf{Unif}[\,(x, g(x))\,]$.

Le but de cette section est de montrer le résultat suivant :

**Proposition 3.67.** *Si* $\mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,] \neq \varnothing$, *alors il existe* $\sigma \in \mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,]$ *tel que* $\sigma \leq \tau$ *pour tout* $\tau \in \mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,]$.

On appelle un tel $\sigma$ un *unificateur le plus général* des couples $(s_1, t_1), \ldots, (s_n, t_n)$. On dira aussi que $\sigma$ un MGU des couples $(s_1, t_1), \ldots, (s_n, t_n)$, où MGU est un acronyme de l'anglais « Most General Unifier ».

**Exemple 3.68.** $\tau = [g(f(w))/x, g(f(w))/y] \in \mathsf{Unif}[\,(f(x, g(z)), f(g(z), y))\,]$, mais $\tau$ n'est pas un MGU de ce problème. En fait, $\sigma = [g(z)/x, g(z)/y]$ est un MGU, et on a $\sigma \leq \tau$, car $\tau = \rho \circ \sigma$, avec $\rho = [f(w)/z]$.

### 3.7.2 Algorithme d'unification

L'algorithme d'unification est illustré en figure 3.7.2. Nous donnons dans la suite des exemples de calcul de cet algorithme sur des problèmes d'unification.

**Exemple 3.69.** Considérez le problème suivant :
$$(f(x, g(z)), f(g(z), x))\,,\ (x, g(z))\,.$$

L'algorithme marche de la facon suivante :

| Ligne appel récursif (ou return) | Entrée | Pile des résultats partiels |
|---|---|---|
| | $(\,f(x, g(z)), f(g(z), x)),\ (x, g(z))\,)$ | |
| 6 | $(x, g(z)),\ (g(z), x),\ (x, g(z))$ | |
| 12 | $(g(z), g(z)),\ (g(z), g(z))$ | $[g(z)/x]$ |
| 6 | $(z, z),\ (g(z), g(z))$ | $[g(z)/x]$ |
| 9 | $(g(z), g(z))$ | $[g(z)/x]$ |
| 6 | $(z, z)$ | $[g(z)/x]$ |
| 9 | | $[g(z)/x]$ |
| 1 | | $[] \circ [g(z)/x]$ |

**Exercice 3.70.** Exercez vous maintenant avec les problèmes suivants :

1. $(f(g(k(x)), y), f(y, g(x)))$,
2. $(f(g(x), x), f(y, g(z))), (g(x), y))$,
3. $(f(y, k(y), g(x)), f(k(x), k(y), y))$.

UNIFIER

Entrée : un problème d'unification $(s_1, t_1), \ldots, (s_n, t_n)$

Sortie :

un MGU de $(s_1, t_1), \ldots, (s_n, t_n)$ si $\mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,] \neq \emptyset$

ECHEC, sinon

```
1  Si n = 0, retourner la substitution identité
2  Sinon, on analyse le couple (s_1, t_1) :
3     si s_1 = f(r_1, ..., r_k) et t_1 = g(r'_1, ..., r'_k') alors
4        si f ≠ g, retourner ECHEC
5        sinon          /* f = g implique k = k' */
6           retourner UNIFIER( (r_1, r'_1), ..., (r_k, r'_k), (s_2, t_2), ..., (s_n, t_n) )
7     si s_1 est la variable x, alors :
8           si t_1 est aussi la variable x,
9              retourner UNIFIER( (s_2, t_2), ..., (s_n, t_n) )
10       si x ∈ VAR(t_1), retourner ECHEC
11       sinon,
12          soit τ le résultat de UNIFIER( (s_2[t_1/x], t_2[t_1/x]), ..., (s_n[t_1/x], t_n[t_1/x]) )
13             si τ = ECHEC, retourner ECHEC
14             sinon retourner τ ∘ [t_1/x]
15    si t_1 est la variable x, alors
16       traitement comme auparavant, avec s_1 à la place de t_1
```

FIGURE 3.2 – Algorithme d'unification

**Terminaison.** Définissons la complexité d'un terme comme suit :

$$\sharp(x) = 1\,,$$
$$\sharp(f(t_1,\ldots,t_n) = 1 + \sum_{i=1,\ldots,n} \sharp t_i\,.$$

La complexité d'un problème est un couple de nombres entiers (non-négatifs) qui se définit comme suit :

$$\sharp((s_1,t_1),\ldots,(s_n,t_n)) = (\,\mathrm{card}(\bigcup_{i=1,\ldots,n} Var(s_i)\cup Var(t_i))\,,\ \sum_{i=1,\ldots,n} \sharp(s_i)+\sharp(t_i)\,)\,.$$

Le lecteur notera qu'à chaque appel récursif, le problème en paramètre a complexité strictement plus petite par rapport à l'ordre lexicographique sur $\mathbb{N}\times\mathbb{N}$. Donc L'algorithme ne peut pas faire une suite infinie d'appels récursifs, et il termine.

### 3.7.3 Correction et completude

Nous souhaitons prouver les propositions suivantes.

**Proposition 3.71** (Correction)**.** *Soit $\pi$ un problème d'unification. Si* UNIFIER($\pi$) *retourne* ECHEC*, alors* $\mathsf{Unif}[\,\pi\,] = \varnothing$ *; si* UNIFIER($\pi$) *retourne une substitution $\sigma$, alors $\sigma \in \mathsf{Unif}[\,\pi\,]$ et de plus $\sigma$ est un* MGU *de $\pi$.*

**Proposition 3.72** (Complétude)**.** *Soit $\pi$ un problème d'unification. Si* $\mathsf{Unif}[\,\pi\,] = \varnothing$*, alors* UNIFIER($\pi$) *retourne* ECHEC *; si* $\mathsf{Unif}[\,\pi\,] \neq \varnothing$*, alors* UNIFIER($\pi$) *retourne un* MGU *de $\pi$.*

La preuve repose sur les lemmes suivants :

**Lemme 3.73.** *Les faits suivants sont vrais :*

1. *Si $f \neq g$, alors* $\mathsf{Unif}[\,(f(r_1,\ldots,r_k), g(r'_1,\ldots,r'_{k'}))\,] = \varnothing$.
2. *Si $x \in Var(t)$ et $t \neq x$, alors* $\mathsf{Unif}[\,(x,t)\,] = \varnothing$.
3. *Si* $\mathsf{Unif}[\,(s,t))\,] = \varnothing$*, alors* $\mathsf{Unif}[\,(s,t),(s_2,t_2),\ldots,(s_n,t_n)\,] = \varnothing$.

**Lemme 3.74.** *On a*

$$\mathsf{Unif}[\,(f(r_1,\ldots,r_k), f(r'_1,\ldots,r'_k)),(s_2,t_2),\ldots,(s_n,t_n)\,] = \mathsf{Unif}[\,(r_1,r'_1),\ldots,(r_k,r'_k),(s_2,t_2),\ldots,(s_n,t_n)\,]\,.$$

**Lemme 3.75.** *Supposons $x \notin Var(t)$. Un* MGU *de*

$$(x,t),(s_2,t_2),\ldots,(s_n,t_n)$$

*est $\rho\circ[t/x]$ où $\rho$ est un* MGU *de*

$$(s_2,t_2)[t/x],\ldots,(s_n,t_n)[t/x]\,.$$

*Si ce dernier problème ne possède pas de solutions, alors il en est de même pour $(x,t),(s_2,t_2),\ldots,(s_n,t_n)$.*

Ce Lemme est un conséquence de la Proposition 3.77 suivante, en raison du fait que $[t/x]$ est évidemment un MGU du problème $(x,t)$.

**Exercice 3.76.** A l'aide des Lemmes 3.73-3.75 complétez une preuve formelle de correction et complétude de l'algorithme d'unification.

Avant la proposition 3.77, introduisons quelques notations qui sera utile :
— $\Delta = \{\,(t,t) \mid t \in \mathcal{T}_{\mathcal{S}_t}(X)\,\}$ et $\Delta^n = \underbrace{\Delta\times\ldots\times\Delta}_{n-\text{fois}}$,
— si $\pi = (s_1,t_1),\ldots,(s_n,t_n)$, alors $\ell(\pi) = n$ et $\pi_\sigma = (s_1\sigma,t_1\sigma),\ldots,(s_n\sigma,t_n\sigma)$.

Avec cette notation remarquez que

$$\sigma \in \mathsf{Unif}[\,\pi\,] \text{ ssi } \pi_\sigma \in \Delta^{\ell(\pi)}\,.$$

**Proposition 3.77.** *Soient $\pi$ et $\psi$ deux problemes d'unification. Supposons que $\sigma$ est un* MGU *$\pi$. Alors*

$$\mathsf{Unif}[\,\pi,\psi\,] = \{\,\rho\circ\sigma \mid \rho \in \mathsf{Unif}[\,\psi_\sigma\,]\,\}\,. \tag{3.2}$$

*Par conséquent, si $\rho$ est un* MGU *de $\psi_\sigma$, alors $\tau = \rho\circ\sigma$ un* MGU *de $\pi,\psi$.*

*Démonstration.* Observons que si $\tau \in \mathsf{Unif}[\,\pi,\psi\,]$ alors $\tau \in \mathsf{Unif}[\,\pi\,]$ et, car $\sigma$ est un MGU de $\pi$, $\sigma \le \tau$, c'est-à-dire $\tau = \rho\circ\sigma$ pour une substitution $\rho$. Car $\tau \in \mathsf{Unif}[\,\psi\,]$, on remarquera que

$$(\psi_\sigma)_\rho = \psi_{\rho\circ\sigma} = \psi_\tau \in \Delta^{\ell(\psi)}\,,$$

donc $\rho$ est un unificateur de $\psi_\sigma$, et $\tau \in \{\,\rho\circ\sigma \mid \rho \in \mathsf{Unif}[\,\psi_\sigma\,]\,\}$. D'autre part, si $\rho \in \mathsf{Unif}[\,\psi_\sigma\,]$, alors

$$\begin{aligned}
\psi_{\rho\circ\sigma} &= (\psi_\sigma)_\rho \in \Delta^{\ell(\psi)}\,, && \text{car } \rho \in \mathsf{Unif}[\,\psi_\sigma\,]\\
\pi_{\rho\circ\sigma} &= (\pi_\sigma)_\rho \in (\Delta^{\ell(\pi)})_\rho && \text{car } \sigma \in \mathsf{Unif}[\,\pi\,]\\
&\subseteq \Delta^{\ell(\pi)}\,,
\end{aligned}$$

donc $\rho\circ\sigma \in \mathsf{Unif}[\,\pi,\psi\,]$. Cela complète la preuve de l'égalité (3.2).

Soient maintenant $\rho$ un MGU de $\psi_\sigma$ et soit $\tau \in \mathsf{Unif}[\,\pi,\psi\,]$. A cause l'égalité (3.2), nous pouvons écrire $\tau = \rho'\circ\sigma$ avec $\rho' \in \mathsf{Unif}[\,\psi_\sigma\,]$ ; on a alors $\rho \le \rho'$, donc $\rho' = \theta\circ\rho$ pour une substitution $\theta$ et, par conséquent, $\tau = \theta\circ\rho\circ\sigma$, ce qui montre que $\rho\circ\sigma \le \tau$. Nous avons donc montré que $\rho\circ\sigma$ est un MGU du problème $\pi,\psi$. □

## 3.8 Résolution

### 3.8.1 Substitution, sur les formules propostionnelles

L'action d'une substitution s'étend aisément aux formules sans quantificateurs :
— $R(t_1, \ldots, t_n)\sigma = R(t_1\sigma, \ldots, t_n\sigma)$,
— $(\neg\varphi)\sigma = \neg(\varphi\sigma)$,
— $(\varphi \circ \psi)\sigma = \varphi\sigma \circ \psi\sigma$, $\circ \in \{ \vee, \wedge, \Rightarrow \}$.

**Lemme 3.78.** *Soient $\varphi$ une formule sans quantificateurs, $\sigma$ une substitution, $\mathcal{M}$ une $\mathcal{S}$-structure et $\mathcal{V}$ une valuation. On a que*

$$\mathcal{M}, \mathcal{V} \models \varphi\sigma \text{ ssi } \mathcal{M}, \mathcal{V}\sigma \models \varphi\,, \tag{3.3}$$

*où $\mathcal{V}\sigma$ est la valuation définie par la règle suivante :*

$$(\mathcal{V}\sigma)(x) = [\![\, \sigma(x) \,]\!]_{\mathcal{M},\mathcal{V}}\,.$$

Notez que si $\sigma = [t_1/x_1, \ldots, t_n/x_n]$, alors $\mathcal{V}\sigma = \mathcal{V}[x_1 := [\![\, t_1 \,]\!]_{\mathcal{M},\mathcal{V}}, \ldots, x_n := [\![\, t_n \,]\!]_{\mathcal{M},\mathcal{V}}]$ est la variante de $\mathcal{V}$ satisfaisant aux lois suivantes :

$$\mathcal{V}[x_1 := [\![\, t_1 \,]\!]_{\mathcal{M},\mathcal{V}}, \ldots, x_n := [\![\, t_n \,]\!]_{\mathcal{M},\mathcal{V}}](y) = \begin{cases} [\![\, t_i \,]\!]_{\mathcal{M},\mathcal{V}}\,, & \text{si } y = x_i, \text{ pour quelques } i, \\ \mathcal{V}(y), & \text{sinon.} \end{cases}$$

*Démonstration du Lemme 3.78.* La preuve de cet énoncé se fait aisément par induction. Nous nous limiterons à illustrer le cas de base. Pour toute variable $y \in X$, nous avons

$$[\![\, y \,]\!]_{\mathcal{M},\mathcal{V}\sigma} = (\mathcal{V}\sigma)(y) = [\![\, \sigma(y) \,]\!]_{\mathcal{M},\mathcal{V}}\,.$$

Ainsi, nous avons

$$\begin{aligned} \mathcal{M}, \mathcal{V} \models R(y_1, \ldots, y_m)\sigma \text{ ssi } & \mathcal{M}, \mathcal{V} \models R(\sigma(y_1), \ldots, \sigma(y_m)) \\ \text{ssi } & ([\![\, \sigma(y_1) \,]\!]_{\mathcal{M},\mathcal{V}}, \ldots, [\![\, \sigma(y_m) \,]\!]_{\mathcal{M},\mathcal{V}}) \in R^{\mathcal{M}} \\ \text{ssi } & ([\![\, y_1 \,]\!]_{\mathcal{M},\mathcal{V}\sigma}, \ldots, [\![\, y_m \,]\!]_{\mathcal{M},\mathcal{V}\sigma}) \in R^{\mathcal{M}} \\ \text{ssi } & \mathcal{M}, \mathcal{V}\sigma \models R(y_1, \ldots, y_m). \end{aligned}$$

□

**Définition 3.79.** Un littéral est ou bien une formule atomique, ou bien la négation d'une formule atomique. Une *clause universelle* est la fermeture universelle d'une disjonction de littéraux.

Désormais, clause sera un synonyme de clause universelle. Bien que une clause soit une formule de la forme

$$\forall x_1, \ldots, \forall x_n (l_1 \vee \ldots \vee l_k)$$

avec $l_i$ des littéraux et $\{ x_1, \ldots, x_n \} = \mathsf{FV}(\{ l_1, \ldots, l_n \})$, il est habituel de laisser l'écriture des quantificateurs implicite. Par exemple, nous allons considérer l'expression

$$R(x, y) \vee \neg Q(f(x), z)$$

comme un raccourci de sa fermeture universelle :

$$\forall x \forall y \forall z (R(x, y) \vee \neg Q(f(x), z))\,.$$

Pour de raison de convenance, nous avons donc décidé d'écrire de la même façon une clause universelle de sa matrice (la sous-formule sans quantificateurs). Au cas nous aurions besoin de distinguer une clause universelle $C$ de sa matrice, nous allons écrire $C_{\mathsf{mat}}$ pour la matrice.

La substitution s'etend de facon formelle aux littéraux et aux clauses :

1. $R(t_1, \ldots, t_n)\sigma = R(t_1\sigma, \ldots, t_n\sigma)$,
2. $(\neg R(t_1, \ldots, t_n))\sigma = \neg(R(t_1, \ldots, t_n)\sigma)$,
3. $(\bigvee_{i=1}^{n} l_i)\sigma = \bigvee_{i=1}^{n} (l_i)\sigma$.

**Lemme 3.80.** *Pour toute substitution $\sigma$, la règle d'inférence suivante est correcte :*

$$\frac{C}{C\sigma}\,\sigma$$

*C'est-à-dire, si $\mathcal{M} \models C$, alors $\mathcal{M} \models C\sigma$, pour toute $\mathcal{S}$-structure $\mathcal{M}$.*

*Démonstration.* Supposons que $\mathcal{M} \models C$ ; pour montrer que $\mathcal{M} \models C\sigma$, nous devons montrer que $\mathcal{M}, \mathcal{V} \models (C\sigma)_{\mathsf{mat}}$, où $\mathcal{V}$ est une valuation arbitraire. En considération que $(C\sigma)_{\mathsf{mat}} = (C_{\mathsf{mat}})\sigma$ et par le Lemme reflemme :subst cela revient à vérifier que $\mathcal{M}, \mathcal{V}\sigma \models C_{\mathsf{mat}}$ ; cette dernière relation et en effet vraie à cause de l'assomption $\mathcal{M} \models C$. □

Un *unificateur* de deux litteraux $l_0$ et $l_1$ est une substitution $\sigma$ telle que $l_0\sigma = l_1\sigma$.

**Lemme 3.81.** *$\sigma$ est un unificateur de $l_0$ et $l_1$ ssi*

1. *$l_0 = R(s_1, \ldots, s_n)$, $l_1 = R(t_1, \ldots, t_n)$, et $\sigma \in \mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,]$, ou bien*
2. *$l_0 = \neg R(s_1, \ldots, s_n)$, $l_1 = \neg R(t_1, \ldots, t_n)$, et $\sigma \in \mathsf{Unif}[\,(s_1, t_1), \ldots, (s_n, t_n)\,]$.*

Du Lemme il en découle tout de suite que l'ensemble des unificateurs de deux littéraux, appelons encore une fois $\mathsf{Unif}[\,l_0, l_1\,]$, ou bien il est vide, ou bien il possède un élément le plus général, qui sera appelé un $\mathsf{MGU}$ de $l_0$ et $l_1$.

### 3.8.2 Les règles du calcul de la résolution

On peut considérer le calcul de la résolution comme une généralisation de la méthode de la coupure propositionnelle. Comme dans le cas propositionnel, la méthode de résolution prend en paramètre un ensemble $\Gamma$ de clauses et essaye de dériver la clause vide $\perp$ depuis les clauses dans $\Gamma$. Les deux règles pour dériver des nouvelles clauses à partir de clauses déjà construites sont les suivantes :

$$\frac{C \vee A_0 \qquad C' \vee \neg A_1}{(C \vee C')\sigma}\ \text{Résolution}$$

où $\sigma$ est un $\mathsf{MGU}$ des formules atomiques $A_0$ et $A_1$, et

$$\frac{C \vee l_0 \vee l_1}{(C \vee l_0)\sigma}\ \text{Factorisation}$$

où $\sigma$ est un $\mathsf{MGU}$ des littéraux $l_0$ et $l_1$.

**Exemple 3.82** (Règle de Résolution)**.** La suivante est une instance de la règle de résolution :

$$\frac{\neg HabiteCL(x) \vee Tue(x, Agate) \qquad \neg Tue(Charles, y) \vee Hait(x, y)}{\neg HabiteCL(Charles) \vee Hait(Charles, Agate)}$$

Ici $l_0 = Tue(x, Agate)$ et $l_1 = Tue(Charles, y)$, $C = HabiteCL(x)$, $C' = Hait(x, y)$. En fait, $[Charles/x, Agate/y]$ est un $\mathsf{MGU}$ de $Tue(x, Agate)$ et $Tue(Charles, y)$.

**Exemple 3.83** (Règle de Factorisation)**.** La suivante est une instance de la règle de factorisation :

$$\frac{\neg HabiteCL(x) \vee Hait(x, y) \vee Tue(x, Agate) \vee Tue(Charles, y)}{\neg HabiteCL(Charles) \vee Hait(Charles, Agate) \vee Tue(Charles, Agate)}$$

Le $\mathsf{MGU}$ est encore une fois $[Charles/x, Agate/y]$.

**Correction.** La méthode de résolution est correcte. En fait, on peut penser que les règles du calcul sont obtenues comme synthèse de deux règles, l'une qui porte sur les quantificateurs universels, et l'autre étant la règle correspondante propositionnelle :

$$\frac{\dfrac{C \vee A_0}{C\sigma \vee A_0\sigma}\,\sigma \qquad \dfrac{C' \vee \neg A_1}{C'\sigma \vee \neg A_0\sigma}\,\sigma}{C\sigma \vee C'\sigma}\ \text{Résolution propositionnelle}$$

où on a $A_0\sigma = A_1\sigma$. De façon semblable :

$$\frac{\dfrac{C \vee l_0 \vee l_1}{C\sigma \vee l_0\sigma \vee l_0\sigma}\,\sigma}{C\sigma \vee l_0\sigma}\ \text{Factorisation propositionnelle}$$

**Complètude.** Nous pouvons aussi démontrer le résultats suivant. Soit $\Gamma$ un ensemble de clauses universelles. Nous disons que $\Gamma$ est *saturé* si toute clause dérivable (via résolution et factorisation) de formules dans $\Gamma$ appartient déjà à $\Gamma$ ; nous disons que $\Gamma$ est *cohérent* si $\bot \notin \Gamma$.

**Théorème 3.84.** *Un ensemble saturé et cohérent de clauses admet au moins un modèle.*

En fait, nous allons montrer la proposition suivante :

**Proposition 3.85.** *Si un ensemble saturé de clauses $\Gamma$ n'admet pas un modèle, alors $\bot \in \Gamma$.*

*Démonstration.* Soit $\mathcal{S} = (\mathcal{S}_{\mathtt{f}}, \mathcal{S}_{\mathtt{r}})$ le langage ; nous allons nous intéresser au *langage propositionnel* caractérisé par le fait que PROP est l'ensemble de proposition atomiques du langage $\mathcal{S}$.

Remarquons d'abord que, étant donnée une valuation (au sens propositionnel) $v$ : PROP $\to \{0, 1\}$, nous pouvons définir une $\mathcal{S}$-structure $\mathcal{M}_v$ de la façon suivante :

— $D_{\mathcal{M}_v} := \mathcal{T}_{\mathcal{S}_{\mathtt{f}}}(X)$ ;
— pour tout $f \in \mathcal{S}_{\mathtt{f}}$, $f^{\mathcal{M}_v}$ est la fonction qui envoie un tuplet $(t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{S}_{\mathtt{f}}}(X)$ vers le terme $f(t_1, \ldots, t_n)$ ;
— pour tout $R \in \mathcal{S}_{\mathtt{r}}$, nous posons

$$R^{\mathcal{M}_v} := \{\, (t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{S}_{\mathtt{f}}}(X)^n \mid v(\, R(t_1, \ldots, t_n)\,) = 1 \,\}\,.$$

Par ailleurs, si $C$ est une clause universelle, on a que

$$\mathcal{M}_v \models C \text{ ssi } v(C\sigma) = 1, \text{ pour toute substitution } \sigma.$$

En fait, une valuation $\mathcal{V}$ est ici rien d'autre qu'une substitution ; en tenant compte que $C$ est implicitement quantifié universellement, la condition $\mathcal{M}_v \models C$ est vraie quand $\mathcal{M}_v, \sigma \models C_{\mathsf{mat}}$, pour toute substitution $\sigma$ ; en se rappelant que $[]$ est la substitution identité, on voit alors (via le Lemme 3.78) que cela est équivalent à $\mathcal{M}_v, [] \models C_{\mathsf{mat}}\sigma$ et donc à $v(C\sigma) = 1$ au sens propositionnel.

Il en découle que l'ensemble de clauses propositionnelles

$$\Delta = \{\, C\sigma \mid C \in \Gamma, \sigma \text{ une substitution} \,\}$$

n'est pas satisfaisable au sens propositionnel. En fait, si $v$ est une valuation vérifiant toutes les formules de cet ensemble, alors $\mathcal{M}_v$ est un modèle satisfaisant toutes les formules de $\Gamma$.

Pour le Théorème de compacité, il existe un sous-ensemble fini $\Delta_f \subseteq \Delta$ tel que $\Delta_f$ n'est pas satisfaisable. Car la méthode de la coupure est complète, il existe une suite de clauses $D_1, \ldots, D_n$ avec $D_n = \bot$ (c'est-à-dire, $D_n$ est la clause vide), telle que, pour tout $i > 0$ :

1. either $D_i \in \Delta_f$, ou
2. $D_i$ est déduite de $D_j$ et $D_k$ (avec $j, k < i$) via la règle de coupure, ou
3. $D_i$ est déduite de $D_j$ (avec $j < i$) via la règle de factorization (propositionnelle).

**Lemme 3.86.** *Pour tout $i = 1, \ldots, n$, il existe $C_i \in \Gamma$ est une substitution $\rho_i$ telle que $D_i = C_i\rho_i$.*

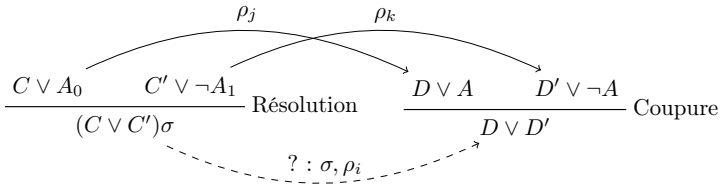


FIGURE 3.3 – Simulation de la coupure par la résolution

En particulier, le Lemme dit que $C_n \in \Gamma$ et que $C_n\rho_n = \bot$. Il est facile à voir que si $C_n \neq \bot$, alors $C_n\rho \neq \bot$ (pour n'importe quel $\rho$) ; par conséquent, nous avons $C_n = \bot \in \Gamma$. La preuve du Lemme est par induction (sur $i = 1, \ldots, n$), et par cas.

1. Si $D_i \in \Delta_f \subseteq \Delta$, alors cela est vrai par définition de $\Delta$.
2. Supposons que $D_i$ est déduite de $D_j$ et $D_k$ (avec $j, k < i$) via la règle de coupure. Par hypothèse d'induction, ils existent $C_j, C_k \in \Gamma$ et deux substitutions $\rho_j, \rho_k$ tels que $D_j = C_j\rho_j$ et $D_k = C_k\sigma_k$.
   Supposons donc que $D_j = D \vee A$, $D_k = D' \vee \neg A$, et $D_i = D \vee D'$. On a alors $C_j = C \vee A_0$, $C_k = C' \vee \neg A_1$, $D = C\rho_j$, $D' = C'\rho_k$, et $A_0\rho_j = A = A_1\rho_k$. Sans perte de généralité, nous pouvons assumer qu'il n'y a pas des variables en commun entre $C_j$ et $C_k$, que $\rho_j$ fixe les variables de $C_k$, et $\rho_k$ fixe les variables de $C_j$. Par conséquent, si $\rho_j = [t_1/x_1, \ldots, t_n/x_n]$ et $\rho_k = [s_1/y_1, \ldots, s_m/y_m]$, alors $\tau = [t_1/x_1, \ldots, t_n/x_n, s_1/y_1, \ldots, s_m/y_m]$ est un unificateur de $A_0$ et $A_1$, $D_j = C_j\tau$ et $D_k = C_k\tau$. Soit $\sigma$ un $\mathsf{MGU}$ de $A_0$ et $A_1$, on a alors $\tau = \rho \circ \sigma$, et
   $$D_i = D \vee D' = C\tau \vee C'\tau = (C \vee C')\tau = [(C \vee C')\sigma]\rho\,.$$
   Nous pouvons alors poser $C_i := (C \vee C')\sigma$ et $\rho_i := \rho$. $C_i \in \Gamma$ car il est déduit de $C_i$ et $C_j$ via la règle de résolution depuis $C_j, C_k \in \Gamma$, et en plus $\Gamma$ est saturé (Voir la Figure 3.3).
3. Supposons enfin que la clause $D_i$ est déduite de $D_j$ (avec $j < i$) via la règle de factorisation propositionnelle. Par hypothèse d'induction, il existe une clause $C_j \in \Gamma$ et une substitution $\rho_j$ telle que $D_j = C_j\rho_j$.
   Si $D_j = D \vee \ell \vee \ell$, alors $C_j = C \vee \ell_0 \vee \ell_1$ avec $D = C\rho_j$, et $\ell = \ell_0\rho_j = \ell_1\rho_j$. La substitution $\rho_j$ est donc un unificateur de $\ell_0$ et $\ell_1$ ; si $\sigma$ est un $\mathsf{MGU}$ de $\ell_0$ et $\ell_1$, alors il existe une substitution $\rho$ telle que $\rho_j = \rho \circ \sigma$.
   Nous pouvons donc poser $C_i := (C \vee \ell_0)\sigma$ et $\rho_i := \rho$ ; $C_i \in \Gamma$ car il a été déduit depuis $C_j \in \Gamma$ via la règle de factorisation (du premier ordre) et $\Gamma$ saturé ; par ailleurs
   $$C_i\rho_i = [(C \vee \ell_0)\sigma]\rho_i = (C \vee \ell_0)(\rho_i \circ \sigma) = (C \vee \ell_0)\rho_j = C\rho_j \vee \ell_0\rho_j = D \vee \ell = D_j.$$

La preuve du Lemme étant complété, nous avons aussi complété la démonstration de la Proposition 3.85. □

Une analyse fine de la preuve de la Proposition 3.85 amène à une preuve du théorème suivant.

**Théorème 3.87.** *Si un ensemble de clauses* $\Gamma$ *n'admet pas un modèle, alors il existe une preuve de* $\bot$ *à partir de* $\Gamma$ *dans le calcul de la résolution.*

**3.8.2.0.1 Indecidabilité.** Bien que le calcul soit correct et complet, nos résultats n'amènent pas à la construction d'un algorithme—c'est à dire une sorte de programme qui *s'arrête toujours* et qui donne la réponse souhaitée à la fin des calculs—pour décider si un ensemble de clauses universelles est satisfaisable ou non. Si nous essayons d'adapter l'algorithme de résolution propositionnelle, cf. **??**, on rencontre un problème majeur : cet algorithme pourrait ne jamais se terminer, en fonction de la possibilité de produire une infinité de nouvelles de clauses. Pour s'en apercevoir, il suffit de considerer le langage $\mathcal{S}$ avec $\mathcal{S}_F = \{\,(o, 0),\,(s, 1)\,\}$ et $\mathcal{S}_R = \{\,(P, 1)\,\}$. Considérons l'ensemble $\mathcal{C}$ de clauses donné par
$$\mathcal{C} := \{\,\neg P(x) \vee P(s(x)),\ P(o)\,\}\,.$$

L'algorithme engendrera, l'une après l'autres, toutes les clauses de la forme

$$P(\underbrace{s(\ldots s(}_{n \text{ fois}} o)\ldots))$$

En fait, nous ne pouvons simplement pas trouver un algorithme ; les prochains théorèmes pourront être mieux compris dans le cadre du chapitre suivant, autour de la calculabilité, où nous formaliserons la notion d'algorithme.

**Théorème 3.88.** *Il n'existe aucun algorithme tel que, étant donné une formule du premier ordre close $\varphi$, il répond oui si $\varphi$ admet un modèle, et non si $\varphi$ est insatisfaisable.*

Car décider de la satisfaisabilité d'un formule du premier ordre se réduit (via la mise en forme prenexe et la Skolemisation) à décider de la satisfaisabilité d'un ensemble de clauses, nous pouvons déduire cette autre théorème à partir du précédent :

**Théorème 3.89.** *Il n'existe aucun algorithme tel que, étant donné un ensemble fini de clauses $\mathcal{C}$, il répond oui si $\mathcal{C}$ admet un modèle, et non si $\mathcal{C}$ est insatisfaisable.*

### 3.8.3 Utilisation d'un démonstrateur automatique

**Exemple 3.90** (L'île misterieuse)**.** Écoutez cette histoire.

> *Chaque habitant de cette île est soit un cavaliers, soit un escroc. Il peut être un loup garou (il est donc dangereux, car il mange les hommes pendant les nuits de lune pleine). Un loup garou est lui aussi soit un cavalier soit un escroc. Les cavaliers disent toujours la vérité, les escrocs mentent toujours. Un explorateur debarque sur cette île et rencontre Albert, Bernard et Charles. Il est au courant qu'un des trois est un loup garou.*
>
> — *Albert prétend que Bernard est un loup ;*
> — *Bernard dit qu'il n'est pas un loup ;*
> — *Charles dit qu'au moins deux entre eux sont des escrocs.*
>
> *Qui doit choisir l'explorateur comme guide de son voyage ?*

Nous avons formalisé cette histoire en logique du premier ordre, dans un fichier prêt à être lu par le démonstrateur automatique `Prover9`. Ce fichier apparaît dans la Figure 3.4. Le prouveur automatique confirme que Albert est un loup garou, et donc l'explorateur ne choisira pas Albert comme guide. La preuve construite par le prouveur apparait dans la Figure 3.5. Le lecteur y reconnaîtra plusieurs instances de la règle de résolution. L'analyse de la preuve montre que l'hypothèse 'Albert n'est pas un loup garou' n'a pas été utilisée. Cela veut dire que la connaissances à disposition de l'explorateur, (qui est modélisée dans la liste des assomptions) est elle même incohérente.

Un procédé analogue peut être utilisé pour montrer que une liste de spécifications d'un programme/logiciel est incohérente, et donc ne peut pas être assuré par n'importe quel programme. Réfléchir avant se mettre à programmer !!! □

**Exemple 3.91** (Le mystère du Château Letot)**.** Écoutez cette autre histoire.

> — *Quelqu'un qui habite Château Letot a tué tante Agate.*
> — *Agate, le majordome, et Charles habitent Château Letot, et ils sont les seuls qui l'habitent.*
> — *Un tueur haït toujours sa victime, et il n'est jamais plus riche que sa victime.*
> — *Charles haït personne que tante Agate haït.*
> — *Agate haït tous sauf le majordome.*
> — *Le majordome haït tous ceux qui ne sont pas plus riches de tante Agate.*
> — *Le majordome haït tous ceux qui tante Agate haït.*
> — *Personne haït tous le monde.*
> — *Agate n'est pas le majordome.*
>
> *Qui a tué tante Agate ?*

```
set(binary_resolution).

formulas(assumptions).
        Cavalier(x) | Escroc(x).
        LoupGarou(albert)  | LoupGarou(bernard) | LoupGarou(charles).
        Cavalier(albert) -> LoupGarou(bernard).
        Escroc(albert) -> -LoupGarou(bernard).
        Cavalier(bernard) -> -LoupGarou(bernard).
        Escroc(bernard) -> LoupGarou(bernard).
        Cavalier(charles) -> (
                (Escroc(albert) & Escroc(bernard))
                | (Escroc(albert) & Escroc(charles))
                | (Escroc(bernard) & Escroc(charles))
                 ).
        Escroc(charles) -> -(
                (Escroc(albert) & Escroc(bernard))
                | (Escroc(albert) & Escroc(charles))
                | (Escroc(bernard) & Escroc(charles))
                 ).
end_of_list.

formulas(goals).
        LoupGarou(albert).
end_of_list.
```

FIGURE 3.4 – Fichier d'entrée pour `Prover9`

Nous avons utilisé le prouveur automatique `Prover9` pour montrer que Agate s'est suicidé ; en fait, 'Agate a tué Agate' est une conséquence logique des faits décrits concernant le Château Letot.

Le procédé est comme auparavant (voir l'île mystérieuse). Nous avons d'abord modélisé l'histoire (*base de connaissances*, ou *ontologie*) en logique du premier ordre, en construisant ainsi un ensemble d'*assomptions* ; la phrase 'Agate a tuée Agate' étant la formule *but* à démontrer [4]. Le code qui a été fourni en entrée à `Prover9` apparaît dans la figure 3.6. `Prover9` transforme d'abord cet ensemble de formules dans un ensemble de clauses universelles. Observez donc l'introduction de nouvelles constantes et de symboles de fonction par élimination de quantificateurs existentiels (Skolemization), la mise sous forme clausale, et l'inclusion du but parmi les assomptions, via sa négation (voir figure 3.7). La preuve que le but est une conséquence logique des assomptions apparaît dans la figure 3.8 Nous avons utilisé les outils GVIZIFY (pour transformer la sortie du `Prover9`—très souvent assez difficile à décrypter—vers un graphe décrit dans le langage `dot`) est GRAPHVIZ (pour dessiner des graphes à partir de leur description en langage `dot`) afin de présenter cette preuve sous forme de diagramme.

Un dernier remarque s'impose. Parmi les symboles de relation de notre langage nous avons utilisé le symbole $=$ sans avoir ajouté, parmi les assomptions les assomptions, aucune hypothèse sur ce symbole. Notamment, nous aurions du ajouter des clauses explicitant le fait que l'égalité est réflexive, transitive, symétrique, et congruentielle. Par exemple, nous aurions du expliciter que si $x = y$ et $Hait(x, z)$ alors $Hait(y, z)$, et tous les inférences de ce type. `Prover9` reconnaît qu'il s'agit du symbole d'égalité et ajoute ces assomptions automatiquement. Car le traitement de l'égalité n'est pas optimal en utilisant la résolution seulement, on se sert aussi de la réglé de paramodulation, que nous présentons ci-dessous.

$$\frac{C \vee t_1 = t_2 \qquad D(t_3)}{(C \vee D(t_2))\sigma}\ \texttt{Paramodulation}$$

où $\sigma$ est un unificateur de $t_1$ et $t_3$.

4. Dans la notation que nous avons utilisé dans le cours, les assomptions correspondent à l'ensemble $\Gamma$ et le but à $\varphi$ quand on se pose la question si $\varphi$ est une conséquence logique de $\Gamma$ ($\Gamma \models \varphi$ ?)

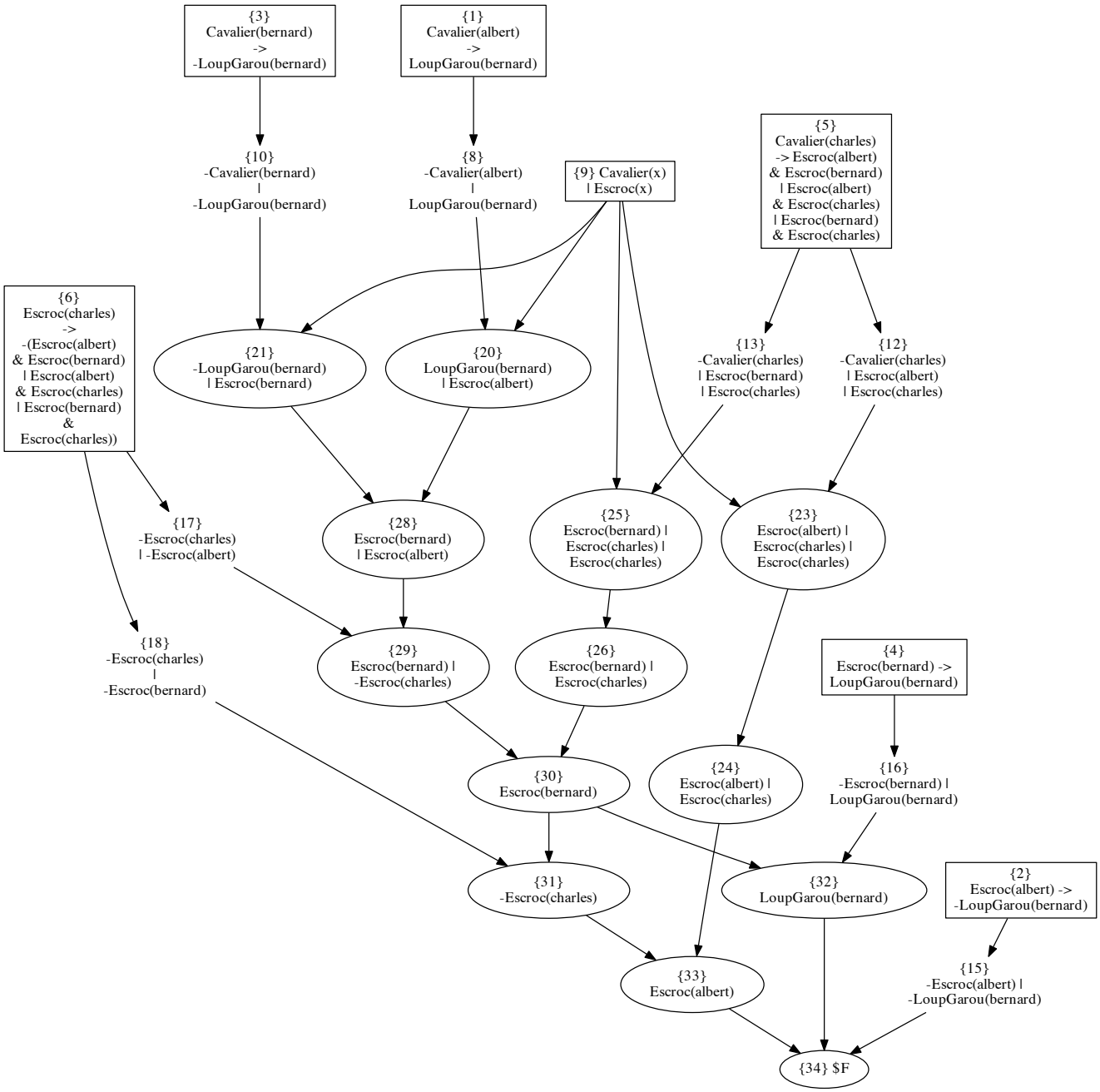


FIGURE 3.5 – Visualisation de la preuve construite par `Prover9`

```
set(binary_resolution).

formulas(assumptions).
        exists x (HabiteCL(x) & Tue(x,agate)).
        HabiteCL(agate) & HabiteCL(maj) & HabiteCL(charles)  &
                    (HabiteCL(x) -> (x = agate | x = maj | x = charles)).
        Tue(x,y) -> (Hait(x,y) & -PlusRiche(x,y)).
        Hait(agate,z) -> -Hait(charles,z).
        (-Hait(agate,x)) -> x=maj.
        x != maj -> Hait(agate,x).
        -PlusRiche(x,agate) -> Hait(maj,x).
        Hait(agate,x) -> Hait(maj,x).
        - (exists x all y Hait(x,y)).
        agate != maj.
end_of_list.

formulas(goals).
        Tue(agate,agate).
end_of_list.
```

FIGURE 3.6 – Fichier entrée pour le château Letot

```
HabiteCL(c1).  [clausify(1)].
Tue(c1,agate).  [clausify(1)].
HabiteCL(agate).  [clausify(2)].
HabiteCL(maj).  [clausify(2)].
HabiteCL(charles).  [clausify(2)].
-HabiteCL(x) | agate = x | maj = x | charles = x.  [clausify(2)].
-Tue(x,y) | Hait(x,y).  [clausify(3)].
-Tue(x,y) | -PlusRiche(x,y).  [clausify(3)].
-Hait(agate,x) | -Hait(charles,x).  [clausify(4)].
Hait(agate,x) | maj = x.  [clausify(5)].
maj = x | Hait(agate,x).  [clausify(6)].
PlusRiche(x,agate) | Hait(maj,x).  [clausify(7)].
-Hait(agate,x) | Hait(maj,x).  [clausify(8)].
-Hait(x,f1(x)).  [clausify(9)].
agate != maj.  [assumption].
-Tue(agate,agate).  [deny(10)].
```

FIGURE 3.7 – Château Letot : forme clausale

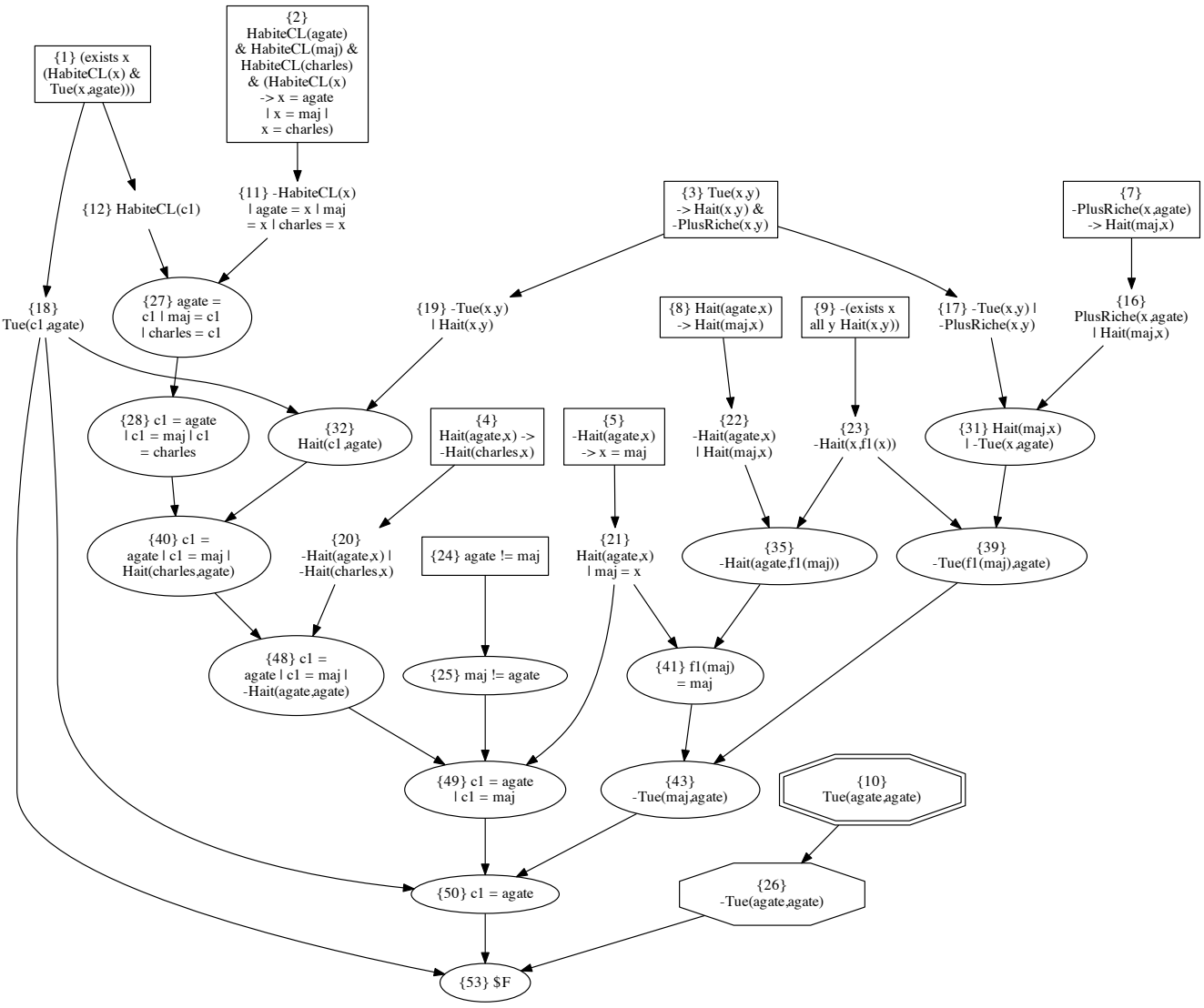


FIGURE 3.8 – La preuve trouvée par `Prover9`

Dans notre exemple, nous avons que l'inférence de la clause 40,

$$\frac{c1 = agate \vee c1 = maj \vee c1 = charles \qquad Hait(c1, agate)}{c1 = agate \vee c1 = maj \vee Hait(charles, agate)}$$

est une instance de la règle de paramodulation. □