Le Langage de Requête Temporel *FTL*

Laurent Herr
INRIA – Projet VERSO
Domaine de Voluceau
Rocquencourt BP 105
78153 Le Chesnay Cedex
Laurent.Herr@inria.fr

Mars 1995

Résumé

Les requêtes sur les bases de données temporelles font référence au temps. Nous nous intéressons ici aux langages de requêtes temporels où cette référence au temps est implicite, c'est-à-dire qu'elle se fait via des opérateurs temporels comme *until* ou *next*. Dans ce contexte, nous proposons un langage de requête *FTL* qui étend la logique temporelle *TL* et la logique temporelle étendue *ETL* en palliant leurs déficiences (faiblesse du pouvoir expressif et/ou nombre infini d'opérateurs temporels). De plus, notre approche permet d'éclairer le pouvoir expressif de *TL* et *ETL* en montrant que ces deux langages utilisent le temps de manière *essentiellement propositionnelle*. Enfin, nous donnons un langage de requête complet pour les bases de données temporelles.

Mots-Clé
Langages de requête, bases de données temporelles, logique temporelle, expressivité.

1 Introduction

Une caractéristique importante des systèmes de gestion de bases de données (SGBD) traditionnels est leur aspect « atemporel », c'est-à-dire qu'ils ne permettent de stocker que des informations non datées, photographies instantanées du monde réel. Or, de plus en plus d'applications ont besoin de SGBD capables de traiter des données retraçant l'histoire du monde réel, et qui permettent à l'utilisateur d'accéder facilement et rapidement aux informations passées. C'est notamment le cas des applications financières (« comment la bourse a-t-elle évolué cette semaine ? »), médicales (« quels sont les antécédents de ce patient ? »), militaires (« quelle est la dernière position connue de la deuxième division blindée ? », « rejouer le dernier exercice ») et des applications temps-réel (« l'altitude de l'avion a-t-elle baissé de plus de 10% sur les dix dernières minutes ? »).

Pour résoudre ce problème, on peut étendre le modèle relationnel classique en un *modèle relationnel temporel* – que nous adopterons dans cet article – en ajoutant dans chaque relation un attribut temporel qui donne la date de validité des n-uplets de cette relation. Cette manière de procéder est présentée formellement dans [Cho94], où l'on constate qu'elle est équivalente d'un point de vue logique aux autres approches les plus courantes [TC$^+$93].

Cependant, le choix d'un langage de requête sur une telle extension n'est pas évident. On peut bien sûr utiliser la logique du premier ordre sur le modèle étendu. Mais le traitement du temps n'est alors pas transparent, puisque l'utilisateur travaille sur un schéma étendu avec des attributs temporels, et non pas sur le schéma initial. De plus, l'utilisateur doit faire référence *explicitement* au temps, en utilisant des variables temporelles.

C'est pourquoi une autre approche est couramment adoptée : il s'agit d'utiliser une logique étendue qui dispose d'opérateurs temporels comme *until* et *since* qui expriment respectivement qu'une propriété sera vérifiée dans le futur jusqu'à un certain événement, et qu'une propriété était vérifiée dans le passé depuis un certain événement. L'avantage de cette solution, outre son élégance, est de permettre une gestion totalement transparente du temps. En effet, l'utilisateur travaille maintenant directement sur le schéma initial de la base de données sans se préoccuper du traitement du temps. D'autre part, il n'a plus besoin de disposer de variables temporelles pour exprimer ses requêtes.

Le plus souvent, dans cette seconde optique, on propose la *logique temporelle TL* comme langage de requête [Cho92], [SW94], [LS87]. Seulement, il s'avère qu'elle a un pouvoir expressif très limité. Par exemple, supposons que le temps est représenté par les entiers 0,1,2,... et que chaque entier correspond à un jour de l'année. On date avec l'entier i toutes les modifications de la base de données qui ont lieu le $i^{\text{ème}}$ jour. Si le jour 0 correspond à un dimanche, le jour 1 à un lundi..., *TL* ne permet pas d'exprimer la requête suivante : « toutes les modifications de la base de données ont-elles été effectuées pendant les jours ouvrables, c'est-à-dire de lundi à vendredi ? »

Une logique temporelle étendue *ETL* a donc proposée dans [Wol83] et [CF84] pour combler cette lacune. Mais *ETL* nécessite une *infinité* d'opérateurs temporels pour disposer de toute sa puissance d'expression ! D'autre part, si cette logique permet d'exprimer la plupart des requêtes usuelles, nous voyons dans cet article qu'elle n'est pas non plus suffisamment expressive. En effet, elle ne peut pas exprimer la requête *À Risque*, que pourrait fort bien poser un médecin :

> Trouver toutes les personnes à risque, une personne étant dite « à risque » si elle a été contaminée dans le passé ou si elle a été en contact avec une autre personne qui était déjà à risque.

Bien sûr, on n'est pas classé à risque lorsqu'on a été en contact avec une personne qui n'était pas encore à risque au moment où on l'a rencontrée, mais qui l'est devenue plus tard.

Nous proposons donc une logique temporelle *FTL* basée sur un opérateur d'itération *foreach*. Cette logique est plus puissante que la logique temporelle étendue *ETL*, et elle permet d'exprimer simplement la requête *À Risque*. D'autre part, cette logique n'utilise qu'un seul opérateur temporel *foreach* décliné dans une variante du futur $foreach^+$, et une variante du passé $foreach^-$. Enfin, elle permet de donner un nouvel éclairage au pouvoir expressif de *TL* et *ETL*. En effet, *ETL* se caractérise comme une restriction élégante de *FTL*, où l'on utilise le temps de manière *essentiellement propositionnelle*. Si on fait de plus une hypothèse inflationniste, on obtient exactement TL.

Enfin, nous voyons que l'opérateur *foreach* de FTL peut être également utilisé pour construire un langage de requête *complet* au sens de Chandra et Harel [CH80], c'est-à-dire un langage qui permet d'exprimer toutes les requêtes calculables et génériques.

Ce papier est organisé de la manière suivante. Dans un premier temps, nous présentons les principaux langages de requêtes temporels où le temps est implicite : la logique temporelle du premier ordre TL et la logique temporelle étendue ETL. Ensuite, dans une deuxième partie, nous présentons le langage FTL dans sa version algébrique et dans sa version logique, puis nous montrons l'équivalence de ces deux versions. Dans une troisième partie, nous voyons à quelles restrictions de FTL correspondent TL et ETL. Enfin, dans une dernière section, nous présentons un langage temporel complet basé sur *foreach*.

La plupart des démonstrations de cet article sont regroupées en annexe.

2 Les principaux langages temporels

2.1 La Logique Temporelle *TL*

Dans cette section, nous présentons la logique temporelle du premier ordre TL.

Syntaxe

Les expressions de TL sont construites à partir des règles de formation suivantes, en admettant qu'un *terme* t est soit une constante c, soit une variable x :

- si R est un symbole de relation, et si $t_1, \ldots, t_n$ sont des termes, alors $R(t_1, \ldots, t_n)$ est une formule ;
- si t_1 et t_2 sont des termes, alors $t_1 = t_2$ est une formule ;
- si φ et ψ sont des formules, alors $\neg\varphi$, $\varphi \wedge \psi$ et $\varphi \vee \psi$ en sont aussi ;
- si φ et ψ sont des formules, alors $\mathit{next}(\varphi)$ et $\varphi\ \mathit{until}\ \psi$ en sont aussi ;
- si φ et ψ sont des formules, alors $\mathit{previous}(\varphi)$ et $\varphi\ \mathit{since}\ \psi$ en sont aussi.

Une *formule du futur* est une formule qui n'utilise que les opérateurs temporels *next* et *until*. De même, une *formule du passé* n'utilise que *previous* et *since*. La sémantique est définie de manière naturelle : par exemple $\varphi\ \mathit{until}\ \psi$ est vraie si ψ est vrai à une date future, et si φ est vrai jusqu'à cette date. Donnons-en maintenant une sémantique formelle.

Sémantique

Une instance d'une BD est la donnée d'un univers infini *dom* et d'une relation finie sur dom^k pour chaque symbole de relation R d'arité k. Pour interpréter les formules de TL, on suppose que le temps est isomorphe à l'ensemble des entiers naturels. Autrement dit, les *instants* forment une suite infinie $i = 0, 1, 2, \ldots$. Une *base de donnée temporelle* est une suite *finie* $\mathbb{D} = (D_0, D_1, D_2, \ldots, D_n)$ d'instances de bases de données de même domaine *dom*, que nous appelons par extension le *domaine de la BD temporelle*. Dans la suite, nous désignons la structure D_i comme étant *l'état* de la base à l'instant i.

Une *valuation* est une fonction v qui associe une valeur $v(x) \in dom$ à chaque variable x. Remarquons que la valeur d'une variable ne change pas avec le temps. Pour alléger les notations, nous étendons les valuations aux constantes en posant $v(c) = c$. Définissons maintenant ce que signifie pour une formule φ d'être vérifiée à l'instant $i \in [0, n]$ pour la valuation v, ce que nous notons $(\mathbb{D}, v, i) \models \varphi$.

- $(\mathbb{D}, v, i) \models t_1 = t_2$ si $v(t_1) = v(t_2)$;
  $(\mathbb{D}, v, i) \models R(t_1, \ldots, t_k)$ si $(v(t_1), \ldots, v(t_k))$ est dans la relation R de D_i ;
- $(\mathbb{D}, v, i) \models \neg\varphi$ si $(\mathbb{D}, v, i) \not\models \varphi$;
  $(\mathbb{D}, v, i) \models \varphi \wedge \psi$ si $(\mathbb{D}, v, i) \models \varphi$ et $(\mathbb{D}, v, i) \models \psi$;
  $(\mathbb{D}, v, i) \models \varphi \vee \psi$ si $(\mathbb{D}, v, i) \models \varphi$ ou $(\mathbb{D}, v, i) \models \psi$;
- $(\mathbb{D}, v, i) \models \forall x\ \varphi$ si pour tout $d \in dom$, $(\mathbb{D}, v[x \leftarrow d], i) \models \varphi$;
- $(\mathbb{D}, v, i) \models next(\varphi)$ si $i < n$ et $(\mathbb{D}, v, i+1) \models \varphi$;
  $(\mathbb{D}, v, i) \models \varphi\ until\ \psi$ s'il existe $j \in [i, n]$ tel que $(\mathbb{D}, v, j) \models \psi$ et pour tout $k \in [i, j], (\mathbb{D}, v, k) \models \varphi$;
- $(\mathbb{D}, v, i) \models previous(\varphi)$ si $i > 0$ et $(\mathbb{D}, v, i-1) \models \varphi$;
  $(\mathbb{D}, v, i) \models \varphi\ since\ \psi$ s'il existe $j \in [0, i]$ tel que $(\mathbb{D}, v, j) \models \psi$ et pour tout $k \in [j, i], (\mathbb{D}, v, k) \models \varphi$.

On peut alors définir l'opérateur « éventuellement dans le futur », noté $\Diamond\varphi$ qui exprime que φ sera vraie au moins une fois dans le futur. L'opérateur « nécessairement dans le futur » est noté $\Box\varphi$ et exprime que φ sera toujours vraie dans le futur. Symétriquement, on définit les opérateurs « éventuellement dans le passé » noté $\blacklozenge\varphi$ et « nécessairement dans le passé » noté $\blacksquare\varphi$. Formellement, on les construit à l'aide de *since* et *until* de la manière suivante :

$$
\begin{aligned}
\Diamond\varphi &= true\ until\ \varphi \\
\Box\varphi &= \neg\Diamond\neg\varphi \\
\blacklozenge\varphi &= true\ since\ \varphi \\
\blacksquare\varphi &= \neg\blacklozenge\neg\varphi
\end{aligned}
$$

Étant donnée une formule $\varphi(x_1, \ldots, x_p)$ dans TL de variables libres $x_1, \ldots, x_p$, la réponse à la requête φ en $(\mathbb{D}, i)$ est la relation formée de l'ensemble des $(v(x_1), \ldots, v(x_n))$ tels que $(\mathbb{D}, v, i) \models \varphi$.

On dit que deux requêtes φ et ψ sont *équivalentes* si elles ont même valeur pour toute base de données $\mathbb{D}$ et en tout instant i.

Exemples

Nous donnons maintenant quelques exemples de requêtes que l'on peut exprimer en logique temporelle du premier ordre. Considérons une base temporelle $\mathbb{D} = (D_0, D_1, \ldots, D_n)$ dans laquelle chaque état D_{i+1} résulte d'une modification de la base D_i. Nous supposons que l'on stocke dans la base des informations sur des étudiants. Un état est constitué des informations *nouvelles* concernant la scolarité d'un certain nombre d'étudiants. On indique par exemple si des étudiants sont entrés à l'université, s'ils ont redoublé ou s'ils ont commencé une thèse...

On peut demander « les élèves qui n'ont jamais redoublé depuis qu'ils sont à l'université » avec la formule du passé ci-dessous, évaluée au temps n :

$$\neg\text{Redoublement}(x)\ \textit{since}\ \text{Nouvel\_Universitaire}(x).$$

On peut également exprimer cette requête avec la formule du futur ci-dessous, évaluée au temps initial :

$$\Diamond(\text{Nouvel\_Universitaire}(x) \wedge \Box\neg\text{Redoublement}(x)).$$

Pour obtenir « les personnes qui ont passé des examens après s'être inscrites en thèse », on peut évaluer au temps initial :

$$\Diamond(\text{Nouveau\_Thésard}(x) \wedge \Diamond\text{Inscrit\_Examen}(x)).$$

Limitations

Si la logique du premier ordre *TL* permet d'exprimer nombre des requêtes usuelles, son pouvoir expressif demeure limité. Par exemple, il n'est pas possible de vérifier dans *TL* que toutes les transactions ont été effectuées lors des jours ouvrables (Lundi à Vendredi), sachant que le temps initial 0 correspond à un dimanche, le temps 1 à un Lundi, etc. Plus simplement, mais plus abstraitement, il n'est pas non plus possible d'exprimer qu'une formule close φ est vraie dans tous les états pairs. En effet, la formule suivante exprime que φ est vraie dans les instants pairs et fausse dans les états impairs, ce qui est beaucoup plus restrictif.

$$\Box(\varphi \leftrightarrow (\textit{first} \vee \textit{previous}(\textit{previous}(\varphi)))).$$

où $\textit{first} = \neg\, \textit{previous}(\textit{true})$ est une formule vraie seulement dans le premier état de la base.

C'est pour pallier ce manque d'expressivité de *TL* que Casanova et Furtado ont proposé le langage temporel étendu *ETL* que nous présentons maintenant.

2.2 La Logique Temporelle Étendue *ETL*

Dans le cas propositionnel, Wolper avait remarqué dans [Wol83] que la logique temporelle classique n'était pas assez expressive pour exprimer des conditions notamment nécessaires à la spécification des programmes concurrents. Pour pallier cette déficience il proposait une logique étendue *ETL* basée sur des opérateurs de grammaires. Comme Wolper se limitait au cas propositionnel, Casanova et Furtado ont généralisé *ETL* au premier ordre dans [CF84]. C'est un aspect de cette généralisation que nous présentons ici, en nous limitant au cas des séquences temporelles *finies* de bases de données, et aux grammaires régulières (à l'instar de Wolper).

Étant donné un alphabet fini $\Sigma = \{v_1, \ldots, v_n\}$, un mot $w = v_{w_i} v_{w_{i+1}} \ldots v_{w_n}$ sur cet alphabet, et une assignation de formules φ_k aux lettres v_k de Σ, on dit qu'une base de données $\mathbb{D} = (D_0, \ldots, D_n)$ satisfait le mot w à l'instant i pour la valuation v et pour l'assignation donnée si, pour tout j dans $[i, n]$, on a $(\mathbb{D}, v, j) \models \varphi_{w_j}$.

Prenons par exemple $\Sigma = \{a, b\}$, et assignons la formule φ à a et la formule *true* à b. Le mot

$$ababababab \cdots$$

est satisfait à l'instant 0 par toute base de données

$$\mathbb{D} = (D_0, D_1, D_2, \ldots, D_n)$$

dans laquelle φ est vraie dans tous les états pairs, et où rien n'est requis pour les états impairs.

À chaque langage rationnel L (c'est-à-dire « langage régulier » [Per90]) sur un alphabet $\Sigma = \{v_1, \ldots, v_m\}$, on associe un opérateur temporel $\mathcal{L}$, appelé *opérateur grammatical*, de la façon suivante. L'opérateur $\mathcal{L}$ a exactement m arguments et $\mathcal{L}(\varphi_1, \ldots, \varphi_m)$ est vraie à l'instant i pour la base de données $\mathbb{D} = (D_0, \ldots, D_n)$ et la valuation v s'il existe un mot $v_{w_i} \cdots v_{w_n}$ de L qui est satisfait par cette base de données à l'instant i quand on assigne φ_1 à v_1, ..., φ_n à v_n.

La logique *ETL* est obtenue en rajoutant à *FO* les opérateurs grammaticaux formés à partir de tous les langages rationnels possibles. On peut montrer que les opérateurs d'éventualité $\Diamond$, de nécessité $\Box$, ainsi que *next* et *until* peuvent être exprimés avec des opérateurs grammaticaux. Ce langage est particulièrement approprié pour exprimer des contraintes d'intégrité.

Remarquons pour conclure cette présentation, que *ETL* est un langage du futur, sans opérateur du passé.

Exemple

Soit L le langage rationnel $v_1 v_2^* v_3$, et soit $\mathcal{L}$ l'opérateur grammatical associé. Si on associe la formule *Nouvel_Universitaire(x)* à v_1, $\neg$*Algorithmique(x)* à v_2, et *Nouveau_Thésard(x)* à v_3, on obtient l'ensemble des étudiants qui n'ont jamais suivi de cours d'algorithmique à l'université avant de commencer leur thèse. Une manière de formuler cette requête dans *ETL* est donc :

$$\mathcal{L}(\text{Nouvel\_Universitaire}(x), \neg\text{Algorithmique}(x), \text{Nouveau\_Thésard}(x)).$$

Pour être parfaitement exact, cette requête donne en fait les étudiants qui n'ont pas suivi de cours d'algorithmique entre le jour suivant leur entrée à l'université et la veille de leur entrée en thèse.

Limitations

Dans le cas propositionnel, ETL est équivalent à la logique temporelle quantifiée QTL [Eme90], qui est suffisamment puissante pour les besoins pratiques. En effet, QTL est équivalent à la logique monadique du second ordre $SOLLO$ [Eme90], et à la logique de point fixe μTL [Var88].

Par contre, pour le premier ordre, auquel nous nous intéressons ici, ETL est assez limité. Par exemple, elle ne permet pas d'exprimer la requête « À Risque » donnée dans l'introduction (nous démontrons ce résultat dans la suite). D'autre part, une infinité d'opérateurs temporels est nécessaire pour en obtenir tout le pouvoir expressif !

3 Le langage temporel *FTL*

FTL est un langage qui étend la logique du premier ordre FO avec un opérateur temporel *foreach*. FTL est plus puissant que TL et ETL, et permet en particulier d'exprimer la requête « À Risque ». D'autre part, FTL pallie la faiblesse principale de ETL : le nombre infini de ses opérateurs temporels.

Dans cette section, nous présentons les versions algébrique et logique de FTL, avant de montrer l'équivalence de ces deux approches.

3.1 Algèbre + *foreach*

L'algèbre relationnelle est un langage procédural qui permet d'affecter à une relation une expression construite à partir des symboles de relation et des opérateurs de sélection σ, de projection π, d'union $\cup$, de négation $\neg$ et de produit cartésien $\times$. L'opérateur *foreach*$^+$ permet d'itérer un programme pour chaque état *futur* de la base de données. Symétriquement, *foreach*$^-$ permet d'itérer un programme pour chaque état *passé* de la base.

Un programme où n'apparaît que des boucles *foreach*$^+$ (resp. *foreach*$^-$) est appelé un programme *du futur* (resp. *du passé*). Dans la suite, nous parlerons de boucles *foreach* sans préciser s'il s'agit de *foreach*$^+$ ou de *foreach*$^-$ lorsque l'on peut prendre indifféremment l'une ou l'autre.

Les boucles *foreach* se construisent suivant la syntaxe suivante :

foreach
    begin
    <corps de la boucle>
    end

Le corps de la boucle est une séquence (1) d'affectations $T := \varphi$ où T est une relation temporaire et où φ est une formule du premier ordre FO, (2) et de boucles *foreach*. Un *programme* est une séquence d'affectations $T := \varphi$ et de boucles *foreach*.

La sémantique d'une boucle *foreach*$^+$ dans la base $\mathbb{D} = (D_0, \ldots, D_n)$ à l'instant i est d'exécuter le corps de la boucle $n - i + 1$ fois, une fois pour chaque état $D_i, \ldots, D_n$ postérieur à l'instant i. À la première étape, on évalue le corps de la boucle dans l'état D_i ; à la deuxième étape, on l'évalue dans l'état D_{i+1} ; ... ; à la dernière étape, on l'évalue dans l'état D_n.

Notons que les relations temporaires T sont globales à tous les états. Leurs valeurs sont indépendantes de l'état sur lequelle elles sont évaluées. Par contre, les relations R de la base sont locales à chaque état. Leurs valeurs dépendent de l'état où elles sont évaluées.

La sémantique des boucles *foreach*$^-$ se définit de manière symétrique : on exécute le corps de la boucle $i+1$ fois, une fois pour chaque état $D_i, D_{i-1}, \ldots, D_0$ antérieur à l'instant i. À la première étape, on évalue le corps de la boucle dans l'état D_i ; à la deuxième étape, on l'évalue dans l'état D_{i-1} ; ... ; à la dernière étape, on l'évalue dans l'état D_0.

Exemples

Prenons l'exemple que nous avons déjà vu plus haut de la base de données d'une université. Le programme suivant, évalué à l'instant 0, calcule l'ensemble des étudiants qui sont passés par cette université. Nous convenons dans la suite d'écrire avec une majuscule les relations de la base de données, et avec une minuscule les relations auxiliaires.

$$
\begin{array}{l}
\textit{étudiants} := \emptyset; \\
\textit{foreach}^+ \\
\qquad \textit{begin} \\
\qquad \textit{étudiants} := \textit{étudiants} \cup \textit{Nouvel\_Universitaire}; \\
\qquad \textit{end}
\end{array}
$$

Détaillons-en l'exécution sur une base $\mathbb{D} = (D_0, \ldots, D_n)$. Après un premier passage dans la boucle *foreach*, la relation *étudiants* contient tous les *Nouveaux_Universitaires* de D_0. Après un second passage, elle contient en plus les *Nouveaux_Universitaires* de D_1. Et ainsi de suite jusqu'à D_n, où elle contient tous les universitaires de la base.

Voici maintenant une manière d'exprimer la requête « À Risque » au temps 0.

$$
\begin{array}{l}
\textit{à\_risque} := \emptyset; \\
\textit{foreach}^+ \\
\qquad \textit{begin} \\
\qquad \textit{à\_risque} := \textit{à\_risque} \cup \textit{Contaminé} \cup \pi_3(\sigma_{1=2}(\textit{à\_risque} \times \textit{En\_Contact})) \\
\qquad \textit{end}
\end{array}
$$

Dans cette boucle, l'affectation exprime qu'une personne est à risque si elle l'était déjà l'instant précédent, ou si elle a été contaminée dans l'instant présent, ou si elle a été en contact avec une personne à risque.

Montrons maintenant comment calculer la requête *even*, dont la réponse *result* est vraie si et seulement si la requête φ est vraie dans tous les états pairs. Pour « l'implantation » dans l'algèbre relationnelle, la valeur booléenne *true* pourrait être représentée par le domaine *dom* et *false* par l'ensemble vide.

```
result := true;
parity := false;
foreach+
        begin
        parity := ¬parity;
        result := (parity ∧ φ ∧ result) ∨ (¬parity ∧ result);
        end
```

Dans ce programme, évalué à l'instant 0, la relation *parity* est vraie dans tous les états pairs et fausse dans tous les impairs. *result* est vraie tant que la portion de la base parcourue ne montre pas que *even* est fausse.

Pour conclure, nous donnons maintenant un exemple de programme mettant en jeu des boucles imbriquées. Le programme suivant donne tous les étudiants qui ont passé des examens après avoir commencé leur thèse.

```
T1 := ∅;
foreach+
        begin
        T2 := Nouveau_Thesard;
        T3 := ∅;
        foreach+
                begin
                T3 := T3 ∪ (T2 ∩ Inscrit_Examen);
                end
        T1 := T1 ∪ T3;
        end
```

Dans ce programme, le résultat de la requête est stocké dans T_1. À la $i^{\text{ème}}$ étape de la boucle extérieure, T_2 donne les thésards qui ont commencé leur thèse au $i^{\text{ème}}$ état de la base. La boucle intérieure calcule alors dans T_3 les thésards de T_2 qui ont passé des examens dans un état futur.

3.2 Logique + *foreach*

Nous présentons maintenant une version de *FTL* plus orientée vers la logique, mais qui a exactement le même pouvoir expressif. Donnons-en maintenant la syntaxe et la sémantique de manière formelle.

Syntaxe

Les expressions de *FTL* sont construites à partir des règles de formations suivantes. Un terme t est soit une constante c, soit une variable v.

- Si R est une relation, et si $t_1, \ldots, t_n$ sont des termes, alors $R(t_1, \ldots, t_n)$ est une formule ;
- si t_1 et t_2 sont des termes, alors $t_1 = t_2$ est une formule ;

– si φ et ψ sont des formules, alors $\neg\varphi$, $\varphi \wedge \psi$ et $\varphi \vee \psi$ en sont aussi ;

– Si φ est une formule et si T est une relation, alors $\textit{foreach}^+_T(\varphi(T))$ et $\textit{foreach}^-_T(\varphi(T))$ sont des formules.

Une formule dans laquelle n'apparaît que des $\textit{foreach}^+$ (resp. $\textit{foreach}^-$) est une formule *du futur* (resp. *du passé*). Nous désignons par FTL^+ et FTL^- les restrictions respectives de FTL aux formules du futur et du passé.

Sémantique

Supposons donnée une formule $\varphi(T)$ dans laquelle apparaît la relation libre T, et dont le résultat a la même arité que T. Pour une base de données $\mathbb{D} = (D_0, D_1, \ldots, D_n)$ et un temps i, nous définissons la suite (finie) $T_0, \ldots, T_{n-i+1}$ de la façon suivante :

$$\begin{array}{ll} T_0 = \emptyset & \\ T_1 = \varphi(T_0) & \text{évalué dans } D_i \\ T_2 = \varphi(T_1) & \text{évalué dans } D_{i+1} \\ \ldots & \\ T_{n-i+1} = \varphi(T_{n-i}) & \text{évalué dans } D_n \end{array}$$

Alors, $\textit{foreach}^+_T(\varphi(T))$ évalué au temps i est égal à T_{n-i+1}. Dans le cas où φ a une arité supérieure à celle de T, on « fixe » les variables supplémentaires et on effectue l'itération comme ci-dessus avec les variables restantes. Si au contraire, φ a une arité inférieure à celle de T, on peut toujours créer des variables fictives qui n'apparaissent nulle part, et écrire φ comme une fonction (constante) de ces variables fictives.

La sémantique des boucles $\textit{foreach}^-$ se définit de manière symétrique. Nous aurons l'occasion d'y revenir plus tard.

Enfin, il est parfois pratique de prendre autre chose que $T_0 = \emptyset$ pour initialiser la boucle. Nous définissons donc $\textit{foreach}_{\Lambda,T}(\varphi(T))$ comme étant le résultat de cette boucle lorsque $T_0 = \Lambda$. Nous verrons plus tard que cette extension ne modifie pas le pouvoir expressif de FTL.

Donnons maintenant la sémantique formelle de FTL. Pour les connecteurs classiques (conjonction, négation, quantification, etc), la sémantique est définie de manière habituelle, comme pour la logique temporelle TL. Nous la rappelons brièvement ci-dessous. Pour *foreach*, la sémantique formelle est inspiré de celle de l'opérateur de point fixe *fixpoint*, par exemple dans [AV90].

Nous définissons inductivement ce que signifie pour une formule φ d'être vérifiée à l'instant $i \in [1, n]$ dans la base de données $\mathbb{D} = (D_0, \ldots, D_n)$ pour la valuation v, ce que nous notons $(\mathbb{D}, v, i) \models \varphi$, comme de coutume.

– $(\mathbb{D}, v, i) \models t_1 = t_2$ si $v(t_1) = v(t_2)$;
$(\mathbb{D}, v, i) \models R(t_1, \ldots, t_p)$ si $(v(t_1), \ldots, v(t_p))$ est dans la relation R de D_i ;

– $(\mathbb{D}, v, i) \models \neg\varphi$ si $(\mathbb{D}, v, i) \not\models \varphi$;
$(\mathbb{D}, v, i) \models \varphi \wedge \psi$ si $(\mathbb{D}, v, i) \models \varphi$ et $(\mathbb{D}, v, i) \models \psi$;

$(\mathbb{D}, v, i) \models \varphi \vee \psi$ si $(\mathbb{D}, v, i) \models \varphi$ ou $(\mathbb{D}, v, i) \models \psi$;

- $(\mathbb{D}, v, i) \models \forall x\ \varphi$ si pour tout $d \in dom$, $(\mathbb{D}, v[x \leftarrow d], i) \models \varphi$;
- Soit T un symbole de relation d'arité p et φ une formule possédant $p+q$ variables libres. On a $(\mathbb{D}, v, i) \models (\mathit{foreach}^{-}_{\Lambda,T}\varphi)(x_1, \ldots, x_p, y_1, \ldots, y_q)$ si $(v(x_1), \ldots, v(x_p))$ est dans la relation T_i, la suite $T_0, \ldots, T_i$ étant définie de la façon suivante :

$$T_0 = \Lambda$$
$$T_{j+1} = \{(c_1, \ldots, c_p) \mid (\mathbb{D}, v[x_1 \leftarrow c_1, \ldots, x_p \leftarrow c_p], i-j) \models \varphi(T_j)\}$$

- Soit T un symbole de relation d'arité p et φ une formule possédant $p+q$ variables libres. On a $(\mathbb{D}, v, i) \models (\mathit{foreach}^{+}_{\Lambda,T}\varphi)(x_1, \ldots, x_p, y_1, \ldots, y_q)$ si $(v(x_1), \ldots, v(x_p))$ est dans la relation T_{n-i+1}, la suite $T_0, \ldots, T_{n-i+1}$ étant définie de la façon suivante :

$$T_0 = \Lambda$$
$$T_{j+1} = \{(c_1, \ldots, c_p) \mid (\mathbb{D}, v[x_1 \leftarrow c_1, \ldots, x_p \leftarrow c_p], i+j) \models \varphi(T_j)\}$$

Exemples

Reprenons l'exemple des étudiants vu plus haut. La formule suivante évaluée au temps initial donne l'ensemble des étudiants qui sont passés par l'université :

$$\mathit{foreach}^{+}_{T}(T(x) \vee \text{Nouvel\_Universitaire}(x)).$$

À la $i^{\text{ème}}$ itération de la boucle *foreach*$^{+}$, T contient l'ensemble des étudiants entrés à l'université aux temps $0, \ldots, i-1$.

La requête « À Risque » devient :

$$\mathit{foreach}^{+}_{T}(T(x) \vee \text{Contaminé}(x) \vee \exists y(T(y) \wedge \text{En\_Contact}(y, x))).$$

À chaque itération, T contient les personnes qui sont à risque au temps $i-1$.

Enfin, la requête *even* s'exprime de la façon suivante :

$$\mathit{foreach}^{+}_{\{a\},T}((T(a) \wedge \varphi \wedge x = b) \vee (T(b) \wedge x = a)).$$

où a et b désignent deux constantes quelconques. Le résultat est non vide si et seulement si la formule φ est vraie dans tous les états pairs. Dans la boucle, T vaut $\{a\}$ aux instants pairs et $\{b\}$ aux instants impairs, tant que la portion de la base de données parcourue vérifiait *even*. Dans le cas contraire, T est vide.

Nous montrons maintenant un lemme qui établit que le choix de la valeur d'initialisation des boucles *foreach* n'a pas d'importance, du point de vue du pouvoir expressif. Ce lemme simplifiera les démonstrations des théorèmes à venir.

Lemme 1 Sans nuire à la généralité, on peut se restreindre aux $\mathit{foreach}_{\Lambda,T}(\varphi(T))$ où $\Lambda = \emptyset$. Autrement dit, avoir le choix de la valeur initiale d'une boucle *foreach* ne rajoute pas de pouvoir d'expression.

Démonstration
Voir en annexe.

Proposition 2 « Algèbre + *foreach* » et « Logique + *foreach* » ont le même pouvoir expressif.

Démonstration
Pour montrer cela, il suffit de voir que le fait de disposer de plusieurs relations temporaires dans l'algèbre ne rajoute pas de pouvoir expressif. En effet : on peut coder p relations $T_1, \ldots, T_p$ en une seule relation T en posant $T = \{a_1\} \times T_1 \cup \cdots \cup \{a_p\} \times T_p$, où $a_1, \ldots, a_p$ désignent des constantes distinctes. □

4 *TL* et *ETL* retrouvés

Dans cette section, nous montrons que FTL étend la logique temporelle TL et la logique temporelle étendue ETL. Plus précisément, nous montrons qu'en se restreignant aux $foreach_B(\varphi)$ où φ est inflationniste et à valeurs finies (c'est-à-dire si l'ensemble des réponses possibles de φ, considérée comme une requête, est fini, indépendamment de la base sur laquelle on l'évalue), on obtient exactement TL. De plus, si on lève l'hypothèse inflationniste, on obtient exactement ETL. En termes intuitifs, on obtient donc que TL et ETL utilisent le temps de manière essentiellement propositionnelle : l'information transmise d'un état à l'autre par la boucle de *foreach* est finie, et peut être représentée par les valeurs binaires d'un ensemble fini de variables propositionnelles.

Théorème 3 La logique temporelle TL est équivalente à FTL_{prop}, où FTL_{prop} désigne la restriction de FTL aux formules ne contenant que des $foreach_B(\varphi)$ avec φ inflationniste et à valeurs finies, c'est-à-dire :

- φ inflationniste : $B \subset \varphi(B)$ pour toute relation B et dans tout contexte $(\mathbb{D}, v, i)$;
- φ à valeurs finies : il existe une relation (finie) F tel que $\varphi(B) \subset F$, pour toute relation B, et pour tout contexte $(\mathbb{D}, v, i)$.

Démonstration
Voir en annexe.

Théorème 4 ETL est équivalent à FTL^{+}_{reg}, où FTL^{+}_{reg} désigne la restriction de FTL aux formules du futur ne contenant que des $foreach^{+}_{B}(\varphi)$ où φ est à valeurs finies.

Démonstration
Voir en annexe.

Corollaire 5 ETL ne peut pas exprimer la requête « À Risque ».

Indication de Démonstration
D'après le théorème précédent, toute formule de ETL est équivalente à une formule de FTL, où on se limite aux $foreach_T(\varphi)$ avec φ à valeurs finies. Une telle formule ne distingue donc qu'un nombre fini d'états futurs. Or, en faisant varier les états futurs, on devrait obtenir un nombre infini de réponses possibles.

Pour une description complète de la technique utilisée pour cette démonstration, on consultera [AHvdB]. □

5 Complétude des bases de données temporelles

Chandra et Harel ont défini la notion de *requête* dans dans [CH80]. Une *requête* est une fonction de l'ensemble des bases de données (traditionnelles) sur lui-même qui est :

– *calculable*, c'est-à-dire qu'il existe une machine de Turing qui prend en entrée le codage d'une base de données et rend en sortie le codage de la réponse à la requête.

– *générique*, c'est-à-dire que la réponse de la machine de Turing ne dépend pas du codage choisi. Plus formellement, on appelle *isomorphisme* d'une base de données D une bijection du domaine sur lui-même qui laisse stable les relations de la base. Une requête Q est alors générique si $Q(h(D)) = h(Q(D))$ pour tout isomorphisme h et toute base de données D.

Un langage de requête est dit *complet* s'il permet d'exprimer toutes les requêtes, dans le sens de Chandra et Harel. Chandra et Harel proposent alors un langage complet baptisé QL.

Ces notions de requêtes et de complétude s'étendent naturellement aux bases de données temporelles. La question se pose alors de trouver un langage de requêtes complet pour les bases de données temporelles. Nous avons le résultat suivant :

Théorème 6 Prenons un langage complet sur les bases de données non-temporelles, comme par exemple QL. Alors, le langage *foreach* + QL est complet pour les bases de données temporelles.

Démonstration
En effet, il est possible de simuler les entiers dans QL. Le programme suivant crée donc une base de données non-temporelle D' équivalente à la base temporelle $\mathbb{D} = (D_0, \ldots, D_n)$:

$D' := \emptyset$;
$k := 0$;
foreach
  begin
  $D' := D' \cup (R \times \{k\})$;
  inc(k);
  end

où R désigne l'unique relation de $\mathbb{D}$ (supposer que $\mathbb{D}$ ne comporte qu'une seule relation ne nuit pas à la généralité). Dans la base non-temporelle D', chaque n-uplet est estampillé avec sa date de la validité. On effectue alors le calcul sur la base D' à l'aide de QL. D'où le résultat. $\square$

Remarque

On peut aussi considérer qu'un langage de requête temporel retourne une base de données temporelle. Pour cela, une requête prend en entrée la base sur laquelle est évaluée la requête, et un entier i. La requête donne alors en sortie le $i^{\text{ème}}$ état de la base temporelle répondant à la requête. Si cet état n'existe pas (parce que i est plus grand que la longueur de la base répondant à la requête), on suppose que la requête rend la réponse *end_of_sequence* ou qu'elle boucle. On suppose de plus qu'on est toujours dans l'une des deux situations :

– il existe un entier i tel que l'évaluation de la requête pour tous les entiers inférieurs à i termine et ne soit pas *end_of_sequence*. Et l'évaluation en i termine et donne *end_of_sequence* ;

– l'évaluation de la requête pour tout entier i donne *end_of_sequence* ou boucle.

Cette restriction est dans l'esprit de QL, où l'on donne une valeur par convention aux programmes qui ne terminent pas.

Alors, le langage QL + *foreach* est encore complet pour ce type de requêtes temporelles.

Remerciements
Je tiens à remercier Serge Abiteboul et Jan Van den Bussche pour les nombreuses discussions sur ce sujet, ainsi que pour leur lecture attentive et critique de versions antérieures de ce papier. Merci aussi à Geneviève Jomier pour sa patience !

6 Conclusion et perspectives

Dans cet article nous avons présenté le langage de requête temporel FTL, qui étend la logique temporelle étendue ETL de Wolper, Casanova et Furtado. Ce langage s'avère suffisamment puissant pour exprimer les requêtes usuelles que l'on ne pouvait pas formuler dans ETL. De plus, son utilisation sur des exemples montre que son maniement est simple et naturel. Enfin, nous avons vu que l'opérateur temporel *foreach* de FTL associé à un langage complet sur les bases de données traditionnelles comme QL fournit un langage complet sur les bases de données temporelles.

En plus de l'intérêt que FTL présente en lui-même pour l'interrogation des bases de données temporelles, nous pensons que l'étude de ce langage pourrait ouvrir de nouvelles perspectives dans la compréhension de la logique temporelle TL. En effet, le pouvoir expressif de TL reste encore mal compris. Par exemple, la question reste toujours ouverte de savoir si TL a le même pouvoir expressif que FO [Cho94] [Kam68] [AHvdB]. La caractérisation de TL comme une restriction de FTL où le temps est utilisé de manière essentiellement propositionnelle pourrait être un nouvel outil pour aborder ce problème.

Enfin, nous avons étudié dans cet article un opérateur temporel du futur $foreach^+$ qui parcourt la base de données temporelle de la gauche vers la droite, et un opérateur temporel du passé $foreach^-$ qui parcourt la base de la droite vers la gauche. Une extension possible des opérateurs *foreach* consiste en une *machine temporelle* qui parcourt la base de données temporelle, et se déplace à chaque étape soit vers la gauche, soit vers la droite, en fonction du résultat d'une requête portant sur l'état courant. Une machine de ce type est étudiée dans [AHvdB].

Références

[AHvdB] S. Abiteboul, L. Herr, and J. van den Bussche. Temporal connectives versus explicit timestamps in temporal query languages. submitted.

[AV90] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41(2):181–229, October 1990.

[CF84] A. Casanova and A. Furtado. On the description of database transition constraints using temporal constraints. In J.-M. Nicolas H. Gallaire, J. Minker, editor, *Advances in Data Base Theory*, volume 2, pages 211–236. Plenum Press, New York, 1984.

[CH80] A. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21:156–178, 1980.

[Cho92] J. Chomicki. History-less checking of dynamic integrity constraints. In *IEEE International Conference on Data Ingineering*, February 1992.

[Cho94] J. Chomicki. Temporal query languages: a survey. In *International Conference on Temporal Logic*, LNAI 827, pages 506–534. Springer-Verlag, July 1994.

[Eme90] E. Emerson. *Handbook of theoretical computer science*, volume B, chapter Temporal and modal logic, pages 996–1071. Elsevier Science Publishers B. V., 1990.

[Kam68] H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[LS87] U. Lipeck and G. Saake. Monitoring dynamic integrity constraints based on temporal logic. *Information Systems*, 12(3):255–269, 1987.

[Per90] D. Perrin. *Handbook of Theoretical Computer Science*, volume B, chapter Finite Automata, pages 1–57. Elsevier Science Publishers B. V., 1990.

[SW94] A. Sistla and O. Wolfson. Temporal triggers in active databases. In *IEEE Transactions on Knowledge and Data Engineering*, 1994.

[TC+93] Tansel, , Clifford, Gadia, Jajodia, Segev, and Snodgrass. *Temporal Databases: Theory, Design, and Implementation.* The Benjamin/Cummings Publishing Company, 1993.

[Var88] M. Vardi. A temporal fixpoint calculus. In *Fiftteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, January 1988.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

A Démonstrations

Démonstration du lemme 1

La démonstration de ce lemme est assez technique et gagnera à ne pas être consultée en première lecture.

Soit φ une formule de FO et Λ une relation. Pour simplifier, et sans nuire à la généralité, nous supposons que φ est monadique. On définit alors une formule binaire ψ dans FO de manière à ce que tout triplet $(\mathbb{D}, v, i)$ satisfait $\mathit{foreach}_{\Lambda,T}(\varphi(T))$ si et seulement s'il satisfait $\pi_1(\mathit{foreach}_T(\psi(T)\backslash\{(b,b)\}))$, en supposant données deux constantes distinctes a et b. Pour obtenir cela, il suffit de prendre pour $\psi(x,y)$ la formule suivante :

$$(x = b \wedge y = b) \bigvee (y = a \wedge \varphi[T(x) \longleftarrow T'(x,\alpha) \vee (T' = \emptyset \wedge \Lambda(x))]).$$

Si $(T_0, \ldots, T_{n-i+1})$ et $(T'_0, \ldots, T'_{n-i+1})$ désignent respectivement les suites associées aux formules $\mathit{foreach}_{\Lambda,T}(\varphi(T))$ et $\mathit{foreach}_{T'}(\psi(T'))$, nous montrons que $T'_k = (T_k \times \{a\}) \cup \{(b,b)\}$ pour $k \in [1, n-i+1]$. En effet, pour $k = 1$, on a :

$$\begin{aligned}
T'_1(x,y) &= \psi[T' \leftarrow \emptyset](x,y)\\
&= (y = a \wedge \varphi[T(x) \leftarrow \Lambda(x)]) \vee (x,y) = (b,b)\\
&= (y = a \wedge T_1(x)) \vee (x,y) = (b,b)\\
&= (T_1 \times \{a\}) \cup (b,b)
\end{aligned}$$

Supposons maintenant le résultat établi au rang $k-1$. Alors :

$$\begin{aligned}
T'_k &= \psi[T' \leftarrow T'_{k-1}](x,y)\\
&= (y = a \wedge \varphi[T(x) \leftarrow T'_{k-1}(x,a)]) \vee (x,y) = (b,b)\\
&= (y = a \wedge \varphi[T(x) \leftarrow T_{k-1}] \vee (x,y) = (b,b) \text{ par l'hypothèse de récurrence}\\
&= (y = a \wedge T_k(x)) \vee (x,y) = (b,b)\\
&= T_k \times \{a\}) \cup \{(b,b)\}
\end{aligned}$$

Ayant établi le résultat annoncé, on en déduit que $\pi_1(\mathit{foreach}_{T'}(\psi(T')\backslash\{(b,b)\})) = \pi_1(T'_{n-i+1}\backslash\{(b,b)\}) = T_{n-i+1} = \mathit{foreach}_{\Lambda,T}(\varphi(T))$. On a donc transformé le $\mathit{foreach}_{\Lambda,T}$ en un $\mathit{foreach}_T$. Une récurrence immédiate montre alors le lemme. □

Démonstration du théorème 3

Commençons par montrer par induction sur la taille des formules que toute formule de TL est équivalente à une formule de FTL_{prop}. Pour cela, on prend $F = \{a, b\}$ où a et b sont deux constantes quelconques. La formule $\varphi\ \mathit{until}\ \psi$ est équivalente à $\mathit{foreach}_B(\theta)(a)$ où θ est définie comme suit :

- si on a $B = \emptyset$, $\varphi = \mathit{true}$ et $\psi = \mathit{false}$, alors θ retourne $\emptyset$;
- si on a $B = \emptyset$, $\varphi = \mathit{true}$ et $\psi = \mathit{true}$, alors θ retourne $\{a\}$;
- si on a $B = \emptyset$ et $\varphi = \mathit{false}$, alors θ retourne $\{a, b\}$;
- si $B \neq \emptyset$, θ retourne B.

Intuitivement, quand on a $B = \emptyset$, cela signifie que la portion de la base de données que l'on a déjà parcourue ne permet pas de conclure que φ *until* ψ est vraie ou fausse. $B = \{a\}$ signifie que cette formule est vraie, et $B = \{a, b\}$ signifie qu'elle est fausse. Formellement, en supposant obtenue par l'hypothèse d'induction deux formules Φ et Ψ de FTL_{prop} équivalentes à φ et ψ, on définit θ de la manière suivante, x désignant une variable libre :

$$\begin{array}{ll} & B = \emptyset \wedge \Phi \wedge \neg\Psi \wedge x \neq x \\ \vee & B = \emptyset \wedge \Phi \wedge \Psi \wedge x = a \\ \vee & B = \emptyset \wedge \neg\varphi \wedge x = a \wedge x = b \\ \vee & B \neq \emptyset \wedge B(x) \end{array}$$

Bien sûr, cette formule est inflationniste. On procède de manière similaire pour *next*, ce qui nous fournit le résultat annoncé.

Montrons maintenant la réciproque, à savoir que toute formule de FTL_{prop} est équivalente à une formule de TL. Pour cela, procédons par induction sur la taille des formules. Prenons donc une formule ψ de TL et supposons que le résultat est établi pour les formules de tailles inférieures. Le seul cas qui pose problème est lorsque ψ est de la forme $foreach_B(\varphi)$ où φ est inflationniste et à valeurs finies sur une relation finie et fixée F. Soit $s = (F_0 \subset F_1 \subset \ldots \subset F_k)$ une suite strictement croissante de sous-ensembles de F (il n'existe qu'un nombre fini de telles suites). Supposons donnée par l'hypothèse d'induction une formule Φ de TL qui est équivalente à φ. Soient alors $\Phi_{i,j}$, pour $i \leq j$, les formules de TL telles que :

$$\Phi_{i,j} \equiv ((B = F_i) \wedge (\Phi(B) = F_j)).$$

Alors, on définit la formule $\Psi_s \in TL$ de la manière suivante :

$$\Psi_s \equiv \Phi_{0,0}\, until(next(\Phi_{0,1}) \wedge next(next(\Phi_{1,1}\, until(next(\Phi_{1,2}) \wedge next(next(\Phi_{2,2}\, until \cdots)))))).$$

Si $B_0, \ldots, B_{n-k+1}$ désigne la suite des itérations associées à la formule $foreach_B(\varphi)$, Ψ_s exprime intuitivement que cette suite est constante égale à F_0 jusqu'à valoir F_1. Qu'elle est alors constante égale à F_1 jusqu'à valoir F_2, et ainsi de suite.

Alors, la disjonction des Ψ_s pour toutes les séquences s possibles donne une formule de TL équivalente à ψ. Ce qui montre le résultat. □

Démonstration du théorème 4

Ce théorème résulte des deux lemmes suivants.

Lemme 7 FTL^{+}_{reg} est inclus dans ETL.

DÉMONSTRATION

Pour montrer cela, on établit par récurrence sur la taille des formules que toute formule de FTL^{+}_{reg} a un équivalent dans ETL. Le seul cas qui pose problème est celui des formules atomiques de la forme $\varphi = foreach_T \psi(T)$. Soit F la relation (finie) sur laquelle ψ prend ses valeurs et soit $|F|$ la cardinalité de cette relation.

Alors, ψ peut-être considérée intuitivement comme la fonction de transition d'un automate fini, qui pose une requête sur l'état courant de la base, et dont le résultat donne le nouvel état de l'automate. Les états de cet automate sont les relations incluses dans F. L'objectif de la démonstration est de trouver l'ensemble des mots que reconnaît cet automate. L'opérateur grammatical associé fournira alors un équivalent de φ dans ETL.

Supposons donc le résultat établi pour les formules de tailles inférieures à celle de φ. Soit p l'arité de T. Pour chaque $R, S \subset F$, soient $a_{R,S}$ et $b_{R,S}$ des lettres distinctes. On considère les alphabets $\Sigma = \{a_{R,S} \mid R, S \subset F\}$ et $\Sigma' = \Sigma \cup \{b_{R,S} \mid R, S \subset F\}$. Notre automate utilise l'alphabet Σ'.

On définit le langage L constitué des mots de la forme

$$a_{\emptyset,R_1} a_{R_1,R_2} a_{R_2,R_3} \cdots a_{R_{m-1},R_m} b_{R_m,R_{m+1}}$$

pour $m > 0$ ainsi que des mots $b_{\emptyset,R}$ (ce qui correspond à $m = 0$). Ce langage est bien rationnel puisqu'il peut s'écrire comme :

$$L = \left(\neg \bigcup_{S \neq R'} \Sigma'^* a_{R,S} \{a_{R',S'}, b_{R',S'}\} \Sigma'^* \right) \bigcap \left(\bigcup_{R,R',S'} a_{\emptyset,R} \Sigma^* b_{R',S'} \cup \bigcup_{S} b_{\emptyset,S} \right).$$

Soit $\mathcal{L}$ l'opérateur grammatical associé. Alors, φ est équivalente à la formule suivante de ETL, où Ψ désigne l'équivalent de ψ dans ETL obtenu par l'hypothèse de récurrence, et où $t = (t_1, \ldots, t_n)$ désigne le vecteur des variables libres de φ :

$$\Phi = \mathcal{L}((a_{R,S} \leftarrow (\Psi(R) = S)), (b_{R,S} \leftarrow (t \in \Psi(R) = S))).$$

En effet, Φ exprime intuitivement que l'on a un mot

$$a_{\emptyset,R_1} a_{R_1,R_2} a_{R_2,R_3} \cdots a_{R_{m-1},R_m} b_{R_m,R_{m+1}}$$

dans L, et une suite $T_0, \ldots, T_m$ telle que $T_0 = \emptyset$, $T_1 = \Psi(T_0)$ dans l'état courant, $T_2 = \Psi(T_1)$ dans l'état suivant, etc. Et le résultat retourné par Φ est alors le dernière relation de la suite. Comme $\Psi = \psi$, on a donc intuitivement bien équivalence entre Φ et φ.

Montrons maintenant plus formellement cette équivalence entre Φ et φ. Prenons pour cela une base de données $\mathbb{D} = (D_0, \ldots, D_n)$ et une valuation v satisfaisant φ à l'instant i. Soit $T_0, \ldots, T_{n-i+1}$ la suite telle que

$$\begin{array}{l} T_0 = \emptyset \\ (\mathbb{D}, v, k+i) \models T_{k+1} = \psi(T_k) \text{ pour } k \in [0, n-i] \end{array}$$

Alors, le mot $a_{T_0,T_1} a_{T_1,T_2} \cdots a_{T_{n-i-1},T_{n-i}} b_{T_{n-i},T_{n-i+1}}$ est bien dans L. De plus, l'hypothèse de récurrence donne que $(\mathbb{D}, v, k+i) \models T_{k+1} = \Psi(T_k)$ pour $k \in [0, n-i]$. Comme de plus $t \in T_{n-i+1}$ pour la valuation v, Φ est bien vérifiée dans $(\mathbb{D}, v, i)$.

Réciproquement, prenons une base de données $\mathbb{D}$ et une valuation v satisfaisant Φ à l'instant i. Alors, il existe un mot $a_{R_0,R_1} \cdots a_{R_{n-i-1},R_{n-i}} b_{R_{n-i},R_{n-i+1}}$ avec $R_0 = \emptyset$ dans L tel que pour tout $k \in [0, n-i]$, $\Psi(i_k) = i_{k+1}$ et $v(t) \in i_n = \Psi(i_{n-1})$. Alors, la suite $T_0, \ldots, T_{n-i+1}$ telle que $T_k = i_k$ pour $k \in [0, n-i+1]$ est telle que $T_0 = \emptyset$ et $(\mathbb{D}, v, k+i) \models T_{k+1} = \psi(T_k)$ pour $k \in [0, n-i]$. Donc, φ est satisfaite dans $(\mathbb{D}, v, i)$. $\square$

Lemme 8 *ETL* est inclus dans FTL^+_{reg}.

DÉMONSTRATION
Comme dans le lemme précédent, pour établir que toute formule Φ de *ETL* possède un équivalent φ dans *foreach*, on procède par récurrence sur la taille de Φ. Le seul cas qui pose problème est celui où $\Phi = \mathcal{L}(\Psi_1, \ldots, \Psi_p)$. Soit alors par récurrence $\psi_1, \ldots, \psi_p$ équivalentes à $\Psi_1, \ldots, \Psi_p$. Soit L le langage associé à l'opérateur grammatical $\mathcal{L}$ et soit $\mathcal{A}$ un automate déterministe d'états $\mathcal{S}$ reconnaissant L. On suppose qu'on dispose d'un codage des sous-ensembles de $\mathcal{S}$ dans la base de données tel que l'ensemble des états initiaux de $\mathcal{A}$ est codé par $\emptyset$. On construit une formule ψ dans *FTL* qui prend en entrée un sous-ensemble S de $\mathcal{S}$ et et rend en sortie un sous-ensemble S' de $\mathcal{S}$ en effectuant les opérations suivantes :

- calculer l'ensemble W des lettres de L dont la formule associée φ_k est vraie dans l'état courant ;
- trouver l'ensemble S' des états de l'automate $\mathcal{A}$ dans lesquels on peut arriver en appliquant une lettre de W à un état de S ;

Alors, $\varphi = \exists t(F(t) \wedge \mathit{foreach}_S \psi(S))$ où F désigne les états finaux de $\mathcal{A}$, est équivalente à Φ. Intuitivement, elle exprime qu'il existe un chemin sur les états de l'automate qui part d'un état initial et qui arrive dans un état final de F, et dont les étiquettes $v_{w_c} \cdots v_{w_{c+m}}$ sont telles que les formules associées $\varphi_{w_c}, \ldots, \varphi_{w_{c+m}}$ sont vérifiées dans $(\mathbb{D}, w_c), \ldots, (\mathbb{D}, w_{c+m})$. $\square$