Grammaires locales étendues principes

Dans le chapitre 3 nous avons étudié les grammaires locales et leur pertinence pour traiter des problèmes liés au traitement automatique des langues (TAL). Nous avons défini une grammaire locale (LG) comme un formalisme de description de règles syntaxiques ou sémantiques sous forme de graphe. Nous avons également étudié la représentation d'une grammaire locale par des réseaux de transitions récursifs (RTNs) qui peuvent être vus comme des grammaires non-contextuelles (CFGs) dans lesquelles on admet que les membres droits des règles soient des automates finis au lieu de simples séquences de symboles.

De plus, nous avons évoqué le fait que ces grammaires peuvent également reconnaître des langages contextuels (type 1) et sans restriction (type 0) en reconnaissant tout d'abord un langage régulier (type 3) ou non-contextuel (type 2) et en appliquant par la suite une opération pour exclure les séquences qui n'appartiennent pas au langage à reconnaître (0 ou 1).

Nous avons vu que le formalisme des LGs a été implémenté dans des outils (Silberztein, 1994, Paumier, 2003a, Silberztein et Tutin, 2005, Blanc et al., 2006) destinés aux tâches du traitement automatique des langues telle que la recherche de motifs dans un texte qui a été éventuellement prétraité par des dictionnaires électroniques. Enfin, nous avons également soulevés certains inconvénients lors de leur utilisation.

Du point de vue de la mise en œuvre des grammaires locales, la possibilité de

simuler des grammaires non contextuelles est une conséquence de leur modélisation par des automates finis. En revanche, la gamme de contraintes représentables dépend de la mise en œuvre. Dans les outils disponibles, elle va d'une absence totale de contraintes jusqu'à un nombre prédéterminé de types de tests sur les variables : longueur, état (initialisée ou non), comparaison, etc, tous ceux-ci définis à l'avance et reconnus par l'analyseur de la grammaire.

Bien que la possibilité d'établir des contraintes sur les variables reste un atout indispensable pour augmenter le pouvoir de reconnaissance des grammaires locales, la non-standarisation des opérateurs parmi les outils disponibles, leur nombre réduit, le fait qu'il soit complexe de mettre en œuvre de nouvelles opérations sans modifier l'algorithme d'analyse syntaxique sous-jacent, ainsi que l'impossibilité de les paramétrer, de les adapter et de les modifier pour s'ajuster à différents types de problèmes, continuent à limiter la puissance de reconnaissance des grammaires locales. Entre autres, est en jeu la capacité d'une grammaire locale à conserver un bon niveau de précision, tout en améliorant leur rappel.

Dans ce chapitre nous introduisons la notion de grammaire locale étendue (ELG) comme une extension des grammaires locales classiques, plus particulièrement nous nous concentrons sur les grammaires locales qui sont représentées par des graphes syntaxiques et qui sont utilisés pour la recherche de motifs. Dans ce sens, il s'agit également d'un formalisme permettant de décrire formellement des ensembles de séquences grammaticales acceptées. Cependant, à la différence d'une grammaire locale, dans une grammaire locale étendue, il est possible d'associer aux étiquettes de sortie des transitions, des symboles non-terminaux qui représentent des appels à des actions conditionnelles, appelées *fonctions étendues*. Ces fonctions sont évaluées à l'extérieur de la grammaire, c'est-à-dire, en dehors de l'analyseur syntaxique.

Ainsi, alors qu'une grammaire locale permet uniquement de décrire des motifs syntaxiques ou sémantiques, une ELG peut en plus effectuer, au cours de l'analyse, des opérations arbitraires basées, par exemple, sur les motifs et sur l'état de l'analyse. Les fonctions étendues ont la possibilité de recevoir des paramètres, de réaliser des traitements, et de renvoyer comme résultat des symboles terminaux à concaténer à la sortie de la transition. Comme telles, les fonctions étendues peuvent être utilisées pour augmenter le pouvoir transformationnel des grammaires locales : repérer – en tolérant les fautes inattendues – des motifs dans un texte, puis les normaliser,les enrichir, les valider, les mettre en relation ou les baliser à la volée.

Au-delà de l'implémentation des grammaires traditionnelles, l'analyseur syntaxique d'une grammaire locale étendue peut être doté de la capacité de déclencher des événements, tel que la lecture d'un symbole d'entrée ou l'échec d'une reconnaissance. Ces actions peuvent également être implémentées sous la forme de fonctions étendues à l'extérieur de la grammaire locale, plus précisément, en dehors de la représentation des graphes.

Une telle possibilité implique un lien fort entre la modélisation des grammaires par des automates finis et l'analyseur syntaxique. Cependant, nous montrons comment ce lien peut à la fois garder une forte cohésion et se caractériser par un faible couplage.

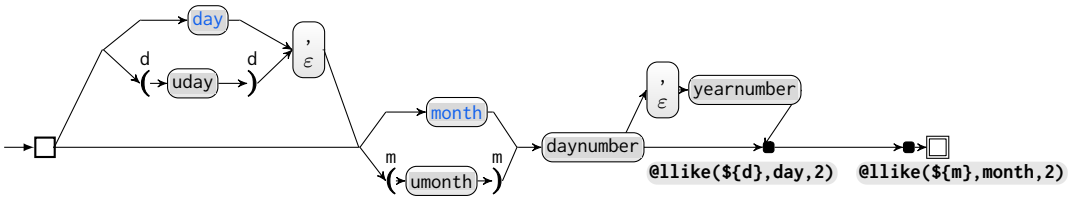
Autrement dit, l'analyseur peut considérer les fonctions étendues, dont les événements, comme des opérations abstraites et arbitraires qui n'ont pas besoin d'être définies à l'avance et qui ne changent pas la logique d'analyse.

Par exemple, les fonctions étendues mises en œuvre peuvent faire référence à des opérations sur les unités lexicales, si l'unité lexicale est 'book', nous avons des opérations comme '@reverse(book)' = 'koob', '@length(book)' = '5', '@equal(book,koob)' = false, '@upper(book)' = 'BOOK' ; sur les unités lexicales numériques comme '@greater(10,2)' = true, 'times(2,3)' = 6 ; sur des listes d'unités lexicales comme '@front(book ;koob)' = book, '@push(book ;koob,BOOK)' = book ;koob ;BOOK, '@get(book ;koob ;BOOK,2)' = koob, sur les masques lexicaux '@isverb(⟨manger.V⟩)' = true, '@similarity(Bordaux, ⟨N+Toponym⟩, 0.9)' ainsi que les positions particulières (absolues et relatives) de la séquence d'entrée '@begin()', '@end()', '@seek()'.

Parmi les avantages des grammaires locales étendues on peut citer la possibilité de réutiliser des fonctions externes à la grammaire afin d'améliorer ou agrandir les capacités de reconnaissance. Ces fonctions externes peuvent mettre en œuvre des approches hybrides au lieu de se limiter à des stratégies fondées sur des règles. Finalement, les fonctions étendues peuvent être organisées dans des bibliothèques réutilisables et partagées pour être intégrées à d'autres grammaires.

4.1 Aperçu général

Le graphe 4.1 représente une grammaire locale étendue pour la reconnaissance de dates. La fonction étendue `llike` (*looks like*) vérifie si une séquence inconnue, stockée dans la variable *d*, est similaire (avec un seuil maximal de 2) à un nom de jour, de même, elle teste, dans la deuxième appelle, si la séquence stockée sur la variable *m* est similaire à un nom de mois.



Graphe 4.1 – Grammaire locale étendue pour la reconnaissance des dates

Des exemples de séquences reconnues par cette grammaire, colonne ELG, sont présentés dans le tableau 4.1.

Séquences	COGITO [1]	LG [2]	TXRZR [3]	STNER [4]	ELG [5]	Valide ? [6]
March 14th	✓	✓	✓	✓	✓	✓
Friday, October 7, 1966	✓	✓	✓	✓	✓	✓
Dceember 21, 1985	×	×	×	✓	✓	✓

Séquences	COGITO	LG	TXRZR	STNER	ELG	Valide ?
Monda Yanary 15, 2007	×	×	✓	×	✓	✓
Jueves, Septiembre 30, 2004	×	×	×	×	×	✓
Championships 2nd 2004	×	×	×	×	×	×

Tableau 4.1 – Exemples de dates reconnues par différents systèmes et par une ELG

L'implémentation de la fonction étendue `llike`, et par conséquent de la notion de ressemblance, est arbitraire et peut dépendre de la nature du problème à traiter. Nous pouvons définir, par exemple, comme présentée plus bas, qu'une séquence ressemble au nom d'un jour si la distance d'édition (Levenshtein, 1966) entre le nom d'un jour et la séquence est inférieure ou égal 2. Par exemple, `distance(Wednesday,Wednesdy) = 1` mais `distance(Wednesday,Wedgelike) = 5`.

```
function llike(input,entity,threshold)
  if input == nil then return true end
    if entity == 'month' then
      if levenshtein({'January','February',...}, input) <= threshold then
        return true
      end
      elseif entity == 'day' then
        if levenshtein({'Sunday','Monday',...}, input)    <= threshold then
      return true
    end
  end
end
```

La fonction étendue `llike` utilise une approche naïve pour rechercher approximativement un motif p (input) dans une liste W (e.g. 'January','February',...). Des autres approches sont plus adéquates à mettre en œuvre. Cependant, ce qui est important à retenir est que l'analyseur syntaxique ne connaît pas à l'avance cette fonction étendue et n'est pas responsable de son évaluation. Cela implique que la fonction peut être modifiée ou améliorée sans changer l'algorithme d'analyse syntaxique, et même sans apporter des changements au graphe graphe 4.1, et donc sans le recompiler.

Le graphe 4.2 illustre un autre exemple de l'utilisation des grammaires locales étendues, il s'agit de la reconnaissance des expressions du type :

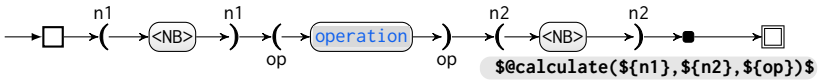
- *2 fois 13*
- *16 divisé par 4*

6. Cogito demo API : http://cogitoapi.com/demo,context:kernel.
6. Grammaire locale classique
6. TextRazor demo : https://www.textrazor.com/demo
6. Stanford NER demo : http://nlp.stanford.edu:8080/ner, classifier : english.muc.7class.
6. Grammaire locale étendu

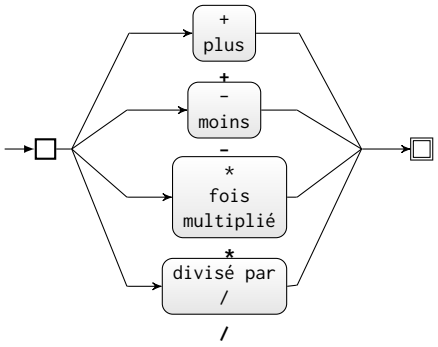
- *25 moins 15*
- *45 + 32*

Afin de produire des sorties comme :

- *2 fois 13* *= 26*
- *16 divisé par 4* *= 4*
- *25 moins 15* *= 10*
- *45 + 32* *= 77*



(a) graphe principal



(b) sous-graphe operation

Graphe 4.2 – Calculatrice

Pour parvenir à produire ce résultat, la fonction étendue `calculate` est utilisée, un exemple de sa mise en œuvre est illustrée ci-après :

```
function calculate (op1, op2, op)
  if not uMatch.start_with_space() then return false end
    if     op == "+" then return tostring(op1 + op2)
    elseif op == "-" then return tostring(op1 - op2)
    elseif op == "*" then return tostring(op1 * op2)
    elseif op == "/" then return tostring(op1 / op2)
    else                  return false
  end
end
```

Dans l'exemple `uMatch.start_with_space()` fait partie des opérations accessibles aux fonctions étendues pour connaître l'état de l'analyse, dans le cas présent, il est requis que le premier *token* du motif qui viens d'être reconnu se trouve au début de l'entrée ou après l'espace.

4.2 Principes

En premier lieu, une grammaire locale étendue inclut les caractéristiques classiques d'une grammaire locale traditionnelle [1] : appel à des sous-graphes, utilisation de variables, de contextes, de filtres morphologiques, de masques lexicaux (Gross, 1993, 1997, Silberztein, 2003, Silberztein et Tutin, 2005, Paumier, 2016), etc.

De plus, une grammaire locale étendue enrichit le modèle des grammaires locales par un nouveau formalisme qui permet d'ajouter, par des transitions étiquetées, des fonctions conditionnelles qui sont externes à l'analyseur syntaxique. Ces fonctions permettent d'inclure ainsi, à la volée, dans les analyses effectuées, la combinaison d'une ou plusieurs approches, comme des calculs mathématiques, des manipulations de chaînes de caractères, des analyses statistiques, l'interrogation de bases de données, ou l'exploitation d'autres ressources utiles.

Dans leur représentation, le graphe est composé également par un groupe de boîtes (les transitions du diagramme d'états-transitions) reliées par des arcs avec une unique boîte désignée comme initiale (l'état initial du diagramme d'états-transitions) et une seule boîte désignée comme finale (l'état final du diagramme d'états-transitions), chaque boîte, à l'exception de celle associée à l'état final, est étiquetée en entrée (contenu de la boîte) et facultativement en sortie (contenu en dessous de la boîte).

Comme dans le cas des LGs, les étiquettes d'entrée peuvent contenir des mots, des méta-symboles, des traits sur les mots (contraintes lexicales sur les mots), le mot vide, l'appel à un sous-graphe (en indiquant son nom), etc.

Les étiquettes de sortie peuvent contenir des suites de symboles, comportant zéro ou plus sous-séquences de symboles non-terminaux toujours sous la forme $\varphi(\Delta)$ représentant chacuns l'appel à une fonction externe à la grammaire. φ est le nom de la fonction étendue, Δ est un n–uplet $(a_1, a_2, \ldots, a_n)$ où chaque a_i est dénommé argument de φ. Un argument est soit une suite finie de symboles terminaux, désormais appelé argument littéral, soit un registre de la grammaire, désormais appelé argument variable.

L'entrée de la grammaire locale étendue est une séquence d'unités telle que des caractères ou des tokens. Lors de leur analyse, c'est-à-dire, l'application d'une ELG sur la séquence d'unités lexicales en entrée, une sous-séquence est reconnue si, en lisant le graphe de gauche à droite, il existe un chemin allant de l'état initial à l'état final où chacune des transitions parcourues reconnaît la sous-séquence présentée en entrée tout en satisfaisant les fonctions étendues rencontrées.

De la même manière que la notion de grammaire locale est comparable à celle d'un RTN, la notion de grammaire locale étendue que nous introduisons peut être rapprochée du modèle du réseaux de transitions augmenté (ATN, Woods, 1970, Bates, 1978). Le modèle d'ATN a été introduit indépendamment par Thorne et al. (1968,

1. Il est utile de rappeler que dans le périmètre de nos travaux, nous parlons des grammaires locales classiques en faisant référence au formalisme des graphes syntaxiques disponibles dans des outils comme UNITEX ou NOOJ

citée par Kaplan, 1972, p. 5) et généralisé par Woods (1970) comme une approche du traitement automatique du langage. Un ATN est une version augmentée du modèle des RTNs par la possibilité qu'offre un tel réseau d'ajouter 1) des registres, 2) des conditions aux arêtes et 3) d'associer à celles-ci des actions. Néanmoins, nous pouvons mettre en avant deux sortes de différences entre les ELG et les ATN. D'une part, on retrouve les mêmes différences que celles existant entre les grammaires locales et les RTNs :

- Les grammaires locales peuvent produire des sorties.
- Les grammaires locales peuvent faire référence à des ressources linguistiques.

D'autre part, on note des différences supplémentaires entre les ATN et les ELG :

- La notion de fonction étendue diffère de celle d'action dans les ATN, d'un part parce que les fonctions étendues peuvent communiquer de forme bidirectionnelle avec l'analyseur syntaxique et aussi entre elles, d'autre part, parce qu'elles peuvent retourner des résultats à concaténer à la sortie de la grammaire.
- Dans une ATN le cycle de vie des actions est limité à leur temps d'exécution, les fonctions étendues sont évaluées à chaque appel, mais peuvent conserver des registres globaux.
- Dans une ATN il n'est pas possible de faire face à des erreurs inattendues, par exemple, des phrases qui échouent à être analysées pour contenir un mot inconnu. En revanche, la notion de fonction étendue et l'ajout des événements permet aux grammaires locales étendues de mettre en place des techniques de correction locale au niveau de la lecture de symboles d'entrée ou des opérations appelées à partir du graphe.
- Les fonctions étendues sont évaluées par une machine abstraite et indépendantes de l'algorithme d'analyse syntaxique, autrement dit, il n'est pas nécessaire que l'analyseur connaisse à l'avance la fonction pour parvenir à la traiter.
- Les fonctions étendues ont accès aux mêmes ressources utilisées par l'analyseur syntaxique, tels que les dictionnaires morphologiques ou les tokens et peuvent les consulter et les modifier. En outre, les fonctions étendues ont aussi accès à des ressources externes, tels que des fichiers ou des bases de données.

4.3 Préliminaires

En plus des notions décrites dans l'annexe A et des définitions données au chapitre 2, nous présentons quelques définitions et conventions supplémentaires que nous utilisons dans la description des ELGs.

Symboles

Nous avons défini un alphabet (cf. définition A.25) comme un ensemble fini et *non-vide* d'éléments appelés symboles. Dans la présentation des ELGs, nous distinguons deux classes de symboles appartenant à un même alphabet :

1. **des symboles terminaux**, et
2. **des symboles non-terminaux**

Alphabets

Nous considérons un alphabet d'entrée (Σ) et un alphabet de sortie Γ. Chacun des alphabets est constitué par deux sous-alphabets disjoints, l'un de symboles terminaux, l'autre de symboles non-terminaux.

Alphabet d'entrée

L'alphabet d'entrée Σ d'une ELG est constitué par l'union de deux sous-alphabets disjoints :

1. $\overline{\Sigma}$: l'alphabet des symboles **terminaux** d'entrée, et
2. Ω : l'alphabet des symboles **non-terminaux** d'entrée.

Nous symbolisons par Σ_ε l'union entre l'alphabet d'entrée Σ et un ensemble ne contentant que le mot vide $\{\varepsilon\}$, soit $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Alphabet de sortie

L'alphabet d'entrée Γ d'une ELG est constitué par l'union de deux sous-alphabets disjoints :

1. $\overline{\Gamma}$: l'alphabet des symboles **terminaux** de sortie, et
2. Φ : l'alphabet des symboles **non-terminaux** de sortie, aussi appelé **alphabet étendu**.

Il est utile de rappeler que nous symbolisons par Γ_ε l'union entre l'alphabet de sortie Γ et un ensemble ne contentant que le mot vide $\{\varepsilon\}$, soit $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.

Enfin, le tableau 4.2 résume les conventions pour représenter les symboles des alphabets des ELGs.

Alphabet	Symboles	Mots
Σ	$\{\sigma_1, \sigma_2, \ldots, \sigma_{\lvert\Sigma\rvert}\}$	
$\overline{\Sigma}$	$\{\overline{\sigma}_1, \overline{\sigma}_2, \ldots, \overline{\sigma}_{\lvert\overline{\Sigma}\rvert}\}$	
Ω	$\{\varrho_1, \varrho_2, \ldots, \varrho_{\lvert\Omega\rvert}\}$	
Γ	$\{\gamma_1, \gamma_2, \ldots, \gamma_{\lvert\Gamma\rvert}\}$	
$\overline{\Gamma}$	$\{\overline{\gamma}_1, \overline{\gamma}_2, \ldots, \overline{\gamma}_{\lvert\overline{\Gamma}\rvert}\}$	
Φ	$\{\phi_1, \phi_2, \ldots, \phi_{\lvert\Phi\rvert}\}$	

Tableau 4.2 – Conventions pour représenter les symboles des alphabets des ELGs

Étiquettes

Au chapitre 3 nous avons vu que dans une LG, les étiquettes des transitions, en plus des symboles terminaux, peuvent contenir des symboles non-terminaux. Dans le cas des ELG que nous présentons par la suite, les **étiquettes de sortie** des transitions peuvent aussi être constituées de symboles terminaux et non-terminaux (vus comme symboles terminaux au regard des opérations sur les automates).

Machines abstraites

Nous définissons une machine abstraite $\mathcal{M}$ comme un 5–uplet comprenant 1) des registres permettant de stocker des valeurs 2) de constantes 3) de variables et 4) un état global, et 5) des opérations. Une opération reçoit des **entrées**, aussi appelées **variables**, et retourne des **sorties**. Plus précisément, une opération est une fonction de X dans Y qui définit une relation entrée-sortie en permettant de faire correspondre aux entrées ($x \in X$) des sorties ($y \in Y$) selon une règle ou un algorithme.

4.4 Modélisation

Une **grammaire locale étendue** définie sur un alphabet d'entrée $\Sigma = \overline{\Sigma} \cup \Omega$, tel que $\overline{\Sigma} \amalg \Omega$, et un alphabet de sortie $\Gamma = \overline{\Gamma} \cup \Phi$, tel que $\overline{\Gamma} \amalg \Phi$, est un n–uplet $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \psi, \theta, \mathfrak{X}, q_0, F)$, où :

1. Q est un ensemble fini d'**états**,
2. Σ est l'alphabet d'**entrée**,
3. Γ est l'alphabet de **sortie**,
4. $\delta : Q \times (\overline{\Sigma} \cup \Omega) \times (\overline{\Gamma} \cup \Phi) \longrightarrow \wp(Q \times (\overline{\Gamma} \cup \Phi))$ est la **fonction de transition**,
5. $\mathfrak{X}$ est le symbole d'échec,
6. $q_0 \in Q$ est l'**état initial**, et
7. $F \subseteq Q$ est l'ensemble des **états finaux**.

Fonction de transition

Rappelons que la fonction de transition d'un automate fini (FSA) décrit comment un automate passe à un autre état en « lisant » un symbole de l'alphabet. Dans les automates finis non-déterministes avec ε–transitions (ε–NFAs), la fonction de transition δ est égale à :

$$Q \times \Sigma_\varepsilon \longrightarrow \wp(Q)$$

Les arguments de la fonction transition sont ainsi définis par un état de départ ($q_i \in Q$) et par un symbole d'entrée égal au mot vide ε ou à un symbole σ de l'alphabet d'entrée ($\sigma \in \Sigma$), donc par une paire ordonnée de la forme (q_i, ε) ou (q_i, σ), tel que :

1. Si c'est (q_i, ε), la transition est spontanée, autrement dit, qu'elle fait passer l'automate de l'état q_i dans un autre état sans qu'un symbole d'entrée ne soit consommé.
2. Si c'est $(q_i, \overline{\sigma})$, la transition est réalisée en consommant le symbole d'entrée terminal $\overline{\sigma}$ [1].
3. Si c'est (q_i, ϱ), la transition est effectuée en consommant, ou pas, un symbole d'entrée, ceci dépend de la sémantique associée au symbole d'entrée non-terminal ϱ [2].

En outre, étant donnée (q_i, σ), la fonction de transition δ renvoie un ensemble de parties (cf. définition A.7) de Q, donc des états d'arrivée parmi les états de Q ou l'ensemble vide $\varnothing$.

Deux classes d'arguments de la fonction de transition sont alors possibles :

a. (q_i, ϕ)
b. (ε, ϕ)

Notons que les cas 2.a (ε–cycle) n'est pas accepté. En effet, le cas $\varepsilon : \phi$ signifierait qu'il est possible d'appeler infiniment une fonction sans avoir lu d'entrée.

De même, observons que la sortie de la fonction de transition est un ensemble de paires ordonnées de la forme (q_j, y) de $\wp(Q \times \Gamma_\varepsilon)$. Ceci indique qu'étant donnée une entrée (q_i, a, x), la fonction de transition peut définir en sortie : 1. plusieurs états d'arrivée parmi les états de Q, 2. pour tout état d'arrivée $q_j \in Q$, un symbole $y \in \Gamma_\varepsilon$ à mettre à jour au sommet de pile lorsque le changement d'état est réalisé.

Par exemple, la règle de transition $\delta(q_i, a, x) = (q_j, y)$, précise que lorsque l'automate se trouve à l'état q_i et *lit* a sur l'entrée, il peut remplacer le symbole x au sommet de pile par y et passer dans l'état q_j. L'étiquette d'une telle transition est notée $a, x \to y$ et est illustré à la figure 2.8.

1. Notons que $\overline{\sigma} \in \overline{\Sigma}$ et $\overline{\Sigma} \subseteq \Sigma$
2. Notons que $\varrho \in \Omega$ et $\Omega \subseteq \Sigma$

- $\delta : Q \times \Sigma_\varepsilon \longrightarrow \wp(Q)$. La fonction de transition δ reçoit un état Q et un symbole d'entrée Σ_e et retourne un ensemble de parties $\wp(Q)$,

- $\psi : Q \times \Sigma_\varepsilon \times Q \longrightarrow \Omega^*$. La fonction de sortie étendue ψ reçoit un état courant Q, un symbole d'entrée Σ_ε et un état suivant Q et retourne une séquence finie de symboles, appelée sortie étendue, de l'alphabet étendu $\Omega = \Phi_\varepsilon \cup \Gamma_\varepsilon$,

- $\mathcal{M} : \Omega^* \longrightarrow \Gamma^* \cup \{\mathfrak{X}\}$. $\mathcal{M}$ est une machine abstraite qui prend en entrée une sortie étendue $\Omega^* = \{\Phi_\varepsilon \cup \Gamma_\varepsilon\}^*$ et qui retourne une suite finie de symboles de l'alphabet de sortie Γ_ε ou le symbole d'échec $\mathfrak{X}$,

- $\theta : Q \times \Sigma_\varepsilon \times \mathcal{M}(\Omega^*) \times Q \longrightarrow \{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$. La fonction de sortie θ reçoit un état courant Q, un symbole d'entrée Σ_ε, la sortie de $\mathcal{M}(\Omega^*)$, et un état suivant Q. Elle retourne l'étiquette de sortie de la transition constituée d'une suite finie de symboles de l'alphabet résultant de l'union de l'alphabet de sortie et d'entrée $\{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$.

Les états (Q) de l'automate sont reliés par des arcs orientés représentant les transitions d'un état à un autre. Chaque transition $t = (p[t], li[t], f[t], lo[t], n[t]) \in$ E est ainsi un arc qui part d'un état source (ou état précédent p) vers un état destination (ou état postérieur n), avec une étiquette d'entrée $li[t]$, une fonction étendue $f[t]$, et une étiquette de sortie $lo[t]$ (cf. figure 4.1).

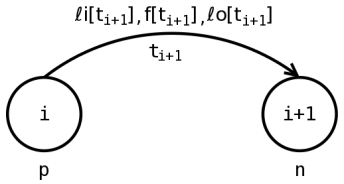


Figure 4.1 – Transition sur une grammaire locale étendue

Une transition peut être étiquetée en entrée soit par le symbole vide (ε), pour indiquer qu'elle ne « consomme » pas d'entrée ; soit par des symboles terminaux, par exemple, des mots (portions du texte en entrée), des méta-symboles, des traits sur les mots (contraintes lexicales sur les portions du texte en entrée), etc ; soit par le nom d'un autre automate (symbol non-terminal) dont l'état associé est l'état initial du sous-automate appelé (cf. figure 4.2). Dans le cas d'un symbole terminal, le critère de transition de i vers $i+1$ est que le mot courant qui est analysé en entrée (x_i) satisfait l'étiquette d'entrée de la transition, par exemple si $li[ti{+}1] = red, < MIN >, < ADJ >$, un changement à l'état $i+1$ sera possible si xi est égal soit au mot « red », soit à un mot en minuscules (par exemple, « of »), soit à un adjectif (par exemple, « strong »). Si la transition est satisfaite, alors le mot xi est consommé, s'il existe encore des séquences en entrée, le mot courant devient alors le mot suivant en entré (xi+1). Dans le cas d'un appel récursif, la transition ne consomme pas des séquences d'entrée et le critère de transition de i vers $i+1$ dépendra alors de l'existence d'au moins un chemin accepteur dans le sous-automate appelé.

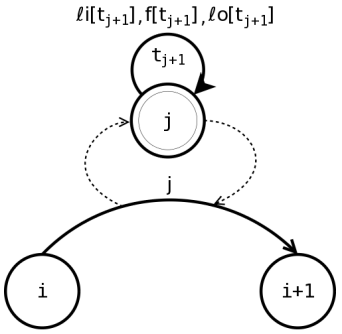


Figure 4.2 – Appel à un sous-graphe sur une grammaire locale étendue

Une fonction f[t] est représentée par une paire « i : o », i étant un ensemble de zéro ou plus paramètres d'entré et o = f(i) étant égal à la valeur qui renvoie la fonction après évaluation. Le critère d'acceptation de « f » est que la valeur retournée soit une sous-étiquette (l[ft]) différente du symbole d'échec ($\mathfrak{X}$) ou une valeur booléenne « vrai », cela implique que la fonction « attachée » à la transition ti doit être satisfaite pour avancer à l'état i. Par convention, les fonctions sont associées aux transitions en écrivant « `@f(i)` » où f est le nom de la fonction à appeler et i l'ensemble paramètres d'entré (cf. figure 4.3).

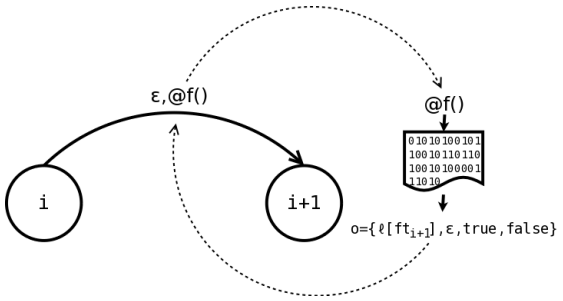


Figure 4.3 – Appel à une fonction sur une grammaire locale étendue

Dans la figure 4.3 la fonction f[ti+1] = @f(i) associée à la transition ti+1 est satisfaite si, après être évaluée, elle renvoie une valeur différente de $\mathfrak{X}$ ou « faux ». En outre, la transition ti+1 est étiquetée en entrée par le symbole vide (ε), ce qui indique qu'elle ne consomme pas d'entrée.

Une transition peut être étiquetée en sortie, sous la forme d'une séquence de caractères, par la concaténation de l[ft] et lo[t], c'est-à-dire, par la sous étiquette retournée après évaluation de ft et l'étiquette de sortie associée à la transition t (cf. figure 4.4).

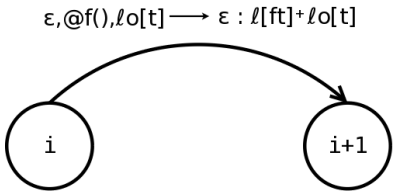


Figure 4.4 – Transition étiquetée en sortie sur une grammaire locale étendue

Un chemin de l'automate est une séquence de transitions t1, t2, t3 ... tn avec n[ti] = p[ti+1] pour i=1, ..., n-1 (cf. figure 4.5). L'étiquette d'entrée d'un chemin est le résultat de la concaténation des étiquettes d'entrée des transitions associées au chemin, soit li[π] = li[t1] ... li[tn]. L'étiquette de sortie d'un chemin est le résultat de la concaténation des étiquettes de sortie des transitions associées au chemin, soit lo[π] = l[f1] + lo[t1] ... l[fn] + lo[tn].

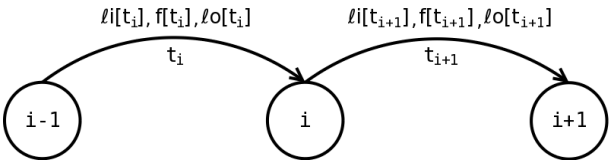


Figure 4.5 – Chemin sur une grammaire locale étendue

L'automate qui modélise la grammaire locale étendue débute dans un état initial (premier cercle en gras à gauche) et change d'état lorsque l'ensemble des conditions attachées à la transition t sont satisfaites. Si au cours de chaque changement l'automate vérifie toutes les conditions, il finira pour aboutir à l'état final (dernier double cercle à droite). Alors, un chemin accepteur π = t1, t2, t3 ... tn sera le chemin partant de l'état initial vers l'état final f $\in$ F (cf. figure 4.6).

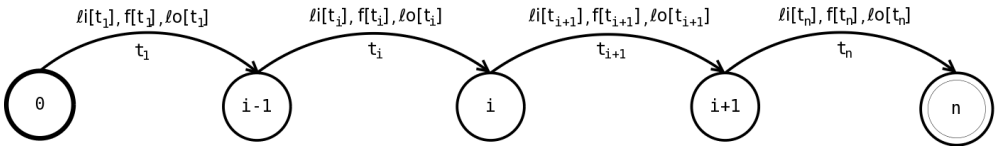


Figure 4.6 – Chemin accepteur sur une grammaire locale étendue

Une sous-séquence x = u1,u2,...,un de caractères ou de tokens donnés en entrée de la grammaire, est acceptée par l'automate s'il existe un chemin accepteur π partant de s et qui est étiqueté comme x : li[π] = x (cf. figure 4.7). Comme pour les grammaires locales, l'automate résultant est localement efficace mais ne satisfait pas, par exemple, un critère de minimisation globale.

Exemple :

1. $Q : \{q_0, q_1, q_2, q_3, q_4, q_5\}$
2. $\Sigma : \{aperçoit, Juliette, Roméo\}$
3. $\Phi : \{inc(), fail()\}$
4. $\Gamma : \{(,), a, b, c, \ldots, z\}$
5. δ :

$\delta(q_0, \varepsilon) = q_1,$	$\delta(q_1, \langle \text{WORD} \rangle) = q_2,$	$\delta(q_1, \langle \text{TOKEN} \rangle) = q_3,$	$\delta(q_2, \varepsilon) = q_5,$
$\delta(q_3, \langle \text{TOKEN} \rangle) = q_4,$	$\delta(q_4, \varepsilon) = q_6,$	$\delta(q_6, \varepsilon) = q_5,$	

6. ψ :
   $\psi(q_0, \varepsilon, q_1) = inc(), \mid \psi(q_2, \varepsilon, q_5) = inc(), \mid \psi(q_4, \varepsilon, q_6) = inc(), \mid \psi(q_6, \varepsilon, q_5) = fail(),$
7. $\theta : \varepsilon$
8. $\mathfrak{X}$
9. q_0
10. $F : \{q_5\}$

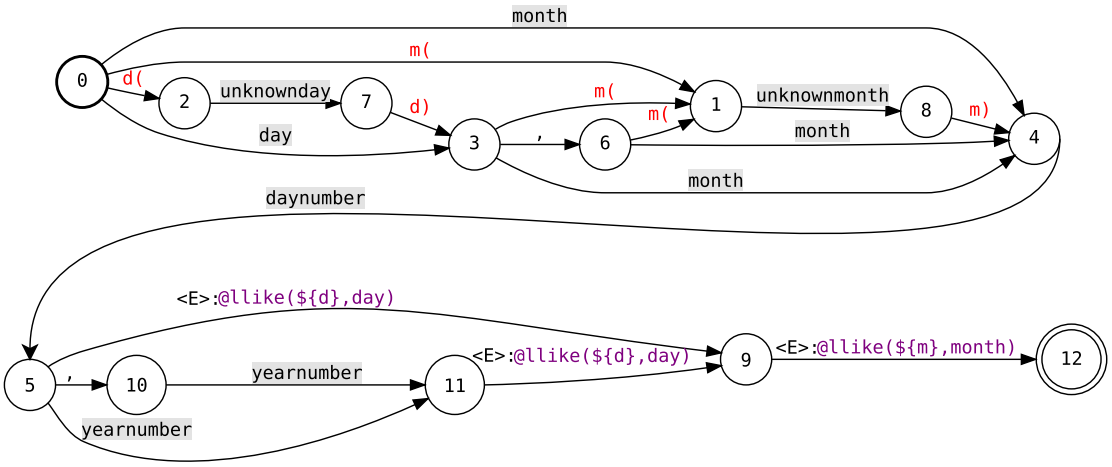


Figure 4.7 – Automate équivalent au graphe 4.1

4.5 Aplatissement

Une grammaire non contextuelle est récursive lorsque une variable à gauche d'une règle se réfère à une autre dans la partie à droite. La dérécursivation est une opération qui permet d'interdire les cycles et donc de transformer une grammaire locale en une machines à états finis. Une des techniques de dérécursivation est l'aplatissement, il consiste à remplacer une transition t étiquetée en entrée par un symbole non-terminal qui désigne l'appel à un sous-graphe par une copie exclusive du sous-graphe. L'automate résultant est obtenu en parcourant la grammaire à partir du graphe principal et, pour chaque étiquette non-terminale qui fait appel à un sous-graphe, en remplaçant récursivement la transition par le sous-graphe appelé. Lors du processus, il est possible d'aboutir à une dépendance cyclique parmi un sous-ensemble de transitions qui appellent un sous-graphe et boucler indéfiniment. La stratégie pour palier à ce problème peut être abordée de façon passive ou active :

- De manière passive, il est nécessaire de vérifier les dépendances cycliques avant l'aplatissement, si des dépendances cycliques sont détectées, alors produire une erreur. Cette démarche est par exemple l'implémentée dans OpenFst pour la commande *replace* en charge de l'aplatissement.
- De manière active, est possible de démarrer l'algorithme en fixant un seuil d'imbrication n au-delà duquel les appels à un sous-graphe sont tronqués et remplacés par des transitions vides. Dans les cas ou le niveau maximal de récursion est atteint, l'automate résultant n'est pas strictement équivalent à la grammaire d'origine. Cette approche est aussi dénommée par *aplatissement contrôlé* (Paumier, 2003a, p. 138) et est utilisée dans Unitex et Outilex pour les commandes *flatten* et *wrtn-flatten*.

4.6 Caractéristiques principales

En plus d'hériter des caractéristiques des LGs, les ELGs ont les caractéristiques principales suivantes :

1. Créer des conditions implicites et explicites sur les transitions.
2. Faire appel à des fonctions arbitraires.
3. Faire référence à des ressources externes.
4. Utiliser les sorties des fonctions pour générer des sorties.
5. Stocker des variables globales.
6. Utiliser des contextes éloignés.
7. Déclencher des événements.
8. Modifier la fenêtre d'analyse.
9. Consulter les dictionnaires morphologiques.

4.7 Évaluation des fonctions étendues

Rappelons qu'une fonction étendue φ reçoit un n–uplet d'arguments Δ et retourne, soit une suite finie de symboles γ de l'alphabet de sortie Γ_ε, soit le symbole d'échec $\mathfrak{X}$:

$$\varphi(\Delta) = \gamma \in \Gamma^* \cup \{\mathfrak{X}\}$$

En outre, dans la sous-section 2.2.2 nous avons défini comme non-déterministe tout automate fini qui à partir d'un symbole donné et d'un état de départ a comme

destination plusieurs états d'arrivée, ou dans lequel il est admis de changer d'état sans lire de symbole.

La faculté des grammaires locales de choisir quel sera le prochain état, ou de passer dans un autre d'état spontanément, implique aussi que dans une grammaire locale étendue, les fonctions qui sont attachées aux transitions doivent être évaluées selon un principe qui empêche la modification des registres associés à la fonction lorsque elle n'est pas satisfaite, soit quand $\varphi(\Delta) = \mathfrak{X}$.

Le principe qui assure que l'évaluation d'une fonction se réalise de façon « *tout ou rien* » peut s'assimiler à la propriété d'*atomicité* dans un système de gestion de base de données[1]. En effet, dans ceux-ci, une opération, appelée transaction, est considérée comme une unité indivisible, de telle sorte que si elle ne s'achève pas, il est requis de laisser les données dans l'état précédent, celui antérieur à sa réalisation.

Ce principe est élargi à l'évaluation d'une sérié de fonctions étendues $\varphi_1, \varphi_2, \cdots, \varphi_n$ qui sont attachés à une suite de transitions menant d'un état p à un état r. Ainsi, si une fonction φ_i n'est pas satisfaite, les modifications effectuées par φ_i, ainsi que par toute autre fonction précédente et satisfaite $\varphi_1, \varphi_2, \cdots, \varphi_{i-1}$, ne doivent être :

1. ni prises en compte lors de une nouvelle évaluation de la fonction,

2. ni visibles lors de l'évaluation d'autres fonctions.

Comme conséquence, si φ_i n'est pas satisfaite, la série de changements $\varphi_{1,i}$ n'est pas prise en compte et les registres globaux sont restaurés tels qu'ils étaient à l'état p. Enfin, il est utile de rappeler que si $p \xrightarrow{\alpha_{i,j}} r$ n'est pas un chemin réussi, la série de fonctions satisfaites $\varphi_1, \varphi_2, \cdots, \varphi_n$ est considérée comme incomplète et leur effets ne sont pas pris en compte. En d'autres termes, le principe d'atomicité implique que l'évaluation d'une série de fonctions étendues attachées à une suite de transitions menant d'un état p à un état r est complète si :

1. $\forall i \in n, \varphi_i(\Delta) \neq \mathfrak{X}$ (toutes les fonctions étendues sont satisfaites)

2. $p \xrightarrow{\forall i \in n, \varphi_i(\Delta) \neq \mathfrak{X}} r$, $p = q_0$ et $r \in F$ (la série de fonctions satisfaites se trouve dans un chemin réussi)

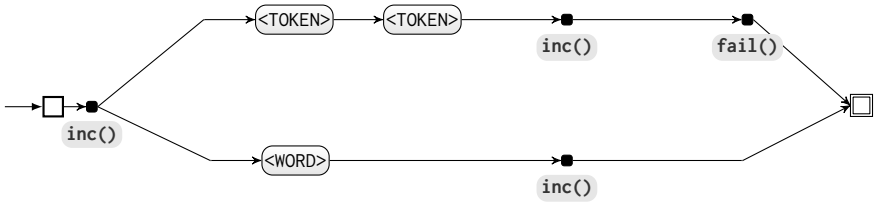
Afin d'illustrer l'atomicité de l'évaluation des fonctions étendues, considérons d'abord l'exemple 4.1 où ce principe n'est pas respecté.

1. En poursuivant le parallèle comparatif, nous pouvons dire, par exemple, qu'une propriété semblable à celle de l'*isolation* des bases de données, mais appliquée aux registres accessibles à travers les fonctions étendues, est garantie dès lors que tout évaluation simultanée est interdite, cette contrainte est respectée comme conséquence de la nature séquentielle de l'algorithme d'analyse de la grammaire. Par ailleurs, nous considérons qu'une propriété comme celle de la *durabilité* des transactions n'est proprement requise.

Exemple 4.1. Analysons la phrase d'exemple 12 :

(12) Roméo aperçoit Juliette

à l'aide de la ELG définie par le graphe 4.3 :



Graphe 4.3 – Évaluation des fonctions étendues

du registre global :

- $i = 0$

et des fonctions étendues `inc` et `fail` définies comme :

- $\varepsilon \xrightarrow[i=i+1]{\texttt{inc}()} \varepsilon$ (ne reçois pas d'arguments, incrémente i de 1, retourne le symbole vide)
- $\varepsilon \xrightarrow[i=i+1]{\texttt{fail}()} \mathfrak{X}$ (ne reçois pas d'arguments, incrémente i de 1, retourne le symbole d'échec [1])

Pour faciliter l'illustration, nous présentons également le graphe 4.3 sous la forme d'un automate fini :

$$\mathcal{L} = (Q, \Sigma, \Phi, \Gamma, \delta, \psi, \theta, \mathfrak{X}, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3, q_4, q_5\}$
2. $\Sigma : \{aperçoit, Juliette, Roméo\}$
3. $\Phi : \{inc(), fail()\}$
4. $\Gamma : \{(,), a, b, c, \ldots, z\}$
5. $\delta :$

$\delta(q_0, \varepsilon) = q_1,$	$\delta(q_1, \langle \text{WORD} \rangle) = q_2,$	$\delta(q_1, \langle \text{TOKEN} \rangle) = q_3,$	$\delta(q_2, \varepsilon) = q_5,$
$\delta(q_3, \langle \text{TOKEN} \rangle) = q_4,$	$\delta(q_4, \varepsilon) = q_6,$	$\delta(q_6, \varepsilon) = q_5,$	

6. $\psi :$

$$\psi(q_0, \varepsilon, q_1) = inc(), \mid \psi(q_2, \varepsilon, q_5) = inc(), \mid \psi(q_4, \varepsilon, q_6) = inc(), \mid \psi(q_6, \varepsilon, q_5) = fail(),$$

7. $\theta : \varepsilon$

1. Autrement dit, la fonction n'est pas satisfaite

8. $\mathfrak{X}$
9. q_0
10. $F : \{q_5\}$

dont la représentation sous forme de diagramme d'états-transitions est montrée à la figure 4.8 :

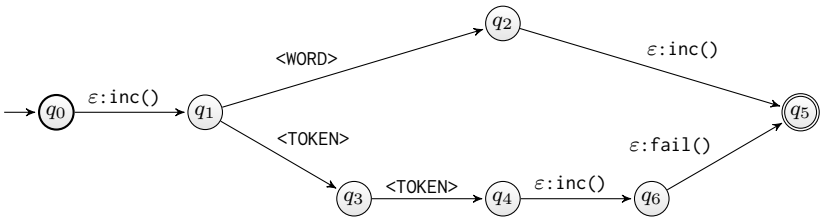


Figure 4.8 – Automate fini équivalent au graphe 4.3

Sachant que :

- $\delta : Q \times \Sigma_\varepsilon \longrightarrow \wp(Q)$. La fonction de transition δ reçoit un état Q et un symbole d'entrée Σ_e et retourne un ensemble de parties $\wp(Q)$,

- $\psi : Q \times \Sigma_\varepsilon \times Q \longrightarrow \Omega^*$. La fonction de sortie étendue ψ reçoit un état courant Q, un symbole d'entrée Σ_ε et un état suivant Q et retourne une séquence finie de symboles, appelée sortie étendue, de l'alphabet étendu $\Omega = \Phi_\varepsilon \cup \Gamma_\varepsilon$,

- $\mathcal{M} : \Omega^* \longrightarrow \Gamma^* \cup \{\mathfrak{X}\}$. $\mathcal{M}$ est une machine abstraite qui prendre en entrée une sortie étendue $\Omega^* = \{\Phi_\varepsilon \cup \Gamma_\varepsilon\}^*$ et qui retourne une suite finie de symboles de l'alphabet de sortie Γ_ε ou le symbole d'échec $\mathfrak{X}$,

- $\theta : Q \times \Sigma_\varepsilon \times \mathcal{M}(\Omega^*) \times Q \longrightarrow \{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$. La fonction de sortie θ reçoit un état courant Q, un symbole d'entrée Σ_ε, la sortie de $\mathcal{M}(\Omega^*)$, et un état suivant Q. Elle retourne l'étiquette de sortie de la transition constituée d'une suite finie de symboles de l'alphabet résultant de l'union de l'alphabet de sortie et d'entrée $\{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$.

Nous débutons l'analyse de la séquence d'entrée $T = \{Roméo, aperçoit, Juliette\}$ en associant à chaque symbole de T, des éléments de l'ensemble $E = \{\langle\text{TOKEN}\rangle\ ,\ \langle\text{WORD}\rangle\}$ selon la relation d'équivalence $\Sigma \longrightarrow \{\langle\text{TOKEN}\rangle, \langle\text{WORD}\rangle\}$. Nous obtenons alors que $E(T) = (\{\langle\text{TOKEN}\rangle\ ,\ \langle\text{WORD}\rangle\}, \{\langle\text{TOKEN}\rangle\ ,\ \langle\text{WORD}\rangle\}, \{\langle\text{TOKEN}\rangle\ ,\ \langle\text{WORD}\rangle\})$.

À t_0 le registre global i est égal à 0 et la ELG se trouve à l'état initial q_0. Nous lisons le premier symbole (*Roméo*) et consultons la fonction de transition δ. Le seul état de destination est définie par $\delta(q_0, \varepsilon) = q_1$, afin de vérifier s'il est possible de réaliser la transition vers q_1, trois actions sont effectuées :

1. Consulter la fonction de sortie étendue ψ. Pour $\delta(q_0, \varepsilon) = q_1$ elle est définie comme $\psi(q_0, \varepsilon, q_1) =$ `inc()`,

2. Évaluer la sortie étendue `inc()`. Puisque `inc()` $\in \Phi$ est une fonction étendue définie comme $\varepsilon \overset{\texttt{inc()}}{\underset{i=i+1}{\Longmapsto}} \varepsilon$, $\mathcal{M}(\texttt{inc()})$ incrémente le registre global i de 1 et retourne ε,

3. Vérifier si la valeur retournée est différente du symbole d'échec $\mathfrak{X}$. Étant donné que $\varepsilon \neq \mathfrak{X}$, la fonction étendue est satisfaite et par conséquent la sortie étendue l'est aussi.

Ensuite, avant de réaliser la transition vers q_1, sans consommer de symbole, nous consultons la fonction de sortie $\theta(q_0, \varepsilon, \varepsilon, q_1)$ qui renvoie ε, valeur utilisée comme étiquette de sortie de la transition.

À présent nous sommes dans l'état q_1, nous consultons à nouveau la fonction de transition δ. Deux transitions sont possibles $\delta(q_1, \textit{Roméo}) = q_2$ et $\delta(q_1, \textit{Roméo}) = q_3$. Nous analysons d'abord $\delta(q_1, \textit{Roméo}) = q_2$ en appliquant les actions 1, 2 et 3 citées auparavant :

1. Consulter la fonction de sortie étendue ψ. Pour $\psi(q_1, \textit{Roméo}, q_2)$, il n'existe pas de sortie étendue,

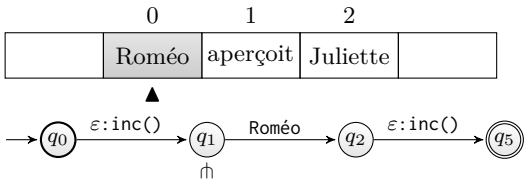
2. Évaluer la sortie étendue ε. La fonction étendue ε est définie comme $\varepsilon \overset{\varepsilon}{\Longmapsto} \varepsilon$, $\mathcal{M}(\varepsilon)$ retourne alors ε,

3. Vérifier si la valeur retournée est différente du symbole d'échec $\mathfrak{X}$. Étant donné que $\varepsilon \neq \mathfrak{X}$, la fonction étendue est satisfaite et par conséquent la sortie étendue l'est aussi.

Comme pour la transition $q_0 \xrightarrow{\varepsilon:\texttt{inc()}} q_1$, avant de réaliser la transition vers q_2, en consommant le symbole *Roméo*, nous consultons la fonction de sortie $\theta(q_1, \textit{Roméo}, \varepsilon, q_2)$ qui renvoie ε, valeur utilisée comme étiquette de sortie de la transition.

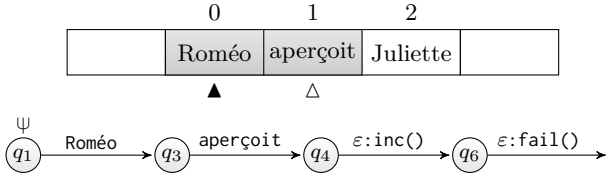
Un processus semblable se répète pour $q_2 \xrightarrow{\varepsilon:\texttt{inc()}} q_5$, qui incrémente le registre global i de 1 avant de passer en q_5. Enfin, étant donné que $q_5 \in F$, la séquence *Roméo* est acceptée et la sortie ε est produite. Le parcours réalisé jusqu'à présent est résumé dans le schéma t_0 suivant :

$\mathbf{t_0}$:	q_0	$\delta(q_0, \varepsilon) = q_1$	$\psi(q_0, \varepsilon, q_1) = \texttt{inc()}$	$\mathcal{M}(\texttt{inc()}) = \varepsilon$	$\theta(q_0, \varepsilon, \varepsilon, q_1) = \varepsilon$	$i = 1$
	q_1	$\delta(q_1, \textit{Roméo}) = q_2$	$\psi(q_1, \textit{Roméo}, q_2) = \varepsilon$	$\mathcal{M}(\varepsilon) = \varepsilon$	$\theta(q_1, \textit{Roméo}, \varepsilon, q_2) = \varepsilon$	
	q_2	$\delta(q_2, \varepsilon) = q_5$	$\psi(q_2, \varepsilon, q_5) = \texttt{inc()}$	$\mathcal{M}(\texttt{inc()}) = \varepsilon$	$\theta(q_2, \varepsilon, \varepsilon, q_5) = \varepsilon$	$i = 2$
	$q_5 \in F$					



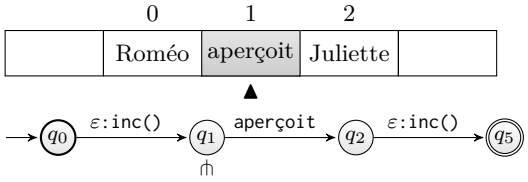
Une fois le parcours de t_0 achevé, en t_1, il est nécessaire de continuer l'analyse de la fonction de transition $\delta(q_1, \textit{Roméo}) = q_3$ que nous n'avons pas encore traitée. Nous procédons de la même manière pour les états $q3, q4, q6$ de telle sorte que lorsque nous arrivons à $q_6 \xrightarrow{\varepsilon:\texttt{fail()}} q_5$, $\mathcal{M}(\texttt{fail()})$ [1] incrémente le registre global i de 1 et retourne $\mathfrak{X}$. Étant donné que la valeur retournée est $\mathfrak{X}$, la fonction étendue n'est pas satisfaite, par conséquent la sortie étendue ne l'est pas non plus et la transition vers q_5 ne se réalise pas. La séquence *Roméo aperçoit* est alors rejetée. Le parcours réalisé est résumé dans le schéma t_1 suivant :

$\mathbf{t_1}$:						
	q_1	$\delta(q_1, \textit{Roméo}) = q_3$	$\psi(q_1, \textit{Roméo}, q_3) = \varepsilon$	$\mathcal{M}(\varepsilon) = \varepsilon$	$\theta(q_1, \textit{Roméo}, \varepsilon, q_3) = \varepsilon$	
	q_3	$\delta(q_3, \textit{aperçoit}) = q_4$	$\psi(q_3, \textit{aperçoit}, q_4) = \varepsilon$	$\mathcal{M}(\varepsilon) = \varepsilon$	$\theta(q_3, \textit{aperçoit}, \varepsilon, q_4) = \varepsilon$	
	q_4	$\delta(q_4, \varepsilon) = q_6$	$\psi(q_4, \varepsilon, q_6) = \texttt{inc()}$	$\mathcal{M}(\texttt{inc()}) = \varepsilon$	$\theta(q_4, \varepsilon, \varepsilon, q_6) = \varepsilon$	$i = 3$
	q_6	$\delta(q_6, \varepsilon) = q_5$	$\psi(q_6, \varepsilon, q_5) = \texttt{fail()}$	$\mathcal{M}(\texttt{fail()}) = \mathfrak{X}$		$i = 4$



À la fin des parcours t_0 et t_1, nous avons une séquence reconnue (*Roméo*), le registre global i est égal à 4 et il ne reste plus de transitions à examiner. Vu que l'analyse que nous effectuons se réalise avec le principe de fenêtre glissante, l'analyse reprend à partir du symbole *aperçoit*. Le parcours réalisé est résumé dans le schéma t_2 suivant :

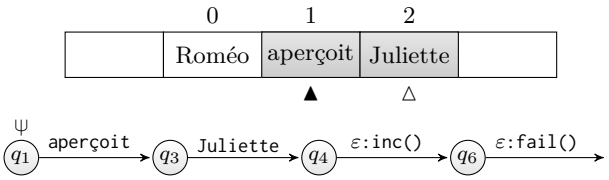
$\mathbf{t_2}$:						
	q_0	$\delta(q_0, \varepsilon) = q_1$	$\psi(q_0, \varepsilon, q_1) = \texttt{inc()}$	$\mathcal{M}(\texttt{inc()}) = \varepsilon$	$\theta(q_0, \varepsilon, \varepsilon, q_1) = \varepsilon$	$i = 5$
	q_1	$\delta(q_1, \textit{aperçoit}) = q_2$	$\psi(q_1, \textit{aperçoit}, q_2) = \varepsilon$	$\mathcal{M}(\varepsilon) = \varepsilon$	$\theta(q_1, \textit{aperçoit}, \varepsilon, q_2) = \varepsilon$	
	q_2	$\delta(q_2, \varepsilon) = q_5$	$\psi(q_2, \varepsilon, q_5) = \texttt{inc()}$	$\mathcal{M}(\texttt{inc()}) = \varepsilon$	$\theta(q_2, \varepsilon, \varepsilon, q_5) = \varepsilon$	$i = 6$
	$q_5 \in F$					



Une fois le parcours t_2 achevé, il est nécessaire de continuer une analyse qui aboutit au rejet de la séquence *aperçoit Juliette*. Ce parcours de rejet est résumé dans le schéma t_3 suivant :

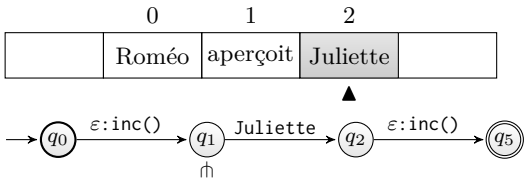
$\mathbf{t_3}$:						
	q_1	$\delta(q_1, \textit{aperçoit}) = q_3$	$\psi(q_1, \textit{aperçoit}, q_3) = \varepsilon$	$\mathcal{M}(\varepsilon) = \varepsilon$	$\theta(q_1, \textit{aperçoit}, \varepsilon, q_3) = \varepsilon$	
	q_3	$\delta(q_3, \textit{Juliette}) = q_4$	$\psi(q_3, \textit{Juliette}, q_4) = \varepsilon$	$\mathcal{M}(\varepsilon) = \varepsilon$	$\theta(q_3, \textit{Juliette}, \varepsilon, q_4) = \varepsilon$	
	q_4	$\delta(q_4, \varepsilon) = q_6$	$\psi(q_4, \varepsilon, q_6) = \texttt{inc()}$	$\mathcal{M}(\texttt{inc()}) = \varepsilon$	$\theta(q_4, \varepsilon, \varepsilon, q_6) = \varepsilon$	$i = 7$
	q_6	$\delta(q_6, \varepsilon) = q_5$	$\psi(q_6, \varepsilon, q_5) = \texttt{fail()}$	$\mathcal{M}(\texttt{fail()}) = \mathfrak{X}$		$i = 8$

1. Rappelons que nous avons défini la fonction étendue $\texttt{fail()} \in \Phi$ comme $\varepsilon \overset{\texttt{fail()}}{\underset{i=i+1}{\longmapsto}} \mathfrak{X}$.



À la fin des parcours t_2 et t_3, nous avons une séquence supplémentaire reconnue (*aperçoit*), le registre global i est égal à 8 et il ne reste plus de transitions à examiner. Avec une fenêtre glissante, le processus reprend à partir du symbole *Juliette*. Le parcours réalisé est résumé dans le schéma t_4 suivant :

$\mathbf{t_4}$:	q_0	$\delta(q_0,\varepsilon)=q_1$	$\psi(q_0,\varepsilon,q_1)=\texttt{inc()}$	$\mathcal{M}(\texttt{inc()})=\varepsilon$	$\theta(q_0,\varepsilon,\varepsilon,q_1)=\varepsilon$	$i=9$
	q_1	$\delta(q_1,\mathit{Juliette})=q_2$	$\psi(q_1,\mathit{Juliette},q_2)=\varepsilon$	$\mathcal{M}(\varepsilon)=\varepsilon$	$\theta(q_1,\mathit{Juliette},\varepsilon,q_2)=\varepsilon$	
	q_2	$\delta(q_2,\varepsilon)=q_5$	$\psi(q_2,\varepsilon,q_5)=\texttt{inc()}$	$\mathcal{M}(\texttt{inc()})=\varepsilon$	$\theta(q_2,\varepsilon,\varepsilon,q_5)=\varepsilon$	$i=10$
	$q_5\in F$					



À la fin du parcours t_4, nous avons une séquence supplémentaire reconnue (*Juliette*), le registre global i est égal à 10 et il ne reste plus de transitions à examiner. Étant donnée qu'il n'existe plus de symboles pour faire glisser la fenêtre d'analyse, le processus est terminé.

Comme résultat de l'analyse, nous avons les séquences reconnues $\{\mathit{Roméo}\}$, $\{\mathit{aperçoit}\}$ et $\{\mathit{Juliette}\}$ chacune associée à une sortie vide ε, ainsi que le registre global i affecté à 10. Il est simple de constater que cette analyse reconnaît les bonnes séquences, cependant, l'évaluation des fonctions étendues ne respecte pas le principe d'atomicité d'évaluation.

Premièrement, nous avons défini que l'évaluation d'une fonction $\varphi \in \Phi$ doit se réaliser de façon « *tout ou rien* », lors que la fonction n'est pas satisfaite, soit quand $\mathcal{M}(\varphi)=\mathfrak{X}$, un registre comme i ne devrait pas être modifié. Cependant dans t_1 et t_3, $q_6 \xrightarrow{\varepsilon:\texttt{fail()}} q_5$, la fonction étendue $\varepsilon \xmapsto[i=i+1]{\texttt{fail()}} \mathfrak{X}$ incrémente i et retourne $\mathfrak{X}$.

Deuxièmement, nous avons dit que ce principe s'étend à l'évaluation d'un série de fonctions étendues $\varphi_1, \varphi_2, \cdots, \varphi_n$ qui sont attachées aux transitions d'un chemin menant d'un état p à un état r. Comme conséquence, dans t_1 et t_3, $q_4 \xrightarrow{\varepsilon:\texttt{inc()}} q_6$, les changements de la fonction étendue $\varepsilon \xmapsto[i=i+1]{\texttt{inc()}} \mathfrak{X}$ qu'incrémente i et retourne ε, ne doivent pas non plus être prises en compte.

En résume, le principe d'atomicité n'est pas respecté, dans t_0 et t_3, l'évaluation des fonctions étendues n'est pas complète :

1. $\exists i \in n, \mathcal{M}(\varphi_i) = \mathfrak{X}$ (il existe des fonctions étendues qui ne sont pas satisfaites)

2. $p \xrightarrow{\exists i \in n, \mathcal{M}(\varphi_i) \neq \mathfrak{X}} r$, $p = q_0$ et $r = q_6 \notin F$ (il existe des fonctions satisfaites qui se trouvent dans un chemin non réussi)

4.8 Machines abstraites et évaluation des fonction étendues

Nous avons défini une machine abstraite $\mathcal{M}$ comme un 5–uplet comprenant 1) des registres, 2) de constantes 3) de variables et 4) un état global, et 5) des opérations. Pour être interprétés, les fonctions étendues doivent être écrites dans un langage utilisant les caractéristiques de $\mathcal{M}$, être stockées et évaluées à la volée. Lorsqu'un chemin de G se trouve sur une transition ti=(p[ti], li[ti], f[ti], lo[ti], n[ti]) ∈ E associée à une fonction étendue, la fonction f[ti], étiquetée comme « @f(i) », est analysée (cf.figure 4.9) afin d'extraire son nom « n » et ses entrées « i », si la fonction « n » existe stockée sur un support d'enregistrement, une pile S (stack) de données est utilisée pour empiler l'état de l'automate (p) ainsi que les entrées (i) et le nom (n) de la fonction (S=p,i,n).

Ensuite, $\mathcal{M}$ fait remonter (i.e. dépile) le nom de la fonction (S=p,i), charge les sources et les compile dynamiquement pour générer les instructions (opérations de $\mathcal{M}$) qui peuvent être exécutées et charger dans les registres les données qui contient les valeurs constantes de la fonction. Les paramètres d'entrée de la fonction 'i' et l'état de l'automate sont ensuite dépilés (S=) et la fonction est exécutée. A la fin de l'exécution la valeur de sortie de la fonction est empilée dans S (S=o) et l'analyse de la transition se poursuit. Si la valeur de « o » est une sous-étiquette (l[ft]) différente du symbole d'échec ($\mathfrak{X}$) et égale à une valeur booléenne « vrai », la transition est satisfaite et l'état suivant sera alors ti+1, si la valeur de sortie est le symbole symbole d'échec ou égale une valeur booléenne « faux », la transition n'est pas satisfaite et par conséquent il n'existe pas de chemin accepteur vers ti+1, et l'exploration recommence (*backtracking*). Dans le cas où la transition est satisfaite, l'étiquette de la transition t (stockée sur une pile T) sera égale à la concaténation de l[ft] et lo[t].

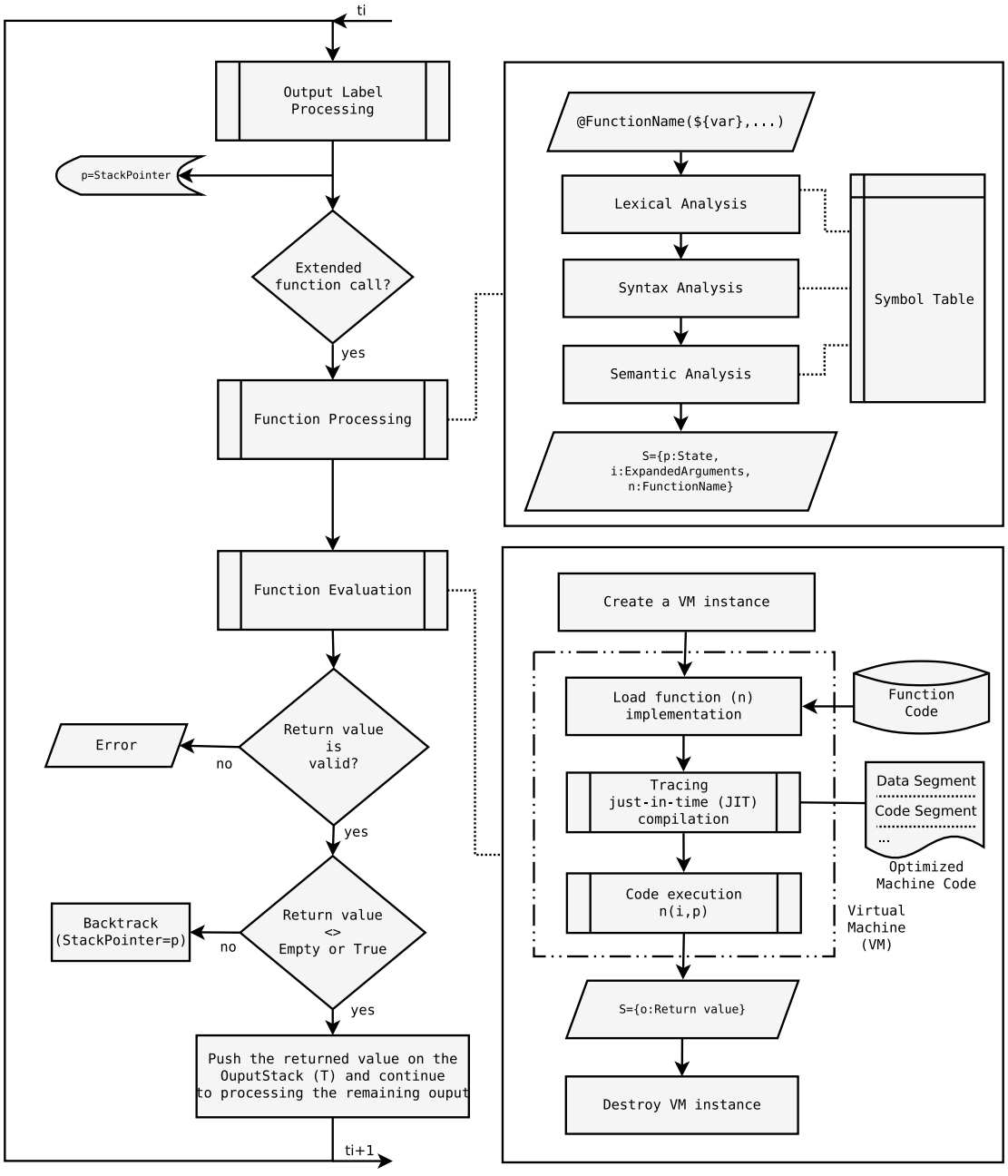


Figure 4.9 – Analyse d'une transition étiquetée par une fonction étendue par une machine abstraite