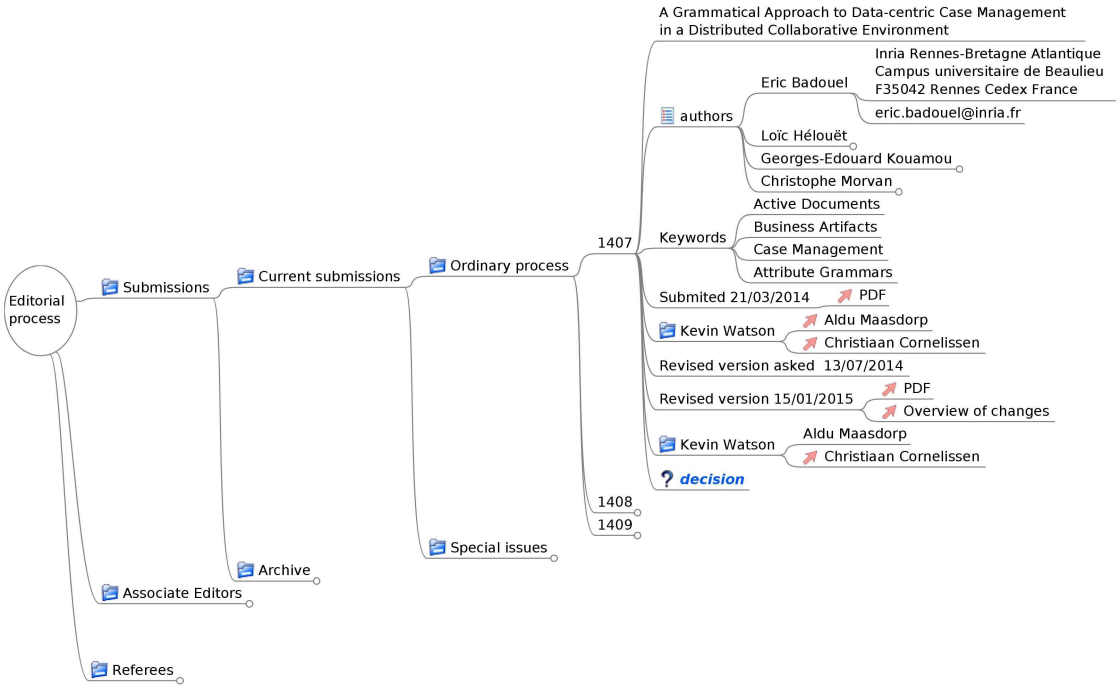
Une note sur les GAG

Eric Badouel

23 mars 2015

1 Espace de travail d'un utilisateur

On représente l'espace de travail d'un utilisateur par une carte heuristique (un arbre) qui donne une vue structurée des tâches qui lui incombent ainsi que des informations qui lui sont utiles pour l'exécution de ces tâches. Par exemple la carte ci-dessous représente l'espace de travail de l'éditeur en chef d'une revue académique.



A COMPLETER

2 Grammaires attribuées gardées

2.1 Un peu de terminologie

Chaque tâche possède un identifiant (appelée «sorte»). Les sortes sont les symboles (non-terminaux) de la grammaire, elles ont des attributs (hérités ou synthétisés) identifiés chacun par un sélecteur. Les valeurs des attributs sont formées à l'aide d'un ensemble de constructeurs et de variables. Chaque argument d'un constructeur est également identifié à l'aide d'un sélecteur. Par exemple

$$\text{Cons}\{\underline{\text{head}} = 3, \underline{\text{tail}} = \text{Cons}\{\underline{\text{head}} = 5, \underline{\text{tail}} = x\}\}$$

représente une liste dont les premières valeurs sont 3 et 5 et le reste de la liste est donnée par la variable x. Cons, 3 et 5 sont les constructeurs de cette expression et x en est la variable, head et tail sont les sélecteurs permettant d'accéder respectivement à l'élément en tête de liste et à la liste résiduelle. On admet des valeurs dont tous les champs ne sont pas renseignés, par exemple

$$\text{Cons}\{\underline{\text{tail}} = \text{Cons}\{\underline{\text{head}} = 5\}\}$$

représente une liste dont le deuxième élément est 5.

On définit deux types d'expressions : les expressions à gauche, ou motifs, et les expressions à droite, ou valeurs, données respectivement comme suit

```
motif   ::=  variable | constructeur {contraction*}
valeur  ::=  variable | constructeur {affectation*}
```

dans lesquelles les contractions et affectations sont définies par

```
contraction  ::=   motif = sélecteur
affectation  ::=   sélecteur = valeur
```

La contraction «motif = sélecteur» revient à décomposer la valeur de l'attribut associé au sélecteur selon le motif donné par l'expression à gauche. En cas de succès cela procure une affectation de valeurs aux variables apparaissant dans le motif. Par exemple la contraction

$$\text{Cons}\{x = \underline{\text{head}}, \text{Cons}\{5 = \underline{\text{head}}, xs = \underline{\text{tail}}\} = \underline{\text{tail}}\} = \underline{\text{list}}$$

vérifie que la valeur associée au sélecteur list (par exemple un attribut hérité d'une tâche, ou un composant particulier d'une donnée) est une liste dont le second élément est un 5, si c'est le cas la variable x reçoit la valeur en tête de cette liste et la variable xs la sous-liste constituée des éléments à partir de la troisième position. Les valeurs des variables de l'expression à gauche sont donc obtenues de façon conditionnelle par filtrage (pattern matching) de la valeur associée au sélecteur.

Souscription. Une contraction dont l'expression à gauche est réduite à une variable, c'est-à-dire de la forme «variable = sélecteur» s'interprète comme le fait que la variable en question souscrit à la valeur associé au sélecteur (mais on ne filtre pas cette valeur).

L'affectation «sélecteur = valeur» revient a donner à l'élément associé au sélecteur la valeur donnée par l'expression à droite. Par exemple l'affectation

$$\underline{\text{list}} = \text{Cons}\{\underline{\text{head}} = x, \underline{\text{tail}} = \text{Cons}\{\underline{\text{head}} = 5, \underline{\text{tail}} = xs\}\}$$

affecte à l'élément désigné par le sélecteur list la liste dont le premier élément est associé à la variable x, le second est un 5 et la liste des éléments suivants est donnée par la variable xs.

Tout noeud ouvert est associé à une tâche qui se présente comme suit

```
tâche  ::=  sorte {(affectation + souscription)*}
```

- Les sortes permettent de catégoriser les tâches, c'est-à-dire de préciser leur nature. A chaque sorte est associée un ensemble d'attributs hérités et synthétisés chacun identifié par un sélecteur.
- Les affectations «sélecteur = rexp» associent des valeurs aux (ou à certains des) attributs hérités de la sorte, donc les sélecteurs en partie gauche d'affectation doivent correspondre à des attributs hérités.
- Les souscriptions «variable=sélecteur» permettent au contraire de lier la valeur qui sera produite en un attribut synthétisé à une variable, les sélecteurs en partie droite d'une souscription doivent donc correspondre à des attributs synthétisés.
- Un sélecteur ne peut apparaitre qu'une fois dans la tâche dans le corps d'une tâche. Une variable ne peut être *définie* qu'une fois, c'est-à-dire ne peut apparaître que dans une souscription. Elle peut être *utilisée*, c'est-à-dire apparaître dans une affectation, un nombre arbitraire de fois. La *source* d'une variable x est le noeud (ouvert) de la carte où cette variable est définie et une *cible* de x est un noeud (ouvert) de la carte où cette variable est utilisée. On maintient à jour un tableau indiquant la source et les cibles de chaque variable apparaissant dans la carte à un moment donné.

Par exemple supposons que la sorte **wait** corresponde à l'attente d'une liste de messages (attribut hérité listVal) et à la production d'un accusé de réception pour chaque message reçu (attribut synthétisé listAcks va collecter la liste de ces accusés de reception). Une instance particulière de cette sorte, c'est-à-dire une tâche de sorte **wait** est par exemple

$$\textbf{wait}\{\underline{\text{listVal}} = \text{Cons}\{\underline{\text{head}} = 2, \underline{\text{tail}} = \text{vals}\}, \text{acks} = \underline{\text{listAcks}}\}$$

dans laquelle une première valeur reçue par l'attribut hérité listVal est 2 et la liste des valeurs suivantes sera fournie par la variable *vals* tandis que la variable *acks* souscrit à la liste des accusés de réception qui seront produits dans l'attribut synthétisé listAcks.

2.2 Profil d'une règle et pattern matching

Une **règle** décrit une façon particulière de réaliser une tâche donnée. Sa partie gauche est son **profil** qui est de la forme suivante

```
profil   ::=  sorte {(contraction + affectation)*}
```

- Un sélecteur ne peut apparaitre qu'une fois dans le profil d'une règle.

— Les motifs des contractions servent à conditionner l'application de la règle pour résoudre une tâche (de la même sorte) en testant la forme de la valeur de certains attributs hérités de la tâche. Les sélecteurs en partie droite d'une contraction doivent par conséquent être des attributs hérités de la sorte.
— Les affectations «sélecteur = rexp» décrivent les résultats qui seront affectés en retour aux attributs synthétisés de la tâche. Les sélecteurs en partie gauche d'une affectation doivent correspondre à des attributs synthétisés.
— Une variable ne peut être *définie* qu'une fois dans une règle, et en particulier elle ne pourra apparaître qu'au plus une fois dans une contraction du profil.

Pour que la procédure puisse s'appliquer à résoudre la tâche il faut que son profil soit compatible avec la tâche. Il faut pour cela d'une part que la sorte soit la même mais aussi que contractions et affectations se correspondent ce qu'on nous définissons plus bas.

2.2.1 Substitutions

Une *substitution* associe une valeur (une expression à droite) à certaines variables. On assimile une substitution à un ensemble d'équations $\{x_1 = u_1, \ldots, x_n = u_n\}$ dans lequel les variables x_i sont distinctes et n'apparaissent dans aucune des valeurs u_i. Les variables x_i sont *définies* par la substitution et les variables apparaissant dans les expressions à droite sont *utilisées* par la substitution. Une variable ne peut donc pas être simultanément définie et utilisée par une substitution. Une expression E contenant des contractions et des affectations est *bien formée* si une variable est définie au plus une fois dans E, c'est-à-dire apparaît au plus une fois dans une contraction (ou dans une souscription ce qui en est un cas particulier). Une substitution σ est applicable en une expression E si aucune variable n'est définie à la fois dans l'expression et dans la substitution. Le résultat $E\sigma$ de l'application de σ à E est obtenue à partir de cette expression en remplaçant toute occurrence de la variable x par la valeur u.

2.2.2 Pattern matching

Les variables qui apparaissent dans une règle et en particulier dans son profil sont liées à cette règle et donc peuvent être renommées injectivement sans en changer la signification. On supposera toujours que ces variables sont distinctes de celles qui apparaissent dans la tâche et plus généralement dans la configuration courante. Sous cette hypothèse on vérifie que le profil $\mathbf{s}\{arg\}$ est compatible avec la tâche $\mathbf{s}\{arg'\}$ en deux étapes.

1. La première étape consiste à vérifier que toutes les variables définies dans le profil, c'est-à-dire apparaissant dans une contraction, se voient attribuer une valeur correspondante provenant de la tâche. Pour cela :
   (a) On vérifie que pour toute contraction $t = \underline{\text{sel}}$ dans arg on trouve une affectation $\underline{\text{sel}} = u$ dans arg' correspondant au même sélecteur et dans ce cas on forme l'équation $t \overset{?}{\lhd} u$. Si ce n'est pas le cas on arrête sur un échec sinon on obtient un ensemble P_{in} d'équations de la forme $t \overset{?}{\lhd} u$.
   (b) On transforme progressivement l'ensemble P_{in} en choisissant une équation $t \overset{?}{\lhd} u$ dans laquelle t n'est pas réduit à une variable, tant qu'il en reste, et en appliquant, selon la forme de cette équation, la transformation correspondante :

i. $\mathrm{C}\{cs\} \overset{?}{\lhd} \mathrm{C}\{as\}$: si pour toute contraction $t = \underline{\text{sel}}$ dans cs on trouve une affectation $\underline{\text{sel}} = u$ qui lui corresponde dans as alors on remplace l'équation $\mathrm{C}\{cs\} \overset{?}{\lhd} \mathrm{C}\{as\}$ par les équations $t \overset{?}{\lhd} u$ correspondantes sinon on arrête sur un échec.

ii. $\mathrm{C}\{cs\} \overset{?}{\lhd} y$ où y est une variable : on arrête sur un échec.

iii. $\mathrm{C}\{cs\} \overset{?}{\lhd} \mathrm{C}'\{as\}$ avec $\mathrm{C} \neq \mathrm{C}'$: on arrête sur un échec.

(c) En cas de succès on se retrouve avec une équation de la forme $x_i \overset{?}{\lhd} u_i$ pour chacune des variables $x_1, \ldots, x_n$ définies dans le profil et on forme la substitution $\sigma_{in} = \{x_1 = u_1, \ldots, x_n = u_n\}$ (le profil et la tâche n'ont pas de variables en commun et donc les variables x_i n'apparaissent dans aucune des valeurs u_j).

2. La deuxième étape consiste à vérifier que toutes les variables définies dans la tâche, c'est-à-dire apparaissant dans une souscription, se voient attribuer une valeur correspondante provenant du profil. Pour cela :

(a) On vérifie que pour toute souscription $x = \underline{\text{sel}}$ dans arg' on trouve une affectation $\underline{\text{sel}} = u$ dans arg correspondant au même sélecteur et dans ce cas on forme l'équation $x \overset{?}{=} u\sigma_{in}$. Si ce n'est pas le cas on s'arrête sur un échec, sinon on forme l'ensemble $E_{out} = E_? \cup E_=$ dans lequel $E_?$ est donné par l'ensemble des équations ainsi obtenues et $E_= = \emptyset$. On transforme progressivement cet ensemble de la manière suivante jusqu'à ce que l'ensemble $E_?$ soit vide. On choissit une équation $x \overset{?}{=} u$ dans $E_?$. Si x est une variable de u alors on s'arrête sur un échec sinon

i. Si $u = y$ est une variable on remplace $x \overset{?}{=} y$ par l'équation $y = x$ et on remplace toutes les occurrences de y par x dans les autres équations de E_{out}.

ii. Si u est une expression qui ne se réduit pas à une variable, on remplace $x \overset{?}{=} u$ par l'équation $x = u$ et on remplace toutes les occurrences de x par u dans les autres équations de E_{out}.

(b) En cas de succès l'ensemble $E_=$ définit, lorsque toutes les équations de $E_?$ ont été traitées, une substitution σ_{out} attribuant une valeur à chacune des variables apparaissant dans les souscriptions de la tâche.

3. Si des deux étapes aboutissent avec succès on dit que le profil $\mathbf{s}\{arg\}$ est compatible avec la tâche $\mathbf{s}\{arg'\}$ avec la substitution $\sigma = \sigma_{in} \cdot \sigma_{out} \cup \sigma_{out}$.

Des commentaires seront utiles pour justifier la différence de traitement entre $x \overset{?}{\lhd} y$ qui génère l'équation $x = y$ et $x \overset{?}{=} y$ qui génère au contraire $y = x$.

Par exemple les deux interfaces suivantes

$$I_1 = \mathbf{wait}\{\mathrm{Cons}(\mathit{value} = \underline{\text{head}}, \mathit{vs} = \underline{\text{tail}}\} = \underline{\text{listVal}}, \underline{\text{listAck}} = \mathrm{Ack}\{\underline{\text{tail}} = \mathit{acks}\}$$
$$I_2 = \mathbf{wait}\{\mathrm{Nil} = \underline{\text{listVal}}\}$$

correspondent aux deux façons possibles de réaliser une tâche de sorte **wait** selon que la valeur de l'attribut $\underline{\text{listVal}}$ contient une valeur (constructeur Cons) ou que le processus de transmission de valeurs s'est arrêté (constructeur Nil). Par exemple la tâche

$$\mathbf{wait}\{\underline{\text{listVal}} = \mathrm{Cons}\{\underline{\text{head}} = 2, \underline{\text{tail}} = \mathit{vals}\}, \mathit{as} = \underline{\text{listAcks}}\}$$

est compatible avec I_1 et produit la substitution suivante :

$$\{value = 2, \quad vs = vals, \quad as = \text{Ack}\{\underline{\text{tail}} = acks\}\}$$

qui lie les variables locales *value* et *vs* du profil aux valeurs correspondantes extraites des paramètres de la tâches (en positions héritées) et inversement la variable *as* de la tâche qui avait souscrit à la valeur du sélecteur listAcks se voit retourner la valeur correspondante $\text{Ack}\{\underline{\text{tail}} = acks\}$ qui correspond à l'envoi d'un accusé de réception (constructeur Ack) suite à la réception de la valeur 2. Le sélecteur tail du constructeur Ack désigne la liste des accusés de réception à venir ; ce sélecteur est lié à la variable *acks* qui n'est pas définie dans le profil. Nous devons donc préciser dans la règle de quelle façon cette variable est calculée, ce que nous détaillons dans la section suivante.

2.3 Forme des règles

Lorsque le profil d'une règle (sa partie gauche) est compatible avec une tâche encore faut il pour résoudre cette tâche décrire de quelle façon elle calcule les valeurs souscrites par la tâche. Ces valeurs sont données dans la partie affectations du profil, mais le plus souvent ces valeurs font intervenir des variables dont le calcul de la valeur est délégué aux sous tâches qui apparaissent en partie droite de la règle. Par exemple les règleszs de profils respectifs I_1 et I_2 qui permettent de résoudre des tâches de sortes **wait** peuvent être les suivantes :

$$\begin{aligned}&\textbf{wait}\{\text{Cons}\{value = \underline{\text{head}}, vs = \underline{\text{tail}}\} = \underline{\text{listVal}}, \underline{\text{listAck}} = \text{Ack}\{\underline{\text{tail}} = acks\}\}\\ &\qquad \rightarrow \text{ReceiveData}\{\ \underline{\text{process}} = \textbf{process}\{\underline{\text{val}} = value\},\\ &\qquad\qquad\qquad\quad \underline{\text{sequel}} = \textbf{wait}\{\underline{\text{listVal}} = vs, acks = \underline{\text{listAck}}\}\}\end{aligned}$$

$$\textbf{wait}\{\text{Nil} = \underline{\text{listVal}}\} \rightarrow \text{StopWaiting}$$

Dans le premier cas la réception d'une valeur provoque la création de deux nouvelles tâches. La première, $\textbf{process}\{\underline{\text{val}} = value\}$, utilise la donnée transmise pour faire un certain traitement qui dépend de cette valeur et la seconde, $\textbf{wait}\{\underline{\text{listVal}} = vs, acks = \underline{\text{listAck}}\}$ est une continuation de la tâche initiale qui attend la liste des valeurs suivantes (variable *vs*) et se charge de produire la liste des accusés de réception à venir. La valeur retournée (souscrite par la tâche initiale) est donnée par $\text{Ack}\{\underline{\text{tail}} = acks\}$ dans le profil. Elle consiste en un accusé de réception (constructeur Ack) et la transmission (en argument de ce constructeur) de la variable *acks* qui contiendra la liste résiduelle des accusés de réception.

La seconde règle a une partie droite vide (la production en partie droite est une contante) ce qui traduit le fait que la tâche de sorte **wait** se termine s'il n'y a plus de valeurs transmises (constructeur

Nil). On peut compléter cet exemple en indiquant l'ensemble des règles de la grammaire :

$$\begin{aligned}
&\mathit{start}: \mathbf{root} \to \mathrm{Root}\{\ \underline{\mathrm{send}} = \mathbf{send}\{\underline{\mathrm{listAck}} = \mathit{acks}, \mathit{vals} = \underline{\mathrm{listVal}}\}\\
&\qquad\qquad\qquad\qquad\qquad \underline{\mathrm{wait}} = \mathbf{wait}\{\underline{\mathrm{listVal}} = \mathit{vals}, \mathit{acks} = \underline{\mathrm{listAck}}\}\}\\
&\\
&\mathit{send}\{\mathit{val} = \underline{\mathrm{arg}}\}: \mathbf{send}\{\mathit{acks} = \underline{\mathrm{listAck}}, \underline{\mathrm{listVal}} = \mathrm{Cons}\{\underline{\mathrm{head}} = \mathit{val}, \underline{\mathrm{tail}} = \mathit{vals}\}\}\\
&\qquad\to\quad \mathrm{Send}\{\underline{\mathrm{arg}} = \mathit{val}, \underline{\mathrm{waitAck}} = \mathbf{waitAck}\{\underline{\mathrm{listAck}} = \mathit{acks}, \mathit{vals} = \underline{\mathrm{listVal}}\}\}\\
&\\
&\mathit{stopSending}: \mathbf{send}\{\underline{\mathrm{listVal}} = \mathrm{Nil}\} \to \mathrm{StopSending}\\
&\\
&\mathit{receiveAck}: \mathbf{waitAck}\{\mathrm{Ack}\{\mathit{acks} = \underline{\mathrm{tail}}\} = \underline{\mathrm{listAck}}, \underline{\mathrm{listVal}} = \mathit{vals}\}\\
&\qquad\qquad\to \mathrm{ReceiveAck}\{\underline{\mathrm{send}} = \mathbf{send}\{\underline{\mathrm{listAck}} = \mathit{acks}, \mathit{vals} = \underline{\mathrm{listVal}}\}\}\\
&\\
&\mathit{receiveData}: \mathbf{wait}\{\mathrm{Cons}\{\mathit{val} = \underline{\mathrm{head}}, \mathit{vs} = \underline{\mathrm{tail}}\} = \underline{\mathrm{listVal}}, \underline{\mathrm{listAck}} = \mathrm{Ack}\{\underline{\mathrm{tail}} = \mathit{acks}\}\}\\
&\qquad\to \mathrm{ReceiveData}\{\ \underline{\mathrm{value}} = \mathit{val},\\
&\qquad\qquad\qquad\qquad\qquad \underline{\mathrm{process}} = \mathbf{process}\{\underline{\mathrm{val}} = \mathit{val}\},\\
&\qquad\qquad\qquad\qquad\qquad \underline{\mathrm{sequel}} = \mathbf{wait}\{\underline{\mathrm{listVal}} = \mathit{vs}, \mathit{acks} = \underline{\mathrm{listAck}}\}\}\\
&\\
&\mathit{stopWaiting}: \mathbf{wait}\{\mathrm{Nil} = \underline{\mathrm{listVal}}\} \to \mathrm{StopWaiting}
\end{aligned}$$

La tâche **send** se charge ainsi de transmettre une suite de valeurs à la tâche **wait** en attendant un accusé de réception de chaque envoi avant de transmettre la valeur suivante, elle peut aussi à tout moment décider d'arrêter le processus d'envoi de messages. On remarque que la règle $\mathit{send}\{\mathit{val} = \underline{\mathrm{arg}}\}$ possède un argument correspondant à la valeur à transmettre. Cet argument est une souscription indiquant que la variable *val* souscrit à la valeur à transmettre. Cette valeur est mémorisée dans la carte ($\mathrm{Send}\{\underline{\mathrm{arg}} = \mathit{val}, \ldots\}$) et est transmise dans la valeur retournée à l'attribut synthétisé ($\underline{\mathrm{listVal}} = \mathrm{Cons}\{\underline{\mathrm{head}} = \mathit{val}, \underline{\mathrm{tail}} = \mathit{vals}\}$). Les arguments d'une règle servent à interfacer une grammaire avec son contexte, en particulier si on veut coupler deux grammaire. un argument peut-être en entrée comme ici (i.e., une souscription) ou en sortie (une affectation). Par défaut l'utilisateur joue le rôle du contexte, et dans ce cas les valeurs sont transmises ou reçues par l'utilisateur. Dans cet exemple, la valeur du sélecteur $\underline{\mathrm{arg}}$ doit être fournie par l'utilisateur lorsqu'il fait le choix d'envoyer une valeur. Ainsi un sélecteur (comme ici $\underline{\mathrm{arg}}$) qui apparait dans une souscription en paramètre d'une règle s'interprète comme un champ à renseigner par l'utilisateur lorsqu'il sélectionne la règle send.

Les règles comportent trois ingrédients :

1. Une production (de la grammaire sous-jacente du GAG) $s \to s_1 \cdots s_n$ indiquant que la tâche s peut être décomposée en les sous tâches $s_1, \ldots, s_n$ en appliquant cette règles.
2. Des gardes conditionnant le déclanchement de la règles donnée par les contractions de s et des règles sémantiques liant les entrées et sorties entre la tâche et ses sous-tâches données par les contractions et affectations de ces différentes tâches.
3. Un contexte en partie droite, dans lequel s'insère les sous tâches $s_1, \ldots, s_n$, qui décrit la façon dont la carte évolue lorsqu'on applique la règle.

La forme des règles suit la syntaxe suivante :

$$
\begin{array}{ll}
\langle \text{regle} \rangle & ::= \text{regle}\{\langle \text{args} \rangle\} : \ \langle \text{profil} \rangle \to \langle \text{rhs} \rangle \\
\langle \text{args} \rangle & ::= (\langle \text{souscription} \rangle | \langle \text{affectation} \rangle)^* \\
\langle \text{profil} \rangle & ::= \textbf{sorte}\{(\langle \text{contraction} \rangle | \langle \text{souscription} \rangle)^*\} \\
\langle \text{rhs} \rangle & ::= (\underline{\text{selecteur}} = \langle \text{rexp} \rangle \mid \underline{\text{selecteur}} = \langle \text{tache} \rangle \mid \underline{\text{selecteur}} = \text{Constructeur}\{\text{rhs}\})^* \\
\langle \text{tache} \rangle & ::= \textbf{sorte}\{(\langle \text{affectation} \rangle | \langle \text{souscription} \rangle)^*\}
\end{array}
$$

Avec les conditions suivantes :

1. Un même sélecteur ne peut pas apparaître plusieurs fois pour la même occurrence d'une production, d'une sorte ou d'un constructeur.
2. Le sélecteur d'une sorte doit correspondre à un attribut hérité s'il apparaît dans une contraction du profil ou une affectation dans le corps de la règle, et à un attribut synthétisé s'il apparaît dans une affectation du profil ou une souscription du corps de la règle.
3. Une variable doit apparaître exactement une fois en position définie, c'est-à-dire dans une contraction ou une souscription et doit apparaître au moins une fois en position utilisée. Les arguments de la règle indiquent les variables qui sont définies à l'extérieur et celles dont la valeur est susceptible d'être utilisées à l'extérieur.

2.4 Notation abrégée

On peut se dispenser de mentionner les sélecteurs de façon explicite si le nombre et la nature des arguments (que ce soit pour les sortes, les constructeurs ou les productions) sont fixés et peuvent donc être identifiés par leur position dans la liste. Pour les attributs d'une sorte il faudra néanmoins regrouper ensemble les attributs hérités (indiqués entre paranthèses) et les attributs synthétisés (indiqués entre crochets) pour éviter la confusion entre contraction et affectation. Par exemple la grammaire suivante qui décrit comment extraire la liste des feuilles d'un arbre binaire lues de la gauche vers la droite

$$
\begin{array}{lll}
\mathit{init}\{\underline{\text{out}} = x\} & : & \textbf{root} \to \text{Root}\{\underline{\text{tree}} = \textbf{bin}\{\underline{\text{inh}} = \text{Nil}, x = \underline{\text{syn}}\}\} \\
\mathit{fork} & : & \textbf{bin}\{x = \underline{\text{inh}}, \underline{\text{syn}} = y\} \to \text{Fork} \left\{ \begin{array}{l} \underline{\text{left}} = \textbf{bin}\{\underline{\text{inh}} = z, y = \underline{\text{syn}}\}, \\ \underline{\text{right}} = \textbf{bin}\{\underline{\text{inh}} = x, z = \underline{\text{syn}}\} \end{array} \right\} \\
\mathit{leaf}\{a = \underline{\text{arg}}\} & : & \textbf{bin}\{x = \underline{\text{inh}}, \underline{\text{syn}} = \text{Cons}\{\underline{\text{head}} = a, \underline{\text{tail}} = x\}\} \to \text{Leaf}\{\underline{\text{arg}} = a\}
\end{array}
$$

pourra s'écrire de façon plus synthétique de la manière suivante

$$
\begin{array}{lll}
\mathit{root}\langle x \rangle & : & \textbf{root}() \to \text{Root}(\textbf{bin}(\text{Nil})\langle x \rangle) \\
\mathit{fork} & : & \textbf{bin}(x)\langle y \rangle \to \text{Fork}(\textbf{bin}(z)\langle y \rangle, \ \textbf{bin}(x)\langle z \rangle) \\
\mathit{leaf}(a) & : & \textbf{bin}(x)\langle \text{Cons}(a, x) \rangle \to \text{Leaf}(a)
\end{array}
$$

De même la grammaire donnée dans la section 2.3 peut s'écrire plus simplement

$$\mathit{start} : \mathbf{root} \;\to\; \mathrm{Root}(\mathit{send}(\mathit{acks})\langle \mathit{vals}\rangle, \mathbf{wait}(\mathit{vals})\langle \mathit{acks}\rangle)$$

$$\mathit{send}(\mathit{val}) : \mathbf{send}(\mathit{acks})\langle \mathrm{Cons}(\mathit{val}, \mathit{vals})\rangle \;\to\; \mathrm{Send}(\mathit{val}, \mathbf{waitAck}(\mathit{acks})\langle \mathit{vals}\rangle)$$

$$\mathit{stopSending} : \mathbf{send}(\mathrm{Nil}) \;\to\; \mathrm{StopSending}$$

$$\mathit{receiveAck} : \mathbf{waitAck}(\mathrm{Ack}(\mathit{acks}))\langle \mathit{vals}\rangle \;\to\; \mathrm{ReceiveAck}(\mathbf{send}(\mathit{acks})\langle \mathit{vals}\rangle)$$

$$\mathit{receiveData} : \mathbf{wait}(\mathrm{Cons}(\mathit{val}, \mathit{vs}))\langle \mathrm{Ack}(\mathit{acks})\rangle \;\to\; \mathit{ReceiveData}(\mathit{val}, \mathbf{process}(\mathit{val}), \mathbf{wait}(\mathit{vs})\langle \mathit{acks}\rangle)$$

$$\mathit{stopWaiting} : \mathbf{wait}(\mathrm{Nil}) \;\to\; \mathrm{StopWaiting}$$

2.5 Application d'une règle

Je ne détaille pas trop cette partie parce qu'il n'y a pas trop de différences avec ce qui est décrit dans notre rapport précédent. En gros pour appliquer une règle on vérifie que son profil (partie gauche) correspond avec la tâche associée au noeud (au sens du paragraphe 2.2.2) ce qui nous procure une substitution. L'utilisateur doit de son côté produire un match pour les affectations et les souscriptions en argument de la règle, c'est-à-dire qu'il introduit des données en paramètre de la règle et indique où doivent être redirigés les résultats produits. Cela complète la substitution. On remplace alors la tâche par la partie droite de la règle et on applique la substitution à l'ensemble de la configuration.

Pour éviter d'avoir à parcourir toute la configuration (carte où s'applique la règle ainsi que les cartes distantes) on maintient pour chaque carte un tableau donnant la source et les cibles de toutes les variables intervenant dans la carte. Ce tableau peut aussi être distribué sur chacun des noeuds en y indiquant les entrées du tableau correspondant aux variables définies en ce noeud de tel sorte que chaque noeud connaisse la liste des noeuds utilisant la valeur des variables qui y sont définies.

PRECISER LA MISE A JOUR DYNAMIQUE DE CES TABLEAUX

Une alternative est de propager les substitutions générées par une récriture le long de l'arbre : lorsqu'un noeud reçoit une telle substitution d'un de ses noeuds voisins (son père ou un de ses fils) il propage cette information vers ses autres noeuds voisins après avoir filtré cette substitution pour ne transmettre à chacun d'eux que l'information utile.

IDENTIFIER L'INFORMATION A CONSERVER EN CHAQUE NOEUD LORS SA CREATION POUR ASSURER CETTE REDIRECTION DE FACON OPTIMALE

2.6 Un exemple : calcul du feuillage d'un arbre binaire

Nous rappelons ci-dessous la grammaire qui décrit comment extraire la liste des feuilles d'un arbre binaire lues de la gauche vers la droite

$$\begin{array}{lcl}
\mathit{init}\{\underline{\mathrm{out}} = x\} & : & \mathbf{root} \to \mathrm{Root}\{\underline{\mathrm{tree}} = \mathbf{bin}\{\underline{\mathrm{inh}} = \mathrm{Nil}, x = \underline{\mathrm{syn}}\}\} \\
\mathit{fork} & : & \mathbf{bin}\{x = \underline{\mathrm{inh}}, \underline{\mathrm{syn}} = y\} \to \mathrm{Fork}\left\{\begin{array}{l} \underline{\mathrm{left}} = \mathbf{bin}\{\underline{\mathrm{inh}} = z, y = \underline{\mathrm{syn}}\}, \\ \underline{\mathrm{right}} = \mathbf{bin}\{\underline{\mathrm{inh}} = x, z = \underline{\mathrm{syn}}\} \end{array}\right\} \\
\mathit{leaf}\{a = \underline{\mathrm{arg}}\} & : & \mathbf{bin}\{x = \underline{\mathrm{inh}}, \underline{\mathrm{syn}} = \mathrm{Cons}\{\underline{\mathrm{head}} = a, \underline{\mathrm{tail}} = x\}\} \to \mathrm{Leaf}\{\underline{\mathrm{arg}} = a\}
\end{array}$$

Partons de la configuration suivante

$$\Gamma_0 = \{X = \mathbf{root},\ Y = \mathbf{output}\{\underline{\text{list}} = \mathit{list}\}\}$$

On y trouve deux éléments. X est un noeud ouvert à partir duquel un arbre binaire va pouvoir être développé, et Y est un noeud ouvert qui possède un attribut hérité $\underline{\text{list}}$ dont la valeur représentée par la variable *list* n'est pour l'instant pas définie (elle n'apparait dans aucune souscription). On peut appliquer la règle $\mathit{init}\{\mathit{list} = \underline{\text{out}}\}$ en X ce qui donne la substitution $\{x = \mathit{list}\}$ et nous conduit à la nouvelle configuration :

$$\Gamma_1 = \{X = \text{Root}\{\underline{\text{tree}} = \mathbf{bin}\{\underline{\text{inh}} = \text{Nil}, \mathit{list} = \underline{\text{syn}}\}\};\ Y = \mathbf{output}\{\underline{\text{list}} = \mathit{list}\}\}$$

ce qui a pour effet d'initialiser l'attribut hérité de l'arbre par la liste vide ($\underline{\text{inh}} = \text{Nil}$) et de lier le feuillage de l'arbre (qui doit être synthétisé en l'attribut $\underline{\text{syn}}$ de $X_1 = X \cdot \underline{\text{tree}}$) à l'attribut $\underline{\text{list}}$ de Y : cet attribut souscrit maintenant à la valeur de $\underline{\text{syn}}$ de X_1.

L'étape suivante consiste à appliquer la règle *fork* au noeud X_1 ce qui procure la substitution $\{y = \mathit{list}, x = \text{Nil}\}$ et nous conduit à la configuration

$$\Gamma_2 = \left\{ \begin{array}{l} X = \text{Root}\left\{\underline{\text{tree}} = \text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \mathbf{bin}\{\underline{\text{inh}} = z, \mathit{list} = \underline{\text{syn}}\} \\ \underline{\text{right}} = \mathbf{bin}\{\underline{\text{inh}} = \text{Nil}, z = \underline{\text{syn}}\} \end{array} \right\}\right\} \\ Y = \mathbf{output}\{\underline{\text{list}} = \mathit{list}\} \end{array} \right\}$$

Si on applique la règle $\mathit{leaf}\{\underline{\text{arg}} = \text{c}\}$ au noeud $X_{12} = X_1 \cdot \underline{\text{right}}$ on obtient la substitution $\{a = \text{c}, x = \text{Nil}, z = \text{Cons}(\text{c}, \text{Nil})\}$ et la nouvelle configuration

$$\Gamma_3 = \left\{ \begin{array}{l} X = \text{Root}\left\{\text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \mathbf{bin}\{\underline{\text{inh}} = \text{Cons}(\text{c}, \text{Nil}), \mathit{list} = \underline{\text{syn}}\} \\ \underline{\text{right}} = \text{Leaf}\{\underline{\text{arg}} = \text{c}\} \end{array} \right\}\right\} \\ Y = \mathbf{output}\{\underline{\text{list}} = \mathit{list}\} \end{array} \right\}$$

Si on applique la règle *fork* au noeud $X_{11} = X_1 \cdot \underline{\text{left}}$ on obtient la substitution $\{x = \text{Cons}(\text{c}, \text{Nil}), y = \mathit{list}\}$ et la nouvelle configuration

$$\Gamma_4 = \left\{ \begin{array}{l} X = \text{Root}\left\{\text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \mathbf{bin}\{\underline{\text{inh}} = z, \mathit{list} = \underline{\text{syn}}\} \\ \underline{\text{right}} = \mathbf{bin}\{\underline{\text{inh}} = \text{Cons}(\text{c}, \text{Nil}), z = \underline{\text{syn}}\} \end{array} \right\} \\ \underline{\text{right}} = \text{Leaf}\{\underline{\text{arg}} = \text{c}\} \end{array} \right\}\right\} \\ Y = \mathbf{output}\{\underline{\text{list}} = \mathit{list}\} \end{array} \right\}$$

Si on applique la règle $\mathit{leaf}\{\underline{\text{arg}} = \text{a}\}$ au noeud $X_{111} = X_{11} \cdot \underline{\text{left}}$ on obtient la substitution $\{a = \text{a}, z = x, \mathit{list} = \text{Cons}(\text{a}, z)\}$ et la nouvelle configuration

$$\Gamma_5 = \left\{ \begin{array}{l} X = \text{Root}\left\{\text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \text{Leaf}\{\underline{\text{arg}} = \text{a}\} \\ \underline{\text{right}} = \mathbf{bin}\{\underline{\text{inh}} = \text{Cons}(\text{c}, \text{Nil}), z = \underline{\text{syn}}\} \end{array} \right\} \\ \underline{\text{right}} = \text{Leaf}\{\underline{\text{arg}} = \text{c}\} \end{array} \right\}\right\} \\ Y = \mathbf{output}\{\underline{\text{list}} = \text{Cons}(\text{a}, z)\} \end{array} \right\}$$

1. Pour alléger les notations, on écrit de façon abrégée $\text{Cons}(h, t)$ pour $\text{Cons}\{\underline{\text{head}} = h, \underline{\text{tail}} = t\}$, de même on omet dans la suite le sélecteur $\underline{\text{tree}}$ de la production Root

Si on applique la règle $leaf\{\underline{\text{arg}} = \text{b}\}$ au noeud $X_{112} = X_{11} \cdot \underline{\text{right}}$ on obtient la substitution $\{a = \text{b}, x = \text{Cons}(\text{c}, \text{Nil}), z = \text{Cons}(\text{b}, \text{Cons}(\text{c}, \text{Nil}))\}$ et la configuration

$$\Gamma_6 = \left\{ \begin{array}{l} X = \text{Root}\left\{ \text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \text{Fork}\left\{ \begin{array}{l} \underline{\text{left}} = \text{Leaf}\{\underline{\text{arg}} = \text{a}\} \\ \underline{\text{right}} = \text{Leaf}\{\underline{\text{arg}} = \text{b}\} \end{array} \right\} \\ \underline{\text{right}} = \text{Leaf}\{\underline{\text{arg}} = \text{c}\} \end{array} \right\} \right\} \\ Y = \textbf{output}\{\underline{\text{list}} = \text{Cons}(\text{a}, \text{Cons}(\text{b}, \text{Cons}(\text{c}, \text{Nil})))\} \end{array} \right\}$$

2.7 Macro-règles

Notons $T \xrightarrow[\sigma_{out}:C]{r} T_1, \ldots, T_n$ lorsque la règle r peut s'appliquer à résoudre la tâche T en produisant la substitution σ_{out} et $rhs(r)\sigma_{in} = C[T_1, \ldots, T_n]$. Les contextes sont donnés par

$$\langle \text{Contexte} \rangle ::= (\underline{\text{selecteur}} = \langle \text{rexp} \rangle \mid \underline{\text{selecteur}} = [\,] \mid \underline{\text{selecteur}} = \text{Constructeur}\{\text{Contexte}\})^*$$

On peut composer ces règles de la manière suivante. L'ensemble des macro-règles est inductivement défini par :

1. $[\,]$ est une macro-règle telle que $T \xrightarrow[\emptyset:[\,]]{[\,]} T$ pour toute tâche T.
2. Une règle r est identifiée à la macro-règle $R = r([\,], \ldots, [\,])$ à n arguments où n est le nombre de tâches en sa partie droite.
3. si r est une règle n-aire et $R_1, \ldots, R_n$ sont des macro-règles alors $R = r(R_1, \ldots, R_n)$ est une macro-règle et si $T \xrightarrow[\sigma_{out}:C]{r} T_1, \ldots, T_n$ et $T_i \xrightarrow[\sigma_{out}^{(i)}:C_i]{R_i} \vec{T_i}$ tels que $\cup_{1 \le i \le n} \sigma_{out}^{(i)}$ est acyclique et définit une substitution σ alors $T \xrightarrow[\sigma_{out} \cdot \sigma:C]{r(R_1,\ldots,R_n)} \vec{T_1}\sigma, \ldots, \vec{T_n}\sigma$.

Une GAG est dite acyclique si la condition apparaissant en (3) est satisfaite pour toutes les tâches accessibles à partir d'une tâche initiale (service de son interface).

CONJECTURE : Si une GAG est acyclique (on a associativité, i.e.) si R est une macro-règle n-aire et $R_1, \ldots, R_n$ sont des macro-règles alors $R(R_1, \ldots, R_n)$ est une macro-règle et si $T \xrightarrow[\sigma_{out}:C]{R} T_1, \ldots, T_n$ et $T_i \xrightarrow[\sigma_{out}^{(i)}:C_i]{R_i} \vec{T_i}$ tels que $\cup_{1 \le i \le n} \sigma_{out}^{(i)}$ définit la substitution σ alors $T \xrightarrow[\sigma_{out} \cdot \sigma:C]{R(R_1,\ldots,R_n)} \vec{T_1}\sigma, \ldots, \vec{T_n}\sigma$.

IDENTIFIER UNE CONDITION DE FORTE ACYCLICITE, VERIFIABLE EN TEMPS POLYNOMIAL PAR UN CALCUL DE POINT FIXE, PROCURANT UNE CONDITION SUFFISANTE D'ACYCLICITE

FAIRE LE LIEN ENTRE ACYCLICITE ET LA POSSSIBILITE DE DISTRIBUER LA GAG.

On peut être plus précis dans la définition des macro-règles du fait que les tâches T pour lesquelles elles s'appliquent peuvent être caractérisées par des motifs

On a par exemple

$$\textbf{bin}\{\underline{\text{inh}} = x, y = \underline{\text{syn}}\} \xrightarrow[\{y=\text{Cons}(a,y')\}:\text{Fork}\{\underline{\text{left}}=\text{Leaf}\{\underline{\text{arg}}=a\},\underline{\text{right}}=[\,]\}]{fork(leaf\{\underline{\text{arg}}=a\},[\,])} \textbf{bin}\{\underline{\text{inh}} = x, y' = \underline{\text{syn}}\}$$

Correspondant à la macro-règle :

$$\mathit{fork}(\mathit{leaf}\{\underline{\text{arg}} = \text{a}\}, [\,]) : \mathbf{bin}\{x = \underline{\text{inh}}, \underline{\text{syn}} = \text{Cons}(a, y')\} \to \text{Fork}\left\{\begin{array}{l}\underline{\text{left}} = \text{Leaf}\{\underline{\text{arg}} = a\} \\ \underline{\text{right}} = \mathbf{bin}\{\underline{\text{inh}} = x, y' = \underline{\text{syn}}\}\end{array}\right\}$$

et

$$\mathbf{bin}\{\underline{\text{inh}} = x, y = \underline{\text{syn}}\} \xrightarrow[\{\,\}:\text{Fork}\{\underline{\text{left}}=[\,],\underline{\text{right}}=\text{Leaf}\{\underline{\text{arg}}=a\}\}]{\mathit{fork}([\,],\mathit{leaf}\{\underline{\text{arg}}=a\})} \mathbf{bin}\{\underline{\text{inh}} = \text{Cons}(a, x), y = \underline{\text{syn}}\}$$

Correspondant à la macro-règle :

$$\mathit{fork}([\,], \mathit{leaf}\{\underline{\text{arg}} = a\}) : \mathbf{bin}\{x = \underline{\text{inh}}, \underline{\text{syn}} = y\} \to \text{Fork}\left\{\begin{array}{l}\underline{\text{left}} = \mathbf{bin}\{\underline{\text{inh}} = \text{Cons}(a, x), y = \underline{\text{syn}}\} \\ \underline{\text{right}} = \text{Leaf}\{\underline{\text{arg}} = a\}\end{array}\right\}$$

3 Opérations de Compositions

Dans cette partie on étudie un certain nombre d'opérations pour la composition de GAG en vue d'aboutir à un langage (graphique ?) permettant de spécifier des GAGs complexes à partir de spécifications simples combinées par ces opérations.

3.1 Composition modulaire (somme)

C'est la forme de composition "la plus naturelle".

A DEVELOPPER SUIVANT CE QUE J'EN AVAIS PRESENTE PENDANT NOTRE REUNION DU 19 MARS

3.2 Couplage d'aspects (produit)

J'ai esquissé la définition de cette opération lors de la réunion du 19 mars. Je suggère que le sujet de notre réunion du 26 mars soit de donner une définition explicite et complète de cette opération

3.3 Composition séquentielle

En 2.7 lors de l'introduction des macro-règles, nous avons présenté les règles d'une GAG sous la forme $T \xrightarrow[\sigma_{out}:C]{r} T_1, \ldots, T_n$ ce qui la fait apparaître comme un transducteur transformant un arbre d'entrée, décrivant une macro-règle (programme d'entrée), en un arbre de sortie, donnant l'arbre produit par application de ce programme à partir d'une configuration initiale donnée. Si l'arbre produit peut s'interpréter comme un programme (macro-règle) pour une autre GAG on peut songer à composer séquentiellement ces deux GAG (de façon paresseuse).

A CREUSER

3.4 GAG d'ordre supérieur et composition descriptionnelle

A l'instar des grammaires attribuées d'ordre supérieur, on peut considérer que la valeur des attributs étant des arbres on peut eux même les doter d'attributs et de règles sémantiques, lesquelles attributs étant des arbres peuvent ... Cela ne devrait compliquer pas trop la définition du modèle, d'un point de vue pratique cela peut-être bien utile pour "séparer des préoccupations", par exemple on peut représenter les informations devant produire un tableau de bord sous forme d'un arbre, décrivant de façon hiérarchique les différents constituants du tableau de bord. Des règles sémantiques (pour la signature initiale) disent comment on génére ce tableau de bord (où plutôt sa description formelle) puis d'autres règles sémantiques opérant sur cette représentation disent comment générer une description de la façon de représenter plus concrètement ce tableau de bord, ce qui peut aussi dépendre d'autres paramêtres (hérités). Sous certaines conditions (single used requirement : la valeur d'un attribut synthétisé ne peut être utilisé qu'en au plus un endroit) il est possible d'aplatir une grammaire attribuée d'ordre supérieur en une grammaire attribuée ordinaire. Cette opération est appelée composition descriptionnelle. Il peut être intéressant d'étudier cette opération pour les GAGs. Par ailleurs les GAG d'ordre supérieur présentent à mon avis un intérêt en soi, d'un point de vue pratique, même si elles ne peuvent pas être transformées en des GAG ordinaires.

3.5 Spécialisation et composition invasive

Cela repose sur une opération de "driving" consistant à construire un graphe décrivant formellement l'exécution d'une spécification. Si le graphe obtenu est fini on doit être capable d'en extraire une nouvelle spécification, équivalente à celle de départ, mais dans lquelle on a effectué de façon statique des calculs qui initialement se faisait dynamiquement (et de façon potentiellement répétitive). On a ainsi optimisé la spécification de départ (en faisant de l'évaluation partielle). Cela peut-être en particulier utile pour "spécialiser" une spécification en remplaçant les services se trouvant dans son interface par des instances particulières de ces services (on a partiellement précisé certaines valeurs de leurs attributs hérités). La composition invasive de composants (composants de base + bus logiciels) consiste à les regrouper (on en fait la somme comme décrite en 3.1), on applique le driving pour optimiser le résultat puis on projette sur chaque composante. Chaque composant est ainsi remplacé par une une de ses spécialisations qui tient compte du contexte particulier dans lequel ce composant est utilisé.