A Grammatical Approach to Data-centric Case Management in a Distributed Collaborative Environment

Eric Badouel
Inria and LIRIMA
Campus de Beaulieu
35042 Rennes, France
eric.badouel@inria.fr

Loïc Hélouët
Inria
Campus de Beaulieu
35042 Rennes, France
loic.helouet@inria.fr

Georges-Edouard Kouamou
ENSP and LIRIMA
BP 8390, Yaoundé, Cameroon
georges.kouamou@lirima.org

Christophe Morvan
Université Paris-Est
UPEMLV, F-77454
Marne-la-Vallée, France
christophe.morvan@u-pem.fr

ABSTRACT

This paper presents a purely declarative approach to artifact-centric case management systems. Each case is presented as a tree-like structure; nodes bear information that combines data and computations. Each node belongs to a given stakeholder, and semantic rules govern the evolution of the tree structure, as well as how data values derive from information stemming from the context of the node. Stakeholders communicate through asynchronous message passing without shared memory, enabling convenient distribution.

Keywords

Business Artifacts, Case Management, Attribute Grammars

1. INTRODUCTION

Case-management consists in assembling relevant information during short collaborative processes that may involve human stakeholders. It is frequently addressed using the notion of *Business Artifacts*, also known as *business entities with lifecycles*, as proposed in [8, 6, 3]. An artifact is a document that conveys all the information concerning a particular case from its inception in the system until its completion. It contains all the relevant information about the entity together with a lifecycle that models its possible evolutions through the business process.

This paper presents a declarative model for the specification of artifact-centric case management systems where the stakeholders interact according to an asynchronous message-based communication schema. Case-management usually consists in assembling relevant information by calling *tasks*, which may in turn call subtasks, etc. Case elicitation needs not be implemented as a sequence of successive calls to subtasks, and several subtasks can be performed in parallel. To allow as much as concurrency as possible in the execution of tasks, we favor a *declarative* approach where tasks dependencies are specified without imposing a particular execution order.

Attribute grammars are particularly adapted to that purpose. The model proposed in this paper is called *Guarded Attributed Grammars* (GAG). A GAG is an extension of an attribute grammars [7, 9] using a notation reminiscent of unification grammars. It is made of production rules, that are used to complete documents (via a standard rewriting process) and at the same time synthesize and propagate information via the attributes of the grammar.

A production of a grammar is as usual described by a left hand side, indicating a non-terminal to expand, and a right hand side, describing how to expand this non-terminal. We furthermore interpret a production of the grammar as a way to decompose a task (the symbol in the left-hand side of the production) into sub-tasks associated with the symbols in its right-hand side. The semantics rules basically serve as a glue between the task and its sub-tasks by making the necessary connections between the corresponding inputs and outputs (associated respectively with inherited and synthesized attributes).

In this declarative model, the lifecycle of artifacts is left implicit. Artifacts under evaluation can be seen as incomplete structured documents, i.e., trees with *open nodes* corresponding to parts of the document that remain to be completed. Each open node is attached a so-called *form* interpreted as a service call. A form consists of a task together with some inherited attributes (data resulting from previous executions) and some synthesized attributes. The latter are variables subscribing to the values that should emerge from task execution.

Productions are *guarded* by patterns occurring at the inherited positions of the left-hand side symbol. Thus a production is enabled at an open node if the patterns match with the corresponding attribute values appearing in the form. The evolution of the artifact thus depends both on previously computed data (stating which production is enabled) and the stakeholder's decisions (choosing a particular production amongst those which are enabled at a given moment, and inputting associated data). Thus GAGs are both

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

data-driven and *user-centric*.

Data manipulated in guarded attributed grammars are of two kinds. First, the tasks communicate using *forms* which are temporary information used for communication purpose only. Second, *artifacts* are structured documents that record the history of cases (log of the system). An artifact grows monotonically (we never erase information) and every part of it is edited by a unique stakeholder (the owner of the corresponding nodes), hence avoiding edition conflicts. These properties are instrumental to obtain a simple and robust model that can easily be implemented on a distributed asynchronous architecture.

This paper is organized as follows : Section 2 introduces the formal notations for GAGs. Section 3 gives a semantics to the model in terms of rewriting steps. Section 4 briefly states some formal properties of the model, before conclusion.

2. GUARDED ATTRIBUTE GRAMMARS

This section introduces a *Guarded Attributed Grammars* a grammatical notation for case management. It is inspired by the work of Deransart and Maluszynski [4] relating attribute grammars with definite clause programs. Throughout the paper, the term *case* will designate a concrete instance of a given business process. We will use the editorial process of an academic journal as a running example to illustrate the various notions and notations. A case for this example is the editorial processing of a particular article submitted to the journal.

The case is handled by various actors involved in the process, the so-called *stakeholders*, namely the editor in chief, an associate editor and some referees. We associate each case with a document, called an *artifact*, that collects all the information related to the case from its inception in the process until its completion. When the case is closed this document constitutes a full history of all the decisions that led to its completion.

We interpret a case as a problem to be solved, that can be completed by refining it into sub-tasks using business rules. This notion of business rule can be modeled by a *production* $P : s_0 \leftarrow s_1 \cdots s_n$ expressing that task s_0 can be reduced to subtasks s_1 to s_n. If several productions with the same left-hand side s_0 exist then the choice of a particular production corresponds to a decision made by some designated stakeholder. For instance, there are two possible immediate outcomes for a submitted article: either it is validated by the editor in chief and it enters the evaluation process of the journal or it is invalidated because its topic or format is not adequate. This initial decision can be reflected by the two following productions:

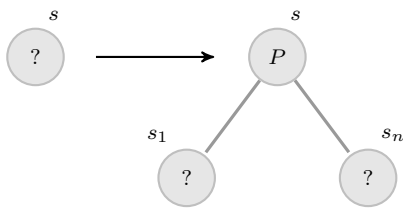
validate : **Proposed_submission** ← **Submission**
invalidate : **Proposed_submission** ←

If P is the unique production having s_0 in its left-hand side, then there is no real decision to make and such a rule is interpreted as a logical decomposition of the task s_0 into substasks s_1 to s_n. Such a production will be automatically triggered without human intervention.

Accordingly, we model an artifact as a tree whose nodes are sorted. We write $X :: s$ to indicate that node X is of sort s. An artifact is given by a set of equations of the form $X = P(X_1, \ldots, X_n)$, stating that $X :: s$ is a node labeled by production $P : s \leftarrow s_1 \cdots s_n$ and with successor nodes $X_1 :: s_1$ to $X_n :: s_n$. In that case node X is said to be a *closed* node defined by equation $X = P(X_1, \ldots, X_n)$ (we henceforth assume that we do not have two equations with the same left-hand side). A node $X :: s$ defined by no equation (i.e. that appears only in the right hand side of an equation) is an *open node*. It corresponds to a pending task.

The lifecycle of an artifact is implicitly given by a set of productions:

1. The artifact initially associated with a case is reduced to a single open node.
2. An open node X of sort s can be *refined* by choosing a production $P : s \leftarrow s_1 \ldots s_n$ that fits its sort. The open node X becomes a closed node $X = P(X_1, \ldots, X_n)$ under the decision of applying production P to it. In doing so the task s associated with X is replaced by n subtasks s_1 to s_n and new open nodes $X_1 :: s_1$ to $X_n :: s_n$ are created accordingly.



3. The case has reached completion when its associated artifact is closed, i.e. it no longer contains open nodes.

However, plain context-free grammars do not model the interactions and data exchanged between the various tasks associated with open nodes. To overcome this problem, we attach additional information to open nodes using *attributes*. Each sort $s \in S$ comes equipped with a set of *inherited* attributes and a set of *synthesized* attributes. Values of attributes are given by *terms* over a ranked alphabet. Recall that such a term is either a variable or an expression of the form $c(t_1, \ldots, t_n)$ where c is a symbol of rank n, and $t_1, \ldots, t_n$ are terms. In particular a constant c, i.e. a symbol of rank 0, will be identified with the term $c()$. We will denote by $var(t)$ the set of variables used in term t.

DEFINITION 2.1. *A* **form** *of sort s is an expression*

$$F = s(t_1, \ldots, t_n)\langle u_1, \ldots, u_m \rangle$$

where $t_1, \ldots, t_n$ (respectively $u_1, \ldots, u_m$) are terms over a ranked alphabet —the alphabet of attribute's values— and a set of variables $var(F)$. Terms $t_1, \ldots, t_n$ give the values of the **inherited attributes** *and $u_1, \ldots, u_m$ the values of the* **synthesized attributes***) attached to form F.*

We can now define productions where the left-hand and right-hand sides of a rule are defined using forms. More precisely, a production is of the form

$$s_0(p_1, \ldots, p_n)\langle u_1, \ldots, u_m \rangle \leftarrow \begin{array}{l} s_1(t_1^{(1)}, \ldots, t_{n_1}^{(1)})\langle y_1^{(1)}, \ldots, y_{m_1}^{(1)} \rangle \\ \cdots \\ s_k(t_1^{(k)}, \ldots, t_{n_k}^{(k)})\langle y_1^{(k)}, \ldots, y_{m_k}^{(k)} \rangle \end{array} \quad \textbf{(1)}$$

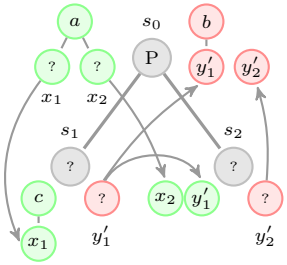
where the p_i's, the u_j's, and the $t_j^{(\ell)}$'s are terms and the $y_j^{(\ell)}$'s are variables. The forms in the right-hand side of a production are *service calls*, namely they are forms $F =$

$s(t_1, \ldots, t_n)\langle y_1, \ldots, y_m\rangle$ where the synthesized positions are (distinct) variables $y_1, \ldots, y_m$ (i.e., they are not instantiated). The rationale is that we invoke a service by filling in the inherited positions of the form (the entries) and by indicating the variables that expect to receive the results returned by the service (the subscriptions).

Any open node is now attached a service call. The corresponding service is supposed to *(i)* construct the tree that will refine the open node and *(ii)* compute the values of the synthesized attributes (i.e., it should return the subscribed values). A service is enacted by applying productions. More precisely, a production such as the one given in formula (1) can apply in an open node X when its left-hand side matches with the service call $s_0(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ attached to node X. For that purpose the terms p_i's are used as patterns that should match the corresponding data d_i's. When the production applies, new open nodes are created and they are respectively associated with the forms (service calls) in the right-hand side of the production. The values of u_j's are then returned to the corresponding variables y_j's that had subscribed to these values. For instance applying production

$$P : s_0(a(x_1, x_2))\langle b(y_1'), y_2'\rangle \leftarrow s_1(c(x_1))\langle y_1'\rangle \;\; s_2(x_2, y_1')\langle y_2'\rangle$$

to a node associated with service call $s_0(a(t_1, t_2))\langle y_1, y_2\rangle$ gives rise to the substitution $x_1 = t_1$ and $x_2 = t_2$. The two newly-created open nodes are respectively associated with the service calls $s_1(c(t_1))\langle y_1'\rangle$ and $s_2(t_2, y_1')\langle y_2'\rangle$ and the values $b(y_1')$ and y_2' are substituted to the variables y_1 and y_2 respectively.



A Guarded Attributed Grammar (GAG for short) is defined as a set of production rules of the form $P : F_0 \leftarrow F_1 \ldots F_k$, where all $F_i's$ are forms.

Definition 2.2 (Guarded Attribute Grammars). *Given a set of sorts S with fixed inherited and synthesized attributes. A* ***guarded attribute grammar*** *is a set of productions $P : F_0 \leftarrow F_1 \cdots F_k$ where the $F_i :: s_i$ are forms. The inherited attributes of left-hand side F_0 are called the* ***patterns*** *of the production. The values of synthesized attributes in the right-hand side are variables. These occurrence of variables together with the variables occurring in the patterns are called the* **input occurrences** *of variables. We assume that each variable has* **at most one input occurrence**.

A GAG is *weel-formed* whenever every output is defined in terms of the inputs. More precisely, the inputs are associated with (distinct) variables and the value of each output is given by a term using these variables. We will refer to these correspondences as the **semantic rules**.

For our running example, a GAG defining the editorial process can be defined as follows: A stakeholder has a specific **role** in the editorial process: he can be an author, the editor in chief, an associate editor or a referee. Each role is associated with a set of services and a set of productions explaining how each service is provided. For instance an associate editor provides the service **Submission**$(article)\langle decision\rangle$ consisting in returning an editorial decision about an article submitted to the journal. The corresponding productions are listed in Table 1. The first two productions mean that an associate editor makes an editorial decision about a submitted paper on the basis of the evaluation reports produced by two different referees. He can ask a report from a reviewer through an invocation of the external service **ToReview**$(article)\langle answer\rangle$. The productions that govern the actions of a reviewer are given in Table 2.

Productions Decline(msg) and Accepts(msg) reflect a nondeterministic choice of a reviewer. They are also a way to input new data by assigning a particular message Msg to variable *msg* resulting in the respective attribute values No(Msg) or Yes(Msg, *report*).

One can group the productions of Table 1 and Table 2 using an additional parameter *reviewer* to make as many disjoint copies of the specification given in Table 2 as there are individuals playing the role of a referee. The resulting set of productions (where call to external services have been eliminated) is given in Table 3. Similarly one has as many instances of the productions in Table 1 as there are associate editors in the editorial board. In the complete specification one should therefore add an additional parameter *associateEditor* to distinguish between all associate editors. If the specification is large and contains many different roles the resulting global grammar can be quite complex. Yet, it is still possible to build an equivalent monolithic grammar without external service calls.

3. BEHAVIOR OF GAGS

Attribute grammars are traditionally applied to abstract syntax trees which can be produced by some parsing algorithm during a previous stage. The semantic rules are then used to decorate the nodes of the tree by attribute values. In our setting the generation of the tree and evaluation of attributes, using the semantic rules, are intertwined since the input tree represents an artifact under construction.

We consider collaborative systems relying on a distributed memory consisting of the current artifacts. A configuration of this memory can be represented as follows:

Definition 3.1 (Configuration). *A* **configuration** Γ *is an S-sorted set of nodes $X \in nodes(\Gamma)$ each of which is associated with a defining equation in one of the following form where $var(\Gamma)$ is a set of variables associated with Γ:*

Closed node: *$X = P(X_1, \ldots, X_k)$ where $P : F_0 \leftarrow F_1 \ldots F_k$ is a production of the grammar and $X :: s$, and $X_i :: s_i$ for $1 \leq i \leq k$. Production P is the* **label** *of node X and nodes X_1 to X_n are its* **successor nodes**.

Open node: *$X = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle$ where X is of sort s and $t_1, \ldots, t_k$ are terms with variables in $var(\Gamma)$ that represent the values of the inherited attributes of X, and $x_1, \ldots, x_m$ are variables in $var(\Gamma)$ associated with its synthesized attributes.*

Each variable in $var(\Gamma)$ occurs at most once in a synthesized position.

We identify a substitution σ on a set of variables $x_1, \ldots, x_k$, called the *domain* of σ, with a system of equations $x_i =$

Table 1: Acting as an associate Editor

$$\begin{array}{l}
\text{DecideSubmission} : \mathbf{Submission}(article)\langle decision\rangle \leftarrow \quad \mathbf{Evaluate}(article)\langle report_1\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{Evaluate}(article)\langle report_2\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{Decide}(report_1, report_2)\langle decision\rangle \\
\text{MakeDecision}(decision) : \mathbf{Decide}(report_1, report_2)\langle decision\rangle \leftarrow \\
\text{AskReview}(reviewer) : \mathbf{Evaluate}(article)\langle report\rangle \leftarrow \quad \mathbf{WaitReport}(answer, article)\langle report\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{Call}(reviewer, \mathbf{ToReview}(article)\langle answer\rangle) \\
\text{CaseNo}\langle msg\rangle : \mathbf{WaitReport}(\text{No}(msg), article)\langle report\rangle \leftarrow \mathbf{Evaluate}(article)\langle report\rangle \\
\text{CaseYes}\langle msg\rangle : \mathbf{WaitReport}(\text{Yes}(msg, report), article)\langle report\rangle \leftarrow
\end{array}$$

Table 2: Acting as a reviewer

$$\begin{array}{rcl}
\text{Decline}(msg) & : & \mathbf{ToReview}(article)\langle\text{No}(msg)\rangle \leftarrow \\
\text{Accept}(msg) & : & \mathbf{ToReview}(article)\langle\text{Yes}(msg, report)\rangle \leftarrow \mathbf{Review}(article)\langle report\rangle \\
\text{MakeReview}(report) & : & \mathbf{Review}(article)\langle report\rangle \leftarrow
\end{array}$$

$\sigma(x_i)$. The set $\text{var}(\sigma) = \bigcup_{1\le i\le k} \text{var}(\sigma(x_i))$ of variables of σ is disjoint from the domain of σ. Conversely a system of equations $\{x_i = t_i\}_{1\le i\le k}$ defines a substitution σ with $\sigma(x_i) = t_i$ if it is in *solved form*, i.e., none of the variables x_i appears in some of the terms t_j. In order to transform a system of equations $E = \{x_i = t_i\}_{1\le i\le k}$ into an equivalent system $\{x_i = t'_j\}_{1\le j\le m}$ in solved form one can iteratively replace an occurrence of a variable x_i in one of the right-hand side term t_j by its definition t_i until no variable x_i occurs in some t_j. This process terminates when the relation $x_i \succ x_j \Leftrightarrow x_j \in \text{var}(\sigma(x_i))$ is acyclic. Then the resulting system of equations $SF(E) = \{x_i = t'_i\}_{1\le i\le n}$ in solved form does not depend on the order in which the variables x_i have been eliminated from the right-hand sides. When the above condition is met we say that the set of equations is *acyclic* and that it *defines* the substitution associated with its solved form.

The composition of two substitutions σ, σ' is denoted by $\sigma\sigma'$ and defined by $\sigma\sigma' = \{x = t\sigma' \mid x = t \in \sigma\}$. Similarly, we let $\Gamma\sigma$ denote the configuration obtained from Γ by replacing the defining equation $X = F$ of each open node X by $X = F\sigma$.

We now define more precisely when a production is enabled at a given open node of a configuration and the effect of applying the production. First note that variables of a production are formal parameters which scope is limited to that production. They can injectively be renamed in order to avoid clashes with variables names appearing in a configuration. Therefore we shall always assume that the set of variables of a production P is disjoint from the set of variables of a configuration Γ when applying production P to Γ. As informally stated in the previous section, a production P applies in an open node X when its left-hand side $s(p_1, \ldots, p_n)\langle u_1, \ldots u_m\rangle$ matches with the definition $X = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$, i.e., the service call attached to X in Γ.

First, the patterns p_i should match with the data d_i according to the usual pattern matching operation given by the following inductive statements

$$\begin{array}{l}
\mathbf{match}(c(p'_1, \ldots, p'_k), c'(d'_1, \ldots, d'_{k'})) \text{ with } c \neq c' \text{ fails} \\
\mathbf{match}(c(p'_1, \ldots, p'_k), c(d'_1, \ldots, d'_k)) \;=\; \sum_{i=1}^{k} \mathbf{match}(p'_i, d'_i) \\
\mathbf{match}(x, d) = \{x = d\}
\end{array}$$

where the sum-substitution, $\sigma = \sum_{i=1}^{k} \sigma_i$, of substitutions σ_i is defined and equal to $\bigcup_{i\in 1..k} \sigma_i$ when all substitutions σ_i are defined and associated with disjoint sets of variables. Note that since no variable occurs twice in the whole set of patterns p_i, the various substitutions $\mathbf{match}(p_i, d_i)$, when defined, range over disjoint sets of variables. Note also that $\mathbf{match}(c(), c()) = \emptyset$.

DEFINITION 3.2. *A form $F = s(p_1, \ldots, p_n)\langle u_1, \ldots u_m\rangle$* **matches** *with a service call $F' = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ (of the same sort) when*

1. *the patterns p_i's match with the data d_i's, defining a substitution $\sigma_{in} = \sum_{1\le i\le n} \mathbf{match}(t_i, d_i)$,*
2. *the set of equations $\{y_j = u_j\sigma_{in} \mid 1 \le j \le m\}$ is acyclic and defines a substitution σ_{out}.*

The resulting substitution $\sigma = \mathbf{match}(F, F')$ is given by $\sigma = \sigma_{out} \cup \sigma_{in}\sigma_{out}$.

DEFINITION 3.3 (APPLYING A PRODUCTION). *Let $P = F \leftarrow F_1 \ldots F_k$ be a production, Γ be a configuration, and X be an open node with definition $X = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ in Γ. We assume that P and Γ are defined over disjoint sets of variables. We say that P is* ***enabled*** *in X and write $\Gamma[P/X\rangle$, if the left-hand side of P matches with the definition of X. Then applying production P in X transforms configuration Γ into Γ', denoted as $\Gamma[P/X\rangle\Gamma'$, where:*

$$\begin{array}{rcl}
\Gamma' & = & \{X = P(X_1, \ldots, X_k)\} \\
& \cup & \{X_1 = F_1\sigma, \ldots, X_k = F_k\sigma\} \\
& \cup & \{X' = F\sigma \mid (X' = F) \in \Gamma \;\wedge\; X' \neq X\}
\end{array}$$

where $X_1, \ldots, X_k$ are new nodes added to Γ' and $\sigma = \mathbf{match}(F, X)$.

Thus the first effect of applying production P to an open node X is that X becomes a closed node with label P and successor nodes X_1 to X_k. The latter are new nodes added to Γ'. They are associated respectively with the instances of the k forms in the right-hand side of P obtained by applying substitution σ to these forms. The definitions of the other nodes of Γ are updated using substitution σ (or equivalently σ_{out}). This update has no effect on the closed nodes because their defining equations in Γ contain no variable.

Table 3: Making a decision on a submitted paper

$$\begin{array}{l}
\text{DecideSubmission} : \mathbf{Submission}(article)\langle decision\rangle \leftarrow \quad \mathbf{Evaluate}(article)\langle report_1\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{Evaluate}(article)\langle report_2\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{Decide}(report_1, report_2)\langle decision\rangle \\
\text{MakeDecision}(decision) : \mathbf{Decide}(report_1, report_2)\langle decision\rangle \leftarrow \\
\text{AskReview}(reviewer) : \mathbf{Evaluate}(article)\langle report\rangle \leftarrow \quad \mathbf{WaitReport}(answer, article)\langle report\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{ToReview}(reviewer, article)\langle answer\rangle) \\
\text{Decline}(msg)\langle reviewer\rangle : \mathbf{ToReview}(reviewer, article)\langle \text{No}(msg)\rangle \leftarrow \\
\text{Accept}(msg)\langle reviewer\rangle : \mathbf{ToReview}(reviewer, article)\langle \text{Yes}(msg, report)\rangle \leftarrow \mathbf{Review}(reviewer, article)\langle report\rangle \\
\text{MakeReview}(report)\langle reviewer\rangle : \mathbf{Review}(reviewer, article)\langle report\rangle \leftarrow \\
\text{CaseNo}\langle msg\rangle : \mathbf{WaitReport}(\text{No}(msg), article)\langle report\rangle \leftarrow \mathbf{Evaluate}(article)\langle report\rangle \\
\text{CaseYes}\langle msg\rangle : \mathbf{WaitReport}(\text{Yes}(msg, report), article)\langle report\rangle \leftarrow
\end{array}$$

One can show that applying a production P in an open node X of a configuration Γ with $\Gamma[P/X\rangle\Gamma'$ cannot create a variable with several occurrences in synthesized position, i.e. the resulting set of equations Γ' is also a configuration. Thus applying an enabled production defines a binary relation on configurations.

DEFINITION 3.4. *A configuration Γ' is **directly reachable** from Γ, denoted by $\Gamma[\rangle\Gamma'$, whenever $\Gamma[P/X\rangle\Gamma'$ for some production P enabled in node X of configuration Γ. Furthermore, a configuration Γ' is **reachable** from configuration Γ when $\Gamma[*\rangle\Gamma'$ where $[*\rangle$ is the reflexive and transitive closure of relation $[\rangle$.*

As already mentioned, an artifact is refined by applying a production to one of its open node. However we also need means to initiate cases. To this extent, we define interfaces for GAGs, that describe how services can initialize new artifacts.

DEFINITION 3.5. *The **interface** of a guarded attribute grammar is given by a subset $\mathcal{I}$ of forms, called its **services**, $F = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle$ where the synthesized positions are (distinct) variables $x_1, \ldots, x_m$. This set is closed by substitutions whose domains are disjoints from the set of synthesized variables, namely $F\sigma \in \mathcal{I}$ whenever $F \in \mathcal{I}$ and σ is a substitution with $\sigma(x_j) = x_j$ for $1 \leq j \leq m$. The invocation of the service produces a new artifact reduced to a single open node defined by F, it is associated with **initial configuration** $\Gamma_0 = \{X_0 = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle\}$. An **reachable configuration** of a guarded attribute grammar is a configuration reachable from one of its initial configurations.*

4. FORMAL PROPERTIES OF GAGS

The GAG model is very powerful. Indeed allowing rewriting using attributes that take values over unbounded terms gives our model a huge expressive power. Unsurprisingly, this expressive power implies that some formal properties are undecidable.

A specification is sound if every case can reach completion no matter how its execution started. A case is a service call in the interface of the GAG (Definition 3.5) which already contains all the information coming from the environment of the guarded attribute grammar.

DEFINITION 4.1. *Given a guarded attribute grammar with its interface, a **case** $c = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle$ is an element of the interface such that $\text{var}(t_i) \subseteq \{x_1, \ldots, x_m\}$. Stated otherwise a case is, but for the variables with a synthesized value, a closed instance of a service.*

DEFINITION 4.2. *A configuration is **closed** if it contains only closed nodes. A guarded attribute grammar is **sound** if a closed configuration is reachable from any configuration Γ reachable from the initial configuration $\Gamma_0(c) = \{X_0 = c\}$ associated with a case c.*

Let γ denote the set of configurations reachable from the initial configuration of some case. We consider the finite sequences $(\Gamma_i)_{0<i\leq n}$ and the infinite sequences $(\Gamma_i)_{0<i}$ of configurations in γ such that $\Gamma_i[\rangle\Gamma_{i+1}$. A finite and maximal sequence is said to be **terminal**, i.e., a terminal sequence leads to a configuration that enables no production. Soundness can the be rephrased by the two following conditions.

1. Every terminal sequence leads to a closed configuration.
2. Every configuration on an infinite sequence also belongs to some terminal sequence.

Soundness can unfortunately be proved undecidable by a simple encoding of Minsky machines.

THEOREM 4.3. *Both the soundness problem (is a given GAG sound) and the reachability problem (is a given configuration reachable from another one, for a given GAG) are undecidable for guarded attribute grammars.*

Despite this result, there exist interesting subclasses of the model enjoy some monotony properties, and are well suited to distribution.

The principle of a *distribution* of a GAG on a set of locations is as follows: A GAG is distributed by partitionning its set of sorts according to *locations*. Each location maintains a local configuration, and subscribes to results provided by other locations. Productions are applied locally. When variables are given a value by a production, the location that computed this value sends messages to the locations that subscribed to this value. Messages are simply equations defining the value of a particular variable. Upon reception of a message, a subscriber updates its local configuration, and may in turn produce new messages sent to subscribers

of affected variables. A formal definition of the distribution framework and a proof of interesting properties of GAGs w.r.t. distribution are provided in appendix B.

Recall that application of a production P to a node X requires a matching condition, that is construction of a pair of matchings σ_{in} and σ_{out}. We say that a production P is **triggered** in node X if substitution σ_{in} is defined, i.e., the patterns p_i match the data d_i. A specification can be considered erroneous when a triggered transition is not enabled because the set of equations $\{y_j = u_j\sigma_{in} \mid 1 \leq j \leq m\}$ is cyclic.

Substitution σ_{in}, given by pattern matching, is monotonous w.r.t. incoming information and thus it causes no problem for a distributed implementation of a model. However substitution σ_{out} is not monotonous: it may be undefined when information coming from a distant location makes the match of output attributes a cyclic set of equations.

DEFINITION 4.4. *A guarded attribute grammar is* ***input-enabled*** *if every production that is* triggered *in a reachable configuration is also* enabled.

For input-enabled GAGs, messages consumptions and application of productions commute (we refer interested reader to appendix B for details). This property means in particular that distribution does not affect the global behavior of an input-enabled GAG.

However, input-enabledness is a property of the whole set of reachable configurations, and is thus undecidale. Nevertheless one can find a decidable sufficient condition for input-enabledness (called *acyclicity*). Acyclicity is similar to the strong non-circularity of attribute grammars [2], and can be checked by a simple fixed-point computation.

DEFINITION 4.5. *Let s be a sort of a guarded attribute grammar with n inherited attributes and m synthesized attributes. We let $(j,i) \in SI(s)$ where $1 \leq i \leq n$ and $1 \leq j \leq m$ if exists $X = s(d_1,\ldots,d_n)\langle y_1,\ldots,y_m\rangle \in \Gamma$ where Γ is a reachable configuration and $y_j \in d_i$. If P is a production with left-hand side $s(p_1,\ldots,p_n)\langle u_1,\ldots,u_m\rangle$ we let $(i,j) \in IS(P)$ if exists a variable $x \in \mathrm{var}(P)$ such that $x \in \mathrm{var}(d_i) \cap \mathrm{var}(u_j)$. The guarded attribute grammar G is said to be* ***acyclic*** *if for every sort s and production P whose left-hand side is a form of sort s the graph $G(s,P) = SI(s) \cup IS(P)$ is acyclic.*

THEOREM 4.6. *An acyclic guarded attribute grammar is input-enabled. Hence it can be distributed on an asynchronous architecture.*

5. CONCLUSION

Guarded attribute grammar is a model of data-centric collaborative systems where emphasis is put on a simple mathematical syntax and semantics which can ease formal reasoning, a clear identification of stakeholder's decisions (the system is totally driven by user interactions), and an implicit lifecycle of artifacts which allows maximal concurrency and a straightforward distribution scheme. Distributed implementation has been considered for other artifact models, such as Guard-Stage-Milestone[1] [5]. However, it may require restructuring the original GSM schema and relies on locking protocols to ensure that the outcome of the global execution is preserved.

An artifact is a structured document with some active parts. Indeed, an open node is associated with a service call that implicitly describes the data to be further substituted to the node. This notion of *active documents* is close to the model of Active XML introduced by Abiteboul et al. [1] which consists of semi-structured documents with embedded service calls. Such an embedded service call is a query on another document, triggered when a corresponding guard is satisfied. The model of active documents can be distributed over a network of machines. This setting can be instancied in many ways, according to the formalism used for specifying the guards, the query language, and the class of documents. The model of guarded attribute grammars is close to this general scheme with some differences: First of all guards in GAG apply to a single node and its attributes, while guards in AXML are properties that can be checked on a complete document. The invocation of a service in AXML creates a temporary document (called the workspace) that is removed from the document when the service call returns. In GAGs, a service call adds new children to a node, and all computations performed for a service are preserved in the artifact. This provides a kind of monotony to artifacts, that can be an useful property for verification techniques.

In the future, we plan to design prototypes to analyze and implement a GAG description together with the required support tools (editor, parser, checker, simulators ...) to develop some representative case studies to check applicability and limitations of the model. In particular, in order to comply with real-life applications, we might have to use *non-autonomous GAG systems*, i.e., systems whose basic layer is given by a guarded attribute grammar but which is coupled with external facilities as making a query to a database or calling a web service. The main concern is then to evaluate the impact of these couplings on the distribution of the model (we should avoid distributed conflits).

[1] A model describing artifacts lifecycles, adopted as a basis for the OMG standard *Case Management Model and Notation* (CMMN).

6. REFERENCES

[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active xml: A data-centric perspective on web services. In *BDA'02*, 2002.

[2] B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes i and ii. *Theor. Comput. Sci.*, 17:163–191 and 235–257, 1982.

[3] E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):22, 2012.

[4] P. Deransart and J. Maluszynski. *A grammatical view of logic programming.* MIT Press, 1993.

[5] R. Eshuis, R. Hull, Y. Sun, and R. Vaculín. Splitting gsm schemas: A framework for outsourcing of declarative artifact systems. In *BPM*, volume 8094 of *LNCS*, pages 259–274. Springer, 2013.

[6] R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM 2008*, volume 5332 of *LNCS*, pages 1152–1163. Springer, 2008.

[7] D.E. Knuth. Semantics of context free languages. *Mathematical System Theory*, 2(2):127–145, 1968.

[8] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42:428–445, July 2003.

[9] J. Paakki. Attribute grammar paradigms - a high-level methodology in language implementation. *ACM Computing Surveys*, 27(2):196–255, 1995.

APPENDIX

The following appendix is not intended to be published in the proceedings of the Conference, where it will be replaced by a reference to a research report containing the corresponding additional materials. The reading of this appendix is left at the discretion of the referees. For readability, propositions and definitions that already appear in the paper are duplicated in the appendix. Appendix A is devoted to the proofs of some results mentioned in Section 3. The second appendix describes formally the distribution of Guarded attribute grammars on an asynchronous architecture, and considers properties of distribution. Finally, Appendix C studies the soundness property and shows that the soundess of GAGs is undecidable.

A. BEHAVIOUR OF GAGS

We first establish the following result mentioned in Sect. 3, namely that *applying a production P in an open node X of a configuration Γ with $\Gamma[P/X\rangle\Gamma'$ cannot create a variable with several occurrences in a synthesized position, i.e. the resulting set of equations Γ' is also a configuration.*

Each variable can have several occurrences in a production. First it may appear once as an input and it may also appear in several occurrences within some output term. The corresponding occurrence is respectively said to be in an *input* or in an *output position.* One can define the following transformation on productions whose effect is to annotate each occurrence of a variable so that $x^?$ (respectively $x^!$) stands for an occurrence of x in an input position (resp. in an output position).

$$\begin{aligned}
!(F_0 \leftarrow F_1 \cdots F_k) &= ?(F_0) \leftarrow !(F_1) \cdots !(F_k) \\
?(s(t_1, \ldots t_n)\langle u_1, \ldots u_m\rangle) &= s(?(t_1), \ldots ?(t_n))\langle !(u_1), \ldots !(u_m)\rangle \\
!(s(t_1, \ldots t_n)\langle u_1, \ldots u_m\rangle) &= s(!(t_1), \ldots !(t_n))\langle ?(u_1), \ldots ?(u_m)\rangle \\
?(c(t_1, \ldots t_n)) &= c(?(t_1), \ldots ?(t_n)) \\
!(c(t_1, \ldots t_n)) &= c(!(t_1), \ldots !(t_n)) \\
?(x) &= x^? \\
!(x) &= x^!
\end{aligned}$$

The conditions stated in Definition 2.2 say that in the labelled version of a production each variable occurs at most once in an input position, i.e., that $\{?(F_0), !(F_1), \ldots, !(F_k)\}$ each variable x has at most one input occruurence $x^?$. Similarly a set of forms Γ is a valid configuration if each variable x has at most one input occruurence $x^?$ in the set $\{!F \mid F \in \Gamma\}$.

PROPOSITION A.1. *If production P is enabled in an open node X_0 of a configuration Γ and $\Gamma[P/X_0\rangle\Gamma'$ then Γ' is a configuration.*

PROOF. Let $P = F \leftarrow F_1 \ldots F_k$ with left-hand side $F = s(p_1, \ldots, p_n)\langle u_1, \ldots u_m\rangle$ and $X_0 = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ be the defining equation of X_0 in Γ. Since the values of synthesized attributes in the forms $F_1, \ldots, F_k$ are variables (by Definition 2.2) and since these variables are unaffected by substitution σ_{in} the synthesized attribute in the resulting forms $F_j\sigma_{in}$ are variables. The substitutions σ_{in} and σ_{out} substitute terms to the variables $x_1, \ldots, x_k$ appearing to the patterns and to the variables $y_1, \ldots, y_m$ respectively. Since x_i appears in an input position in P, it can appear only in an output position in the forms $!(F_1), \ldots !(F_k)$ and thus any variable of the term $\sigma_{in}(x_i)$ will appear in an output position in $!(F_i\sigma_{in})$. Similarly, since y_i appears in an input position

in the form $!(s(u_1,\ldots,u_n)\langle y_1,\ldots,y_m\rangle)$, it can only appear in an output position in $!(F)$ for the others forms F of Γ. Consequently any variable of the term $\sigma_{out}(y_i)$ will appear in an output position in $!(F\sigma_{out})$ for any equation $X = F$ in Γ with $X \neq X_0$. It follows that the application of a production cannot produce new occurrences of a variable in an input position and thus there cannot exist two occurrences $x^?$ of a same variable x in Γ'. □ *Prop.* A.1

In order to prove Proposition A.3 we first recall some fact about unification.

RECALL A.2 (ON UNIFICATION). *We consider sets $E = E_? \uplus E_=$ containing equations of two kinds. An equation in $E_?$, denoted as $t \stackrel{?}{=} u$, represents a unification goal whose solution is a substitution σ such that $t\sigma = u\sigma$, i.e., substitution σ unifies terms t and u. $E_=$ contains only equations of the form $x = t$ where variable x occurs only there, i.e., we do not have two equations with the same variable in their left-hand side and such a variable cannot either occur in any right-hand side of an equation in $E_=$. A solution to E is any substitution σ whose domain is the set of variables occurring in the right-hand sides of equations in $E_=$ such that the compound substitution made of σ and the set of equations $\{x = t\sigma \mid x = t \in E_=\}$ unifies terms t and u for any equation $t \stackrel{?}{=} u$ in $E_?$. Two systems of equations are said to be equivalent when they have the same solutions. A unification problem is a set of such equations with $E_= = \emptyset$, i.e., it is a set of unification goals. On the contrary E is said to be in solved form if $E_? = \emptyset$, thus E defines a substitution which, by definition, is the most general solution to E. Solving a unification problem E consists in finding an equivalent system of equations E' in solved form. In that case E' is a most general unifier for E.*

Martelli and Montanari Unification algorithm [2] *proceeds as follows. We pick up non deterministically one equation in $E_?$ and depending on its shape we apply the corresponding transformation:*

1. $c(t_1,\ldots,t_n) \stackrel{?}{=} c(u_1,\ldots,u_n)$*: replace it by equations* $t_1 \stackrel{?}{=} u_1,\ldots,t_1 \stackrel{?}{=} u_1$.
2. $c(t_1,\ldots,t_n) \stackrel{?}{=} c'(u_1,\ldots,u_m)$ *with* $c \neq c'$*: halt with failure.*
3. $x \stackrel{?}{=} x$*: delete this equation.*
4. $t \stackrel{?}{=} x$ *where t is not a variable: replace this equation by* $x \stackrel{?}{=} t$.
5. $x \stackrel{?}{=} t$ *where* $x \notin \mathrm{var}(t)$*: replace this equation by $x = t$ and substitute x by t in all other equations of E.*
6. $x \stackrel{?}{=} t$ *where* $x \in \mathrm{var}(t)$ *and $x \neq t$: halt with failure.*

The condition in (5) is the occur check. Thus the computation fails either if the two terms of an equation cannot be unified because their main constructors are different or because a potential solution of an equation is necessarily an infinite tree due to a recursive statement detected by the occur check. System E' obtained from E by applying one of these rules, denoted as $E \Rightarrow E'$, is clearly equivalent to E. We iterate this transformation as long as we do not encounter a failure and some equation remains in $E_?$. It can be proved that all these computations terminate and either the original unification problem E has a solution (a unifier) and every computation terminates (and henceforth produces a solved set equivalent to E describing a most general unifier of E) or E has no unifier and every computation fails. We let

$$\sigma = \mathbf{mgu}(\{t_i = u_i\}_{1\leq i\leq n}) \ \textbf{iff}\ \{t_i \stackrel{?}{=} u_i\}_{1\leq i\leq n} \Rightarrow^* \sigma$$

□ *Recall A.2*

Note that (5) and (6) are the only rules that can be applied to solve a unification problem of the form $\{y_i \stackrel{?}{=} u_i\}_{1\leq i\leq n}$, where the y_i are distinct variables. The most general unifier exists when the occur check always holds, i.e., rule (5) always applies. The computation amounts to iteratively replacing an occurrence of a variable y_i in one of the right-hand side term u_j by its definition u_i until no variable y_i occurs in some u_j. This process terminates precisely when the relation $y_i \succ y_j \Leftrightarrow y_j \in u_i$ is acyclic. When this condition is met we say that the set of equations $\{y_i = u_i \mid 1 \leq i \leq n\}$ *is acyclic* and we say that it *defines* the substitution $\sigma = \mathbf{mgu}(\{y_i = u_i \mid 1 \leq i \leq n\})$.

Recall that a substitution σ unifies a set of equations E if $t\sigma = t'\sigma$ for every equations $t = t'$ in E. A substitution σ is more general than a substitution σ' if $\sigma' = \sigma\sigma''$ for some substitution σ''. If a system of equations has a some unifier, then it has (up to an bijective renaming of the variables in σ) a *most general unifier*. In particular a set of equations of the form $\{x_i = t_i \mid 1 \leq i \leq n\}$ has a unifier if and only if it is acyclic. In this case, the corresponding solved form is its most general unifier.

PROPOSITION A.3. *If $F = s(p_1,\ldots,p_n)\langle u_1,\ldots,u_m\rangle$, left-hand side of a production P, matches with the service call $s(d_1,\ldots,d_n)\langle y_1,\ldots,y_m\rangle$ attached to an open node X then the substitution $\sigma = \mathbf{match}(F, X)$ is the most general unifier of the set of equations*

$$\{p_i = d_i \mid 1 \leq i \leq n\} \cup \{y_j = u_j \mid 1 \leq j \leq m\}$$

.

PROOF OF PROPOSITION A.3.
If a production P of left-hand side $s(p_1,\ldots,p_n)\langle u_1,\ldots u_m\rangle$ is triggered in node X_0 defined by $X_0 = s(d_1,\ldots,d_n)\langle y_1,\ldots,y_m\rangle$ then by Definition 3.3

$$\{p_i \stackrel{?}{=} d_i\}_{1\leq i\leq n} \cup \{y_j \stackrel{?}{=} u_j\}_{1\leq j\leq m} \Rightarrow^* \sigma_{in} \cup \{y_j \stackrel{?}{=} u_j\sigma_{in}\}_{1\leq j\leq m}$$

using only the rules (1) and (5). Now

$$\sigma_{in} \cup \{y_j \stackrel{?}{=} u_j\sigma_{in}\}_{1\leq j\leq m} \Rightarrow^* \sigma_{in} \cup \mathbf{mgu}\{y_j = u_j\sigma_{in}\}_{1\leq j\leq m}$$

by applying iteratively rule (5) if the set of equations $\{y_j = u_j\sigma_{in}\}_{1\leq j\leq m}$ satisfies the occur check. Finally $\sigma_{in} + \sigma_{out} \Rightarrow^* \sigma$ again by using rule (5). □ *Prop.* A.3

Note that the converse does not hold. Namely, one shall not deduce from Proposition A.3 that the relation $\Gamma[P/X\rangle\Gamma'$ is defined whenever the left-hand side, $\mathrm{lhs}(P)$, of P can be unified with the definition $\mathrm{def}(X,\Gamma)$ of X in Γ with Γ' defined as in Definition 3.3 where $\sigma = \mathbf{mgu}(\mathrm{lhs}(P), \mathrm{def}(X_0,\Gamma))$ is the corresponding most general unifier. Indeed, when unifying $s(d_1,\ldots,d_n,y_1,\ldots,y_m)$ with $s(p_1,\ldots,p_n,u_1,\ldots,u_m)$

[2] Alberto Martelli and Ungo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2): 258–282, 1982.

one may generate an equation of the form $x = t$ where x is a variable in an inherited data d_i and t is an instance of a corresponding subterm in the associated pattern p_i. This would correspond to a situation where information is sent to the context of a node through one of its inherited attribute. Otherwise stated some parts of the pattern p_i are actually used to filtered out the incoming data value d_i while some other parts of the same pattern would be used to transfert synthesized information to the context.

B. DISTRIBUTION OF A GAG

B.1 Input-enabled GAGs

We say that a production P is **triggered** in node X if substitution σ_{in} is defined, i.e., the patterns p_i match the data d_i. One can suspect an error in the specification when a triggered transition is not enabled due to the fact that the system of equations $\{y_j = u_j\sigma_{in} \mid 1 \leq j \leq m\}$ is cyclic. This situation also impacts the distributability of a grammar as shown by the following example.

EXAMPLE B.1. *Let us consider the GAG with the following productions:*

$$\begin{array}{ll} P : s(\,)\langle\,\rangle & \leftarrow s_1(x)\langle y\rangle \;\; s_2(y)\langle x\rangle \\ Q : s_1(z)\langle a(z)\rangle & \leftarrow \\ R : s_2(u)\langle a(u)\rangle & \leftarrow \end{array}$$

Production P is enabled in configuration $\Gamma_0 = \{X_0 = s(\,)\langle\,\rangle\}$ with $\Gamma_0[P/X_0\rangle\Gamma_1$ where

$$\Gamma_1 = \{X_0 = P(X_1, X_2);\ X_1 = s_1(x)\langle y\rangle,\ X_2 = s_2(y)\langle x\rangle\}$$

In configuration Γ_1 productions Q and R are enabled in nodes X_1 and X_2 respectively with $\Gamma_1[Q/X_1\rangle\Gamma_2$ and $\Gamma_1[R/X_2\rangle\Gamma_3$ where

$$\begin{array}{l} \Gamma_2 = \{X_0 = P(X_1, X_2);\ X_1 = Q,\ X_2 = s_2(a(x))\langle x\rangle\} \\ \Gamma_3 = \{X_0 = P(X_1, X_2);\ X_1 = s_2(a(y))\langle y\rangle,\ X_2 = R\} \end{array}$$

Now production R is triggered but not enabled in configuration Γ_2 because of the cyclicity of $\{x = a(a(x))\}$. There is a conflict between the application of productions R and Q in configuration Γ_1, which makes this specification non-implementable in case nodes X_1 and X_2 have distinct locations.

Substitution σ_{in}, given by pattern matching, is monotonous w.r.t. incoming information and thus it causes no problem for a distributed implementation of a model. However substitution σ_{out} is not monotonous since it may become undefined when information coming from a distant location makes the match of output attributes a cyclic set of equations, as illustrated by example B.1.

DEFINITION 4.4 *A guarded attribute grammar is **input-enabled** if every production that is triggered in a reachable configuration is also enabled.* □ *Def. 4.4*

It is difficult to verify input-enabledness as the whole set of reachable configurations are involved in this condition. Nevertheless one can find sufficient condition for input-enabledness, similar to the strong non-circularity of attribute grammars [2], that can be checked by a simple fixed-point computation.

DEFINITION 4.5 *Let s be a sort of a guarded attribute grammar with n inherited attributes and m synthesized attributes. We let $(j,i) \in SI(s)$ where $1 \leq i \leq n$ and $1 \leq j \leq m$ if exists $X = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle \in \Gamma$ where Γ is a reachable configuration and $y_j \in d_i$. If P is a production with left-hand side $s(p_1, \ldots, p_n)\langle u_1, \ldots, u_m\rangle$ we let $(i,j) \in IS(P)$ if exists a variable $x \in \mathrm{var}(P)$ such that $x \in \mathrm{var}(d_i) \cap \mathrm{var}(u_j)$. The guarded attribute grammar G is said to be **acyclic** if for every sort s and production P whose left-hand side is a form of sort s the graph $G(s,P) = SI(s) \cup IS(P)$ is acyclic.* □ *Def. 4.5*

THEOREM 4.6 *An acyclic guarded attribute grammar is input-enabled.*

PROOF. Suppose P is triggered in node X with substitution σ_{in} such that $y_j \in u_i\sigma_{in}$ then $(i,j) \in G(s,P)$. Then the fact that occur check fails for the set $\{y_j \mid 1 \leq j \leq m\}$ entails that one can find a cycle in $G(s,P)$. □ *Prop.* 4.6

Relation $SI(s)$ still takes into account the whole set of reachable configurations. The following definition provides an overapproximation of this relation given by a fixed point computation.

DEFINITION B.2. *The **graph of local dependencies** of a production $P : F_0 \leftarrow F_1 \cdots F_\ell$ is the directed graph $GLD(P)$ that records the data dependencies between the occurrences of attributes given by the semantics rules. We designate the occurrences of attributes of P as follows: we let $k(i)$ (respectively $k\langle j\rangle$) denote the occurrence of the i^{th} inherited attribute (resp. the j^{th} synthesized attribute) in F_k. If s is a sort with n inherited attributes and m synthesized attributes we define the relations $\overline{IS(s)}$ and $\overline{SI(s)}$ over $[1,n] \times [1,m]$ and $[1,m] \times [1,n]$ respectively as the least relations such that :*

1. *$\overline{SI(s)} = SI(s)$ if s is an axiom, i.e., it is given by the set of pairs (j,i) such that $y_j \in \mathrm{var}(d_i)$ for some service $F = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ of sort s in the interface of the guarded attribute grammar.*
2. *For every production $P : F_0 \leftarrow F_1 \cdots F_\ell$ where form F_i is of sort s_i and for every $k \in [1,\ell]$*

$$\left\{(j,i) \;\middle|\; (k\langle j\rangle, k(i)) \in GLD(P)^k\right\} \subseteq \overline{SI(s_k)}$$

where graph $GLD(P)^k$ is given as the transitive closure of

$$\begin{array}{c} GLD(P) \cup \left\{(0\langle j\rangle, 0(i)) \;\middle|\; (j,i) \in \overline{SI(s_0)}\right\} \\ \cup \left\{(k'(i), k'\langle j\rangle) \;\middle|\; k' \in [1,\ell],\ k' \neq k,\ (i,j) \in \overline{IS(s_{k'})}\right\} \end{array}$$

3. *For every production $P : F_0 \leftarrow F_1 \cdots F_\ell$ where form F_i is of sort s_i*

$$\left\{(i,j) \mid (0(i), 0\langle j\rangle) \in GLD(P)^0\right\} \subseteq \overline{IS(s_0)}$$

where graph $GLD(P)^0$ is given as the transitive closure of

$$GLD(P) \cup \left\{(k(i), k\langle j\rangle) \;\middle|\; k \in [1,\ell],\ (i,j) \in \overline{IS(s_k)}\right\}$$

*The guarded attribute grammar G is said to be **strongly-acyclic** if for every sort s and production P whose left-hand side is a form of sort s the graph $\overline{G(s,P)} = \overline{SI(s)} \cup IS(P)$ is acyclic.* □ *Def. B.2*

PROPOSITION B.3. *A strongly-acyclic GAG is acyclic and hence input-enabled.*

PROOF. The proof is analog to the proof that a strongly non-circular attribute grammar is non-circular and it goes as follows. We let $(i,j) \in IS(s)$ when $\mathrm{var}(d_i\sigma) \cap \mathrm{var}(y_j\sigma) \neq \emptyset$ for some form $F = s(d_1,\ldots,d_n)\langle y_1,\ldots,y_m\rangle$ of sort s and where σ is the substitution induced by a firing sequence starting from configuration $\{X = F\}$. Then we show by induction on the length of the firing sequence leading to the reachable configuration that $IS(s) \subseteq \overline{IS(s)}$ and $SI(s) \subseteq \overline{SI(s)}$. □ *Prop.* B.3

Note that the following two inclusions are strict

strongly-acyclic GAGs $\subsetneq$ acyclic GAGs $\subsetneq$ input-enabled GAGs

Indeed the reader may easily check that the guarded attribute grammar

$$\begin{cases} A(x)\langle z\rangle \leftarrow B(a(x,y))\langle y,z\rangle \\ B(a(x,y))\langle x,y\rangle \leftarrow \end{cases}$$

is cyclic and input-enabled whereas guarded attribute grammar with productions

$$\begin{cases} A(x)\langle z\rangle \leftarrow B(y,x)\langle z,y\rangle \\ A(x)\langle z\rangle \leftarrow B(x,y)\langle y,z\rangle \\ B(x,y)\langle x,y\rangle \leftarrow \end{cases}$$

is acyclic but not strongly-acyclic. Attribute grammars arising from real situations are almost always strongly non-circular so that this assumption is not really restrictive. Similarly we are confident that most of the guarded attribute grammars used in practise are input-enabled (at least it is the case for all the concrete examples we have but those constructed specially for that purpose) and that most of the input-enabled guarded attribute grammars are in fact strongly-acyclic. Thus most of the specifications are distributable and most of those can be proved so by checking the strong non-circularity condition.

B.2 Some Properties of input-enabled GAGs

We call the *substitution induced by a sequence* $\Gamma[*\rangle\Gamma'$ the corresponding composition of the various substitutions associated respectively with each of the individual steps in the sequence. If X is an open node in both Γ and Γ', i.e., no productions are applied to X in the sequence, then we get $X = s(d_1\sigma,\ldots,d_n\sigma)\langle y_1,\ldots,y_m\rangle \in \Gamma'$ where $X = s(d_1,\ldots,d_n)\langle y_1,\ldots,y_m\rangle \in \Gamma$ and σ is the substitution induced by the sequence.

PROPOSITION B.4 (MONOTONY). *Let Γ be a reachable configuration of an input-enabled guarded attribute grammar, $X = s(d_1,\ldots,d_n)\langle y_1,\ldots,y_m\rangle \in \Gamma$ and σ the substitution induced by some sequence starting from Γ. Then $\Gamma[P/X\rangle\Gamma'$ implies $\Gamma\sigma[P/X\rangle\Gamma'\sigma$.*

PROOF. Direct consequence of Definition 2.2 due to the fact that

1. $\mathbf{match}(p, d\sigma) = \mathbf{match}(p,d)\sigma$, and
2. $\mathbf{mgu}(\{y_j = u_j\sigma\}_{1\leq j\leq m}) = \mathbf{mgu}(\{y_j = u_j\}_{1\leq j\leq m})\sigma$.

The former is trivial and the latter follows by induction on the length of the computation of the most general unifier (relation $\Rightarrow^*$ using rule (5) only). Note that the assumption that the guarded attribute grammar is input-enabled is crucial because in the general case it could happen that the set $\{y_j = u_j\sigma_{in}\}_{1\leq j\leq m}$ satisfies the occur check whereas the set $\{y_j = u_j(\sigma_{in}\sigma)\}_{1\leq j\leq m}$ does not satisfy the occur check. □ *Prop.* B.4

Proposition B.4 is instrumental for the distributed implementation of guarded attribute grammars. Namely it states that new information coming from a distant asynchronous location refining the value of some input occurrences of variables of an enabled production do not prevent from applying that production. Thus a production that is locally enabled can freely be applied regardless of information that might further refine the current local configuration. It means that conflict arises only from the existence of two distinct productions enabled in the same open node. Hence the only form of non-determinism corresponds to the decision of a stakeholder to apply one particular production among those enabled in a configuration. This is expressed by the following confluence property.

COROLLARY B.5. *Let Γ be a reachable configuration of an input-enabled GAG. If $\Gamma[P/X\rangle\Gamma_1$ and $\Gamma[Q/Y\rangle\Gamma_2$ with $X \neq Y$ then $\Gamma_2[P/X\rangle\Gamma_3$ and $\Gamma_1[Q/Y\rangle\Gamma_3$ for some configuration Γ_3.*

Note that the artifact contains a full history of the case in the sense that one can reconstruct from the complete artifact all the sequence that corresponds to the resolution of the case (up to the commutation of independent transitions).

We might have considered a more symmetrical presentation in Definition 2.2 by allowing patterns for synthesized attributes in the right-hand sides of productions with the effect of creating forms in a configuration with patterns in their co-arguments. These patterns express constraints on the synthesized values. This extension could be acceptable as long as one sticks to purely centralized models. However, as soon as one wants to distribute the model on an asynchronous architecture, one cannot avoid such a constraint to be further refined due to a transformation occurring in a distant location. Then the monotony property (Proposition B.4) is lost: a locally enabled production can later be disabled when a constraint on a synthesized value gets a refined value. This is why we required synthesized attributes in the right-hand side of a production to be given by plain variables in order to prohibit the expression of constraints on synthesized values.

B.3 Distribution of an input-enabled GAG

The principle of a distribution of a GAG on a set of locations is as follows: Each location maintains a local configuration, and subscribes to results provided by other locations. Productions are applied locally. When variables are given a value by a production, the location that computed this value sends messages to the locations that subscribed to this value. Messages are simply equations defining the value of a particular variable. Upon reception of a message, a subscriber updates its local configuration, and may in turn produce new messages.

More formally, a GAG can be distributed by specifying a partition $S = \uplus_{1\leq \ell\leq p} S_\ell$ of the set of sorts. The projections Γ_ℓ, called the local configurations associated with sites S_ℓ, are defined as follows. Each site S_ℓ has a namespace $ns(S_\ell)$ used for the nodes X whose sorts are in S_ℓ and for the variables x representing attributes of these nodes but also for

references to variables belonging to distant sites (subscriptions). Hence we have name generators that produce unique identifiers for each newly created variable for each site. For each equation $X = P(X_1, \ldots, X_n)$ with $X :: s$ and $X_i :: s_i$ we insert equation $X = P(\overline{X}_1, \ldots, \overline{X}_n)$ in Γ_ℓ where $s \in S_\ell$ and variable $\overline{X}_i$ is X_i if $s_i \in S_\ell$ or is a new variable in the namespace of S_ℓ if $s_i \in S_{\ell'}$ with $\ell' \neq \ell$. In the latter case we add equation $\overline{X}_i = X_i$ in Γ_ℓ. Similarly for each equation $X = s(t_1, \ldots, t_n)\langle y_1, \ldots, y_m\rangle$ in Γ we add equation $X = s(\bar{t}_1, \ldots, \bar{t}_n)\langle \overline{y}_1, \ldots, \overline{y}_m\rangle$ in Γ_ℓ where $s \in S_\ell$ and $\bar{t}$ is obtained by replacing each variable x in term t by $\overline{x}$ where variable $\overline{x}$ is x if $x :: s'$ with $s' \in S_\ell$ else is a new variable in the namespace of S_ℓ. In the latter case one adds equation $\overline{x} = x$, called a **subscription**, to $\Gamma_{\ell'}$. Similarly for the variables y_j. Hence a local configuration contains the usual equations associated with their closed and open nodes (and containing only local variables) together with equations of the form $X = Y$ and $y = x$ where x and X are local names and y and Y belongs to distant sites. Clearly the global configuration can be recovered as $\Gamma = \Gamma_1 \oplus \cdots \oplus \Gamma_n$ where operator $\oplus$ consists in taking the union of the systems of equations given as arguments and simplifying the resulting system by elimination of the copy rules: we drop each equation of the form $X = Y$ (respectively $y = x$) and replace each occurrence of X by Y (resp. of y by x). Therefore the global configuration Γ may be identified with the vectors of local configurations $(\Gamma_1, \ldots, \Gamma_p)$.

Each production can then be locally applied: we write $\Gamma_\ell \xrightarrow[M]{P/X} \Gamma'_\ell$ when application of production P at node X results in a new configuration Γ'_ℓ and the sending of a set of messages M. More formally, $\Gamma_\ell \xrightarrow[M]{P/X} \Gamma'_\ell$ when

$$X = s(t_1, \ldots, t_n)\langle y_1, \ldots, y_m\rangle \in \Gamma_\ell$$

and $P = F \leftarrow F_1 \cdots F_k$ is a production whose left-hand side matches with X and

$$\begin{array}{rcl}
\Gamma'_\ell & = & \{X = P(X_1, \ldots, X_k)\} \\
& \cup & \{X_i = F_i\sigma \mid X_i :: s_i \text{ and } s_i \in S_\ell\} \\
& \cup & \{X' = F\sigma \mid (X' = F) \in \Gamma_\ell \wedge X' \neq X\} \\
& \cup & \{y' = y_j\sigma \mid (y' = y_j) \in \Gamma_\ell \text{ and } y_j\sigma \text{ is a variable}\} \\
M & = & \{X_i = F_i\sigma \mid X_i :: s_i \text{ and } s_i \notin S_\ell\} \\
& \cup & \{y' = y_j\sigma \mid (y' = y_j) \in \Gamma_\ell \text{ and } y_j\sigma \text{ not a variable}\}
\end{array}$$

where $X_1, \ldots, X_k$ are new names in $ns(S_\ell)$ and $\sigma = \mathbf{match}(F, X)$. This relation means that applying production P in X in site S_ℓ generates messages M send to distant sites. The reception of a message may generate new messages and is described by relation $\Gamma_\ell \xrightarrow[M]{m} \Gamma'_\ell$ where

1. If $m = \{X = s(t_1, \ldots, t_n)\langle y_1, \ldots, y_q\rangle\}$ with $X \in ns(S_{\ell'})$, $s \in S_\ell$ with $\ell' \neq \ell$ then

   $$\Gamma'_\ell = \Gamma_\ell \cup \{\overline{X} = s(\bar{t}_1, \ldots, \bar{t}_n)\langle \overline{y}_1, \ldots, \overline{y}_q\rangle\} \cup \{y_j = \overline{y_j} \mid 1 \leq j \leq q\}$$

   where $\overline{X}$, the variables $\overline{x}$ for $x \in var(t_i)$ and the variables $\overline{y}_j$ are new names in $ns(S_\ell)$ and $\bar{t} = t[\overline{x}/x]$, and

   $$M = \{\overline{x} = x \mid x \in var(t_i) \;\; 1 \leq i \leq n\} \cup \{X = \overline{X}\}$$

2. If $m = \{x = t\}$ with $x \in ns(S_\ell)$ then $\Gamma'_\ell = \Gamma_\ell[x = t[\overline{y}/y]]$ where $\overline{y}$ are new names in $ns(S_\ell)$ associated with the variables y in t and $M = \{\overline{y} = y \mid y \in var(t)\}$.

3. If $m = (X = Y)$ with $X \in ns(S_\ell)$ then $\Gamma'_\ell = \Gamma_\ell \cup \{X = Y\}$ and $M = \emptyset$.

4. If $m = (y = x)$ with $x \in ns(S_\ell)$ then $\Gamma'_\ell = \Gamma_\ell \cup \{y = x\}$ and $M = \emptyset$.

The global dynamics of the system can then be derived as follows, where e stands for P/X or a message m:

1. If $\Gamma_\ell \xrightarrow[M]{e} \Gamma'_\ell$ then $\Gamma \xRightarrow[M]{e} \Gamma'$ with $\Gamma_{\ell'} = \Gamma'_{\ell'}$ for $\ell' \neq \ell$.
2. If $\Gamma \xRightarrow[M]{e} \Gamma'$ and $\Gamma' \xRightarrow[M']{m} \Gamma''$ for $m \in M$ then $\Gamma \xRightarrow[M \setminus \{m\} \cup M']{e} \Gamma''$

Input-enabled GAGs possess useful properties with respect to distribution, namely messages consumptions and application of productions commute, as shown in the following proposition:

PROPOSITION B.6. *For an input-enabled guarded attribute grammar:*

1. *If* $\Gamma \xRightarrow[M]{P/X} \Gamma'$ *then there exists a substitution* σ_M *such that* $\Gamma \xRightarrow[\emptyset]{P/X} \Gamma'\sigma_M$.
2. $\Gamma[P/X\rangle\Gamma'$ *if and only if* $\Gamma \xRightarrow[\emptyset]{P/X} \Gamma'$
3. *Let* $\Gamma \xRightarrow[M_1]{P_1/X_1} \Gamma_1$ *and* $\Gamma \xRightarrow[M_2]{P_2/X_2} \Gamma_2$ *with* $X_1 \neq X_2$. *One can assume w.l.o.g that* M_1 *and* M_2 *have no common variables (the name generator chooses different names for the new variables in both cases). Then the diagram in Fig. 1, where* $\rightsquigarrow$ *denotes messages consumption, commutes.*

Intuitively, proposition B.6 and in particular (3) mean that distribution does not affect the global behavious of an input-enabled GAG.

PROOF. We first prove (1): whenever $\Gamma \xRightarrow[M]{P/X} \Gamma'$ then there exists a substitution σ_M such that $\Gamma \xRightarrow[\emptyset]{P/X} \Gamma'\sigma_M$.

Let us assume that $\Gamma \xRightarrow[M]{P/X} \Gamma'$, and examine how consuming messages in $M = \{m_1, \ldots, m_q\}$ affects Γ'. Messages can be of several kinds :

- if $m_i = (y = x)$ (or $m_i = (X = Y)$) then consuming the message results in adding an equation $\sigma_{m_i} = \{y = x\}$ (resp. $\sigma_{m_i} = \{X = Y\}$) to the local configuration that receives this message, and generates no new message.
- if $m_i = \{x = t\}$, then consumption of the message results in production of new variables, and a new (finite) set of messages M_i that are all of the form $\{\bar{y} = y\}$ and can then be consumed without producing new messages by the location that has subscribed to this value. We can denote by σ_i the substitution that replaces every x in the local configuration that receives m_i.
- if m_i is of the form $\{X = s(t_1, \ldots, t_n) < y_1, \ldots, y_n >\}$, then consuming m_i results in adding new equations to the local configuration that receives it, and generating a set of messages $M_i = \{m_{i,1}, m_{i,q}\}$, that are of the form $\{\bar{y} = y\}$ and $\{X = \bar{X}\}$ and can hence be consumed by the location that will receive them without generating new messages.

These observations show that, after application of a production, message consumption is a finite process. We have $\Gamma \xRightarrow[M]{P/X} \Gamma' \xRightarrow[M_1]{m_1} \Gamma_1 \xRightarrow[\emptyset]{M_1} \Gamma'_1 \ldots \xRightarrow[M_{|M|}]{m_{|M|}} \Gamma_{|M|} \xRightarrow[\emptyset]{M_{|M|}} \Gamma'_{|M|}$

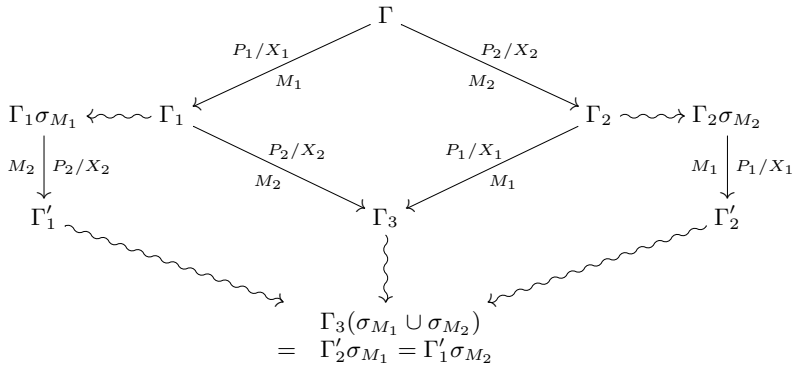


Figure 1: Confluence property

Now, the difference between each Γ_i and Γ'_i is a set of subscriptions, that are appended to some local configurations, and erased during the step. Therefore, the global configurations Γ_i and Γ'_i are identical. Similarly, we have $\Gamma_1 = \Gamma'\sigma_{m_1}$, and $\Gamma_i = \Gamma'_{i-1}\sigma_{m_i}$ for every $i \in [2, |M|]$. Hence, the substitution $\sigma_M = \sigma_{m_1} \dots \sigma_{|M|}$ is such that $\Gamma'\sigma_M = \Gamma'_{|M|}$. Hence $\Gamma \xRightarrow[\emptyset]{P/X} \Gamma'\sigma_M$.

We now give a proof for (2). We have to establish the following equivalence: $\Gamma[P/X\rangle\Gamma'$ if and only if $\Gamma \xRightarrow[\emptyset]{P/X} \Gamma'$. First, whenever $\Gamma \xRightarrow[\emptyset]{P/X} \Gamma'$, then, by definition, $\Gamma[P/X\rangle\Gamma'$. Conversely, $\Gamma[P/X\rangle\Gamma'$ implies the existence of ℓ and M such that $\Gamma_\ell \xRightarrow[M]{P/X} \Gamma^1_\ell$, thus, by definition $\Gamma \xRightarrow[M]{P/X} \Gamma^1$ (with $\Gamma^1_{\ell'} = \Gamma_{\ell'}$ for $\ell' \neq \ell$). Hence, by (1) we have $\Gamma \xRightarrow[\emptyset]{P/X} \Gamma^1\sigma_M$. Note that productions applications are deteministic, and messages consumption too. Hence, it suffices to prove $\Gamma' := \Gamma^1\sigma_M$ to obtain the desired result. In fact the nodes replacement performed to obtain Γ' and Γ^1 are identical, since the same production is applied to the same node. Let us denote by $\Gamma_{[}P/X]$ the configuration obtained by replacement of X. We have to show the equality $\Gamma' = \Gamma_{[}P/X]\sigma = \Gamma_{[}P/X]\sigma_l\sigma_M$, where σ is the usual substitution applied during production application, σ_l is the substitution resulting from applying production locally to Γ_l, and σ_M is the substitution obtained by consumption of messages in M. Now, one can notice that all substitutions in σ_M replace a variable y by a term t whenever y is a subscription to some value produced in Γ_l. The effect is exactly the same as applying σ to nodes that differ from X in Γ. As additional subscription generated by messages consumption is not considered in the product, we have $\Gamma' = \Gamma_{[}P/X]\sigma = \Gamma_{[}P/X]\sigma_l\sigma_M$.

The last statement, (3), expresses the Confluence property (Fig. 1). We first consider the commutativity of the center:

$$\Gamma \xRightarrow[M_1]{P_1/X_1} \Gamma_1 \xRightarrow[M_2]{P_2/X_2} \Gamma_3 \text{ commutes into } \Gamma \xRightarrow[M_2]{P_2/X_2} \Gamma_2 \xRightarrow[M_1]{P_1/X_1} \Gamma_3.$$

This follows directly from the properties of input-enabled grammars: Γ_3 is simply Γ where both open nodes X_1 and X_2 have been replaced respectively by the closed nodes $X_1 = P_1(Y^1_1, \dots, Y^1_m)$ and $X_2 = P_2(Y^2_1, \dots, Y^2_k)$ (since M_1 and M_2 have no common variables they are unaffected by each other).

Let us consider the left hand-side of the diagram. From (1), we have that $\Gamma \xRightarrow[M_1]{P_1/X_1} \Gamma_1 \rightsquigarrow \Gamma_1\sigma_{M_1}$. And by (2), this implies that $\Gamma[P_1/X_1\rangle\Gamma_1\sigma_{M_1}$. Using Proposition B.4, and the fact that $\Gamma_1[P_2/X_2\rangle\Gamma_1$, whe have that P_2 is triggered and enabled in $\Gamma_1\sigma_{M_1}$. Hence, $\Gamma_1\sigma_{M_1} \xRightarrow[M_2]{P_2/X_2} \Gamma'_1$. Using again Proposition B.4, configuration Γ'_1 is simply $\Gamma_3\sigma_{M_1}$. Furthermore, from (1), we have $\Gamma_1\sigma_{M_1} \xrightarrow[M_2]{P_2/X_2} \Gamma'_1 \rightsquigarrow \Gamma'_1\sigma_{M_2}$. From $\Gamma'_1 = \Gamma_3\sigma_{M_1}$ it follows $\Gamma'_1\sigma_{M_2} = \Gamma_3\sigma_{M_1}\sigma_{M_2}$. By a symmetric argument, we obtain: $\Gamma'_2\sigma_{M_1} = \Gamma_3\sigma_{M_2}\sigma_{M_1}$. Since these substitutions have disjoint support, we have

$$\sigma_{M_1}\sigma_{M_2} = \sigma_{M_2}\sigma_{M_1} = \sigma_{M_1} \cup \sigma_{M_2}$$

Thus, $\Gamma_3\sigma_{M_1}\sigma_{M_2} = \Gamma_3\sigma_{M_2}\sigma_{M_1} = \Gamma_3\sigma_{M_1} \cup \sigma_{M_2}$.

Now, the last arrow: $\Gamma_3 \rightsquigarrow \Gamma_3(\sigma_{M_1} \cup \sigma_{M_2})$ this is obvious from the definitions of σ_{M_1} and σ_{M_2}.

□ *Prop.* B.6

C. SOUNDNESS

A specification is sound if every case can reach completion no matter how its execution started. A case is a service call in the interface of the GAG (Definition 3.5) which already contains all the information coming from the environment of the guarded attribute grammar.

DEFINITION 4.1 *Given a guarded attribute grammar with its interface, a* ***case*** $c = s(t_1, \dots, t_n)\langle x_1, \dots, x_m\rangle$ *is an element of the interface such that* $\text{var}(t_i) \subseteq \{x_1, \dots, x_m\}$. *Stated otherwise a case is, but for the variables with a synthesized value, a closed instance of a service.* □ *Def. 4.1*

DEFINITION 4.2 *A configuration is* ***closed*** *if it contains only closed nodes. A guarded attribute grammar is* ***sound*** *if a closed configuration is reachable from any configuration* Γ *reachable from the initial configuration* $\Gamma_0(c) = \{X_0 = c\}$ *associated with a case c.* □ *Def. 4.2*

Let γ denote the set of configurations reachable from the initial configuration of some case. We consider the finite sequences $(\Gamma_i)_{0<i\leq n}$ and the infinite sequences $(\Gamma_i)_{0<i}$ of configurations in γ such that $\Gamma_i[\rangle\Gamma_{i+1}$. A finite and maximal sequence is said to be **terminal**, i.e., a terminal sequence

leads to a configuration that enables no production. Soundness can the be rephrased by the two following conditions.

1. Every terminal sequence leads to a closed configuration.
2. Every configuration on an infinite sequence also belongs to some terminal sequence.

We now turn to the proof of Theorem 4.3:

THEOREM 4.3 *Both the soundness problem (is a given GAG sound) and the reachability problem (is a given configuration reachable from another one, for a given GAG) are undecidable for guarded attribute grammars.*

We first prove the undecidability of soundness.

PROPOSITION C.1. *Soundness of guarded attribute grammar is undecidable.*

PROOF. We consider the following presentation of the Minsky machines. We have two registers r_1 and r_2 holding integer values. Integers are encoded with the constant **zero** and the unary operator **succ**. The machine is given by a finite list of instructions $instr_i$ for $i = 1, \ldots, N$ of one of the three following forms

1. **INC(r,i):** increment register r and go to instruction i.
2. **JZDEC(r,i,j):** if the value of register r is 0 then go to instruction i else decrement the value of the register and go to j.
3. **HALT:** terminate.

We associate such a machine with a guarded attribute grammar whose sorts corresponds bijectively to the lines of the program, (i.e., $S = \{s_1, \ldots, s_N\}$) with the following encoding of the program instructions by productions:

1. If $instr_k = \mathrm{INC}(r_1, i)$ then add production
$$\mathrm{Inc}(k, 1, i) : s_k(x, y) \leftarrow s_i(\mathbf{succ}(x), y)$$
2. If $instr_k = \mathrm{INC}(r_2, i)$ then add production
$$\mathrm{Inc}(k, 2, i) : s_k(x, y) \leftarrow s_i(x, \mathbf{succ}(y))$$
3. If $instr_k = \mathrm{JZDEC}(r_1, i, j)$ then add the productions
$$\begin{array}{ll} \mathrm{Jz}(k, 1, i) & : s_k(\mathbf{zero}, y) \leftarrow s_i(\mathbf{zero}, y) \\ \mathrm{Dec}(k, 1, j) & : s_k(\mathbf{succ}(x), y) \leftarrow s_j(x, y) \end{array}$$
4. If $instr_k = \mathrm{JZDEC}(r_2, i, j)$ then add the productions
$$\begin{array}{ll} \mathrm{Jz}(k, 2, i) & : s_k(x, \mathbf{zero}) \leftarrow s_i(x, \mathbf{zero}) \\ \mathrm{Dec}(k, 2, j) & : s_k(x, \mathbf{succ}(y)) \leftarrow s_j(x, y) \end{array}$$
5. If $instr_k = \mathrm{HALT}$ then add production
$$\mathrm{Halt}(k) : s_k(x, y) \leftarrow$$

Since there is a unique maximal firing sequence from the initial configuration $\Gamma_0 = \{X_0 = s_1(\mathbf{zero}, \mathbf{zero})\}$ the corresponding guarded attribute grammar is sound if and only if the computation of the corresponding Minsky machine terminates. □ *Prop.* 4.3

Adapting the proof of Proposition C.1 to the reachability problem completes the proof of Theorem 4.3.