ISSN 0254–2757

QI QUÆSTIONES INFORMATICÆ

Volume 6 • Number 1

May 1988

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereeniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Production

Mr Q H Gee
Department of Computer Science
University of the Witwatersrand
Johannesburg
Wits
2050

Subscriptions

The annual subscription is

	SA	US	UK
Individuals	R20	$ 7	£ 5
Institutions	R30	$14	£10

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial

Volume six of QI heralds several changes. The most visible is the change in format. The black on red cover has been changed to a more readable blue on white, but we have retained the style of the old cover, for the sake of continuity. The papers are now set in a tighter format, using double columns, which will enable more papers to be published for the same cost.

For authors, the most significant change is that as from Volume 6 Number 2 (the next issue), a charge will be made for typesetting. The charge is quite modest – R20 per page – and will enable us to keep up the high standards that we have become used to with QI. It is worth recording that the alternative to this suggestion was that authors should present camera-ready typescript, as is done for *Quæstiones Mathematicæ*. Given that document preparation and electronic typesetting is one of the areas of computer science that we can feel proud of, it seemed right that our journal should use the most modern techniques available. Fortunately, the two controlling bodies, the CSSA and SAICS, eventually agreed to our proposal and the result is the professional journal you have in front of you now.

Supporters of QI may be interested in a few statistics that I compiled when I took over the editorship from Gerrit Wiechers in April this year. In the past two years (June 1985 to June 1988), 73 papers have been received. Of these 39 (53%) have appeared, 19 have been rejected or withdrawn (26%) and 15 (21%) are either with authors for changes or with referees. If we look at the complete picture for Volumes 4 and 5, we find the following:

Volume	Issues	Papers	Pages	Ave. pages per paper
5	3	27*	220	7.7
4	3	21	136	6.4

Although this issue contains one very long paper of 18 pages, the future policy of QI will be to restrict papers to 6 or 7 printed pages, and prospective authors are asked to bear this in mind when submitting papers.

For the future, we are hoping to move towards more special issues. Many of the papers being published at the moment were presented at the 4th SA Computer Symposium in 1987. Instead of continuing the policy of allowing such papers to be accepted by QI without further refereeing, we are hoping to negotiate with Conference organisers to produce special issues of QI. Thus the proceedings would *ab initio* be typeset by QI and all the papers would be in a single issue. Given the competitive charges of QI, there will be financial gains for both parties in such an arrangement.

As this is my first editorial, it is fitting that it should close with a tribute to the previous QI team. My predecessor as editor was Gerrit Wiechers. Gerrit took over the editorship in 1980 and served the journal well over the years. With his leadership, the number and quality of the papers increased to its present healthy state. I must also extend a big thank you to Conrad Mueller and the University of the Witwatersrand who pioneered desk top publishing of QI in August 1985, using the IBM mainframe and its laser writer. Without Conrad's diligence and the excellent facilities provided by the Wits Computer Centre and subsequently the Computer Science Department, QI would easily have degenerated into a second-rate magazine. Quintin Gee, also of the Wits Computer Science Department, has taken over from Conrad and has raised the production quality of QI to new heights, as this issue testifies.

I look forward to your help and support in the future. Long live QI!

Judy M Bishop
Editor
June 1988

Using NLC-Grammars to Formalise the Take/Grant and Send/Receive Security Models

D P de Villiers and S H von Solms
Department of Computer Science, Rand Afrikaans University, P O Box 524, Johannesburg, 2000

Abstract

In this paper the Send/Receive and Take/Grant logical security models are formalised using results from formal language theory. By using the graph rewriting facilities of NLC-grammars, and by extending these facilities to take different types of context conditions into account, the actions within the Send/Receive and Take/Grant models are simulated.
Keywords: *Graph grammars, Node-Label-Controlled graph grammars, Grammatical protection systems, Take/Grant models, Send/Receive models*

Received March 1987, Accepted July 1987

1. Introduction

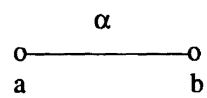
The goal of this study is the definition of a context-dependent logical security model. This model will use the current state of the protected environment to determine whether a new access relation between two entities in the environment may be established, or whether such an existing relationship may be altered. Furthermore, when new entities are introduced into the environment, the current protection state will also be used in determining the relationship between the new and existing entities. A motivation for this is to formalise the definition of the access rights of new users in a protected system. Instead of having to manually determine the access rights of each user, the protection system will define the access rights in view of the current system context. This study will provide the basis for a logical formulation of such a model.

Firstly, current models of access control were studied, to find a suitable framework for development of the envisaged model. It was found that the T/G (Take/Grant) and S/R (Send/Receive) models [4,3] showed promise, for the following reasons

- it is possible to ignore the subject/object distinction.
- the graph-theoretic approach and related graph-rewriting rules pointed to the possible incorporation of a suitable graph grammar, that would make a large body of proven theory available for use in the study.
- the T/G and S/R models are enjoying considerable attention. New results, that appear quite frequently, may lead to new ideas in this study.

We will now give short summaries, first of the T/G model (from [4]), and then of the S/R model (from [3]).

The object of the T/G model is to model a protection system. This is done by using a two-coloured, directed graph, wherein subjects and objects are represented by different coloured nodes of the graph, and where the capability of node a to access node b is indicated by an edge from a to b. The set α of rights that a can exercise in accessing b (such as read, write) is denoted by labelling the edge from a to b with α, as follows:



The set α of rights is a subset of the fixed and finite set of rights R = {*read*, *write*, *append*, *execute* (programs), *take*, *grant*}. The *read*, *write*, *append* and *execute* rights are called the inert rights, while the *take* and *grant* rights may be seen as "active" rights, in that they allow the modification of the protection graph structure.

The T/G model also defines certain graph rewriting rules, two of which correspond to the *take* and *grant* rights. Informally, if node a has the *take* right to node b, and the *grant* right to node c, node a may take any subset of the rights of node b for itself, and give (grant) any subset of its own rights to node c.

There are two more rewriting rules in this formulation of the T/G model, namely the create and remove rules. The create rule adds a new node to the graph, with a labelled edge from the node that initiated the creation operation to the newly created node. The remove rule alters the labelling of an edge of the graph (i.e. an alteration of the access rights of the start node of the edge) by removing a subset of rights from the set of rights denoted by the edge label.

The T/G operations will later be more formally defined.

Like the Take/Grant (T/G) model, the purpose of the Send/Receive model is the modelling of the transfer of access rights between subjects in a protected system.

In the S/R model for protection, several shortcomings of the Take/Grant model are addressed, including *selectivity*, *locality/modularity*, and *unidirectionality* of flow of rights (see [3] for more detail).

We shall now give a brief definition of the S/R model, from [3]. In this model, a protected system consists of typed objects. With each object **x** we associate a set of (*attribute*, *value*) tuples, describing the attributes of the object, as well as a (possibly empty) set of (*object*, *access right set*) tuples, called *tickets*, which denotes the fact that **x** can access the given *object* with any of the rights in the set *access right set*. The *access right set* is a subset of a fixed and finite set of rights R = {*read*, *write*, *append*, *execute* (programs), *send*, *receive*}. Finally, an object also possesses a (potentially empty) set of *rule rights*, called *activators/rules*. An activator takes the following (simplified) form:

CAN A(a_1, ..., a_k) IF Q(a_1, ..., a_k).

This means that the holder of the activator may perform action A on the arguments (logical entities/objects) a_1, ..., a_k, provided that the predicate Q is satisfied.

In the rest of this paper, we shall refer to activators as rules.

Three basic rules are defined in the S/R model.

The first rule, called the CAN-SEND rule, facilitates the control of movement of rights out of an object's domain, enabling the holder thereof to send rights to another object. The rule takes the following form:

CAN SEND *p*:*P* TO *s*:*S* IF Q(*p*, *s*).

p and *s* are free variables and are only present if they are used within the predicate Q. *P* and *S* are tuples, called *templates*, of the form (*T*, *R*), where (for *P*) *T* represents the type of the object *p*, and *R* represents the set of rights of *p* that may be sent to the object *s*. These templates (called *patterns* in [3]) are matched by a ticket (*object*, *access right set*) when the *object* is of type *T* and the *access right set* is contained in the set *R*. *S* will always have the form (*T*, *send*). This type of rule may be activated only if its holder possesses tickets that match *P* and *S*, and Q(*p*, *s*) is satisfied.

The second rule enforces control over the movement of rights into an object's domain, enabling its holder to receive rights from another object. The rule takes the following form:

CAN RECEIVE *p*:(*T*, *R*) FROM *s*:(*T*, *receive*) IF Q(*p*, *s*).

This rule may be activated only if its holder possesses the *receive* right to the object *s* from which it wants to receive a ticket of the form (*T*, *R*), and if the predicate Q is satisfied.

Finally, there is a CREATE operation for the creation of new objects. It is assumed that if an object **x** wants to create a new object **y**, **x** receives the tickets (**y**, *send*), (**y**, *receive*), and **y** receives the tickets (**x**, *send*), (**x**, *receive*), so that full bidirectional transfer of rights is possible. The form of the CAN-CREATE rule is not defined in [3], and for the purposes of this article we define it as follows:

CAN CREATE *p*:(*T*, $\mathcal{R}$) WHEN (*T*, *create*) IF Q(*p*, *s*).

This means that the holder of the rule can create an object *p* of type *T* subject to the condition that the predicate Q is satisfied. The set $\mathcal{R}$ is a subset of the set of rights possessed by the holder of the CAN-CREATE rule.

Finally, in order for a ticket (*t*, *r*) to be transferred from an object **x** to an object **y**, the following must be true:

- **x** must possess the tickets (*t*, *r*) and (**y**, *send*), as well as an appropriate CAN-SEND rule, that is satisfied by these tickets.
- **y** must possess the ticket (**x**, *receive*), as well as an appropriate CAN-RECEIVE rule, that is satisfied by the tickets (**x**, *receive*) and (*t*, *r*).

As was said earlier, the incorporation of a formal graph grammar is indicated by the graph-theoretic approach and rewriting rules of the T/G model. Of existing graph grammars, the so-called NLC-(Node-Label Controlled) grammars [1] seem to be the most applicable. An important motivation is that the grammars were extended by von Solms [6,7,8] to include permitting and forbidding node contexts in their productions. As context dependency is a fundamental aspect of the grammar to be used, the grammar was found to be very acceptable in this regard.

We shall now proceed with the necessary definitions for simulating the T/G model, and then discuss some of the aspects of the resulting model, which we will call the CSM (Context-dependent Security Model). This model is fully described in [10].

2. Extending NLC-grammars to Formalise the T/G Model

For the definition of a node-labelled undirected graph, an NLC grammar and related concepts, see [1].

Definition 2.1 (from [1])

A NLC-grammar with node context is a system

G = (Σ, Δ, P, C, Z) where

Σ is a nonempty finite set of labels, called the total alphabet

Δ is a nonempty finite subset of Σ, called the terminal alphabet

P is a finite set of productions, of the form (d, D, (Σ_1 ; Σ_2)) with

$d \in \Sigma$

D a graph over Σ

$\Sigma_1, \Sigma_2 \subseteq \Sigma$

$\Sigma_1 \cap \Sigma_2 = \emptyset$, where

Σ_1 is called the permitting- and Σ_2 the forbidding (node) context

C is a subset of $\Sigma \times \Sigma$, called the connection relation

Z is a graph over Σ, called the axiom.

Definition 2.2

Let R be a finite, nonempty set of symbols, called the set of rights. Let H be a graph over Σ, and let e = l_1 —— l_2 be a subgraph of H, consisting of any two connected nodes of H and the edge between them. We associate with the edge of e an $\alpha \subseteq R$, called the edge classification. We denote this by a tuple (l_1, l_2, α), called an edge context. The edge context is denoted graphically by adding the labels l_1,α to the edge, as follows:

l_1,α

l_1 —— l_2

The edge context (l_2, l_1, β) is written

l_2,β

l_2 —— l_1

We say that l_1 owns the set α of rights with respect to l_2. The "owner" of the rights is therefore indicated by being the first element of the tuple.

The set Λ_H of all possible edge contexts of a graph H is defined as

$\Lambda_H = \{(l_i, l_j, \alpha) \mid l_i$ —— l_j is an edge of H, $i \neq j, \alpha \subseteq R\}$,

where R is the set of rights associated with the grammar that generated H.

For a grammar G, Λ_G is defined as

$\Lambda_G = \{(l_i, l_j, \alpha) \mid l_i, l_j, \in \Sigma, l_i \neq l_j, \alpha \subseteq R\}$,

where R is the set of rights associated with G.

Definition 2.3

A NLC-grammar with node- and edge context is a system

G = $(\Sigma, \Delta, P, C, Z, R)$ where

Σ is a nonempty finite set of labels, called the total alphabet

Δ is a nonempty finite subset of Σ, called the terminal alphabet

P is a finite set of productions, of the form

$(d, D, (\Sigma_1 ; \Sigma_2), (\Lambda_1 ; \Lambda_2))$ with

$d \in \Sigma$

D a graph over Σ, where with every edge of D we associate an edge classification $\alpha \in R$, applicable to one of the nodes of the edge.

$\Sigma_1, \Sigma_2 \subseteq \Sigma$

$\Sigma_1 \cap \Sigma_2 = \varnothing$

Σ_1 is called the permitting and Σ_2 the forbidding node context

$\Lambda_1, \Lambda_2 \subseteq \Lambda_G$. Λ_1 and Λ_2 is called the permitting and forbidding edge contexts, respectively. Also, $\Lambda_1 \cap \Lambda_2 = \varnothing$

C is a subset of $\Sigma \times \Sigma$, called the connection relation

Z is a graph over Σ, called the axiom

R is a finite, nonempty set of symbols, called the set of rights.

The meaning of the permitting and forbidding node/edge contexts in definition 2.3 is as follows: A production may be applied if

- all of the nodes specified in the permitting node context Σ_1 appear somewhere in the current graph
- none of the nodes specified in the forbidding node context Σ_2 appears anywhere in the current graph
- for every edge context (l_1, l_2, α) that appears in the set Λ_1 of permitting edge contexts, there exists an edge somewhere in the current graph between two nodes labelled l_1 and l_2, having an edge classification α.
- for every edge context (l_1, l_2, β) that appears in the set Λ_2 of forbidding edge contexts, there does not exist an edge anywhere in the current graph between two nodes labelled l_3 and l_4, having an edge classification β.

Three types of productions can be identified:

1. Normal productions
2. Edge generation productions
3. Edge removal productions.

A direct derivation step is performed by applying a normal production. Let H be a graph over Σ, with $v \in V_H$, and $\Psi_H(v) = l$. Choose a production $(l, D, (\Sigma_1 ; \Sigma_2), (\Lambda_1 ; \Lambda_2)) \in P_H$, the contexts of which are all satisfied. Apply the production as follows:

(i) Remove v and all edges incident to v from H, resulting in the graph H\v.

(ii) Replace v with an isomorphic copy $\bar{D}$ of D.

(iii) Establish edges between H\v and $\bar{D}$ in the following manner:

Let $x \in V_{\bar{D}}$, $y \in V_{H\setminus v}$, with $\Psi_{\bar{D}}(x) = l_1$ and $\Psi_H(y) = l_2$.

Create an edge between x and y iff there was an edge between y and v in H and $(l_1, l_2) \in C$.

In order to prevent unauthorised access from taking place, connection of nodes in H\v to nodes in D may be forced to be done explicitly. In such a situation, step (iii) in the application of a normal production, as defined above, would be replaced by

(iii) Establish edges between H\v and v in the following manner:

Let $x \in V_{H\setminus v}$.

Create an edge between x and v iff there was an edge between x and v in H.

The set of connection relations, C, would therefore not be used at this stage in the determination of connections between nodes in H\v and nodes in D. After (the modified) step (iii) has been executed, any

node in H\v wishing to gain access to a node in D would explicitly initiate an edge creation production to create an edge to the node in D. In this way all access to new nodes can be simply but rigorously controlled.

The different types of productions will now be discussed in greater detail.

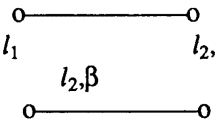
1. A normal production
is of the form

$(l, D, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$

where D is an arbitrary graph over the set Σ. Let $\bar{D}$ be the isomorphic copy of D that is to replace v; then $V_{\bar{D}} \cap V_{H\setminus v} = \emptyset$. This type of production is denoted by P_n.

This type of production is used where at least one new node is to be introduced into the graph (created). The graph D may consist of one or more nodes, and may contain isolated (unconnected) nodes. A normal production cannot establish edges between existing nodes in the graph; the edge generation production is to be used in such a case.

2. An edge generation production
is of the form



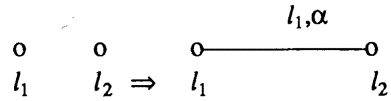
with l_1 $l_2 \in \Lambda_2, \forall \beta \in R, l_2 \in \Sigma_1$, and is denoted by P_g.

It is assumed that identical labels in the above production refer to the same nodes.

The function of this type of production is the generation of an edge between two existing nodes labelled l_1 and l_2. This does not exclude the possibility that an edge between the nodes l_1 and l_2 already exists, although we shall see later that it is possible to establish a precondition that no edge may exist between two nodes at the time that a new edge is added between the two nodes.

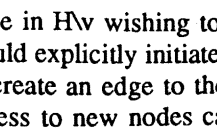
The creation of the edge is initiated by the node labelled l_1 in order to gain access to the node labelled l_2. The owner of the edge (l_1) gives the *take*, *grant*, as well as all other "inert" (*read*, *write*, *append*, *execute*) rights to the edge classification.

The edge generation production can be graphically illustrated as follows:



where the nodes labelled l_1 and l_2 are existing nodes in an arbitrary graph.

3. An edge removal production
is of the form

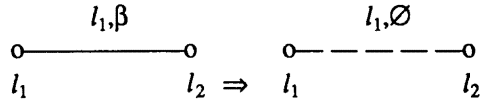


with $(l_1,l_2,\beta) \in \Lambda_1, \beta \in R$, and is denoted by P_r.

It is assumed that identical labels in the above production refer to the same nodes.

The function of the edge removal production is to remove an existing edge from the graph, in order to indicate that no relation between the two nodes exists anymore.

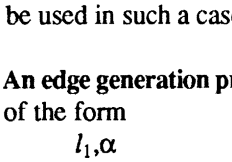
The edge removal production can be graphically illustrated as follows:



where the nodes labelled l_1 and l_2 are existing and connected nodes in an arbitrary graph.

A fourth production type may also be introduced.

4. Edge classification update production
This type of production is of the form

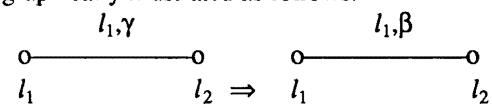


$(l_1,l_2,\gamma) \in \Lambda_1$, and is denoted by P_u.

It is assumed that identical labels in the above production refer to the same nodes.

The function of this type of production is to reflect a change in the relationship between two nodes by altering the edge classification, while leaving the nodes and the edge between them intact. When the edge classification becomes empty, it is indicated that no relationship between the two nodes currently exists, although the edge between the nodes is still shown.

The edge classification update production can be graphically illustrated as follows:



where the nodes labelled l_1 and l_2 are existing (connected) nodes in an arbitrary graph.

Another way to handle the removal of edges is to use the edge classification update production to modify the classification of an edge to be the empty set. In this case, an empty edge classification will denote the situation in which no relation is defined between two nodes, although an edge is shown between them.

Although it may seem that the edge removal production, in contrast to the edge classification update production, would make it possible to distinguish between an isolated node and a node that is connected but with no defined access relations, there is no practical difference between these two situations.

Another remark that is relevant to productions, is that in the NLC-grammars, the sets $V_{H\setminus v}$ and $V_{\bar{D}}$

must be disjoint, i.e. the only node of the graph (to which the production will be applied) that may appear in the right-hand side of the production is the node that appears on the left-hand side of the production, i.e. the node to be replaced. This condition is removed from the CSM, so that the necessary T/G operations may be naturally modelled. It is possible, however, to replace any production that does not obey this rule with a set of productions that do. For more detail, see [8].

Lastly, the following note can be made about the labels of nodes. It is assumed that every label denotes a class of node, e.g. the class of all students. In order to be able to refer in a production to a specific node, a production must first be executed that assigns a unique label to the node. Identification of the node that should receive the unique label can be done by using the node and edge contexts. When the operations necessitating the unique identification of the node have been carried out, the original label of the node may be restored.

3. Modelling of the T/G operations in the modified NLC-grammar

The basic T/G operations will now be formalised in the modified NLC-grammar.

There are four basic operations in the T/G model, namely the Take, Grant, Create, and Remove operations. They are defined as follows: [3]

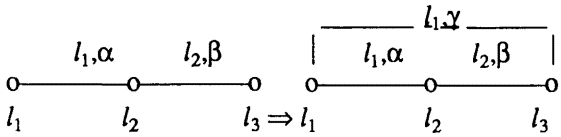
Let H be graph, with x, y, and z $\in V_H$, $\Psi_H(x) = l_1$, $\Psi_H(y) = l_2$, and $\Psi_H(z) = l_3$.

Let $R = \{t, g, c, d, r, w, a, e\}$ with

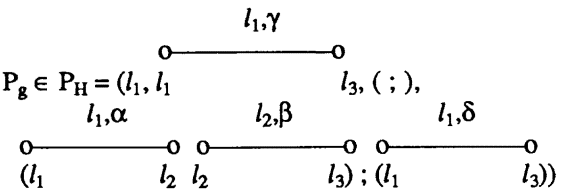
t = "*take*" right
g = "*grant*" right
c = "*create*" right
d = "*remove*" ("delete") right
r = "*read*" right
w = "*write*" right
a = "*append*" right
e = "*execute*" right

1. Take

Let there be an edge between x and y with edge classification α with $t \in \alpha$, as well as an edge between y and z with an edge classification β with $\gamma \subseteq \beta$. An application of the Take rule establishes a new edge between x and z with classification γ. The rule can be read as "x takes (γ to z) from y". The graphical representation of this rule is as follows:



This operation is modelled in the CSM by means of the following production:



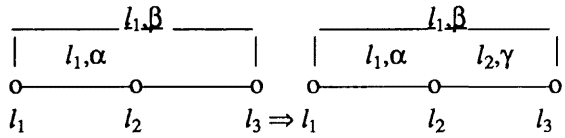
$\forall \delta \subseteq R, t \in \alpha$, using the graphical representation of a production, and

$P_g = (l_1, (l_1, l_3, \gamma), (;), (l_1, l_2, \alpha), (l_2, l_3, \beta) ; (l_1, l_3, \delta)) \forall \delta \subseteq R, t \in \alpha$

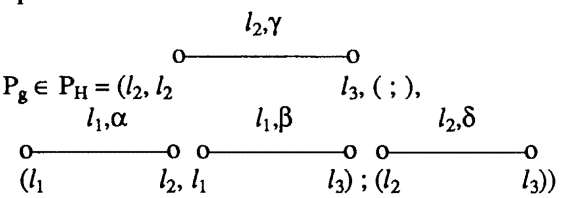
using the "edge context" representation of a production. The graphical representation will be used in the rest of this paper, because it is more legible.

2. Grant

Let there be an edge between x and y with edge classification α, with $g \in \alpha$, as well as an edge between x and z with edge classifcation β, with that $\gamma \subseteq \beta$. The Grant rule defines a new edge between y and z with classification γ. This operation can be read as "x grants (γ to z) to y". The graphical representation is as follows:



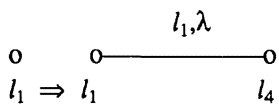
The following CSM production models this operation:



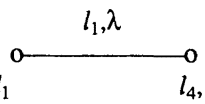
$\forall \delta \subseteq R, g \in \alpha, \gamma \subseteq \beta$.

3. Create

Let there be a node x $\in V_H$ in the current graph. The Create rule adds a new node n, with $\Psi_H(n) = l_4$, to the graph, and creates a new edge with classification λ between x and n. This can be read as "x creates (λ to n)". The graphical representation of this rule is as follows:



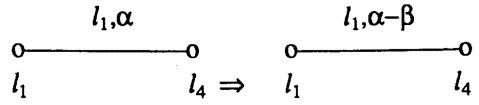
The following CSM production models this operation:



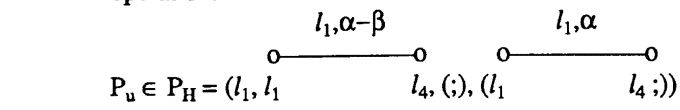
4. Remove

Let there be an edge between x and n with an edge classification α, with $\beta \subseteq \alpha$. The Remove operation removes the set β of rights from α. This rule may be read as "x removes (β to n)". This operation can be

handled by the CSM edge classification update production. The graphical representation is as follows:



The following CSM production models this operation:



A similar operation can be used for the addition of rights to an existing edge classification.

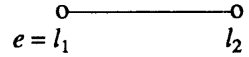
The basic T/G operations have now been defined. Note that the node- and edge contexts in the above operations may contain additional nodes and/or edge contexts in real situations, in order to model other constraints dictated by such a situation.

We now proceed to extend the CSM to model the S/R situation. This model is fully described in [11].

4. Extending NLC-grammars to Formalise the S/R Model

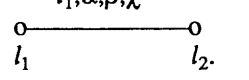
Definition 4.1

Let R be a finite, nonempty set of symbols, called the *set of rights*. Let H be a graph over Σ, and let



be a subgraph of H, consisting of any two connected nodes of H and the edge between them. We associate with the edge of e an α, a β and a $\chi \subseteq R$. α, β and χ together constitute the *classification* of the edge.

We denote this by a tuple $(l_1, l_2, \alpha, \beta, \chi)$, called an *edge context*. The edge context is denoted graphically by adding the labels l_1,α,β,χ to the edge, as follows:



We say that l_1 owns the set α of rights with respect to l_2 and the set β serves the function of *qualifying* the rights in α. That is, if there are any rights within α that operate on other rights, then the rights that are operated upon is given in β and χ respectively, as will be shown later. The "owner" of the rights is indicated by being the first element of the tuple.

The set Λ_H of all possible edge contexts of a graph H, and the set Λ_G of all possible edge contexts of all graphs generated by a grammar G, is defined in the sane manner as in the CSM.

Before we define the grammar that will be used to formalise the S/R model (we shall call it the ECSM - Extended Context-dependent Security Model), a few other preliminary definitions are necessary. We associate with each label l in the protection graph a set of attributes, denoted by $l\{A_1, \ldots, A_k\}$. The value of the attribute A_j is denoted by a_j. The set of all possible values of an attribute a_j is denoted by $\mathcal{D}(A_j)$. We define the attribute set $\mathcal{A}$ as the set of all possible attributes in the system. The power set of $\mathcal{A}$ denoted by $\mathcal{P}(\mathcal{A})$.

Definition 4.2

An *attributed label* l is a label that has associated with it a set of attributes, denoted by
$l\{A_1, \ldots, A_k\}$,
such that $A_1, \ldots, A_k \in \mathcal{A}$, $l \in \Sigma$, where $\mathcal{A}$ is a set of attributes, and Σ is a nonempty, finite set of node labels.

Definition 4.3

A NLC-grammar with node- and edge context is a system

$G = (\Sigma, \Delta, P, C, Z, R, \mathcal{A})$ where

Σ is nonempty finite set of attributed labels, called the total alphabet

Δ is a nonempty finite subset of Σ, called the terminal alphabet

P is a finite set of productions, of the form

$(d, D, (\Sigma 1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$

$d \in \Sigma$

D a graph over Σ, where with every edge D we associate an edge classification α, β, χ with $\alpha, \beta, \chi \subseteq R$, applicable to one of the nodes of the edge;

$\Sigma_1, \Sigma_2 \subseteq \Sigma$

$\Sigma_1 \cap \Sigma_2 = \varnothing$

Σ_1 is called the permitting and Σ_2 the forbidding node context

$\Lambda_1, \Lambda_2 \subseteq \Lambda_G$. Λ_1 and Λ_2 are called the permitting and forbidding edge contexts, respectively. Also, $\Lambda_1 \cap \Lambda_2 = \varnothing$.

C is a subset of $\Sigma \times \Sigma$, called the connection relation

Z is a graph over Σ, called the axiom

R is a finite, nonempty set of symbols, called the set of rights

$\mathcal{A}$ is a finite, nonempty set of attributes.

Definition 4.4

Let $x,y \in V_H$, where H is a graph over Σ, with $\Psi_H(x) = l_1$ and $\Psi_H(y) = l_2$. We say that **x** *matches* **y** if

i) $l_1 = l_2$

ii) The value a_j of every attribute A_j of l_1 is the same as the value of the corresponding attribute of l_2.

The meaning of the permitting and forbidding node/edge contexts in definition 4.4 is as follows. A production may be applied if

- all of the nodes specified in the permitting node context Σ_1 are matched by nodes in the current graph
- none of the nodes specified in the forbidding node context Σ_2 is matched by any of the nodes in the current graph

- for every edge context (l_1, l_2, α, β, χ) that appears in the set Λ_1 of permitting edge contexts, there exists an edge somewhere in the current graph between two nodes labelled l_1 and l_2 having an edge classification α, β, χ.
- for every edge context (l_3, l_4, ε, ϕ, γ) that appears in the set Λ_2 of forbidding edge contexts, there does not exist an edge anywhere in the current graph between two nodes labelled l_3 and l_4, having an edge classification ε, ϕ, γ.

As in the CSM, four types of productions can be identified:

1. Normal productions
2. Edge generation productions
3. Edge removal productions
4. Edge classification update productions.

A direct derivation step is performed by following the same steps as in the CSM.

The different types of productions will now be discussed in greater detail.

1. A normal production

is of the form

$(l, D, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$

where D is an arbitrary graph over the set Σ. Let $\bar{D}$ be the isomorphic copy of D that is to replace v, with $V_{\bar{D}} \cap V_{H\setminus v} = \varnothing$. This type of production is denoted by P_n.

This type of production is used where at least one new node is to be introduced into the graph (created). The graph D may consist of one or more nodes, and may contain isolated (unconnected) nodes. A normal production cannot establish edges between existing nodes in the graph; the edge generation production is to be used in such a case.

2. An edge generation production

is of the form

l_1,α,β,χ

o——————o

$(l_1, l_1$ $l_2, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$

$l_1,\varepsilon,\phi,\gamma$

o——————o

with l_1 $l_2 \in \Lambda_2$, $\forall\ \varepsilon,\phi,\gamma \in R$, $l_2 \in \Sigma_1$,

and is denoted by P_g.

It is assumed that identical labels in the above production refer to the sane nodes.

The function of this type of production is the generation of an edge between two existing nodes labelled l_1 and l_2. The creation of the edge is initiated by the node labelled l_1 in order to gain access to the node labelled l_2. The owner of the edge (l_1) gives the *send* and *receive*, as well as all other "inert" (*read*, *write*, *append*, *execute*) rights to the edge classification.

The edge generation production can be graphically illustrated as follows:

l_1,α,β,χ

o o o——————o

l_1 l_2 $\Rightarrow$ l_1 l_2

where the nodes labelled l_1 and l_2 are existing nodes in an arbitrary graph.

3. An edge removal production

is of the form

$l_1,\varnothing,\varnothing,\varnothing$

o— — — — —o

$(l_1, l_1$ $l_2, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$

with $(l_1,l_2,\alpha,\beta,\chi) \in \Lambda_1$, $\alpha,\beta,\chi \in R$, and is denoted by P_r.

It is assumed that identical labels in the above production refer to the same nodes.

The function of the edge removal production is to remove an existing edge from the graph, in order to indicate that no relation between the two nodes exists anymore.

The edge removal production can be graphically illustrated as follows:

l_1,α,β,χ $l_1,\varnothing,\varnothing,\varnothing$

o——————o o— — — — —o

l_1 l_2 $\Rightarrow$ l_1 l_2

where the nodes labelled l_1 and l_2 are existing and connected nodes in an arbitrary graph.

4. Edge classification update production

This type of production is of the form

l_1,α,β,χ

o——————o

$(l_1, l_1$ $l_2, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$ with

$(l_1,l_2,\varepsilon,\phi,\gamma) \in \Lambda_1$, while at least one of the following statements is true:

i) $\alpha \neq \varepsilon$

ii) $\beta \neq \phi$

iii) $\chi \neq \gamma$

This type of production is denoted by P_u.

It is assumed that identical labels in the above production refer to the sane nodes.

The function of this type of production is to reflect a change in the relationship between two nodes by altering the edge classification, while leaving the nodes and the edge between then intact. When all the sets in the edge classification becomes empty, it is indicated that no relationship between the two nodes currently exists, although the edge between the nodes is still shown.

The edge classification update production can be graphically illustrated as follows:

$l_1,\varepsilon,\phi,\gamma$ l_1,α,β,γ

o——————o o——————o

l_1 l_2 $\Rightarrow$ l_1 l_2

where the node labelled l_1 and l_2 are existing (connected) nodes in an arbitrary graph.

The observations that were noted in relation to the CSM productions also apply here.

5. Modelling the S/R Operations within the NLC-grammars

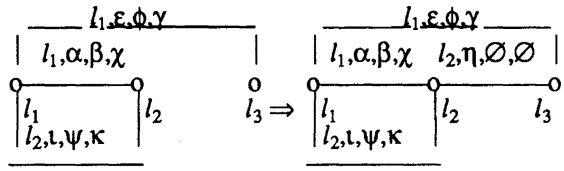
Let H be a graph, with x, y, and z $\in V_H$, $\Psi_H(x) = l_1$, $\Psi_H(y) = l_2$, $\Psi(z) = l_3$.

Before we start to model the S/R operations in the defined grammar, a brief explanation of the edge classification is in order. We said that, given an edge between two nodes labelled l_1 and l_2 with edge context $(l_1, \alpha, \beta, \chi)$, the role of the sets β and χ is to *qualify* certain rights within the set α. We now define these sets to be used as follows: if the *send* right appears within α, the set β contains the set of rights that l_1 may send to l_2. Likewise for χ and the *receive* right: if *receive* $\in \alpha$, χ contains the set of rights that l_1 may receive from l_2. If any other rights appear within α, they have their usual meaning, and need not be qualified.

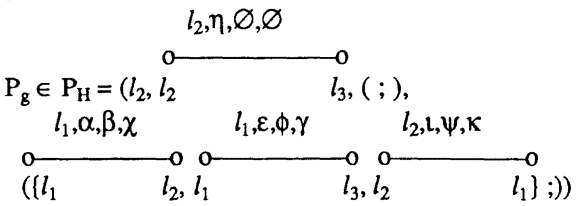
1. Send

Let there be an edge between x and y with edge context $(l_1, l_2, \alpha, \beta, \chi)$ with *send* $\in \alpha$. Let there also be an edge between x and z with edge context $(l_1, l_3, \varepsilon, \phi, \gamma)$ with $\eta \subseteq \varepsilon$ and $\eta \subseteq \beta$. Lastly, let there be another edge between y and x with edge context $(l_2, l_1, \iota, \psi, \kappa)$ with *receive* $\in \iota$, $\eta \subseteq \kappa$.

The send operation establishes a new edge between y and z with edge context $(l_2, l_3, \eta, \varnothing, \varnothing)$. (The reason for the empty sets in this context is that they correspond to qualifiers or rules in the S/R model, and the transportation of these rules is not possible in our formulation of the model). In this production, x sends y the set η of rights, which x has on z, for y to operate on z. The graphical representation of this operation is as follows:



We model this operation by means of the following production:

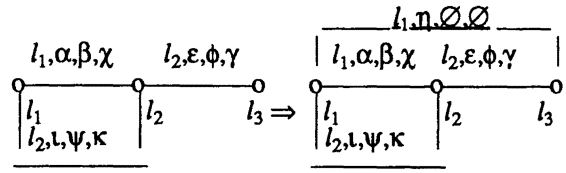


with $\eta \subseteq \varepsilon$, $\eta \subseteq \beta$, $\eta \subseteq \kappa$, *send* $\in \alpha$, *receive* $\in \iota$.

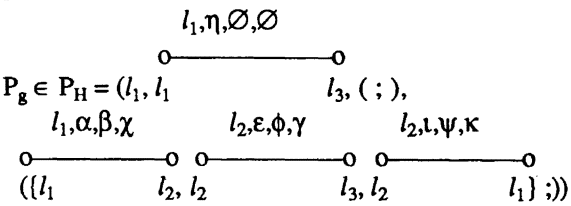
2. Receive

Let there be and edge between x and y with edge context $(l_1, l_2, \alpha, \beta, \chi)$ with that *receive* $\in \alpha$. Let there also be an edge between y and z with edge context $(l_2, l_3, \varepsilon, \phi, \gamma)$, with $\eta \subseteq \varepsilon$, and $\eta \subseteq \chi$. Lastly, let there be another edge between x and y with edge context $(l_2, l_1, \iota, \psi, \kappa)$ with *send* $\in \iota$, $\eta \subseteq \psi$.

The receive operation establishes a new edge between x and z with edge context $(l_1, l_3, \eta, \varnothing, \varnothing)$. (The reason for the empty sets in this context is the same as given above). In this production, x receives from y the set η of rights with which to operate on z. The graphical representation of this operation is as follows:



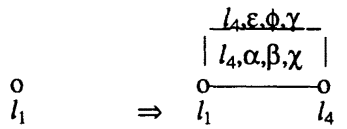
We model this operation by means of the following production:



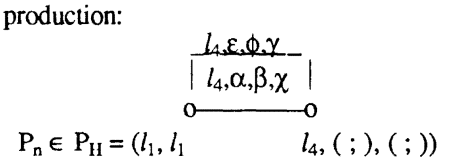
with $\eta \subseteq \varepsilon$, $\eta \subseteq \chi$, $\eta \subseteq \psi$, *send* $\in \iota$, *receive* $\in \alpha$.

3. Create

Let x be a node in a graph H. The Create operation adds a new node n to the graph, with $\Psi_H(n) = l_4$, as well as one edge between x and n with edge context $(l_1, l_4, \alpha, \beta, \chi)$ with $\alpha = R$, $\beta = \chi = R - \{send, receive\}$, and a second edge between n and x with edge context $(l_4, l_1, \varepsilon, \phi, \gamma)$ such that $\varepsilon = \alpha$, $\phi = \beta$, $\gamma = \chi$ (this is done to make full bidirectional transfer of rights possible). This operation is denoted graphically a follows:



This operation is implemented by the following production:



with $l_4 \in \Sigma_2$.

6. Conclusion

There are still some issues to be investigated, and the most important of these would be the development of a grammar suitable for the modelling of protection heuristics, and for the definition of a security model that incorporates the use of expert systems theory.

References

[1] D. Janssens and G. Rozenberg, [1980], On the structure of Node-Label-Controlled graph languages, *Information Sciences*, **20**, 191-216.

[2] R.J. Lipton and L. Snyder, [1977], A linear time algorithm for deciding subject security, *Journal of the Association for Computing Machinery*, **24** (3), 455-464.

[3] N.H. Minsky, [1984], Selective and locally controlled transport of privileges, *ACM Transactions on Programming Languages and Systems*, **6** (4), 573-602.

[4] L. Snyder, [1981], Formal models of capability-based protection systems, *IEEE Transactions on Computers*, **C-30** (3), 172-181.

[5] L.Snyder, [1981], Theft and conspiracy in the Take-Grant model, *Journal of Computer and System Sciences*, **23**, 333-347.

[6] S.H. von Solms, [1984], Node-Label-Controlled graph grammars with context conditions, *International Journal of Computer Mathematics*, **13**.

[7] S.H. von Solms, [1980], Rewriting systems with limited distance Permitting context, *International Journal of Computer Mathematics*, **8** (A), 223-231.

[8] S.H. von Solms, [1982], Rewriting systems with limited distance forbidding context, *International Journal of Computer Mathematics*, **11** (A), 227-239.

[9] D.P. De Villiers, [1986], Logiese Sekuriteits-model gebaseer op NLC-grammatikas, M.Sc. Dissertation, Rand Afrikaans University.

[10] D.P. de Villiers and S.H. von Solms, A logical security model based on NLC-grammars, Submitted.

[11] D.P. de Villiers and S.H. von Solms, Formalizing the Send/Receive security model using NLC-grammars, Submitted.

SOUTH AFRICAN INSTITUTE OF COMPUTER SCIENTISTS

Members in Arrears

Mrs S Berman
Prof C H Bomman
Mr M J Chapman
Mr A K Cooper
Mr W A Cronin
Mr W R Jones
Mr S M Kaplan
Prof P S Kritzinger
Mr D N Lubowitz
Prof K J MacGregor
Mr D J Malan
Mr R J Mann
Mr J C Marnewick
Mr S A Matsoukis
Mr G D Oosthuizen
Mr R P Perold
Mrs A E G Potgieter
Mr C D Reynecke
Mrs C M Richfield
Mr I S Shaw
Mej F Spoelstra
Mr S L Stoch
Mr T Turton
Mr W van Biljon
Prof R van den Heever
Mr W L van Niekerk
Dr D E Wolvaardt
Prof J S Wolvaardt

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Professor J M Bishop
Department of Computer Science
University of the Witwatersrand
Johannesburg
Wits
2050

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review Categories.

Manuscripts may be provided on disc using any Apple Macintosh package or in ASCII format.

For authors wishing to provide camera-ready copy, a page specification is freely available on request from the Editor.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

[1] E. Ashcroft and Z. Manna, [1972], The Translation of 'GOTO' Programs to 'WHILE' programs, *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255.

[2] C. Bohm and G. Jacopini, [1966], Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM,* **9**, 366-371.

[3] S. Ginsburg, [1966], *Mathematical Theory of Context-free Languages,* McGraw Hill, New York.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimise the risk of the author's contribution having to be held over to a later issue.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.