LOGICAL LANGUAGE OF CERTIFICATE-BASED ACCESS CONTROL IN SECURITY MODELS

Mikhail M. Kucherov Nina A. Bogulskaya
Siberian Federal University
26B Kirenskogo Krasnoyarsk, Russia 660074
{mkucherov,nbogulskaya}@sfu-kras.ru

ABSTRACT

Over the last decades, we have seen several policy models, including role-based access control and more recently, certificate-base control. These models are based on the important notion "flow relation". In this work, we present a logical language of certificate-based access control. Our model presents the formal method of reasoning for discretionary access and defines logic to express a discretionary policy. We introduce, instead, material implication widely used in mathematics, and we show in a case study its ease in every sense. We find it allows the policy specifications to be interpreted more conveniently by practitioners and implemented in a simple way. Our evaluation shows that policies defined with material implication can be used for creation of the specification of a trust relationships policy and for checking safety of any computer system.

CCS Concepts

• **Security and privacy~Access control**
• *Security and privacy~Information flow control*
• Security and privacy~Software security engineering.

XML code:

```
<ccs2012>
<concept>
<concept_id>10002978.10002991.10002993</concept_id>
<concept_desc>Security and privacy~Access control</concept_desc>
<concept_significance>500</concept_significance>
</concept>
<concept>
<concept_id>10002978.10003006.10011608</concept_id>
<concept_desc>Security and privacy~Information flow
control</concept_desc>
<concept_significance>300</concept_significance>
</concept>
<concept>
<concept_id>10002978.10003022.10003023</concept_id>
<concept_desc>Security and privacy~Software security
engineering</concept_desc>
<concept_significance>100</concept_significance>
</concept>
</ccs2012>
```

Keywords

access control policy languages; access control model; authorization; logic functions; information flow model

1. INTRODUCTION

Monitoring of access is understood as methods or mechanisms which define whether the request for access to any resource shall be resolved or forbidden. It is known that each distributed access control system should contain an information protection subsystem, which must be based on precisely defined mathematical models for controlling access to this information.

In our article we propose the flow-based logic model for interpreting the basic events and properties of the distributed access control systems. Our goal is to develop the logic and the formal language that can be used for making a security policy specification and for checking any computer system security. We can prove some important properties of this logic and show on a case study how our logical language can express some access control policies proposed so far. This can be achieved with introducing certificates. The certificate-based access control is aimed at specifying security policies for access to resources from untrusted sources, e.g. via the Internet.

Recently, the work on logic-based access models and certificate-based authorization has been intensified. Formal reasoning techniques on security models and access policy specifications have been presented, e.g. [1–5]. These models are usually based on the modal logic and cannot be automated in a simple way. The logic-based approaches generally fail to directly map to an implementation and are not easily interpreted by practitioners. It is intuitively clear that a system with total security is a system that does not allow the information flows among its users. Thus, the system is called safe under a certain policy if all its transactions are confirmed with an ideal security system, except for permitted in policy.

2. LOGICAL MODEL

It is known that the permitted flow of information in a system can naturally be represented as a lattice-ordered set, $(S, \geq)$, where S is a given set of security classes and «→» is a flow relation specifying permissible flows between pairs of classes [6]. Objects are bound to these security classes. Information may flow from object x to y through any sequence of operations if and only if $A" \rightarrow " B$, where A and B are the objects' security classes. Information can be passed by copying, assignment, I/O, parameter passing, message sending, etc. We concerned with information flow on "legitimate" and "storage" channels, not "covert" channels. Binding of objects to security classes can be static or dynamic. With static binding, the security class of an object never changes. With dynamic binding, the object's security class can change based on the content of the object. A process can also be bound to a security class.

A lattice-ordered set is a poset $(S, \geq)$ in which each two-element subset $\{x, y\}$ has a greatest lower bound, denoted $\inf\{x,y\}$, and a

least upper bound, denoted sup{*x*,*y*}. Lattice-ordered sets abound in mathematics and its applications.

Let *A* and *B* be security classes. $A \oplus B$ refers to the security class of the result of any binary function on values *x* and *y* (*x* = *A*, *y* = *B*). Operator $\oplus$ is function independent.

Under the reasonable assumptions that there is a finite number of security classes, that the flow relation is reflexive, and that the flow relation is transitive, we may suppose that $(S, \geq)$ is a lattice, $\langle S, \rightarrow \rangle$. If $\langle S, \rightarrow \rangle$ is not a lattice, it may be transformed into one by adding new classes as necessary without changing the flows among the original classes. In this model, a system is secure if no flow of information violates the flow relation.

3. DERIVATION OF LATTICE

There is a natural relationship between lattice-ordered sets and lattices. Indeed, a lattice $\langle S, \rightarrow \rangle$ is obtained from a lattice-ordered poset $(S, \geq)$. Suppose $\geq = "\rightarrow"$ is the relation. *C* is an upper bound of *A* and *B* if $A \rightarrow C$ and $B \rightarrow C$. *C* is a least upper bound of *A* and *B* if for any upper bound *D* of *A* and *B*, $D \rightarrow C$. Lower bounds and greatest lower bounds work the same way.

First we show that $\langle S, \rightarrow \rangle$ is a poset. It is reflexive: $A \rightarrow A$ (for consistency sake); transitive: if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$ (for consistency sake); antisymmetric: if $A \rightarrow B$ and $B \rightarrow A$, then $A = B$ (otherwise, one has a superfluous security class).

Second, we assume *S* is finite because we dealing with the real world. Third, we can assume that $x \wedge y = \inf\{x,y\} = L$. There exists a greatest lower bound *L* on *S* without loss of generality. If needed, we can insert *L* with no object. Or, perhaps we could fill it with constant. Fourth, we can assume that $x \vee y = \sup\{x,y\} = H$ is a least upper bound operator for any $x,y \in S$.

A lattice-ordered set is bounded provided that it is a bounded poset, i.e., if it has an upper bound and a lower bound. For a bounded lattice-ordered set, the upper bound is frequently denoted 1 and the lower bound is frequently denoted 0. Given an element *x* of a bounded lattice-ordered set $(S, \geq)$, we say that *x* is complemented in $(S, \geq)$ if there exists an element $y \in S$ such that $\inf\{x,y\} = 0$ and $\sup\{x,y\} = 1$.

Also, from a lattice $\langle S, \rightarrow, \vee, \wedge \rangle$, one may obtain a lattice-ordered poset $(S, \rightarrow)$ by setting $x \rightarrow y$ iff $x = x \wedge y$. One obtains the same lattice-ordered poset $(S, \rightarrow)$ from the given lattice by setting $x \rightarrow y$ iff $y = x \vee y$. One may prove that for any lattice, $\langle S, \rightarrow, \vee, \wedge \rangle$, and for any two members *x* and *y* of *S*, $x \wedge y = x$ iff $y = x \vee y$.

Thus, we have established that $\langle S, \rightarrow, \vee, \wedge \rangle$ form universally bounded lattice with greatest lower bound *L*, and the least upper bound *H*. Lattices are different from a lattice-ordered sets because lattices are algebraic structures that form an variety, but lattice-ordered sets are not algebraic structures, and therefore do not form a variety [7].

It is tempting to present the security objects through logical language. In modern times, many researchers have proposed logic diagrams, especially as representations of logical reasoning (e.g., [8]). "Diagrams are a kind of … knowledge representation mechanism that is characterized by correspondence between the structure of the representation and the structure of the represented" [9]. Their advantages include effective control of the reasoning process, and understandability by users. It is suggested that the ability to manipulate logic diagrams could be provided even to nonspecialists.

Therefore, we saw *material implication* as an abstract counterpart of the empirical flow relation, "$\rightarrow$", and the close resemblance above allows modeling the flow relation in all relevant cases. It follows, that $x \rightarrow y \leftrightarrow \neg x \vee y$.

4. LOGIC FORMULAE

Consider the example (from [2]):

- Tipically, every student is authorized to use every device.
- Those who have abused a device before lose access to that device.
- John is a student and a printer is device.
- John is authorized to use a printer.

This access control policy can be written with flow relation, as: $\forall x$ (*Student*(*x*) "$\rightarrow$" $\exists y$ (*Device* (*y*) $\wedge \neg$*Abused* (*x*, *y*))). We have

1. $x \rightarrow y \leftrightarrow \neg x \vee y$;

1.¬*Student* (John) $\vee$ (*Device* (printer) $\wedge \neg$*Abused* (John, printer));

2. *Device* (printer) $\wedge \neg$*Abused* (John, printer);

3.¬*Abused* (John, printer).

"*Student* (*x*) receives ′John′" means applying "John" to *Student*(*x*) and, according to the structure of the formula, also to *Abused*(*x*, *y*). The notion of *information flow* here accomplishes this notion of application; as in mathematics, replacing a variable by a value means replacing it across the equation.

Since John is a student, he is authorized to use a printer. Using the symbols "1" and "0" for security classes, "authorized" or not, we get the truth value of ¬*Student*(*x*) is processed according to "not" truth table, resulting in *false* to "$\vee$", and it is not processed further. On the other hand, assigning a value, say, printer, to *y*, causes replacing in *Abused* (John, printer). Hence, creating *true* in *Device* (printer) causes it to flow to "$\wedge$" and is processed to end in ¬*Abused* (John, printer). The resultant of the latter is *false*, as well as the example of access control policy.

5. OPERATIONS WRITE AND READ

The basic idea in this paper is to conceptualize the truth values as things that can be used as a set of functional characteristics of objects. This conceptualized flow is, to some degree, analogous to information flow. Every security object has two subspheres: the object itself and the truth value given its class of security. The truth value can flow to another term by such logical connective as material implication.

The "write" operation can be interpreted as a flow of information to the object, as Figure 1 shows.

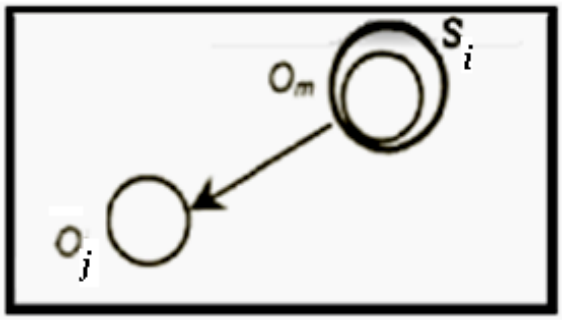

Figure 1: The "write" operation is modeled by material implication between two variables, one of which is security level of object O_m related to subject S_i (writer), and another is security level of object O_j

Then we can describe the writing and reading operations in notions of finite-state automata, as usual. The resultant of material

implication is such that "1" represents "enabled", and "0"– "forbidden" (in the 3d column, see Table 1).

Table 1. Truth table for the "write" operation

O_m (S_i)	O_j	$O_m \rightarrow O_j$
0	0	1
0	1	1
1	0	0
1	1	1

The "read" operation can be interpreted as a flow of information from the object O_m, as Figure 2 shows.

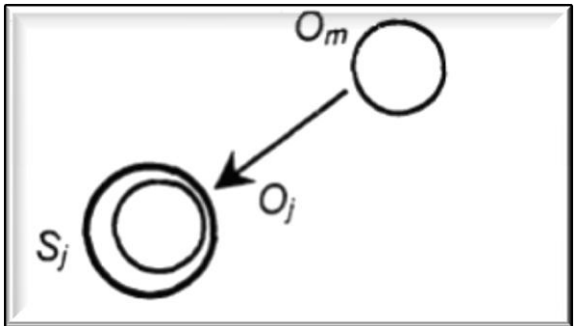

Figure 2: The "read" operation is modeled by material implication between two variables, one of which is security level of object O_m, and another is object O_j related to subject S_j (reader)

Using the symbols "1" and "0" for security classes, similarly to the preceding case we get following truth table, see Table 2.

Table 2: Truth table for the "read" operation

O_m	$O_j(S_j)$	$O_m \rightarrow O_j$
0	0	1
1	0	0
0	1	1
1	1	1

By duality we can formulate the rules of Biba integrity model, giving "flow relation" as $\neg (\neg x \vee y) = x \wedge \neg y$. For example, using the symbols "1" and "0" for integrity classes, where "1" represents, as before, a higher level of integrity than "0", a subject *y* must not read an object *x* at a lower integrity level, accordingly, a subject *x* at a low level of integrity must not write to any object *y* at a higher level of integrity.

The goal, of course, of deriving this logical model is for it to help us enforce security. To do this, we must monitor all information flow causing operations. We must monitor explicit flow (assignment, I/O) and implicit flow. We want to represent a program or statement *Q* in a way that easily allows us to evaluate whether or not it is secure. Define *Q* recursively: *Q* is an elementary statement (assignment, I/O); $Q = Q_1; Q_2$; $Q = c: Q_1, \ldots, Q_m$ (*c* is *m*-valued variable).

For elementary statements, *Q* is secure if any explicit flow caused by *Q* is secure. For $Q = Q_1; Q_2$, *Q* is secure if both Q_1 and Q_2 are secure. For $Q = c: Q_1, \ldots, Q_m$, *Q* is secure if each Q_k is secure and all implicit flows from *c* are secure.

6. LANGUAGE OF LOGIC

Let's enter language of our logic. First we determine atomic formulas on the basis of the following predicates:

1) *grant_key* (*K*, *P*, *Q*) – the principal *P* grants the key *K* to the principal *Q* . It means automatic issuing the certificate of the key which is valid until it is revoked.

2) *grant_right* (*R*, *P*, *Q* , *O*) – the principal *P* grants to the principal *Q* the access right *R* to the object *O*.

3) *give_obright* (*P*, *O*, *R*, *A*) – the principal *P* determines that the condition of using the object *O* with the access right *R* was the possessing of the attribute *A* by any subject. The statement is valid until it is revoked.

4) *take_key* (*K*, *P*, *Q*) – the principal *P* takes from the principal *Q* the key *K*. The key *K* is added to the key revocation list.

5) *take_right* (*R*, *P*, *Q* , *O*) – the principal *P* takes from the principal *Q* the access right *R* to an object *O*.

6) *take_obright* (*P*, *O*, *R*, *A*) – the principal *P* withdraws the condition of using the object *O* with the access right *R* and the attributes *A* from all possessing subjects.

When considering the security properties of a distributed system it is convenient to use concept based on histories. The system interacts with its environment through events. These events correspond to actions, done by system or its environment. We define them by α. Sequence of events corresponding to possible sequence of actions determines a history. System events are defined in terms of histories.

By a local history of principal $P_m\ (m = 1, 2, \ldots, n_p)$ at the moment *k*, we mean a sequence of actions $(\alpha_1, \alpha_2, \ldots, \alpha_k)$, executed by the principal. Number *k* indicates a point of time (discrete time), by which P_m has performed the actions above. We denote the local history of principal P_m by ζ_m^k. We select the point $k = 0$, where all local histories are empty sequences. To simplify, we assume that all the clocks in the network are synchronized.

Each principal P_m at any point of time *k* has:

LP_m^k – the set that states the principal activity. This set is empty at the time $k = 0$. After the execution of action *create_principal* (*P*, P_m), the element "+" is added to the set. After performing action *delete_principal* (*P*, P_m), it becomes empty, and all contents of all following sets are deleted.

RP_m^k – the set of access rights to objects that P_m has at the point of time *k*. After action, like *grant_right* (*R*, *P*, *Q*, *O*), or *take_right* (*R*, *P*, *Q*, *O*), each of the sets RP_m^{k+1} is enlarged or decreased with corresponding quadruples.

A local state of principal P_m at the point of time *k* consists of

1. local history ζ_m^k,
2. set of activities LP_m^k,
3. set of keys K_m^k,
4. set of access rights RP_m^k.

Let us denote local principal states at the point of time *k* as s_m^k. A global state of principals at the point of time *k* represents the sequence of local principal states $s^k = s_1^k, s_2^k, \ldots, s_n^k$.

At any point of time *k* each object O_m has:

LO_m^k – the set that states the object activity. This set is empty at the point of time $k = 0$. After the execution of action *create_object* (*P*, O_n), the element "+" is added to the set. After performing step *delete_object* (*P*, O_n) it becomes empty. After performing action *delete_object*(*P*, O_n) all contents of all following set are deleted.

RO_m^k – the set stating the conditions of use of the object O_n with given access rights at the point of time *k*. RO_m^k RO_m^kAfter executing an action like like $RO_m^{k+1}$is enlarged or decreased with corresponding quadruples.

By a local state of objects at the point of time *k* we mean a sequence of pairs

$$o^k = \left(\left(RO_1^k, LO_1^k\right), \left(RO_2^k, LO_2^k\right), \ldots, \left(RO_{n_o}^k, LO_{n_o}^k\right)\right).$$

By global system state so^k we mean a pair of $so^k = (s^k, o^k)$.
A run η represents any sequence of global states of a system

$$\eta = (so^1, so^2, \ldots, so^n).$$

Program execution is a sequence of system states. In many cases a local state of principal is strongly associated with the available keys. For example, the formula "Q has key *K*" could be written as follows:

$$HKey(Q,K) \overset{df}{\leftrightarrow}$$
$$\bigvee_{P_1 \in S_p} grant\_key(K, P_1, Q) \Lambda \bigwedge_{P_2 \in S_p} \neg take\_key(K, P_2, Q)$$

This formula means: "Q has key *K* iff there is some principal which confirms that the key *K* belongs to principal Q and nobody has revoked this key."

7. CASE STUDY

Let us consider a company that has information divided in two compartments:
1. financial (e.g., product pricing)
2. product (e.g., product designs).

Each file in the computer system is labeled to belong to one of these compartments. Every principal is given a clearance for one or both compartments. For example, the company's policy might be as follows: the company accounts have clearance for reading and writing files in the financial compartment, the company engineers have clearance for reading and writing files in the product compartment, and the company product managers have clearance for reading and writing files in both compartments.
The principals of the system interact with the files through programs, which are untrusted. We want ensure that information flows only to the company's policy. To achieve this goal, every subject records the labels of the compartments for which the principal is cleared; this clearance is stored in S_{lseen}. Furthermore, the system remembers the maximum compartment label of data the subject has seen, S_{lmax}. Now the information flow control rules can be implemented as follows.
In our interpretation the read rule is:

- Before reading an object with labels *O*, check that

$$S_{\text{lmax}} \cap O = O.$$

- If so, set $S_{\text{lseen}} \coloneqq S_{\text{lseen}} \cup O$, and allow access.

The subject is not allowed to have access to information in compartments for which it has no clearance.
Also in our interpretation the corresponding write rule is:

- Allow a write to an object with clearance *O* only if

$$S_{\text{lseen}} \cup O = O.$$

Every object written by a subject that read data in compartments *L* must be labeled with *L*'s labels. This rule ensures that if a subject *S* has read information in a compartment other than the ones listed in *L* than that information doesn't leak into the object *O*.
These information rules can be used to implement a wide range of policies. For example, the company can create more compartments, more principals, or modify the list of compartments a principal has clearance for. These changes in policy don't require changes in the information flow rules. The standard policy of discretionary access claims that an object can be read only by those principals which have access rights on reading for this object [10].
These requirements can be written in our logic. Let's denote

$$Stream(S_j, O_m) \rightarrow O_j$$

a flow of information from object O_m to object O_j, as Figure 2 shows. In the definition it is stressed that a flow of information is not between subject and object, but only between two objects, for example, those related by input-output operations. The active role of a subject is expressed in the realization of this stream (this operation is localized in a subject, and displayed in a state of its associated objects). From the logical point of view operation $Stream(S_j, O_m) \rightarrow O_j$ can be represented in the case of reading, as

$$O_m \rightarrow O_j,$$

where the arrow expresses material implication. The object O_j, associated with the subject S_j, after read operation, has the security level: $O_j \coloneqq O_j \cup O_m$. For write operation, we have also, as Figure 1 shows:

$$O_m \rightarrow O_j.$$

These operations can be represented as predicates:

$$give\_access\ (R, Q, P, O) \overset{df}{\leftrightarrow}$$
$$\bigvee_{P_1 \in S} grant\_right(R, Q, P_1, O) \Lambda \bigwedge_{P_2 \in S} \neg take\_right(R, P_2, Q, O),$$

This formula means:" Q has an access to object *O* iff there is some principal which confirms that the right *R* belongs to principal Q and nobody has revoked this right." It is easy to see that the function $take\_right(R, P, Q, O)$ is simulated in the case of read operation, $R = read_r$, as Figure 2 shows,

$$take\_right(read_r, P, O_j, O_m) \leftrightarrow O_m \cap \neg\, O_j,$$

or write operation, $R = write_r$,

$$take\_right(write_r, P, O_m, O_j) \leftrightarrow O_m \cap \neg\, O_j,$$

as Figure 1 shows, where O_m and O_j say for security levels of corresponding objects. We can also extend these formulas to the case of multiple reads of object O_m or writes to object O_j.

8. SUMMARY AND CONCLUSION

By access control we understand methods or mechanisms that decide whether requests to access some resource should be granted or denied. For example, operating systems need to control which subjects and applications can read, write, or delete which files; managers need to control which employees can perform which workflows within an organization. Prior forms of policy, such as access control matrix, allow you to specify only the access requests should be granted. Other queries were then denied. This approach does not allow the administration explicitly uphold access rights and restrictions. The language must support the ability to explicitly express both permissions and prohibitions.
Originally, the languages of access control were invented driven by certain application; for example, operating systems. This led to the redundancy of effort in design. A language of access control should therefore support the deeper layer [11] that encapsulates domain-specific structure, assumptions, or knowledge. Its

composition mechanisms should so facilitate the applicability of access control patterns across application domains.

In this paper, we have considered the development of information security model, derived from the discretionary security model. Our model represents the formal method of reasoning for discretionary model (we read that as well for other models, for example, role access control) and defines logic to formalize a discretionary policy, and also offers a decision algorithm which can be used for check by a direct and automatic method of coherence of a policy of access or its logical investigations.

We have defined a language for certificate-based access control based on Boolean logic, and shown that it thoroughly handles problems in security models. The analysis was shown to reduce to validity checks in propositional logic, and we support it with assume-guarantee reasoning. We have shown how the use of our language can help in the analysis of sertificate-based policies and of policies for discretionary access.

We reiterate key elements of our work. By basing our language on classical Boolean logic, the properties of security classes and precise structure of accesses can be expressed as simple, purely result of evaluation of propositional forms. We give case study for the use of such a context for the purpose of illustration.

The logic which we offer is powerful, indicative and allows creating and using simple expressions even with qualifiers.

The offered logic has the expressiveness of the first-order logic. Analysis methods on the basis of this logic can be used at a design stage and checks of any computer network access.

Practical systems need access and logical control. Offering a logical model, the authors proceeded from the fact that the main purpose of information security models – provide formalization of security policies. The more general models of information flows and finite automata were used for a description of information security models.

In summary, the use of logic functions for the formulation of information security requirements makes it possible to use the developed apparatus of mathematical logic to determine the correct implementation of security policies in each specific case.

9. REFERENCES

[1] Abadi, M., Burrows, M., Lampson, B., and Plotkin, G. 1993. A Calculus for Access Control in Distributed Systems. *ACM TOPLAS*, 15(4) (Sept. 1993), 706 -734, 1993. DOI = https://doi.org/10.1145/155183.155225.

[2] Basseda, R., Gao, T., Kifer M., Greenspan, S. and Chell C. Representing Flexible Role-Base Access Control Policies Using Objects and Defeasible Reasoning. In *Rule Technologies: Foundations, Tools, and Applications*. Springer, 376-387, 2015.

[3] Bertino, E., Ferrari, E., Buccafurri, F. and Rullo, P. A Logical Framework forReasoning on Data Access Control Policies. In *Proceeding of the 12th IEEE Computer Security Workshop* (June 1999), 175-191, 1999. DOI = https://doi.org/10.1109/CSFW.1999.779772.

[4] Bruns, G., Dantas, D.S. and Huth., M. 2007. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering* (Nov. 2007), 12-21, 2007. DOI = https://doi.org/10.1145/1314436.1314439

[5] Kurkowski, M. and Pejas, J. A Propositional Logic for Access Control in Distributed Systems. In *9th Artificial Intelligence and Security in Computing Systems*. Springer Science+Business Media New York,175-189, 2003.

[6] Denning, D. E. 1976. A lattice model of secure information flow.*CACM*, 19(5), 236-243, 1976. DOI = https://doi.org/10.1145/360051.360056.

[7] Grätzer, G. 1998. *Lattice Theory: Foundation*. Springer Science & Business Media, Berlin/Heidelberg, Germany.

[8] Glasgow, J., Narayanan, N. H. and Chandrasekaran, B.eds. 1995. Diagrammatic Reasoning. MIT Press, Cambridge, MA.

[9] Kulpa, Z. 1997. Diagrammatic representation of Interval Space in Proving Theorems about Interval Relations. *Reliable Computing*, 3, 209 - 217, 1997. DOI = https://doi.org/10.1023/A:1009919304728

[10] Saltzer, J. H., Kaashoek, M. F. 2009. *Principles of Computer System Design*. MIT Press, Cambridge, MA.

[11] Chomsky, N. 1965 *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.