Simulator Design for Security Systems

Kaninda Musumbu
LaBRI (UMR 5800 du CNRS),
Université Bordeaux 1, France
351, cours de la Libération, F-33.405 TALENCE Cedex
musumbu@labri.fr

Abstract

This software is a simulator of security system developed with Java and providing a graphic interface built with Java Swing. This simulator is bound to be used to check an existing system or a system being currently devised.
A security system is represented by a logigram. A logigram consists of events linked by rules. The rules stand for triggering links between the events. The simulator provides a set of processes on security systems. Circuit search and useless rules search (subsuming rules) check the topology of the system. Two search through methods simulate the flow of triggering on events. The forward chaining stands for the simulation of the actual behavior of the system. The backward chaining looks for possible causes of the occurrence of an event. The simulator permits, thanks to those functionalities, the exhaustively check the design of a security system. Furthermore, the graphic display makes data handling user-friendly and ensures better understanding.

Key-words: Simulation, logigram, resolution, forward and backward chaining

1 Introduction

A system of security consists of elements or sensors allowing the detection of incidents intervening within a restricted environment. It manages the security using methods of release making correspond the activation of an element to a given time. With the detection of a whole of sensors, the system starts in cascade of other elements until its stabilization. The use of systems of security is generalized in the significant zones; there thus appears useful to check a system of security being, and especially essential to validate a system in the course of design. This software thus constitutes a tool of assistance to the checking of systems of security, through the simulation of its operations.

A system of security can be represented by a logigram. A logigram is a constitued by the set of rules of the links between events. These rules define the behavior of a system, where time is an essential data. This definition is an abstraction of the system.

The software that we carried out, ALOSE ("Automatisation de Logigrams de Sécurité")[1], is a tool for checking of systems of security. The checking is based on the simulation of the operation of the system. The checks carried out by this simulator rest on the abstract model of representation which is the logigram. Although the concept of time between concerned in simulation, it does not imply the checking in real time. Time is thus not the keystone of the simulator carried out. The checking carried out by the Alose software passes by three great stages, distinct but dependent. The first of all about a tool for checking automated of the topology of the system. This one is considered not ambiguous if there are not circuits in the release of the elements of the system, it makes it possible to carry out a back chaining, opposite simulation of the preceding one, generally from final events to the initial ones. This makes it possible to know the whole of the events which can be the source of the release of other events. The results of the two types of chaining carried out by the software constitute a source of relevant information for the user (designer). This software is based on a significant theoretical part (heuristic, complexities...). Indeed, simulation is operated on an abstract representation of the real system of security. This theoretical base presents the modeling of a system of security in the form of logigram. This abstract model is used for all the theoretical part

[1] We note Alose for conveniences of reading

of simulation. The objective of our paper is to discuss a number of important issues for the design of timed security systems and to contribute to give a backward semantics.

A system of security does not make it possible to tolerate possible inconsistencies. Hence, our software play a great rule in complement of a thorough design of the system, a means of validation. For that, it is necessary to be able to detect any conjunction of events being able to lead to a badly managed situation. The goal is to test in an exhaustive way the behavior of the complex systems. To test the operation of the real system in the event of incident is indeed a process rather not easily possible. On the other hand, a tool for simulation makes it possible to carry out tests which are at the same time multiple, fast and especially very easily skeletal. The cover of the tests is thus definitely more significant and much less expensive.

The results of the two types of chaining carried out by the software constitute a source of relevant information for the user. A logigram is a set of rules that define the behavior of such systems, where the time is very important. In these system, events are not raised or canceled, they are *scheduled*. To give a formal description of logigrams, at any time, the current event of system can be described by a simple structure called *script*. Scripts have many properties that seem to make them a useful support for working on problems like backward-chaining. The paper is divided in two parts. The section 2 gives the basis In section 3, we give a methodological overview of our approach. We insist of its wide applicability and theoretical soundness. Then, in section 4, we provide a short discussion about the nontrivial problem of backward-chaining.

2 Simulation

The simulation of the system aims at testing the operations of the system of security. The user will visualize the sequence of the release of the events in order to check the correctness of the system. This simulation is an observation of the behavior of the system. It must make it possible to detect a possible abnormal operation.

By parameterizing simulation, the user can limit his observation to a subset of the system. This enables him, by stage successive to refine its observation and thus to determine in a precise way the driving context with an abnormal operation.

Marking time makes it possible to follow the evolution of the state of the system. It scheduling reflects the constraints to trigger events. Marking the time produces a simulation very close to the real operation of the system of security.

3 Notations and Formalisms

3.1 Scripts

Let E be the set of **events** Let $\mathcal{N}$ be the set of natural integers. We call **time** any element of $\mathcal{N}$.

Definition 1 A **script** is a total function

$$s : E \longrightarrow \mathcal{N} \cup \{\infty\}$$

Let e be an event ($e \in E$). $s(e)$ is the **setting time** of e. If $s(e) = \infty$ we say that e is not scheduled.

Example 1 Let $\{(a,1),(b,2)(c,0)\}$ be a script. It means that events have to be set in the following condition: a at time 1; b at time 2; c at time 0.

Let S be the set of scripts and $\sqsubseteq$ the relation on the over scripts, such that

$$s_i \sqsubseteq s_j \leftrightarrow \forall e \in E, s_i(e) < s_j(e)$$

It is obvious (clear) that $\sqsubseteq$ is an order. Moreover, $\sqsubseteq$ is well-formed. We can also define the preorder $\sqsubseteq$:

$$s_i \sqsubseteq s_j \Leftrightarrow \forall e \in E,_i (e) \leq s_j(e)$$

The order $\sqsubseteq$ is not total, but we can define the operator $\sqcap$, such that

$$s = s_i \sqcap s_j \leftrightarrow \forall e \in E, s(e) = min(s_i(e), s_j(e))$$

It is easy to see that $\sqcap$ is the greatest lower bound of s_i and s_j. Moreover, $\sqcap$ is always defined. Thus, the set of scripts is a lattice. If we generalize the definition of $\sqcap$ by saying that $\sqcap_{s \in S} s = s_{min}$ such that for each $e \in E$, $s_{min} = min_{s \in S} s(e)$, then S is a complete lattice. We can then define $\perp$, such that $\perp \in S$ and $\forall e \in E, \perp (e) = 0$. So, $\perp$ is the least element of S.

3.2 Rules

Definition 2 A **rule** R consists of: a set $condition(R)$, an integer $delay(R)$, an event $consequence(R)$ such that $condition(R) = C \in \mathcal{P}(E)$, $delay(R) = d \in \mathcal{N}$, $consequence(R) = c \in E$. noted $C \xrightarrow{d} c$.

Definition 3 A **normalized rule** is a rule defining by only one *condition* and only one *consequense*.

3.3 Logigram

Definition 4 A **logigram** is a set of normalized rules.

Remark 1 This definition is sufficiently expressive. Indeed, if a rule has disjunction in its condition, or conjunction in its consequence, we can write it as a set of several rules: $c_i \xrightarrow{t} d$

4 Semantics of Simulation

A simulation aim at deriving information about the actual operational behavior of a system. Let a basis model given by a set of rules, we can consider it as a transition system. The evolution of the system between events is make by an fix-point operator.

4.1 Rules applicable

4.1.1 One rule applicable

Let R be a rule. We define the function

$$latest_R : S \longrightarrow E$$

such that :

- $latest_R(s) \in condition(R)$
- $\forall e \in condition(R), s(e) \leq latest_R(s)$

$latest_R(s)$ is the event in $condition(R)$ of which the setting time in s is maximal. If for a given s we have $latest_R(s) = \infty$ then at least bone of the events in $condition(R)$ is not scheduled in s: we say that rule R is **not applicable** for s.

Definition 5 Let R a rule. An **application** of R is a function $p_R : S \longrightarrow S$, such that :

- if $\exists e \in condition(R), s(e) = \infty$ then $p_R(s) = s$
- if $\forall e \in condition(R), s(e) < \infty$ then $p_R(s) = s'$,where s' is a script such that:
  - $\forall e \in E$ such that $e \neq consequence(R), s'(e) = e$,
  - $s'(consequence(R)) = min(s(consequence(R)), s(latest_R(s) + delay(R)))$.

If an event on the condition of R is not scheduled, that is, if $latest_R(s) = \infty$, then R is not applicable. So $p_R(s) = s$. If R is applicable, then it only changes the setting time of the consequence of R: the new time is the time of latest event in the condition added to the delay of R. But this is only applied if this new time is lower than the old one.

Remark 2 For all R and s, $p_R \sqsubseteq s$. If R is applied, the setting time of $consequence(R)$ is decreased. A setting time is never increased.

4.1.2 Application of a set of rules

Let $\mathcal{R}$ be a set of rules. The application of $\mathcal{R}$ is the application of all rules in $\mathcal{R}$ simultaneously, as defined below:

$$\begin{array}{rll} p_R : & S \longrightarrow & S \\ & s \longrightarrow & \sqcap_{R \in \mathcal{R}} p_R(s) \end{array}$$

Property 1 Monotonicity. We can see that, for all script s and all set of rule $\mathcal{R}$, we also have $p_R(s) \sqcap s$. Indeed, setting time never increased.

Fix-points A fix-point of p_R is a script s_{fix} such that $s_{fix} = p_R(s_{fix})$, *i.e.* if R is a rule of $\mathcal{R}$, either R is not applicable to s_{fix}, or R has no effect on s_{fix}.

Remark 3 $\perp$ is a fix-point of p_R for any $\mathcal{R}$. In fact, any rule is always applicable to $\perp$, and $\perp(e) = 0 \; \forall e \in E$. So, $\perp$ will not be modified after the application of any rule. Of course, for all $\mathcal{R}$, $\perp$ is the least fix-point of p_R.

4.2 Forward-chaining

4.2.1 Motivation

The implementation of the forward-chaining is done in a practically intuitive way starting from the structure of the logigram used. The starting point of a chaining is a script. This script describes the initial state of the logigram,(i.e. the initial times of release of events). The evolution of the logigram will be carried out starting from this initial state. Thus, one can consider that script constitutes the means of parameter setting of the system. It is significant to understand that an initial script describes the planning of events in the course of time. Thus, a script describes in fact in an exhaustive way which had releases has external constraints with the system. Indeed, in a system of security, the initial plannification of the release of events cannot be modified. The parameter setting in the course of forward-chaining corresponds to a processing developed to improve the possibilities of tests.

4.2.2 Evolution of the system

The system evolves/moves starting from its initial state. This evolution is done by stage. Each stage corresponds to the release of new rules. Let us sketch the approach from a general point of view. The operational semantics of any system consists of (or can be rephrased as)

1. a set of events Σ (events are noted σ);
2. an immediate transition relation between events denoted $\longmapsto$ where $\sigma \longmapsto \sigma'$ means that σ' is a possible successor event of σ;
3. a delay associate with transition which presents a temporal constraint. unary predicate on events, denoted final(σ),
4. a unary predicate on events, denoted final(σ), meaning that execution terminates in event σ.

The following semantics characterizes the set Output(S) of final events reachable from an initial set of events S:

$$\begin{array}{rcl} \text{Output}(S) & = & \{\sigma : \sigma \in S \text{final}(\sigma)\} \\ & \cup & \text{Output}(\{\sigma' : \sigma \in S \& \sigma \longmapsto \sigma'\}) \end{array}$$

This recursive definition is computable by usual recursive evaluation for two reasons at least: S is finite and sequences of transitions are finite.

However it can be given a mathematical meaning as the least fix-point of transformation τ such that

$$\begin{array}{rcl} (\tau f)(S) & = & \{\sigma : \sigma \in S \text{final}(\sigma)\} \\ & \cup & f(\{\sigma' : \sigma \in S \sigma \longmapsto \sigma'\}) \end{array}$$

It is straightforward to show from the operational semantics that this definition correctly maps each set of events onto the set of final events reachable from them.

In the case of our simplified language, we define two global operations[2]:

$$\begin{array}{rcl} \text{FS}(S) & = & \{\sigma : \sigma \in S \& final(\sigma)\}, \\ \text{DR}(S) & = & \{\sigma : \exists \sigma' \in S : \sigma' \longmapsto \sigma\}. \end{array}$$

We can then rephrase the definition of τ :

$$(\tau f)(S) = \text{FS}(S) \cup f(\text{DR}(S)).$$

The above presentation looks very simple because it was assumed that relation $\longmapsto$ and predicate final are primitive operations of the language. Let $\mathcal{L}$ be the complete logigram(that is the set of all defined rules). Let us consider the iteration of $\tau f_{\mathcal{L}}$:

$$\tau f(s) = f_{\mathcal{L}}(f_{\mathcal{L}}(\ldots(f_{\mathcal{L}}(s)\ldots))$$

We deduce from the property 1 that, for any script s, the transformation τf is well defined, *i.e.* the iteration of $\tau f_{\mathcal{L}}$ converges (in a finite number of iteration). In other words, if we consider the sequence $s_0, s_1, \ldots,$ such that $s_0 = 0$ and $\forall n > 0, s_n = f_{\mathcal{L}}(s_{n-1})$, we can say that there exists a N such that $s_N = s_{N-1}$, *i.e.* s_N is a fix-point of $\tau f_{\mathcal{L}}$. Indeed, we know that the sequence is decreasing. So, if we suppose there is no fix-point, then the sequence(s_n) is strictly and infinitely decreasing, which is not possible since the order $\subseteq$ is well-founded. $\tau f(s)$ is the result at of forward-chaining of the script s. It is obvious that forward-chaining is not invertible.

4.3 Backward-chaining

4.3.1 Problems of Backward-chaining

The backward-chaining is the most significant functionality of our simulator. It's principal problem is the great number of operations to be carried out. During the forward-chaining, the possibilities tend to be reduced progressively. It is not the case of the backward-chaining because of the conjunctive nature of rules, in progress of process, the possibilities will be growing. Indeed, an event can have many causes. Within the framework of the project, only the primary causes are significant. The objective is thus only search of the initial events leading to the events given.

Assuming that we can easily compute forward-chaining, it is also easy to verify if a given script s is such that $\tau f(s) = s'$ (where s' is the goal script).

The first problem of the backward-chaining is that there is a great number of potential solutions. We have to determine which ones are really useful. This could be done by clearly identifying properties of such scripts. For instance, we can say that candidates for being a solution should be "initial" events, assuming such events are defined (a subset of these events could be spotted interactively by the user). It is necessary to identify them, and to be able to compute efficiently the classes.

With no further theory, all that we can do is to follow backward the rules by making hypothesis, and eliminates wrong scripts while computing. These is similar to the O'Keefe's algorithm. [5] solves finite sets of equations of the form $x_i = expr_i \quad (1 \leq i \leq n)$ where $x_1, \ldots, x_n$ are distinct variables, ranging on lattices of finite depth $T_1, \ldots, T_n$, and $expr_1, \ldots, expr_n$

[2]FS and DR stand for "Final Events" and "Directly reachable", respectively.

are well-typed monotonic expressions possibly involving $x_1, \ldots, x_n$. The algorithm computes the least fixpoint of the transformation

$$\begin{array}{rlll} \tau: & T_1 \times \ldots \times T_n & \mapsto & T_1 \times \ldots \times T_n \\ & \langle x_1, \ldots, x_n \rangle & & \langle expr_1, \ldots, expr_n \rangle \end{array}$$

It proceeds as follows. Variables x_1 to x_n are initialized to $\perp$ and pushed onto a stack. Then variables are popped from the stack until it becomes empty. Each time a variable x_i is popped its value is recomputed. If the new value is greater than the previous one, all variables that depends on x_i and are not on the stack are pushed on it.

One of the remarks we can have is that, by this method, "eligible" script can be computed recursively. The following semantics characterizes the set Input(S) of initial events from which we can reach a goal(event) S:

$$\begin{array}{rcl} \text{Input}(S) & = & \{\sigma : \sigma \in S \& \text{initial}(\sigma)\} \quad \cup \\ & & \text{Input}(\{\sigma' : \sigma \in S \ \sigma' \longmapsto \sigma\}) \end{array}$$

This recursive definition is not computable by usual recursive evaluation for more reasons at least the number of sequences of transitions is exponential .Let us notice that, if one carries out a chaining starting from all the final events of a logigram, all the events will be traversed. Since for each traversed event all the rules coming with an event will be examined. If one counts the number of addition in $\sqcup$, it will be equal to:

$$\sum_{r \in \mathcal{R}} card(conditions(r))$$

In other words, complexity is equal to the number of rules multiplied by the average number of antecedents per rule, or the total number of branches of rules. For a logigram of big size, if the average number of antecedents per rule is high and/or if the number of rules is large, complexity will be an obstacle with the simulation.

4.3.2 Introduction of pruning

For a logigram having many branches of rules an improvement is possible. Indeed, when an event is added to the list $\sqcup$, all the branches of rules which would make it possible to go up until this same event by another path are likely to complicate calculations. Consequently, it is useful systematically to remove all the branches of downward rules starting from this event. This pruning can also be continued in a downward way. Indeed, if one of these suppression places an event in position of initial event, then this one is useless with the chaining (it does not have a previous initial state). One is thus capable effective, starting from this event, same pruning without losing solution. Pruning can then continue in a downward way. Added to the preceding algorithm, this optimization does not find its optimality. This is due to the fact that the course is done in width. It results from this that pruning removes most of the time branches of already traversed rules. One can then reorganize the course so as to exploit this new fully improvement. The ideal would be to arrive as soon as possible at the first initial event, bus from this one the pruning would remove all the other paths which leads only to it, then to quickly find the second initial one and so on An in-depth course makes it possible to arrive at this result.

The new algorithm thus obtained has the double advantage of the strong reduction of complexity (see below), and to remove some choices of rules not influencing the result of the chaining. However it is not necessarily optimal, indeed:

- of the useless courses is always carried out.
- of the non-relevant choices is sometimes proposed to the user.
- complexity varies according to geometry of the logigram.

The logigrams of the figure 1 illustrate the lower limit of the utility of pruning. However, for this kind of logigrams, complexity is close to the number of events. Such a complexity is completely reasonable, one can conclude from it that pruning does not complicate to in no case the back chaining.

4.3.3 Characterization of the algorithm

1. **Comparison with the naive chaining**
   The two algorithms has a similar complexity on a particular type of logigram: when the number of events is close to the number of branches of rules. Indeed, such a logigram is a practically arborescent structure starting from the initial states. In such a logigram few events present several ascending branches.

   For these type of logigrams, pruning does not bring anything again. Indeed, it is necessary to traverse all events in order to arrive at the initial. However, for this kind of logigrams, complexity is close to the number of events. Such a complexity is completely reasonable, one can conclude from it that pruning does not complicate to in no case the back chaining.

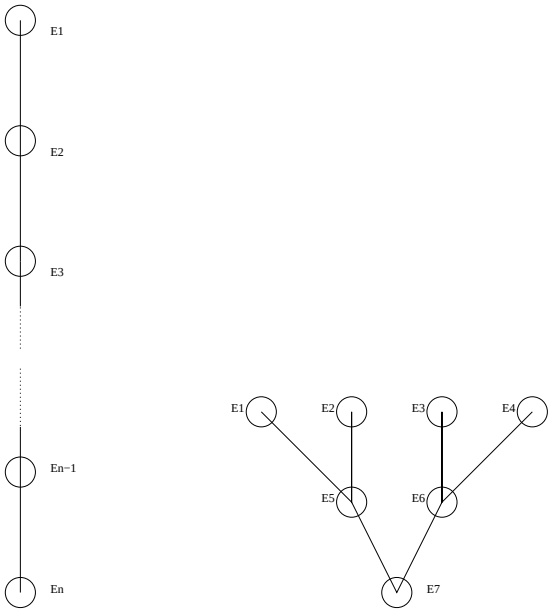

Figure 1: For these two logigrams, pruning does not bring anything again. Indeed, it is necessary to traverse all events in order to arrive at the initial events.

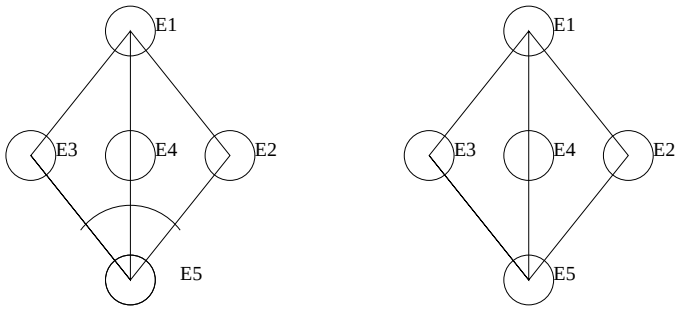

Figure 2: Application of the backward-chaining on the logigram of left comprises 5 against operation 6 for that of right-hand side This highlights the average number of antecedents by rules influences complexity.

2. **Variations of complexity according to the logigram**

   This algorithm of back chaining has a complexity varying with the geometry of the logigram. The logigrams of the figure 2 are two cases having the same number of branches of rules and for which the back chaining displays a different effectiveness.

   This variability comes owing to the fact that when a rule has several antecedents, it is not necessary to be interested in its started event as long as the rules has still at least a branch. Thus the performance go increasing with the number of antecedents by rules.

3. **Variations within the same logigram**

   The logigram of the figure 3 is a case of variability.

   It is thus noted that the command of course is determining. Even if this variability is low, one could imagine logigrams where it would be significant. For example on the figure **??** by adding events similar to `E3 E4` (dependent them also on `E2 E5`) one can increase the differences in results. This consideration carried out us to carry out tests on average on the result.

The optimality of the chaining never postpones

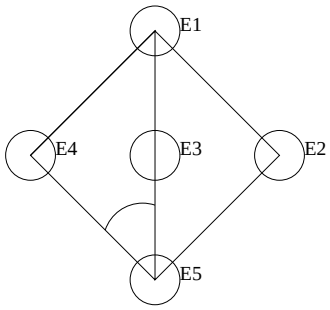

Figure 3: In this logigram, the first branch of rule chosen starting from `E7` determines calculation. If the branch of right-hand side is traversed in first, the branch of left will be traversed only by pruning. However this last operation is less expensive than the back course. What would complicate a departure on the left is the fact that `E5` has two antecedents. Let us note moreover than on this logigram, the chaining will propose a choice not justified

.

with pruning don't be assured. The result will be always right but aspects like the complexity and the choices suggested to the user are not necessarily the best. As we highlighted, these questions of optimality find variable results for the same logigram, this constitutes a **heuristic** introduced by pruning.

4.3.4 Complexity of the chaining postpones

One can note that for the back chaining, the two principal factors of complexity are the number of branches of rules and the average number of antecedents by goal. Any event will be examined once at least during a back chaining starting from all the event terminals. For each rule, its started event will be examined once more if the last branch is removed while going down (at the time of an iteration of pruning). Thus one obtains an upper limit of complexity of the back chaining which is the sum of the number of events and the number of rules. If it is considered that these two numbers are bound, complexity is in $\mathcal{O}(2n)$ or n is the number of events. The criteria of complexity chosen here are not most relevant as soon as the number of rules is much higher than the number of event. However, such a logigram would not be really of interest for a system of fire protection.

Conclusion

We describe a model for managing time-oriented security system and methods to perform simulation on this systems like forward and backward-chaining. Ad hoc solutions may be obtained but often fail to address various issues. Or method in practice, provides a new trend for design and verify a security system, to ensure themselves of maintain integrity walls of compartments and to ensure of the correct operation of those. while verifying that the good configuration of the system allows resister well have temperatures predefine. Forward-chaining can be computed efficiently by having two remarks: We never have to apply any rule more than once and at any moment of the simulation, we can say what rules are applicable. Indeed, it is easy to say to know as each event is set, when any rule become applicable. The be st strategy minimizes the amount of computation (roughly speaking, the number m of iterations). Clearly, this is very problem dependent. The model described above is still being implemented using the monotonicity property, we restrict search domain into intervals bounded by fix-points By clearly identifying and study the properties of backward-chaining instance, we hope to gain efficiency when computing them A certain amount of improvements can be done in the same way, but we see that the rule-backward-following method has defaults. We can add two more:

- a relatively high complexity. By selecting eligible nodes(which suppose a verification for each node, by computing forward-chaining), we get the answer in a quadratic time per solution;
- as a solution, we may get redundant scripts. Even if we manage to efficiently check script equivalence, it remains that we shall be computing useless branches of the hypothesis tree, which makes our complexity problem heavier.

In order to improve accuracy and complexity, it is better to look for a good characterization of scripts, which seem to be actual reasonable structure to work with.

References

[1] R. Alur and A.L. Dill A theory of timed automata. *Theoretical Computer Science*, 126(2):183-235, 1994.

[2] E. W. Mayr An algorithm for the general Petri net reachability problem *SIAM J. Comput.* 13(3):441-460, 1984

[3] B. Le Charlier and P. Van Hentenryck. A universal top-down fixpoint algorithm. Technical Report 20/92, Institute of Computer Science, University of Namur, Belgium, (also Brown University), April 1992.

[4] K.F. Man, K.S. Tang and S. Kwong Genetic Algorithms Springer 1999

[5] R.A. O'Keefe. Finite fixed-point problems. In J-L. Lassez, editor, *Proceedings of the Fourth International Conference on Logic Programming (ICLP'87)*, pages 729–743, Melbourne, Australia, May 1987. MIT Press.