Títol: Online algorithms for change detection in information systems and web environments

Volum: 1

Alumne: Iván Nikolic Fraguela

Director: Josep Carmona Vargas

Codirector: Ricard Gavaldà Mestre

Departament: Llenguatges i Sistemes Informàtics

Data: 21/06/2013
DADES DEL PROJECTE

Títol del Projecte: Online algorithms for change detection in information systems and web environments
Nom de l'estudiant: Iván Nikolic Fraguela
Titulació: Enginyeria Informática
Crèdits: 37,5
Director: Josep Carmona Vargas
Codirector: Ricard Gavaldà Mestre
Departament: Llenguatges i Sistemes Informàtics

MEMBRES DEL TRIBUNAL (nom i signatura)

President: Enric Rodríguez Carbonell
Vocal: Maria Angela Grau Gotes
Secretari: Josep Carmona Vargas

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.2 Adwin Methods</td>
<td>74</td>
</tr>
<tr>
<td>A.3 Swig</td>
<td>75</td>
</tr>
<tr>
<td>A.3.1 Example of Swig usage</td>
<td>76</td>
</tr>
<tr>
<td>A.4 Python Snakes</td>
<td>78</td>
</tr>
<tr>
<td>A.4.1 Example</td>
<td>78</td>
</tr>
<tr>
<td>A.5 Flask</td>
<td>79</td>
</tr>
<tr>
<td>A.5.1 Example</td>
<td>79</td>
</tr>
<tr>
<td>A.6 JQuery</td>
<td>79</td>
</tr>
<tr>
<td>A.6.1 Examples</td>
<td>80</td>
</tr>
<tr>
<td>B Instructions</td>
<td>81</td>
</tr>
<tr>
<td>C Drift Detector Python</td>
<td>82</td>
</tr>
<tr>
<td>D Example of a PNML file</td>
<td>90</td>
</tr>
</tbody>
</table>
1 Introduction

The development process for information systems starts by having a set of requirements that the final implementation must cover. Then the design of the system starts, and a formal model is built in order to specify each one of the components of the system. This can be done using formal languages such as UML. The programmers that write the implementation use that model to guide the development of the system.

The process does not stop there, usually a system is modified a lot of times after it’s first implementation. This can be to fix bugs of the first iteration, or to add new functionalities to the system. The problem here is that during one of this iterations an unexpected and unwanted change could happen to the system, where the programmers inadvertently changed something in the system that makes it differ from the original specification. This kind of changes could be very hard to detect in some cases, and they have to be detected as soon as possible since the system is not doing what it is supposed to do.

In this project we will explain how to detect this changes quickly and in an automatic way. The tools used are part of a discipline called Process Mining. A set of algorithms are going to be used to reach this goal and an implementation of those algorithms will be provided.

1.1 Process Mining

Process mining is a recent discipline. It has two main components, the first one sits between data mining and computational intelligence, and the second one is centered in process modeling and analysis. The main idea of process mining is to analyze and discover processes. The information extracted from it can be used to improve the analyzed process. The information to do such things is
extracted from event logs. One of the advantages of process mining is that most of today’s big systems already log their operations, so the data needed to start working is already there. In process mining there are different goals that can be achieved, there is automatic process discovery, conformance checking, organizational mining, automated construction of simulation models, case prediction, model extension, model repair, and history-based recommendations.

Automatic process discovery is the part that tries to extract a model of the process that is being executed through the information extracted from the log. This model can be represented in different ways, but a common one is using Petri Net. This model can help to understand better what the process is really doing, and it could be useful to fix bugs or improve the program that is being analyzed.

Conformance checking means comparing the log and the model, and check for deviations in their execution. You can then decide if the system is still functioning as expected or if it changed it’s behavior. This is useful to detect bugs that were inadvertently inserted in the code. This is the part on which this work is going to expand more.

Organizational mining[1] aims to extract information of the organization of a company just by looking at the logs of their information systems. The knowledge extracted represents the organigram of the company, such as which departments are above which others, or who is the boss of some employee.

One of the things you can do with a model of the system is use it to perform a simulation. This simulations could have different uses, for example it could be useful to predict possible executions of the system. If the model represents the behavior of a group of people using the system instead of the system itself (e.g how a group of users navigate through a website) this can be used to predict their behavior beforehand and then react using this knowledge to improve their experience.
With model extension and model repair[1] basically the model of the system is used to add more functionality missing from the original one and change parts of it that were not intended to be.

The model can also be used to give some insights on the system from the history of the execution, and then do some recommendations to improve it.

### 1.2 Concept Drift Example

An information system is constantly evolving, having bugs fixed and new features added. During this evolving process some unwanted changes could be added to the system. Whenever one of those changes happen, we say that the system suffered a concept drift.

To understand this better we are going to use an example.
This models are Petri nets, they are explained in deep at section 2.5. This model represents the flow of the possible actions that a user can do. It defines which events have to be sequential and which events can be done concurrently.

Lets start with the first model. The process always start with Register, and then there are two different paths to follow. This means that the different paths can be done concurrently, there is no precedence between the path above and the path below.

After the registration we can do for example ”Create Questionnaire” and then continue with ”Decide High/Low”. Then we can continue, and the different possibilities are ”High Claim Split”, ”Low Claim Split” and ”Send Questionnaire”. When there are two different paths converging we have to wait until both paths are complete before being able to continue with the path after the join.

A trace is ordered list of events that happen in a model. The model only accepts a set of traces, this are some of the traces that the first model accepts:

- Register, Create Questionnaire, Send Questionnaire, Decide High/Low, Low Claim Split, Low Medical History Check, Low Insurance Check, Low Claim Join, Send Notification, Archive
- Register, Create Questionnaire, Decide High/Low, Low Claim Split, Low Insurance Check, Low Medical History Check, Low Claim Join, Send Notification, Send Questionnaire, Archive
- Register, Decide High/Low, High Claim Split, High Insurance Check, High Medical History Check, High Claim Join, Send Notification, Create Questionnaire, Send Questionnaire, Archive
- Register, Decide High/Low, High Claim Split, Create Questionnaire, High Insurance Check, High Medical History Check, Send Questionnaire, High Claim Join, Send Notification, Archive
Each trace is different but they all are valid in the first model represented above, because if two events are sequential they are always in the right order.

If we focus in the second model, we can see that there is a small difference in the top, before ”High Insurance Check” and ”High Medical History Check” where concurrent, the order didn’t matter, but now they are sequential, ”High Insurance Check” always goes first.

This means that are some traces that were valid in the first model and are not valid in this one, for example:

- Register, Decide High/Low, High Claim Split, High Medical History Check, High Insurance Check, High Claim Join, Send Notification, Create Questionnaire, Send Questionnaire, Archive
- Register, Decide High/Low, High Claim Split, Create Questionnaire, High Medical History Check, High Insurance Check, Send Questionnaire, High Claim Join, Send Notification, Archive

In the first model these two traces are okay, but in the second one they are not possible, because the order of the events ”High Insurance Check” and ”High Medical History Check” is not right. The first and second models accept a different set of traces. When this happens we say there is a concept drift, the model is not exactly the same, the behavior is now different. This is what the project is about, the goal is to create a set of tools to automatically detect these changes.

This is only one example of what a concept drift looks like. There are other kinds of concept drifts, for example a new event could appear, or an old one could disappear. Also, there are concept drifts that happen abruptly and others that slowly appear in the system. The later is more difficult to detect.
2 Preliminaries

We are going to define the notation that will be used for the rest of the document. $T$ is a set of events, and a log is defined as $L : T^* \rightarrow \mathbb{N}$. A sequence of $\sigma \in T^*$ is called a trace. A trace $\sigma$ is contained in a log if $L(\sigma) \geq 1$. A trace $\sigma$ is contained in a log if $L(\sigma) \geq 1$. Given a trace $\sigma = t_1, t_2, ..., t_n$, and a natural number $1 \leq k \leq n$, the sequence $t_1, t_2, ..., t_k$ is called the prefix of length $k$ in $\sigma$. Given a log $L$, Pref($L$) is the set of all prefixes of traces in $L$. Finally, $\#(\sigma, e)$ is the number of times that activity $e$ occurs in sequence $\sigma$.

2.1 Logs

All serious information systems have some kind of logging. This is the process of recording events from the system itself and then writing these events to a file as they happen. A lot of information can be written in logs. For example:

- Name of the event
- User that triggered the event
- Time of the event
- Number of concurrent processes in the system
- ...

All this information is stored in a file. This is the source of data from where the algorithms explained here were extracted to find useful information in order to detect a concept drift.

A preprocessing of the logs is done to filter the data and leave only the information that the algorithms explained here use. The information used is
the events in order, separated in traces. The order of can be extracted from the
time of each event. A trace is a list of this ordered events, where the first event
is the start of the interaction of a user with the system, and the last event is
the ending of this interaction.

A simple example of processed log can be:

- OpenGame ChooseLevel1 Play Win CloseGame
- OpenGame ChooseLevel2 Play Lose CloseGame
- OpenGame ChooseLevel2 Play Win ChooseLevel3 Play Lose CloseGame
- ...

This is an example of a preprocessed log of a generic game.

### 2.2 Parikh Vectors

**Definition 1** (Parikh vector). Given a trace $\sigma \in \{t_1, t_2, ..., t_n\}^*$, the Parikh
vector of $\sigma$ is defined as $\hat{\sigma} = (\#(\sigma, t_1), \#(\sigma, t_2), ..., \#(\sigma, t_n))$.

From each trace $\sigma$ of size $n$ in a log, we can extract $n$ prefixes. We can then
use each of the prefixes to obtain a Parikh vector. To do so we need to convert
each event into a dimension(a position in the vector). For example, from the
trace

OpenGame ChooseLevel1 Play Win CloseGame

It is possible to extract 6 prefixes
OpenGame

OpenGame ChooseLevel1

OpenGame ChooseLevel1 Play

OpenGame ChooseLevel1 Play Win

OpenGame ChooseLevel1 Play Win CloseGame

If this game has only 3 levels, a possible mapping from event to dimension can be:

- Dimension 1: OpenGame 1
- Dimension 2: ChooseLevel1
- Dimension 3: ChooseLevel2
- Dimension 4: ChooseLevel3
- Dimension 5: Play
- Dimension 6: Win
- Dimension 7: Lose
- Dimension 8: CloseGame

Each Parikh vector represents a point in a multidimensional space. From each trace in a log it is possible to extract $n$ such points, one for each prefix. Each point has the form $\hat{\sigma} = (\#(\sigma, t_1), \#(\sigma, t_2), \ldots, \#(\sigma, t_n))$. Then each Parikh vector of the prefixes stated before would be:

- $(0,0,0,0,0,0,0,0)$
- $(1,0,0,0,0,0,0,0)$
• (1,1,0,0,0,0,0,0)
• (1,1,0,0,1,0,0,0)
• (1,1,0,0,1,1,0,0)
• (1,1,0,0,1,1,0,1)

But also, each point can be seen as a polyhedron $P_\sigma = \bigcap_{i=1}^{n} (x_i = \#(\sigma, t_i))$. This means that for each prefix of a trace $\sigma$ there exists a polyhedron that represents it.

### 2.3 Abstract Interpretation

Abstract interpretation is useful to generate an approximated model of the data that is being analyzed. There are different ways to represent such an approximation.
Figure 3: Approximating a set of values with several abstract domains

The difference between them is that they allow more or less values as part of the model, the area representing the model is bigger or smaller. A bigger area means that the model is going to contain more errors, since there are points outside the real model that are part of this approximation. In figure 3 you can see some of the possibilities. The one that fits the best is Convex polyhedron, as it has a smaller area than the others and still contains the points of the set. The problem is that a Polyhedron is more complex than the other representations, because more constraints are needed to define it. This is clear in table 1.
In this work the models used to represent the data (in this case is the behavior of the system) are two, Convex Polyhedron and Octagon. Basically the algorithm will use polyhedra whenever it’s possible and if the data is too complex then it will use octagons, so it doesn’t take too long to do the calculations. This is because is more difficult to obtain all the constraints that are needed to represent the polyhedron, and as the complexity of the system being analyzed grows, the number of constraints needed for the model that represent the system also grows.

There are certain operations that can be done with Polyhedra and Octagons. In this Project we only need two:

Meet($\cap$): Given convex polyhedra $P$ and $Q$, their meet is the intersection $P \cap Q$. Notice that this operation is exact, e.g., the meet or intersection of two convex polyhedra is always a convex polyhedron.

<table>
<thead>
<tr>
<th>Interval</th>
<th>Octagon</th>
<th>Convex polyhedron</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq x \leq 3$</td>
<td>$0 \leq x \leq 3$</td>
<td>$y - x \geq 0$</td>
</tr>
</tbody>
</table>
| $0 \leq y \leq 5$ | $0 \leq y - x \leq 2$ | $3x - y \leq 4$
|  |  | $y - 5/3x \leq 0$ |

Table 1: Constraints (inequations) of the different models
Join(∪): Given convex polyhedra $P$ and $Q$, we would like to compute the union of $P$ and $Q$. Unfortunately the union of convex polyhedra is not necessarily a convex polyhedron. Therefore, the union of two convex polyhedra is approximated by the convex hull, the smallest convex polyhedron that includes both operands, denoted by $P \cup Q$. The example on the left shows $P \cup Q$ in gray.

Figure 5: Meet

From a log we can extract a set of Parikh vectors. Each Parikh vector can represent a Polyhedron. Then we have a way to use Abstract interpretation to represent the log of the system.

2.4 Adaptive Windowing

One of the most difficult problems in data and process mining is dealing with changes that happen in the data that is being analyzed. To mine or learn such data, one needs strategies for the following three tasks, at least: 1) detecting when change occurs 2) deciding which examples to keep and which ones to forget and 3) revising the current model when a significant change has been detected.

The standard way of dealing with this is to keep a window with a portion of the data, normally you keep the most recent examples and drop the old ones. The contents of the window can be used for the three tasks above: 1) to detect change, 2) to obtain updated statistics from the recent examples, and 3) to have data to rebuild or revise the model.
Normally you have to choose the size of the sliding window, and you have to decide if you want a small window to have an accurate version of the current distribution, or if you want a large window to have more examples to work on and have more accuracy in stability periods.

The importance of Adaptive Windowing is that the user does not have to specify a size for the window, the window will grow and shrink automatically to have a better representation of the distribution. In periods of stability the window becomes large, therefore obtaining accuracy, and if the distribution is changing the window becomes small to have a good representation of the last change of the distribution.

2.4.1 Adwin

Adwin[2] is an implementation of adaptive windowing (ADaptive WINdowing). The algorithm has a sliding window that has a dynamic size, this means that the size is recomputed according to the rate of changes observed in the data of the window itself. If the data remains stable for a long period then the window will become large to have more precision, but if changes are taking place in the distribution of the data then the window shrinks so it’s able to discard old data. The creators of Adwin provide rigorous guarantees of performance as rates of false positives and false negatives. The algorithm is time and memory efficient.

The input for the algorithm is a confidence value $\delta \in (0, 1)$ and a (possibly infinite) sequence of real values $x_1, x_2, \ldots, x_t, \ldots$, where each $x_i$ is a real number between 0 and 1. Each $x_t$ is only available at time $t$ and they are generated according to a distribution $D_t$. The fact that each $x_t$ has to be between 0 and 1 does not limit the algorithm since a simple rescaling is enough to be able to use it. Other values of the distribution such as $\mu_t$ or $\sigma^2_t$ are unknown for all $t$.
2.4.2 Overview of the ADWIN algorithm

The algorithm keeps a window $W$ and it keeps getting larger while $\mu_t$ remains constant in $W$. To check if this condition holds true the window is divided in two "large enough" consecutive windows $W_0$ and $W_1$. A value $\mu_W$ is calculated for each of the windows and when the difference between those to values is greater than a threshold, then the older part of the window $W$, $W_0$ is dropped. To decide the threshold for the difference or when a window is "large enough" an appropriate distribution has to be chosen, so that the values are precise. This decision often involves the value $\delta$. To get a more deep explanation of the algorithm and its proof look at [2].

2.5 Petri Nets

**Definition 2** (Petri Net). A Petri net is a tuple $(P,T,F,M_0)$ where $P$ and $T$ represent finite sets of places and transitions, respectively, and $F : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ is the weighted flow relation. The initial marking $M_0 \in \mathbb{N}^{|P|}$ defines the initial state of the system.

Petri nets are used to model a distributed system. The representation of a Petri net is a directed bipartite graph. There are two type of nodes in the graph,
there are nodes that represent places and nodes that represent transitions. Each node contains a number of tokens and each transition consumes tokens from $n$ different places and produces tokens towards $m$ different places each time it is fired. A transition can never be fired if the required tokens are not available at a given time. The firing of a transition is atomic. If there are different transitions available then either of them can be fired, the order in this case is not defined and is representing parallel execution. Therefore they are well suited to represent concurrent distributed systems. This different kinds of nodes are connected by arcs, and depending on this arcs a transition gets tokens from a subset of places or puts tokens to other subset. Any given distribution of the tokens on the Petri net is called a marking.

Example of a Petri Net:

In this example we can see 4 places (P1,P2,P3,P4) and 2 transitions (T1,T2). The arcs define the connection between them. The black balls inside each place represent the tokens. In this case we can see that P1 has 1 token, P2 has no tokens, P3 has 3 and P4 has 1 token.

For T1 to be available (i.e. it is possible to fire T1) we need to have at least one token in P1, since there is an arc from P1 to T1 that represents the dependency. After T1 is executed P2 would have 1 token and P3 3 tokens, since there is an arc from T1 to P2 and to P3.
At the beginning it was only possible to fire T1 since T2 didn’t have all the tokens to be fired. Now, after firing T1 its possible to fire T2 since the tokens are now available.

We can use this kind of modeling to represent the systems that are being analyzed. Part of the project is simple program that uses the Abstract Interpretation model as a source to generate a Petri Net.
3 Online Concept Drift detection

The online concept drift detection algorithm is the core of this work. All the information given here is based on [3]. The algorithm has 3 parts, in the first one it learns a model from a set of traces of a log. After that the algorithm does an estimation of the model. Finally it monitors the log and compares the estimation to the one obtained in the previous step.

3.1 Learning the model

The algorithm will then generate \( n \) polyhedrons for each trace.

A polyhedron \( P = \bigcup_{i \in \{1...m\}} P_{a_i} \) is then built from the polyhedrons obtained before. The operation to merge polyhedrons is calculate the convex hull that contains both of them.

During the learning phase of the algorithm, a polyhedron is created by merging all the polyhedrons form the prefixes of all the traces used for learning. Since this is an expensive operation, subsampling is used and only a part of the log is used to calculate the polyhedron that will be used as the abstract interpretation.
One of the main problems here is how to know when to stop the learning phase. A simple way is just using a predefined number of traces to use for the learning process. But this number may be different for different systems, making it difficult to adapt the algorithm quickly. Other way could be using relative numbers, such as using 1/3 of the log as initialization. This decision is going to be made during the experiments, depending on the results of each method.

3.2 Estimation

Once the model is learned then the next step is the estimation step. The polyhedron learned before represents a set of traces, the traces from the subsampling used to generate the model.

Using another set of traces, not used in the learning stage, the algorithm creates new points. Each point is a Parikh vector, and it can be used to create a polyhedron, just like before. So again with each prefix of each trace $\sigma$ a polyhedron is created, and then using the algorithm checks if whether this new polyhedrons are inside or outside the one generated during the learning process.

This boolean information is then sent to the Adwin instance, and it calculates the estimation for the value. in this case it will be a number from 0 to 1.

The estimation step continues to do this until the estimation is stable. To define the stability of the estimation, the user has to set two parameters. The precision for the estimation $\alpha$ and the number of points that the estimation has to be stable for $n$ points. Then the estimation is done and after being stable for $n$ points without changing more than $\alpha$ at each update it finishes. In case that the estimation reaches the end of the log before stabilizing then the algorithm...
ends, with an output saying that it couldn’t stabilize.

3.3 Monitoring

At last, after the model is estimated then the monitoring phase begins. During this phase the algorithm behaves much like in the estimation step, but with a few changes.

It creates polyhedrons from each of the prefixes of the traces it reads from the log, and then checks whether this polyhedrons are contained in the learnt polyhedron from the learning step. This boolean information is then sent to the Adwin instance, doing an update operation. This update operation returns a boolean value that says if there was a drift after the last update. The monitoring phase is executed until a drift is detected or after the end of the log is reached. If a drift is detected then the algorithm restarts from the beginning. If the end of the log is reached the the algorithm stops with an output saying that there is no drift in the log.

3.4 Multiple Adwin

Until now all the work described here was just a new implementation of the drift detection algorithm. In this section we are going to talk about an extension of the algorithm, developed during this project. This extension is intended to add precision to the original algorithm, so it’s easier to find drifts in logs.

This extension uses multiple Adwins to know when a drift is found in a log. Instead of calculating if the polyhedron represented by a point is inside the main polyhedron, this version uses the main polyhedron constraints.

Each constraint of a polyhedron represents a face in the multidimensional
space that contains the polyhedron. Then instead of checking if the new polyhedron fits inside the main polyhedron, this version checks whether the new polyhedron crosses one of the main polyhedron boundaries (faces) or not. In practice this is the same as before, since if a polyhedron crosses one face then it is not inside the polyhedron. The difference is that this version is more precise, since it is possible to know exactly which constraint is not satisfied and then you can extract more information from it. It could be used to mark a part of the Petri network generated later for example, so it’s easy to know where exactly is the system having this changes that ended up producing a drift.

Figure 8: Multiple Adwin Example
3.5 Drift detector for websites

Another extension made to the original algorithm is the web version. This enables the algorithm to gather information from websites to generate the log. This version is distributed since the events happen on the computer of the users that are visiting the website. The main purpose of the algorithm is the same, detect concept drifts in a log, but instead of tracking an information system, this algorithm works with websites.

* The Web Logger is a Javascript program, so it is executed in the computer of each user and not in the website server.
The first problem that arises here is that using web logs from http servers is not good enough, since the information given is vague in terms of what the users are doing. In the website version of the algorithm instead of detecting drifts in a program what it tries to do is detect changes in the behavior of the users that visit the website.

Since web servers don’t give information about the particular users visiting the website (they mostly log information about the server itself) then another way of getting logs is needed.

3.5.1 Web logger

Web logger is a new program developed to track the users from a website. It basically generates traces from each user that navigates through the website that is being tracked and then sends the information to a server that keeps all the traces from different users. This traces are finally merged in one log so the normal online drift detection algorithm can be used. The client part is executed as a script in the clients browser. More details are given in section 4

3.5.2 Server

The server for the web version keeps track of all the events sent from the web logger and stores all the events from the same user and website under the same file. When a trace is finished then it’s appended to the end of the website log. A log is considered as finished if the user leaves the webpage for more than 5 minutes or if a user doesn’t have events for 2 hours.

Since this version has to keep track of websites it has multiple logs, one for each website that are being constantly updated. The server then goes from log to log, loading the information to work with each website and processing the log.
until the end of it is found, then it goes to the next one. This loop works great for websites, because the logs here are asynchronous, so while you wait until one log gets more traces you can work with other logs. You could also have an instance of the algorithm program for each log, but then with a lot of websites the amount of memory needed to have all the models at once would be a lot.

3.6 Model generator

In this project the main work is the concept drift detection algorithm, but there is also a small algorithm for the process discovery part in process mining. This algorithm generates a more complete model from the abstract interpretation model of the drift detection algorithm.

The model generated is a Petri net. Petri net are a really useful way to represent systems. It’s easy to have concurrent and distributed systems written as a Petri net. The information extracted from the abstract interpretation model comes from the constraints of each model. From each constraint you can generate a place, you can know the number of tokens needed from each place and the distance between the transition and the places.

Not all the information that can be extracted from this is useful, and then we have to deal with a lot of not relevant information inside the model. To improve this, the algorithm filters the constraints that are not important for the final model, and uses only the important information.

For each constraint of the abstract interpretation model a transition is created. Each constraint is an inequation, for example:

\[ e_1 - 3e_2 + 2e_5 \leq 3 \]

If we split the coefficients into positive and negative coefficients, constraint
$i$ can be represented in the following way:

$$
\sum_{a_{ij} > 0} a_{ij} x_j + \sum_{a_{ij} < 0} a_{ij} x_j \leq b_i
$$

that can be transformed into:

$$
\sum_{a_{ij} > 0} a_{ij} x_j - b_i \leq \sum_{a_{ij} < 0} -a_{ij} x_j
$$

A constraint $i$ is a *causality constraint* if the following conditions hold:

- There is at least one positive coefficient, and
- $b_i \leq 0$

Hence causality constraints can be described as:

$$
\sum_{a_{ij} > 0} a_{ij} x_j - c_i \leq \sum_{a_{ij} < 0} -a_{ij} x_j
$$

with $c_i = b_i \geq 0$. The intuition behind causality constraints is that they represent real causalities observed in the log which can be explicit in the derived PN. Hence if we assume indexes $n_1, ..., n_k$ range over the indexes of variables with negative coefficients and $p_1, ..., p_l$ range over the variables with positive coefficients, it can be modeled in a PN as:

![Diagram](https://via.placeholder.com/150)

where $c_i$ inside the place denotes $c_i$ tokens, and $a_{ij}$ in an arc represents the weighted flow relation $F$ for the arc.
4 Design & Specification

4.1 Goals of the project

The goal of this project is to have a functional implementation of the online drift detection algorithm, with its web version. Also, an implementation of the model generator algorithm will be written to, which can be used to generate the models for the system both after and before a drift.

The requirements are:

- A program or script that can create a model from a log, do an estimation of this model and then monitor it until it finds a drift in the log
- A web client that can send events to a server to create new logs
- A web server that stores the events from each user of each website being tracked, and then creates a big log merging all the finished traces. This web server must be able to call the program or script that detects drifts to search for them in the new created logs.
- An implementation of the model creator algorithm that can be used for the web or normal version of the drift detector algorithm. It has to create valid Petri nets from the abstract interpretation model used in the drift detection algorithm.

Decisions:

- For creating the abstract interpretation model the PPL library will be used. This library offers all the operations needed to work with polyhedrons and octagons, that are the models which will represent the system.
- Adwin will be used to detect a drift from the distribution of binary values that represent which points are inside a model and which are not. Using this library there is no need to predefine the size of the window in which to check for changes and there is mathematic proof of its correctness.

- To represent the final model, before and after a drift, Petri nets will be used. This allows the representation of concurrent and distributed systems.
4.2 System Overview
This is the structure of the project. It all starts with the system being analyzed. This system generates a Log full of traces from the operations it does. The log is simply a file where there are multiple lines, each line representing a trace, and each trace contains 1 or more events.

When a log is available then the Main program is executed. The main program is the one that coordinates the online drift detection algorithm with the model generator. This Main program starts by calling the drift detection algorithm, with the name of the log file.

The drift detection algorithm then does as explained before. It creates a new model from the log, does the estimation and then starts monitoring. To create the new polyhedrons or check if a polyhedron is inside the other it uses the Parma Polyherda Library.

The Parma Polyhedra Library has two files, one containing the model generated after the first step of the algorithm, and the other containing the constraints of the model. This files are generated using PPL functions.

The drift detection algorithm also has one or more Adwin objects. It will have one in the simple version, and multiple Adwins in the version that check each constraint separately. Each Adwin object has a file where all the information is saved. This file is only created if the program has to be stopped, so it’s possible to resume it from the same point.

The Main program has to wait until the drift detection algorithm finishes, and then it calls the Model generator to create a model of the new version of the system. To do so, the model generator reads the constraints created by PPL. This constraints represent the polyhedron used for the abstract interpretation and also can be used to extract a Petri net. The model generator then creates n different Petri nets, depending on the distance of the nodes. The most relevant one will be Petri net k1.
4.3 Online Drift Detection Algorithm

This part of the program is the one that have all the logic to do the three steps of the drift detection algorithm. The functions are:

```plaintext
setLogFile(logPath)
initialize(ntraces, polyhedronFolder)
setMultipleAdwin(active)
estimate(precision, npoints)
monitor()
load(polyhedronFolder)
save()
```

The first function is to set the log file that will be analyzed. There are three functions representing each step of the algorithm, initialize, estimate and monitor. With that functions you can control the algorithm and change it's parameters. There is also the function setMultipleAdwin that lets you change between the version with only one Adwin or the version with multiple Adwins.

There are also the save and load functions, to save the state of the algorithm to disk. These functions will also call the save operations for each Adwin object used and for the polyhedron or octagon of PPL.

4.4 Model Generator

The model generator can work independently of the other components, since it is not needed to check the log for drifts, and it doesn’t need to know when a drift has happened. However, it is useful to have the Petri net for each model, before and after a drift.

This program only needs the constraints of the abstract interpretation model
as an input and then it generates a set of Petri nets. This is because a lot of the information extracted from the constraints is not really useful, and it makes the final Petri net harder to read. The separation in different Petri nets is basically depending on the number of tokens of each place. The places with only one token give much more information than the places with say 10 tokens. Each Petri net kn represents the constraints that define places with n tokens.

4.5 Main Module

The program has a Main component that is in charge of doing the coordination between the other modules and is the one that starts and stops the algorithms.

This is the program that the user is going to interact with, the user has to set up a log to be analyzed, the folder to keep all the files for the polyhedron and the Adwins. The folder for the Petri nets has to be specified here too.
4.6 Drift detector for websites
The web version is almost the same as the other one, but with a few changes. To generate the log the web version needs traces representing the events that each user visiting a website has done. To get those traces we use a WebLogger that is executed in the computer of the users, in the browser. It has events such as the url that an user is visiting, or if they click some menu button. All that information is then sent to the main WebServer that sorts the events so each user has its own trace. After the trace of a user is complete it is added to the end of the main log of the website, the one that will be analyzed.

The Main program is executed on a regular schedule, for example each hour, so it can process the data gathered during that time. Since the traces here may take a longer time to get to the main log this is a good approach so the program is not executing while it is not needed.

The Main program here is also a bit different. When it starts it lists the folders from all the websites being tracked and when it finds some website with data in its log then it starts processing. If there is previous data to be loaded (Adwins and a model) then it loads it and continues as usual. In case there is no such data it waits until the log has at list the number of traces needed to initialize a polyhedron. If it’s not long enough it skips the folder and the same will be done on the next iteration. After the Main program finishes processing a Log it saves all the data again (Adwins and Model) so it can continue next time.

All this information is saved in a folder structure, each website has the structure:

```
/website
    log
    data/
        model
        constraints
```
Where \text{usrtokenN} is a random sequence of numbers and characters, and it contains the trace of the user whose cookie is \text{usrtokenN}. This is needed to keep track of each user separately, since each trace represents the behavior of a single user.
5 Implementation

5.1 Online concept drift detection algorithm

In order to have abstraction from the polyhedrons library used, and its version this module is separated in two different classes. One has all the model related functions and the other one has all the main functions to control the drift detection algorithm.

![Drift Detection Algorithm Module](image)

Figure 10: Drift Detection Algorithm Module

5.1.1 Model related functions

There is a class written in C++ which is in charge of communicating with the Parma Polyhedra Library. The class is called DriftDetector.

This class contains this functions:

```c++
// Object save and load
void saveConstraints(std::string path);
void saveToFile(std::string path);
```

The main function of this class is to abstract all communication with the library, so if at some point a better library or a more suited one is found the only code that has to be changed is in this class. It handles the save and load operations, the creation of a new model, the insertion of new traces (which means modify the current model to contain them) and a function to check is a point is inside the model or not. This class also decides whether to use a polyhedron or an octagon depending on the number of dimensions. In this way if later on a new kind of model is needed, the core of the application remains the same and the changes are only in this class. The class is written in C++ because the PPL interface is for C++. Since the core of the program is written in Python this class is wrapped using Swig, and then it is usable from the core of the application.

5.1.2 Core of the algorithm

The other half of this module is a Python class. This class is the core of the program, it has all the functions to run each step of algorithm, and it uses the DriftDetector class for all the model related functions.

This class has the following methods:

setLogFile(logPath)
initialize(ntraces, polyhedronFolder)
setMultipleAdwin(active)
estimate(precision, npoints)
monitor()
load(polyhedronFolder)
save()

Using this functions you can control the algorithm flow. You can start it, go through each step, save the state and pause the algorithm, and reload the state to resume the analysis.

This functions are implemented using multiprocessing in order to calculate multiple values at once, and the reorder it to find the drift. This is done by using the multiprocessing library of Python.

There is a Pool of processes with the same number of processes as cores in the machine and a semaphore to control the number of traces read, so it is not all in memory. The while loop then calls the functions asynchronously. There are different functions, one for single adwins and another one for multiple adwins.

This is an example from the monitor function:

```python
p = Pool(nthreads)
while trace != [''] and not drift:
    parikh = [0] * ndim
    for event in trace:
        pool_sema.acquire()
        if not drift:
            dimIndex = dimensions[event]
            parikh[dimIndex] += 1
        if madwin:
```
The estimation function is similar but it has other functions as callbacks.

This are the functions that evaluate a point:

```python
def evalPoint(point, npoint):
    global dd, ndim
    parikh = driftdetector.VectorInt(ndim, 0)
    for i in range(len(point)):
        parikh[i] = point[i]
    contained = dd.modelContains(parikh)
    return (npoint, contained)

def evalPointConstraints(point, npoint):
    contained = evalConstraints(point)
    return (npoint, contained)
```

46
def evalConstraints(point):
    global constraints
    res = []
    for constraint in constraints:
        val = 0
        for i in range(len(point)):
            val = val + point[i] * constraint[i]
        if val < constraint[len(constraint) - 1]:
            res.append(False)
        else:
            res.append(True)
    return res

The first one is for a single adwin and only calls the function modelContains from the C++ module.

The other funtions are for multiple adwins and they use the constraints from the polyhedron or octagon, and create an array containing booleans for each constraint.

To order this results an ordered queue is used, in python this is a heapq:

This are the callbacks for the monitor step:

def monitorDrift(contained):
    global nextPoint, drift, monTrace, eraser, results, totalTrace
    heapq.heappush(results, contained)
    while len(results) > 0 and results[0][0] == nextPoint and not drift:
result = heapq.heappop(results)
drift = adwins[0].update(result[1])
if not drift:
    print "(M) Estimation\n" +
    str(adwins[0].getEstimation()) +
    "trace\n" + str(totalTrace) + eraser + "\n",
    #print result
    nextPoint += 1
pool_sema.release()

def monitorMultipleDrift(contained):
    global nextPoint, drift, monTrace, eraser,
    results, totalTrace
heapq.heappush(results, contained)
while len(results) > 0 and results[0][0] == nextPoint
    and not drift:
        result = heapq.heappop(results)
        drifts = []
        drift = False
        for i in range(len(result[1])):
            drifts.append(adwins[i].update(result[1][i]))
            if drifts[i]:
                drift = True
        if not drift:
            print "(M) Estimation\n" +
            str(totalTrace) + eraser + "\n",
            #print result
            nextPoint += 1
pool_sema.release()
This functions order the results and the check for a drift. This allows the algorithm to run faster in a multicore computer. The structure used here is very important since at the beginning of the project, multiprocessing was not a goal, but it was clear that it will be really useful to process a lot faster the logs.

The code here is just a part of it but it gives the basic idea of how the parts are connected and how it is possible to calculate different results in parallel while keeping the final result the same. A larger version of the important bits of the code is attached in appendix C.

5.2 Model Generator

This class uses a python library called snakes to generate the Petri nets. This library is able to export the Petri net to different formats, in this case the one used is PNML. A set of files .pnml are created, each one containing a different Petri net depending on the number of tokens on each place. This .pnml files can be visualized using external programs or further analyzed with specialized tools. The model generator class has files dependencies also, since it reads the Model Constraints file generated by PPL. This is not a problem though, the file generated by PPL is a simple text file containing one dependency inequation per line, so no special treatment is required.

5.3 Main

The Main file is the one in charge of executing each algorithm, control them and set the folders for the log file and for the outputs of the algorithms. There are two different Main files, one for analyzing an information system and another one for websites.

The first one only has a few lines to set the parameters and execute each
step of the drift detection algorithm. After each drift it restarts the algorithm and creates a Petri model of the system after each initialization.

The web version is called by Cron, and it iterates over different folders containing the logs of different websites. It first updates the website log by adding the new finished traces that were added since the last execution. After that it tries to load the model of the system and continue the execution until it reaches the end of the log. In case that a model does not exist it will wait until there are enough traces to create a new model, and then begin the initialization.

5.4 WebServer

The web server is programmed in python. Flask is used in order to create an HTTP service to receive the traces of each user visiting the websites being analyzed.

Since the requests are crossdomain requests, a special decorator is needed in Flask so it allows requests that are not from the same domain of the server itself.

5.5 WebClient

The webclient is a simple javascript file that uses JQuery[5] to detect some events that the user visiting the webpage triggers. This events can be clicks in specific parts of the webpage or also the event of leaving the webpage.

This script also detects the url that the user is visiting and sends that too. Depending on the type of webpage that is being analyzed, this kind of event can be too descriptive and may not be a good event at all, but the server is the one that decide which events to use and which not to.
6 Experimentation

In this section there is the description and results of the experiments done for each of the modules developed in this project. Those modules are the drift detection algorithm module and the model generator module. The experiments will be centered in each one separately, since the results are independent.

Since there are two versions of the drift detection algorithm, one for analyzing information systems and another one for websites, there are also two sets of experiments. The first set of experiments is for the single information system version. The execution of this tests are simpler since only require one computer and the system analyzed is local.

6.1 Drift detector

The aim of these tests is to see how well the algorithm perform. What the experiments will try to find is:

- Is the algorithm able to detect drifts?
- How much time does the algorithm need?
- How many false positives does it find?

6.1.1 Test Logs

For this experiments there are a group of logs, each one with a different set of events and a different length for each trace. For each of these logs a set of changes are made.

- rem: Remove some events in a set of traces
• conc: Some events that were sequential now are concurrent.

• conf: A conflict is created between events that were sequential before.

• switch: There are switched events from different traces.

The experiments will try to answer the questions stated before for each pair of log and change.

6.1.2 Results Single Adwin

![Figure 11: Concatenation of logs](image)

The first test will be done using only one adwin. For this test the logs will be the concatenation of the normal log and the log with changes. For each log there will be 4 concatenations, one for each change of the ones described above. The result will be if the algorithm is able to detect a change and if a drift is detected how many traces were needed to detect it.

Being L the number of traces in the log and N the average number of events per line, the initialization of the model is set to L/4 traces, and the stabilizations is set to a precision of 0.01 and a number of points of (L/8)*N
In this results there are cases where the drift was detected quickly and some other cases where it couldn’t detect the drift. This is because in those logs the stabilization took too long and the end of the file was reached before having a good estimation. Some other times the estimations was done with only a few traces left of log, and there was not enough traces to detect a drift.

If the parameters of the algorithm are readjusted to let the algorithm stabilize faster then the result obtained is:

- Means no drift

x Means no stabilization
The main conclusion here is that the parameters depend on the type of log that is being analyzed, and a single small change can be really important in order to obtain good results. To obtain this improved results the logs with problems were analyzed by hand trying to see the difference. For example some logs had more variability than others, then a good value to change is the precision of the estimation. Some others had drifts that were not very drastic, and then a good way to detect them is to increase the precision of the Adwin library. In order to analyze a system with the drift detection algorithm we first need to know some parameters of the system itself that can be learnt from the log. This parameters are the ones that describe how much an estimation change during time and how much the estimation change during a drift. Using those values the algorithm can be tuned to have a better performance for that specific system.
6.1.3 False drifts with single Adwin

In order to detect false drifts the same test will be done to the logs but this time concatenating the same log twice, without any changes. If a drift is detected then this will be considered a false drift in this case.

<table>
<thead>
<tr>
<th>log</th>
<th>false drift</th>
</tr>
</thead>
<tbody>
<tr>
<td>a12f0n00</td>
<td>no</td>
</tr>
<tr>
<td>a22f0n00</td>
<td>no</td>
</tr>
<tr>
<td>a32f0n00</td>
<td>no</td>
</tr>
<tr>
<td>a42f0n00</td>
<td>no</td>
</tr>
<tr>
<td>cycles4_2</td>
<td>no</td>
</tr>
<tr>
<td>cycles5_2</td>
<td>no</td>
</tr>
<tr>
<td>kmg8_2</td>
<td>no</td>
</tr>
<tr>
<td>kmg9_2</td>
<td>no</td>
</tr>
<tr>
<td>prodcons8_5_5</td>
<td>no</td>
</tr>
<tr>
<td>prodcons9_6_6</td>
<td>no</td>
</tr>
<tr>
<td>sharedres6_3</td>
<td>no</td>
</tr>
<tr>
<td>sharedres8_5</td>
<td>no</td>
</tr>
<tr>
<td>t32f0n00</td>
<td>no</td>
</tr>
</tbody>
</table>
We can see in this results that the algorithm does not detect any false drift.

### 6.1.4 Results Multiple Adwin

The first time the tests were executed using multiple adwins a problem was found with the estimation. Since the estimation was considered stable once all the adwin objects were stable, the algorithm had a hard time stabilizing, and sometimes it wouldn’t stabilize at all. After realizing that some changes were done to the algorithm so not all the adwins needed to be stable, but only a percentage of them. After that change the tests were executed again, obtaining this results.

Being L the number of traces in the log and N the average number of events per line, the initialization of the model is set to L/4 lines, and the stabilizations is set to a precision of 0.01 and a number of points of (L/8)*N.

<table>
<thead>
<tr>
<th>log</th>
<th>conc</th>
<th>conf</th>
<th>rem</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>a12f0n00</td>
<td>-</td>
<td>38</td>
<td>39</td>
<td>48</td>
</tr>
<tr>
<td>a22f0n00</td>
<td>20</td>
<td>12</td>
<td>17</td>
<td>31</td>
</tr>
<tr>
<td>a32f0n00</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>a42f0n00</td>
<td>18</td>
<td>16</td>
<td>29</td>
<td>15</td>
</tr>
<tr>
<td>cycles4_2</td>
<td>31</td>
<td>31</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>cycles5_2</td>
<td>44</td>
<td>44</td>
<td>-</td>
<td>71</td>
</tr>
<tr>
<td>kmg8_2</td>
<td>91</td>
<td>71</td>
<td>83</td>
<td>x</td>
</tr>
<tr>
<td>kmg9_2</td>
<td>51</td>
<td>47</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td>prodcons8_5_5</td>
<td>47</td>
<td>49</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>prodcons9_6_6</td>
<td>46</td>
<td>47</td>
<td>49</td>
<td>46</td>
</tr>
<tr>
<td>sharedres6_3</td>
<td>12</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>sharedres8_5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>t32f0n00</td>
<td>17</td>
<td>19</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>
We can see in the results that using multiple Adwins the drifts are detected, in average, faster than with only one Adwin. This is because having more Adwins the algorithm is more sensitive to subtle changes that only affect a small amount of the constraints.

The algorithm performs a lot better using multiple Adwins. There are a lot less cases where the algorithm couldn’t find the drift here, and as before it is possible to do adjustments to the log in particular where there are problems.

After such adjustments the results are:

<table>
<thead>
<tr>
<th>log</th>
<th>conc</th>
<th>conf</th>
<th>rem</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>a12f0n00</td>
<td>23</td>
<td>38</td>
<td>39</td>
<td>48</td>
</tr>
<tr>
<td>a22f0n00</td>
<td>20</td>
<td>12</td>
<td>17</td>
<td>31</td>
</tr>
<tr>
<td>a32f0n00</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>a42f0n00</td>
<td>18</td>
<td>16</td>
<td>29</td>
<td>15</td>
</tr>
<tr>
<td>cycles4_2</td>
<td>31</td>
<td>31</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>cycles5_2</td>
<td>44</td>
<td>44</td>
<td>53</td>
<td>71</td>
</tr>
<tr>
<td>kmg8_2</td>
<td>91</td>
<td>71</td>
<td>83</td>
<td>85</td>
</tr>
<tr>
<td>kmg9_2</td>
<td>51</td>
<td>47</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td>prodcons8_5_5</td>
<td>47</td>
<td>49</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>prodcons9_6_6</td>
<td>46</td>
<td>47</td>
<td>49</td>
<td>46</td>
</tr>
<tr>
<td>sharedres6_3</td>
<td>12</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>sharedres8_5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>t32f0n00</td>
<td>17</td>
<td>19</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

6.1.5 False drifts with multiple Adwin

The same as before is done in order to detect false drifts, a concatenation of the log without changes.
<table>
<thead>
<tr>
<th>log</th>
<th>false drift</th>
</tr>
</thead>
<tbody>
<tr>
<td>a12f0n00</td>
<td>no</td>
</tr>
<tr>
<td>a22f0n00</td>
<td>no</td>
</tr>
<tr>
<td>a32f0n00</td>
<td>no</td>
</tr>
<tr>
<td>a42f0n00</td>
<td>no</td>
</tr>
<tr>
<td>cycles4_2</td>
<td>no</td>
</tr>
<tr>
<td>cycles5_2</td>
<td>no</td>
</tr>
<tr>
<td>kmg8_2</td>
<td>no</td>
</tr>
<tr>
<td>kmg9_2</td>
<td>no</td>
</tr>
<tr>
<td>prodcons8_5_5</td>
<td>no</td>
</tr>
<tr>
<td>prodcons9_6_6</td>
<td>no</td>
</tr>
<tr>
<td>sharedres6_3</td>
<td>no</td>
</tr>
<tr>
<td>sharedres8_5</td>
<td>no</td>
</tr>
<tr>
<td>t32f0n00</td>
<td>no</td>
</tr>
</tbody>
</table>

6.2 Drift detector web

In this section the tests will evaluate the web version of the algorithm, and how useful it is to detect drifts in the behavior of the users of a web page.

- Which events are the most useful ones?
- Is it possible to detect drifts in the users behavior?

6.2.1 Test Website

First a website to analyze is needed. This website has to have a fair amount of users every day in order to get enough data. In this case we use a blog, since is a type of website that users visit periodically. The name is AndroidBlog, and it’s url is http://www.androidblog.es.
CONTENIDO DESTACADO

Ya puedes descargar Angry Birds Friends

La nueva versión de Angry Birds de la que os hablamos hace unas semanas ya esta disponible en Google Play y en el App Store. En esta nueva versión podéis jugar contra amigos. Al principio tenéis que hacer login con Facebook y no parece que haya un modo de usarlo sin el login del usuario. Usando la cuenta de Facebook. Leer más

ÚLTIMAS NOTICIAS

Sigue el Google I/O con AndroidBlog

Hoy es el Google I/O, el gran evento donde Google presenta sus nuevos productos y servicios. Podrás verlo en directo usando esta widget, y además los anuncios de Google aparecerán en la parte de abajo. El Keynote empieza a las 16:00 UTC. Te esperamos!

Continuar leyendo...

LINE Band: junta a tus amigos en grupos para hablar en privado

Esta aplicación es otro de los muchos que ya tiene la gran empresa Line.

LINE Band nos va a permitir crear grupos o mejor dicho "band" de amigos con los que podremos compartir lo que queremos y sólo lo podrán ver ellos.

Por poner un ejemplo sería como en el patio del colegio en el que siempre hay grupos en los que se habla (o calla) sobre algo. Aunque también tenemos el ejemplo 2.0 que sería como los círculos de Google+ Continuar leyendo...
The first problem here is how to decide the events that are going to be used. The first kind of event that comes to mind is the url that the user is viewing, in this case each one would represent a post in the blog or a category for example. The problem with that kind of event is that since this is a blog, each new post will have a new url that is not inside the model. That means that each time a new post is published the algorithm will detect a drift. This drift is actually true, the users are now looking at content that they weren’t looking before, but it is not the kind of drift that is wanted for this website.

What can be done to fix this? The answer is relativize the events. This is what was done for the experiments. The main idea is that each post has a relative position in the main page, we can extract this position with javascript and send it to the server. Also, if somebody clicks in a menu button it can be an event.
Contenido Destacado

Ya puedes descargar Angry Birds Friends

La nueva versión de Angry Birds de la que os hablamos hace un par de semanas ha sido descargada 1 millón de veces en Google Play y en el App Store. En esta nueva versión podremos jugar con amigos. Al principio tenéis que hacer login con Facebook y ya podéis jugar con los amigos que hayan dado sus datos. Usando la cuenta del FB.

Post 1

Últimas Noticias

Sigue el Google I/O con AndroidBlog

Hoy es el Google I/O, el gran evento donde Google presenta sus nuevos productos y servicios. Podrás verlo en directo usando esta widget, y además los anuncios de Google aparecerán en la parte de abajo. El Keynote empieza a las 16:00 UTC, te esperamos!

PUBLICADO POR: [Fecha] [Facebook] [Twitter]

LINE Band: junta a tus amigos en grupos para hablar en privado

Esta aplicación es otra de las muchas que ya tiene la gran empresa Line.

LINE Band nos va a permitir crear grupos o mejor dicho "bands" de amigos con los que podemos compartir lo que queramos y sólo lo podran ver ellos.

Por poner un ejemplo sería como en el patio del colegio en el que siempre hay grupos en los que se habla (o critica) sobre algo. Aunque también tenemos el ejemplo 2.0 que sería como los círculos de Google+ [Continuar leyendo...]

PUBLICADO POR: [Fecha] [Facebook] [Twitter]
The result was that there were a really small amount of events. The reason is that most of the people that visits the blog went straight to the news from Google, without reaching the main page. After that they moved between posts using the related posts. Most of the users then are left out of the experiment. A new approach had to be taken and then the events were rethought. Even if people don’t go to the main page they always have the menu on top, so the menu events were changed so they could work in every page. The posts cannot be used as single events since that would fire a drift each time a new post is published, so all posts are treated as an event, \textit{view.post}. Each category has also pages, and that does not change so often so it can be used. The new events are \textit{category.page}, for example \textit{devices}. Now there are different events that can be tracked even if the users come directly from Google.

The result was that most of the traces were too small, only one event in most of them. The other ones were longer but with all the events being \textit{view.post}. This is because the users navigated the website from the outside entirely. For example reading the posts as they show in their Facebook stream, or as a tweet in Twitter.

Unfortunately, the events were not good enough, since it was not possible to track all of the users movement, since most of it was happening outside the blog itself, and was Google, Facebook or Twitter where the movement between the posts was.

This does not mean that the program is not useful, but some things have to be changed in order to be useful. Also, it depends on the website you are trying to track, maybe a blog is difficult to track with this system, but a webapp where people has to touch the buttons on the website itself is easier to control.
6.3 Model Generator

The model generator always can create a new Petri net by design, so we know that it will always work. In this section a Petri net will be generated from the model of one of the previous logs. This is mostly to show how this module works and what is the result obtained from running the algorithm.
6.3.1 Test Model

The model used is the one from the log a12f0n00.tr. This is a simple model so it does not have a lot of constraints. The constraints of the model are:

\[-A \geq -1,\]
\[I - J - L \geq 0,\]
\[A - C - I \geq 0,\]
\[L - M \geq 0,\]
\[C - D \geq 0,\]
\[J - K + M \geq 0,\]
\[D - F \geq 0,\]
\[J \geq 0,\]
\[C - E \geq 0,\]
\[A - B + C + D + E + F + G + I + J + K + L + M \geq 0,\]
\[M \geq 0,\]
\[E - G \geq 0,\]
\[F - G \geq 0,\]
\[B - C - D - E - F - G - H - I - J - K - L - M \geq 0,\]
\[H \geq 0,\]
\[G \geq 0,\]
\[G - H + K \geq 0,\]
\[K \geq 0.\]

This is basically the input file for the model generator algorithm. Using this constraints it will create a Petri net that represents the system being analyzed, the one that created the log.

The algorithm generates different files, each one with different constraints. In each file Petri-n a Petri net is represented where all the places have exactly \(n\) tokens. Since the input constraints all have 0 tokens, only one file is created where all places have 0 tokens.
This Petri net can be visualized with programs such as Woped.

Figure 14: Petri net created by the model generator

The file .pnml generated that represents this Petri net is in the appendix D, which was generated using the model generator from the constraints listed above.
7 Conclusions & future work

Process mining is a fairly new discipline that is rapidly evolving and I think is a great moment to start making progress in it’s different areas. This work intends to add new tools to use for process mining, and also add a different point of view so it does not only works with local systems but also with a group of webpages.

During this project tests we found out that the web version of the algorithm is not as useful as it seemed at first. However this does not mean that this work cannot be applied to that area, but it does mean that maybe there are some changes that have to be done first.

There are also some really good results, we found out that the multiple Ad-win version of the algorithm is a lot more precise and requires less adjustments in order to detect drifts.

Overall, the goals of the project had been reached, the algorithms are all implemented and the new approaches that we wanted to try at first were also tested. There is still a long path ahead with a lot of room for improvement, but I hope that the work done here can add some information to direct the search to the right places.

7.1 Personal Conclusions

Thanks to this project I learnt about Process Mining, a discipline that I didn’t even know before. The first I had to do was read papers about the research being done in this field, and it was my first contact with research reading too.

I think Process Mining has a huge potential and a lot of room for improvement and new tools to be developed. It is something common nowadays to have a lot of system logs that are not being used, and being able to actually extract
useful information from them is really important.

During my degree I never had a class that talked about Process Mining, and neither heard of it from other sources until my project director introduced me to it. I hope this document can help other students to discover this discipline.

Thanks to this project I learnt new algorithms, which is always really useful since it means I have some new ways of thinking future problems I could encounter as a programmer. Also I added new tools to my toolbelt, just as swig, python-snakes, PPL...

It has been a really enriching experience, and I hope it can offer the same experience to the people that read this document.

7.2 Future Work

Since Process Mining is a big discipline, it will always have a lot of improvements that can be done to the current methods and algorithms used. This project is not an exception. All the algorithms and implementations described here can be improved.

The drift detector algorithm can improve by finding new ways of detecting a drift (e.g. calculating the difference between centers of polyhedrons). It is also possible to automate more the values that now the user is responsible for (i.e. number of traces for the initialization, precision, etc). It is also possible to improve the current implementation to be a little bit faster in a multicore system. This implementation already uses multiple processes but it happens sometimes that one core has a little bit less work than others.

The model generator algorithm can be improved so the Petri nets contain more useful information, by filtering states or transitions that are not really important. Also it will be useful to have the weights of each part of the Petri
net, since int can be the case that only one part of it is really used, and the other ones are just used once in a while.
8 Cost Study

This project started in September of 2012. I spent the first two weeks reading about process mining and the drift detection algorithm. Then I also started reading the documentation of PPL to understand how the library works. I also had an example of usage of the Adwin library.

After those few weeks learning how every tool I would use for the project worked, I started programming the first version of the algorithm.

The coding started being only the algorithm in C++, since the libraries were in that language, but then it expanded to Python to integrate the tools with other environments. During the coding phase a Git repository was used to keep track of the changes.

<table>
<thead>
<tr>
<th>Month</th>
<th>Work</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>September 2012</td>
<td>Read documentation</td>
<td>50h</td>
</tr>
<tr>
<td>October 2012</td>
<td></td>
<td>50h</td>
</tr>
<tr>
<td>November 2012</td>
<td>Implementation of the drift detection algorithm</td>
<td>30h</td>
</tr>
<tr>
<td>December 2012</td>
<td></td>
<td>30h</td>
</tr>
<tr>
<td>January 2013</td>
<td>Implementation of the model generation algorithm</td>
<td>100h</td>
</tr>
<tr>
<td>February 2013</td>
<td></td>
<td>100h</td>
</tr>
<tr>
<td>March 2013</td>
<td>Implementation of the web version</td>
<td>120h</td>
</tr>
<tr>
<td>April 2013</td>
<td>Experiments</td>
<td>140h</td>
</tr>
<tr>
<td>May 2013</td>
<td>and report</td>
<td>100h</td>
</tr>
</tbody>
</table>

Table 2: Time study

Assuming that the price per hour is 30€, the total cost of the hours worked is 21,600€.
Tools used to develop the project:

<table>
<thead>
<tr>
<th>Software</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian Wheezy</td>
<td>Free</td>
</tr>
<tr>
<td>Parma Polyhedra Library</td>
<td>Free</td>
</tr>
<tr>
<td>Adwin</td>
<td>Free</td>
</tr>
<tr>
<td>Swig</td>
<td>Free</td>
</tr>
<tr>
<td>Gummi</td>
<td>Free</td>
</tr>
</tbody>
</table>

Table 3: Software cost study

The computer used for this project has a value of 700€. A cheaper computer could be used, but the experiments would have taken longer. This price is used only as a guide.

<table>
<thead>
<tr>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours of work</td>
</tr>
<tr>
<td>Software</td>
</tr>
<tr>
<td>Hardware</td>
</tr>
</tbody>
</table>

Table 4: Total cost
References


9 Appendix

A Libraries Used

A.1 Parma Polyhedra

Parma Polyhedra Library (PPL)\cite{7} is a library for working with multidimensional objects such as Convex Polyhedrons and Octagons. This is the library in which all the abstract interpretation is based. It has interfaces for different languages, but in this work the one used is the C++ one. It also has different versions, some of them are in linux distribution repositories, and you can build the last version from the source code that is available in the website.

Since the changes in the last versions don’t affect the work that is being done here, the version used is the one in Debian testing (wheezy) repositories. The version is libppl0.11-dev. The implementation of the algorithm uses both Convex Polyhedron and Octagon from this library. They are pretty similar, the difference is the one stated in section 2.3. The operations needed for the algorithm to work are available in both models.

A.1.1 Convex Polyhedron

Convex polyhedrons are the implementation of multidimensional polyhedrons in the Parma Polyhedra Library. This implementation allows the creation of new polyhedrons, specifying all it’s coordinates, and also allow the merging of two polyhedrons, doing the convex hull of them both. The implementation of the drift detection algorithm also uses another function to know whether a polyhedron is inside another polyhedron or not. This are the only operations used from this model. The problem is that with complex logs doing the convex
hull takes too long, and online analysis becomes unfeasible under that circumstances. When this happens the algorithm uses Octagons, since the calculations are faster than with polyhedrons.

### A.1.2 Octagons

Octagons are the implementation of multidimensional octagons in the Parma Polyhedra Library. The operations used by the drift detector algorithm are almost the same as in the case of polyhedrons, the only difference is that they defer in name. With this model you can create new octagons giving the full coordinates of the octagon, and you can merge two octagons into one. There is also a function to know whether an octagon is inside another octagon or not. Since the calculations are faster than with polyhedrons, octagons allow the algorithm to work online even in hard cases where it cannot do it with polyhedrons. The drawback in this case is the precision, since octagons will have a bigger area as a representation of the model. This means that there will be cases where polyhedrons would have detected drifts but octagons fail to do so.

### A.2 Adwin Methods

```cpp
Adwin(int _M);

double getEstimation() const;

bool update(const double &value);

void save(std::string path) const;

void load(std::string path);
```

The first one is to initialize the Adwin object with a number of maximum buckets.
The update method is the one that does most of the part. You only need pass it the parameter value. This variable is the one that Adwin is going to track, and after each update it returns a boolean that specifies if there is a drift.

The method getEstimation returns the estimation that Adwin currently has for the value, after all the subsequent updates. When there is a drift then the estimation will change abruptly.

Finally there are two methods that were added during this project, as they were needed to save and load the state of an Adwin object. This is used for the web version of the drift detector algorithm, which is explained in section 3.5.

A.3 Swig

SWIG (Simplified Wrapper and Interface Generator)[8] is an open source library used to connect code created in C or C++ to other code in a scripting language such as PHP, Perl, Python... and other languages such as Java, Javascript,GO,C#...

With Swig you can create wrappers to be able to call the functions created originally in C or C++ from any of the other languages supported. To do so, the programmer has to write an interface file containing the list of functions that will be made visible to an interpreter. Swig uses this interface file to generate code in C/C++ and code in the target programming language. The code to transform simple data types from one language to the other will be generated by SWIG. If the functions have complex data types the code to do the conversion must be written by the programmer.
A.3.1 Example of Swig usage

Lets say we have a c code like this one:

```c
#include <time.h>

double My_variable = 3.0;

int fact(int n) {
    if (n <= 1) return 1;
    else return n*fact(n-1);
}

int my_mod(int x, int y) {
    return (x%y);
}

char *get_time() {
    time_t ltime;
    time(&ltime);
    return ctime(&ltime);
}
```

Then an interface is needed in order to access this functions from a different programming language, such as Python. The interface file would look like this:

```c
*/ example.i */
%module example
%
/* Put header files here or function
```
Then, to build a Python module you have to use this commands:

```bash
unix % swig -python example.i
unix % gcc -c example.c example_wrap.c \ 
       -I/usr/local/include/python2.1
unix % ld -shared example.o example_wrap.o -o _example.so
```

To use the newly created module you have to import the generated file from Python, and then use it as any other Python library:

```python
>>> import example
>>> example.fact(5)
120
>>> example.my_mod(7,3)
1
>>> example.get_time()
'Sun Feb 11 23:01:07 1996'
```

---

*declarations like below */

```c
extern double My_variable;
extern int fact(int n);
extern int my_mod(int x, int y);
extern char *get_time();
%
```

```c
extern double My_variable;
extern int fact(int n);
extern int my_mod(int x, int y);
extern char *get_time();
```
A great advantage of using swig is that it also allows to import a C++ header file in the interface file. This is really helpful since the functions are already defined in the header file and the interface is much simpler. The only thing that you have to add then is the conversion for complex data types.

A.4 Python Snakes

Python-snakes[9] is a library to work with Petri nets in python. The library allows the programmer to create places and transitions in an easy way. The library allows to have tokens that are Python objects. Using that feature you can for example create Petri nets whose tokens are also Petri nets (so called nets in nets). The library also has a plugin system that is really useful to have new features. There are plugins to draw Petri nets and to export them to different formats. During this work the part that is going to be used is the simple interface to create and modify Petri nets, using transition and places objects. Since the final format wanted for the Petri nets is PNML, a plugin is also needed. This plugin is called snakes-pnml, and it adds to the library the functionality to export a Petri net to that format in only one line.

A.4.1 Example

```python
>>> from snakes.nets import *
>>> n1 = PetriNet('First Net')
>>> n1.add_place(Place('p1', [0,1]))
>>> n1.add_place(Place('p2', [0]))
>>> n1.add_transition(Transition('t'))
>>> n1.add_input('p', 't', Variable('x'))
>>> n1.add_output('p', 't', Variable('y'))
```

This creates a Petri net like this one:
A.5 Flask

Flask[10] is a microframework for web development using Python. It offers a simple interface to create an HTTP server and set the function that will be responding to each suburl of the webpage.

A.5.1 Example

A simple Hello World example of flask is:

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!

if __name__ == "__main__":
    app.run()
```

A.6 JQuery

JQuery[5] is a javascript library that allows you to detect user events easily. With this library you can also modify the DOM by using CSS selectors. The DOM is the document of the webpage, its HTML code. Using JQuery you can
then modify the HTML code dynamically in real time. Between the hundreds of functions that this library has, there are some functions to do AJAX calls. An AJAX call is an asynchronous call that you can make from javascript to your backend server. Doing this is not necessary to refresh the web page in order to have communication between the client and the server. After the call you can modify the DOM of the webpage dynamically so the user knows how the call went. You can also do all of this without the user knowing that an exchange of information is happening.

### A.6.1 Examples

**DOM Manipulation:**

```javascript
$( "button.continue" ).html( "Next Step..." )
```

**Event Handling:**

```javascript
var hiddenBox = $( "#banner-message" );

$( "#button-container button" ).on( "click", function(event) {
    hiddenBox.show();
});
```

**AJAX Calls:**

```javascript
$.ajax({
    url: "/api/getWeather",
    data: {
        zipcode: 97201
    },
    success: function( data ) {
```
B Instructions

First you need to prepare the environment to use the project. You need:

- Python 2.6
- Flask
- python-snakes
- libppl
- Swig
- g++
- make

Using the repository of your Linux distribution you should be able to download most of them, the only one that wasn’t in my Debian repository was python-snakes. To download it you should go to https://code.google.com/p/python-snakes/

Once you have all the tools, you should compile the files under driftdetector. There is Makefile available, so you only have to do make in that folder. After that you only have to set the file to analyze in the pddMain.py file.
To use the web version you need to start the server first. Go to the server folder and run "python server.py". You may need to open your ports if you want to use the server from outside your local network. Then the server is ready to receive new traces.

To use the javascript client add the logger.js and jquery library under the a js/ folder and paste this code in the website you want to analyze:

```html
<!-- driftdetector code -->
<script type="text/javascript">
  var tracking_code = "somecode"
</script>
<script type="text/javascript" src="js/jquery-1.8.3.min.js"></script>
<script type="text/javascript" src="js/logger.js"></script>
<!-- end of driftdetector code -->
```

C Drift Detector Python

```python
def evalPoint(point, npoint):
    global dd, ndim
    parikh = driftdetector.VectorInt(ndim, 0)
    for i in range(len(point)):
        parikh[i] = point[i]
    contained = dd.modelContains(parikh)
    return (npoint, contained)

def evalPointConstraints(point, npoint):
```

81
contained = evalConstraints(point)
return (npoint, contained)

def evalConstraints(point):
    global constraints
    res = []
    for constraint in constraints:
        val = 0
        for i in range(len(point)):
            val = val + point[i] * constraint[i]
        if val < constraint[len(constraint) - 1]:
            res.append(False)
        else:
            res.append(True)
    return res

def updateEstimation(contained):
    global nextPoint, estTrace, eraser, estStable,
    results, totalTrace, adwins, estimations,
    acceptedError, wantedStable
    heapq.heappush(results, contained)
    while len(results) > 0 and results[0][0] == nextPoint
        and not estStable:
        result = heapq.heappop(results)
        adwins[0].update(result[1])
        estimation = adwins[0].getEstimation()
        if estimations[0] != -1 and math.fabs(estimations
            [0] - estimation) <= acceptedError:
currStable[0] += 1

else:
    currStable[0] = 1
estimations[0] = estimation
estStable = (currStable[0] >= wantedStable)

print "(E) nutmEstimation = str(estimation) + ",
      "+str(totTrace) + "("+str(currStable[0])+")" +eraser + ",

# print result
nextPoint += 1
pool_sema.release()

def updateMultipleEstimation(contained):
    global nextPoint, estTrace, eraser, estStable,
            results, totalTrace, adwins, estimations,
            acceptedError, wantedStable
heapq.heappush(results, contained)

while len(results) > 0 and results[0][0] == nextPoint
    and not estStable:
    result = heapq.heappop(results)
    stableAux = True
    countStable = 0

    for i in range(len(result[1])):
        adwins[i].update(result[1][i])
        estimation = adwins[i].getEstimation()

        if estimations[i] != -1 and math.fabs(
            estimations[i] - estimation) <=
            acceptedError:
            currStable[i] += 1
else:
    currentStable[i] = 1
    estimations[i] = estimation

if not (currentStable[i] >= wantedStable):
    stableAux = False
else:
    countStable += 1
    if float(countStable) / float(len(result[1])) < 0.8:
        estStable = False
    else:
        estStable = True

print "(E) Estimation in Multiple trace = " + str(totalTrace) + eraser + ",
# print result
nextPoint += 1
pool_sema.release()


def monitorDrift(contained):
    global nextPoint, drift, monTrace, eraser, results, totalTrace
    heapq.heappush(results, contained)
    while len(results) > 0 and results[0][0] == nextPoint
        and not drift:
        result = heapq.heappop(results)
        drift = adwins[0].update(result[1])
        if not drift:
            print "(M) Estimation in Multiple trace = " + str(adwins[0].getEstimation()) + "trace = " + str(
```python
def monitorMultipleDrift(contained):
    global nextPoint, drift, monTrace, eraser, results, totalTrace
    heapq.heappush(results, contained)
    while len(results) > 0 and results[0][0] == nextPoint:
        result = heapq.heappop(results)
        drifts = []
        drift = False
        for i in range(len(result[1])):
            drifts.append(adwins[i].update(result[1][i]))
            if drifts[i]:
                drift = True
        if not drift:
            print "(M) Estimation of Multiple trace = " +
            str(totalTrace) + eraser + "\r",
            #print result
            nextPoint += 1
            pool_sema.release()
        else:
            print "Estimation of Multiple trace = " +
            str(totalTrace) + eraser + "\r",
            #print result
            nextPoint += 1
            pool_sema.release()
```

```
if folder[len(folder) - 1] != "/":
    folder += "/"
baseFolder = folder
dd = driftdetector.driftDetector()

# Calculate Dimensions
dimensions = dict()
nextDim = 0
log = []
for i in range(traces):
    trace = readTrace()
    log.append(trace)
    for j in range(len(trace)):
        if not trace[j] in dimensions:
            dimensions[trace[j]] = nextDim
            nextDim += 1
ndim = len(dimensions)
dd.createModel(ndim)

# Insert null prefix only once
nullPrefix = driftdetector.VectorInt(ndim, 0)
dd.insertParikh(nullPrefix)

# Insert parikh vectors
for i in range(len(log)):
    parikh = driftdetector.VectorInt(ndim, 0)
    for j in range(len(log[i])):
        dimIndex = dimensions[log[i][j]]
        parikh[dimIndex] += 1
    dd.insertParikh(parikh)

def estimate(vacceptedError, vwantedStable):
global p, estTrace, estStable, pool_sema, dimensions, 
    madwin, wantedStable, currentStable,
    acceptedError
p = Pool(nthreads)
acceptedError = vacceptedError
wantedStable = vwantedStable
trace = readTrace()
npoint = 1
estTrace = 1
estStable = False
setMultipleAdwin(madwin)
print "Init Estimation:

while trace != [''] and not estStable:
    parikh = [0] * ndim
    for event in trace:
        pool_sema.acquire()
        if not estStable:
            dimIndex = dimensions[event]
            parikh[dimIndex] += 1
            if madwin:
                p.apply_async(evalPointConstraints,
                              args=(
                                    list(parikh), npoint),
                              callback=
                                    updateMultipleEstimation)
            else:
                p.apply_async(evalPoint,
                              args=(
                                  list(parikh), npoint),
                              callback=
                                    updateEstimation)
            npoint += 1
        else:
            break
pool_sema.release()

trace = readTrace()
estTrace += 1

p.close()
p.join()

if estStable:
    if madwin:
        print "\nEstimation done, trace=\n" + str(totalTrace)
    else:
        print "\nEstimation done, stable at\n" + str(adwins[0].getEstimation()) + "trace=\n" + str(totalTrace)
else:
    print "\nThe estimation didn’t stabilize"

def monitor():
    global p, drift, monTrace, pool_sema, nextPoint,
        results, madwin, dimensions
    p = Pool(nthreads)
    trace = readTrace()
    npoint = 1
    drift = False
    monTrace = 1
    nextPoint = 1
    results = []
    print "Init Monitoring:
    while trace != [''] and not drift:
        parikh = [0] * ndim
for event in trace:
    pool_sema.acquire()
    if not drift:
        dimIndex = dimensions[event]
        parikh[dimIndex] += 1
        if madwin:
            p.apply_async(evalPointConstraints, 
            args=(
                list(parikh), npoint), callback= 
                monitorMultipleDrift)
        else:
            p.apply_async(evalPoint, args=(
                list(parikh), npoint), callback= 
                monitorDrift)
        npoint += 1
    else:
        pool_sema.release()
    trace = readTrace()
    monTrace += 1
p.close()
p.join()
if drift:
    print "\nDrift detected at trace\n" + str(totalTrace)
else:
    print "\nNo drift detected"

D Example of a PNML file

<?xml version="1.0" encoding="utf-8"?>
<pnml>
  <net id="model0">
    <place id="p0">
      <type domain="universal" />
      <initialMarking>
        <multiset/>
      </initialMarking>
    </place>
    <place id="p1">
      <type domain="universal" />
      <initialMarking>
        <multiset/>
      </initialMarking>
    </place>
    ...
    <place id="p10">
      <type domain="universal" />
      <initialMarking>
        <multiset/>
      </initialMarking>
    </place>
    <place id="p11">
      <type domain="universal" />
      <initialMarking>
        <multiset/>
      </initialMarking>
    </place>
    <transition id="A" />
    <transition id="C" />
    ...
  </net>
</pnml>
<transition id="M"/>
<transition id="L"/>
<arc id="A:p1" source="A" target="p1">
  <inscription>
  <variable>x</variable>
  </inscription>
</arc>
<arc id="A:p7" source="A" target="p7">
  <inscription>
  <variable>x</variable>
  </inscription>
</arc>
...
<arc id="L:p7" source="L" target="p7">
  <inscription>
  <variable>x</variable>
  </inscription>
</arc>
<arc id="L:p2" source="L" target="p2">
  <inscription>
  <variable>x</variable>
  </inscription>
</arc>
</net>
</pnml>