Faculty of ENCS, Concordia University, Montreal, QC, Canada

The Need to Support of Data Flow Graph Visualization of Forensic Lucid Programs, Forensic Evidence, and their Evaluation by GIPSY

Serguei A. Mokhov, Joey Paquet, Mourad Debbabi

INITIAL BACKGROUND

Abstract

A Forensic Lucid intensional programming language has been proposed for intensional cyberforensic analysis. In large part, the language is based on various predecessor and codecessor Lucid dialects bound by the higher-order intensional logic (HOIL) that is behind them. This work formally specifies the operational aspects of the Forensic Lucid language and compiles a theory of its constructs using Isabelle, a proof assistant system.

1 Introduction

As a part of the Intensional Cyberforensics project, we define a functional-intensional programming/specification language, called Forensic Lucid. The language is under active design and development including its syntax, semantics, the corresponding compiler, run-time, and interactive "development" environments [1, 2] that we refer to as General Intensional Programming System (GIPSY) [3]. We approach the problem using Isabelle [4] as a proof assistant.

1.1 Intensional Logics and Programming

1.1.1 An Example of Using Temporal Intensional Logic.

Temporal intensional logic is an extension of temporal logic that allows to specify the time in the future or in the past.

(1) E_1 := it is raining **here today**

Context: {`place`:**here**, `time`:**today**}

(2) E_2 := it was raining **here** *before*(**today**) = *yesterday*

(3) E_3 := it is going to rain *at* (altitude **here** + 500 m) *after*(**today**) = *tomorrow*

Let's take E_1 from (1) above. Then let us fix **here** to **Montreal** and assume it is a *constant*. In the month of February, 2008, with granularity of day, for every day, we can evaluate E_1 to either *true* or *false*:

```
Tags:   1 2 3 4 5 6 7 8 9 ...
Values: F F T T T F F F T ...
```

If one starts varying the **here** dimension (which could even be broken down to X, Y, Z), one gets a two-dimensional evaluation of E_1:

```
City: /   1 2 3 4 5 6 7 8 9 ...
Montreal  F F T T T F F F T ...
Quebec    F F F F T T T F F ...
Ottawa    F T T T T T F F F ...
```

2 Forensic Lucid

The end goal is to define our Forensic Lucid language where its constructs concisely express cyberforensic evidence, which can be initial state of a case towards what we have actually observed as a final state. The implementing system (i.e. GIPSY) has to backtrace intermediate results in order to provide the corresponding event reconstruction path, if it exists. The result of the expression in its basic form is either *true* or *false*, i.e. "guilty" or "not guilty" given the context per explanation with the backtrace. There can be multiple backtraces, that correspond to the explanation of the evidence (or lack thereof).

2.1 Properties

```
foo @
{
  [ final observed event, possible initial observed event ],
  [                ],
  [                ]
}
```

Listing 1: Intensional Storyboard Expression

2.2 Primitive Operators

The collection of the translated operators denoted in **`monospaced font`**, while we provide their equivalence to the original Lucid operators, denoted as SMALL CAPS.

The primitive operators are founding blocks to construct more complex case-specific functions that represent a particular investigation case as well as more complex so-called *forensic operators*.

- A stream of first elements of stream X:
  $\text{FIRST}\ X = (x_0, x_0, \ldots, x_0, \ldots)$

$$\texttt{first}\ X = X@0 \tag{1}$$

- A stream of second elements of stream X:
  $\text{SECOND}\ X = (x_1, x_1, \ldots, x_1, \ldots) = \text{FIRST NEXT}\ X$
- A stream of last elements of stream X:
  $\text{LAST}\ X = (x_n, x_n, \ldots, x_n, \ldots)$

$$\texttt{last}\ X = X@(\#@(\#\texttt{iseod}(\#) - 1)) \tag{2}$$

- A stream of elements one before the last one of stream X:
  $\text{PRELAST}\ X = (x_{n-1}, x_{n-1}, \ldots, x_{n-1}, \ldots) = \text{LAST PREV}\ X$
- A stream of elements of stream X other than the first:
  $\text{NEXT}\ X = (x_1, x_2, \ldots, x_{i+1}, \ldots)$

$$\texttt{next}\ X = X@(\# + 1) \tag{3}$$

- A stream of elements of stream X other than the last:
  $\text{PREV}\ X = (x_{n-1}, \ldots, x_{i+1}, x_i, x_{i-1}, \ldots)$

$$\texttt{prev}\ X = X@(\# - 1) \tag{4}$$

- First element of X followed by all of Y:
  $X\ \text{FBY}\ Y = (x_0, y_0, y_1, \ldots, y_{i-1}, \ldots)$

$$\begin{aligned} X\ \texttt{fby}\ Y &= \textbf{if}\ \# = 0\ \textbf{then}\ X\ \textbf{else}\ Y@(\# - 1) \\ &= \textbf{if}\ \texttt{isbod}\ X\ \textbf{then}\ X\ \textbf{else}\ \texttt{prev}\ Y \end{aligned} \tag{5}$$

- First element of X preceded by all of Y:
  $X\ \text{PBY}\ Y = (y_0, y_1, \ldots, y_{i-1}, \ldots, y_n, x_0)$

$$\begin{aligned} X\ \texttt{pby}\ Y &= \textbf{if}\ \texttt{iseod}\,\#\ \textbf{then}\ X\ \textbf{else}\ Y@(\# + 1) \\ &= \textbf{if}\ \texttt{iseod}\ Y\ \textbf{then}\ X\ \textbf{else}\ \texttt{next}\ Y \end{aligned} \tag{6}$$

- WVR stands for *whenever*. WVR chooses from its left-hand-side operand only values in the current dimension where the right-hand-side evaluates to *true*.
  $X\ \text{WVR}\ Y =$
  if $\text{FIRST}\ Y \neq 0$
  then $X\ \text{FBY}\ (\text{NEXT}\ X\ \text{WVR}\ \text{NEXT}\ Y)$
  else $(\text{NEXT}\ X\ \text{WVR}\ \text{NEXT}\ Y)$

$$\begin{aligned} X\ \texttt{wvr}\ Y = X@T\ &\texttt{where} \\ &T = U\ \texttt{fby}\ U@(T+1) \\ &U = \textbf{if}\ Y\ \textbf{then}\ \#\ \textbf{else}\ \texttt{next}\ U \\ &\texttt{end} \end{aligned} \tag{7}$$

- ASA stands for *as soon as*. ASA returns the value of its left-hand-side as a first point in that stream as soon as the right-hand-side evaluates to *true*.
  $X\ \text{ASA}\ Y = \text{FIRST}\ (X\ \text{WVR}\ Y)$

$$X\ \texttt{asa}\ Y = \texttt{first}\ (X\ \texttt{wvr}\ Y) \tag{8}$$

2.3 Forensic Operators

```
/**
 * Append given e to each element
 * of a given stream e under the
 * context of d.
 *
 * @return the resulting combined stream
 */
combine(s, e, d) =
  if iseod s then eod;
  else (first s fby.d e) fby.d combine(next s, e, d);
  fi
```

Listing 2: The **`combine`** Operator

```
/**
 * Append elements of s2 to element of s1
 * in all possible combinations.
 */
product(s1, s2, d) =
  if iseod s2 then eod;
  else combine(s1, first s2) fby.d product(s1, next s2);
  fi
```

Listing 3: The **`product`** Operator

2.4 Operational Semantics

$$\mathbf{E_{cid}} : \frac{\mathcal{D}(id) = (\texttt{const}, c)}{\mathcal{D}, \mathcal{P} \vdash id : c}$$

$$\mathbf{E_{opid}} : \frac{\mathcal{D}(id) = (\texttt{op}, f)}{\mathcal{D}, \mathcal{P} \vdash id : id}$$

$$\mathbf{E_{did}} : \frac{\mathcal{D}(id) = (\texttt{dim})}{\mathcal{D}, \mathcal{P} \vdash id : id}$$

$$\mathbf{E_{fid}} : \frac{\mathcal{D}(id) = (\texttt{func}, id_i, E)}{\mathcal{D}, \mathcal{P} \vdash id : id}$$

$$\mathbf{E_{vid}} : \frac{\mathcal{D}(id) = (\texttt{var}, E) \qquad \mathcal{D}, \mathcal{P} \vdash E : v}{\mathcal{D}, \mathcal{P} \vdash id : v}$$

$$\mathbf{E_{op}} : \frac{\mathcal{D}, \mathcal{P} \vdash E : id \qquad \mathcal{D}(id) = (\texttt{op}, f) \qquad \mathcal{D}, \mathcal{P} \vdash E_i : v_i}{\mathcal{D}, \mathcal{P} \vdash E(E_1, \ldots, E_n) : f(v_1, \ldots, v_n)}$$

$$\mathbf{E_{fct}} : \frac{\mathcal{D}, \mathcal{P} \vdash E : id \qquad \mathcal{D}(id) = (\texttt{func}, id_i, E') \qquad \mathcal{D}, \mathcal{P} \vdash E'[id_i \leftarrow E_i] : v}{\mathcal{D}, \mathcal{P} \vdash E(E_1, \ldots, E_n) : v}$$

$$\mathbf{E_{c_T}} : \frac{\mathcal{D}, \mathcal{P} \vdash E : true \qquad \mathcal{D}, \mathcal{P} \vdash E' : v'}{\mathcal{D}, \mathcal{P} \vdash \texttt{if}\ E\ \texttt{then}\ E'\ \texttt{else}\ E'' : v'}$$

$$\mathbf{E_{c_F}} : \frac{\mathcal{D}, \mathcal{P} \vdash E : false \qquad \mathcal{D}, \mathcal{P} \vdash E'' : v''}{\mathcal{D}, \mathcal{P} \vdash \texttt{if}\ E\ \texttt{then}\ E'\ \texttt{else}\ E'' : v''}$$

$$\mathbf{E_{tag}} : \frac{\mathcal{D}, \mathcal{P} \vdash E : id \qquad \mathcal{D}(id) = (\texttt{dim})}{\mathcal{D}, \mathcal{P} \vdash \#E : \mathcal{P}(id)}$$

$$\mathbf{E_{at}} : \frac{\mathcal{D}, \mathcal{P} \vdash E' : id \qquad \mathcal{D}(id) = (\texttt{dim}) \qquad \mathcal{D}, \mathcal{P} \vdash E'' : v'' \qquad \mathcal{D}, \mathcal{P} \dagger [id \mapsto v''] \vdash E : v}{\mathcal{D}, \mathcal{P} \vdash E\ @E'\ E'' : v}$$

$$\mathbf{E_{w}} : \frac{\mathcal{D}, \mathcal{P} \vdash Q : \mathcal{D}', \mathcal{P}' \qquad \mathcal{D}', \mathcal{P}' \vdash E : v}{\mathcal{D}, \mathcal{P} \vdash E\ \texttt{where}\ Q : v}$$

$$\mathbf{Q_{dim}} : \frac{}{\mathcal{D}, \mathcal{P} \vdash \texttt{dimension}\ id : \mathcal{D} \dagger [id \mapsto (\texttt{dim})], \mathcal{P} \dagger [id \mapsto 0]}$$

$$\mathbf{Q_{id}} : \frac{}{\mathcal{D}, \mathcal{P} \vdash id = E : \mathcal{D} \dagger [id \mapsto (\texttt{var}, E)], \mathcal{P}}$$

$$\mathbf{Q_{fid}} : \frac{}{\mathcal{D}, \mathcal{P} \vdash id(id_1, \ldots, id_n) = E : \mathcal{D} \dagger [id \mapsto (\texttt{func}, id_i, E)], \mathcal{P}}$$

$$\mathbf{QQ} : \frac{\mathcal{D}, \mathcal{P} \vdash Q : \mathcal{D}', \mathcal{P}' \qquad \mathcal{D}', \mathcal{P}' \vdash Q' : \mathcal{D}'', \mathcal{P}''}{\mathcal{D}, \mathcal{P} \vdash Q\ Q' : \mathcal{D}'', \mathcal{P}''}$$

$$\mathbf{E_{E.did}} : \frac{\mathcal{D}(E.id) = (\texttt{dim})}{\mathcal{D}, \mathcal{P} \vdash E.id : id.id}$$

$$\mathcal{D} \vdash E : v$$

$$\mathcal{D}, \mathcal{P} \vdash E : v$$

type	form
dimension	(`dim`)
constant	(`const`, c)
operator	(`op`, f)
variable	(`var`, E)
function	(`func`, id_i, E)

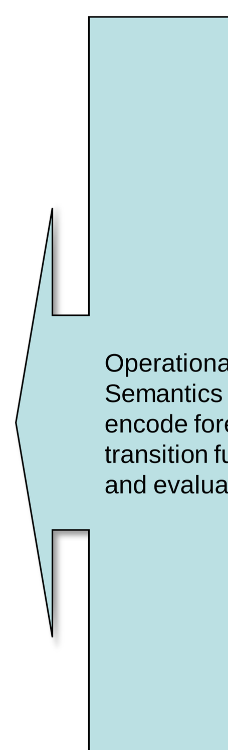
$$\mathcal{P} : \mathbf{Id} \rightarrow \mathbf{N}$$

Each type of identifiers can only be used in the appropriate situations.

$$\mathbf{E_{\#(cxt)}} : \frac{}{\mathcal{D}, \mathcal{P} \vdash \# : \mathcal{P}}$$

$$\mathbf{E_{construction(cxt)}} : \frac{\begin{array}{c} \mathcal{D}, \mathcal{P} \vdash E_{d_j} : id_j \qquad \mathcal{D}(id_j) = (\texttt{dim}) \\ \mathcal{D}, \mathcal{P} \vdash E_{i_j} : v_j \qquad \mathcal{P}' = \mathcal{P}_0 \dagger [id_1 \mapsto v_1] \dagger \ldots \dagger [id_n \mapsto v_n] \end{array}}{\mathcal{D}, \mathcal{P} \vdash [E_{d_1} : E_{i_1}, E_{d_2} : E_{i_2}, \ldots, E_{d_n} : E_{i_n}] : \mathcal{P}'}$$

$$\mathbf{E_{at(cxt)}} : \frac{\mathcal{D}, \mathcal{P} \vdash E' : \mathcal{P}' \qquad \mathcal{D}, \mathcal{P} \dagger \mathcal{P}' \vdash E : v}{\mathcal{D}, \mathcal{P} \vdash E\ @\ E' : v}$$



3 Conclusion

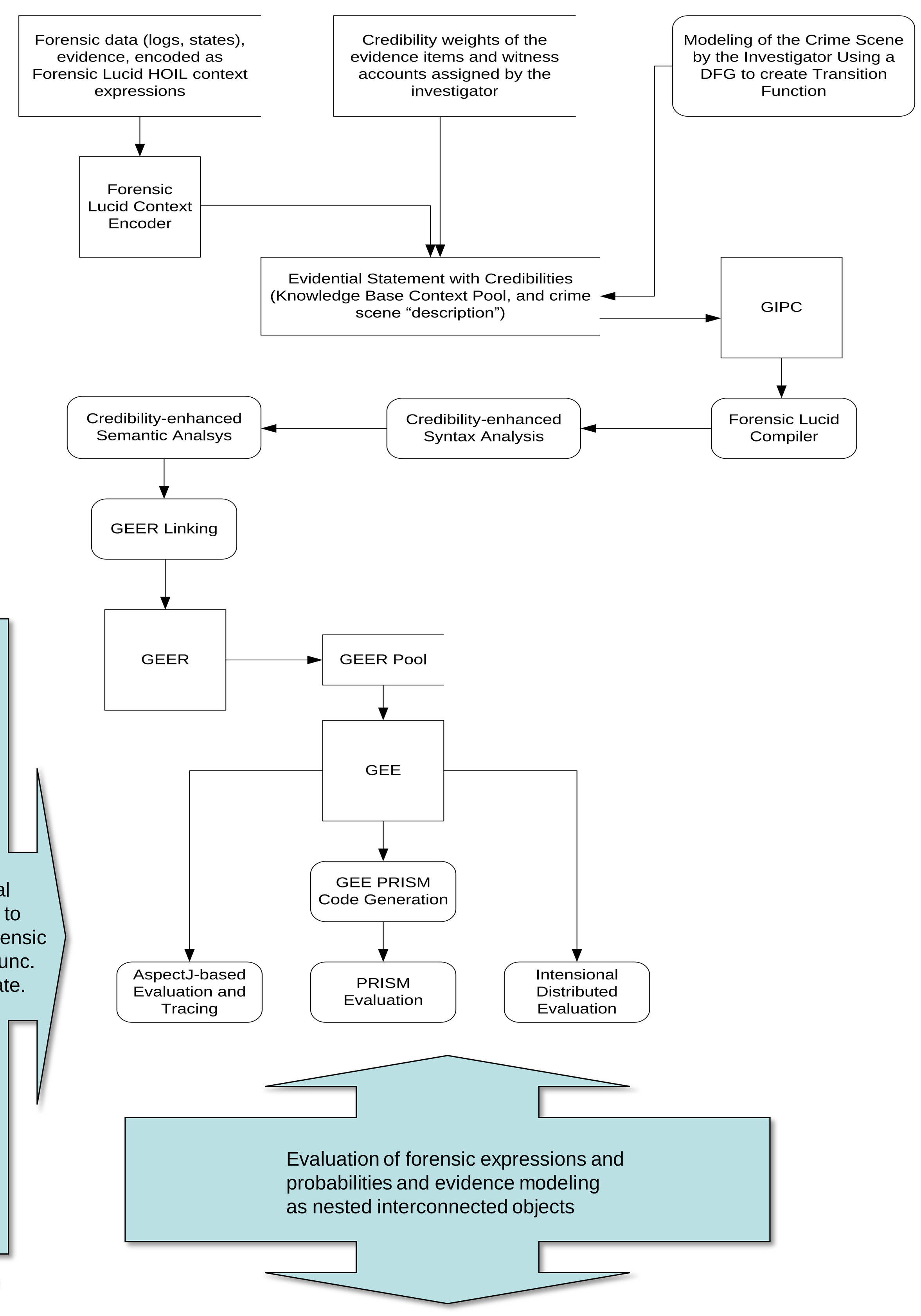
Isabelle is the tool of choice of complete specification of the mainstream Lucid dialects, and more specifically Forensic Lucid, focusing on the correctness of the operational aspects. It is a work in progress. The completed work will have a complete list of the files publicly available and submitted to the AfP [27].

3.1 Future Work

The near-future work will consist primarily of the following items:

- Complete semantics of all the mentioned Lucid dialects and their formalization with Isabelle.
- Augment the language specification to include the Depmster-Shafer theory [30, 31] of evidence to allow weights for claims, credibility, belief, and plausibility parameters.
- Prove semantic rules involving intensional data warehouse.
- Implementation of the Forensic Lucid compiler, run-time and interactive development environments.

THE GIPSY PROCESS



Abstract. This work refines the theoretical structure and formal model of the observation tuple with the credibility weight and other factors for cyberforensic analysis and event reconstruction. The model first proposed for finite-state automata approach by Gladyshev et al. [1, 2] and later extended and realized in the first iteration of Forensic Lucid, an intensional forensic case specification language [3, 4]. In the ongoing work, it is being augmented with the Dempster-Shafer theory of mathematical evidence to include the credibility factors and the like that are lacking in the Gladyshev's model. It's practically being implemented within the General Intensional Programming System (GIPSY) and the probabilistic model-checking tool PRISM to compile the Forensic Lucid model into the PRISM's syntax and check it with the PRISM tool at run-time.

1 Introduction

1.1 Background

In one of the first formal approaches about automated cyberforensic case reasoning and event reconstruction Gladyshev et al. created a finite-state automata (FSA) model [1, 2] to encode the evidence and the stories "told" by witnesses in order to combine them into what they refer to as evidential statement, then model the FSA of a particular case, and, given the FSA, verify if certain claim agrees with the evidential statement or not and if it does what were possible event sequences that explain that claim. On the other hand, an earlier work suggested a mathematical theory of evidence by Dempster, Shafer and others [5, 6], where factors like credibility, trustworthiness, and the like play a role in the evaluation of mathematical expressions, which Gladyshev lacked. Thirdly, an even earlier work on intensional logics and programming provided a formal model that throughout development placed the context as a first-class value in logical and programming expressions in the system, called Lucid that has produced various Lucid dialects and context-aware systems, such as GIPSY [7 14]. Thus, in this work we blend the three together – we augment the Gladyshev's formalization with the credibility weight and other properties derived from the mathematical theory of evidence and we encode it as a context in the Forensic Lucid language, a Lucid derivative for forensic case management, evaluation, and event reconstruction. What follows are the details of our solution along with the details of the related work. We intend to translate a Forensic Lucid specification into the PRISM specification, which is a probabilistic automata evaluation and model-checking system [15].

DEVELOPMENTS TO EHANCE FORENSIC LUCID WITH CREDIBILITY AND VISUALIZATION

1.2 Higher Order Context

HOCs represent essentially nested contexts, e.g. as conceptually shown in Figure 1 modeling evidential statement for forensic specification evaluation. Such a context representation can be modeled as a tree in an OO ontology or a context set. In Forensic Lucid it is expressed following the traditional Lucid syntax with modifications adapted from MARFL [16], e.g. with the main program illustrating the beginning of th Printer Case specification [17] in Listing 1.1.

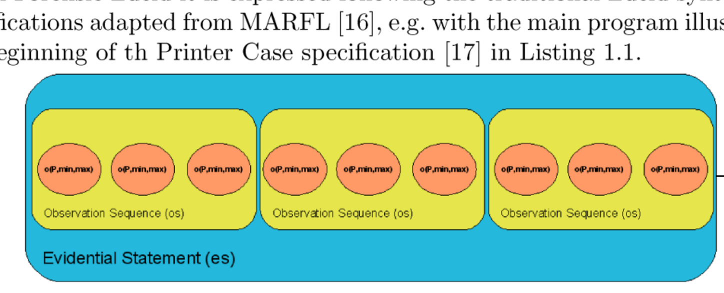


Fig. 1. Nested Context Hierarchy Example for Cyber Investigation

```
alice_claim @ es
where
  evidential statement es = [ printer, manuf, alice ];
  observation sequence printer = F;
  observation sequence manuf = [Oempty, $];
  observation sequence alice = [Oalice, F];
  observation F = (``B_deleted'', 1, 0);
  observation Oalice = (P_alice, 0, +inf);
  observation Oempty = (``empty'', 1, 0);

  // No ``add_A''
  P_alice = unordered {``add_B'', ``take''};

  invpsiacme(F, es);
end;
```

Listing 1.1. Developing the Pinter Case Example

1.3 Computing Credibility Weights

We augment the notion of observation to be formalized as:

$$o = (P, \min, \max, w, t) \tag{1}$$

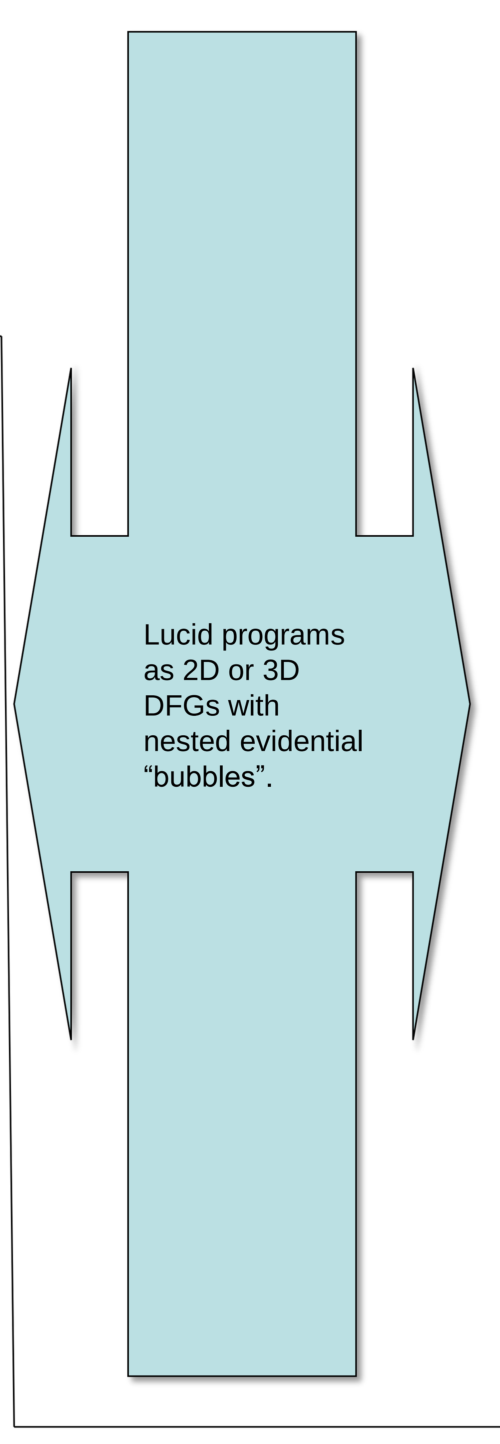
with the w being the credibility or trustworthiness weight of that observation, and the t being an optional wall-clock timestamp. With $w = 1$ the o would be equivalent to the original model proposed by Gladyshev. We define the total credibility of an observation sequence as an average of all the weights in this observation sequence.

$$W_{naive} = \frac{\sum(w_i)}{n} \tag{2}$$

A less naive way of calculating weights is using some pre-existing functions. What comes to mind is the activation functions used in artificial neural networks (ANNs), e.g.

$$W_{ANN} = \sum \frac{1}{(1 + e^{-nw_i})} \tag{3}$$

The witness stories or evidence with higher scores of W have higher credibility. With lower scores therefore less credibility and more tainted evidence.



The Need to Support of Data Flow Graph Visualization of Forensic Lucid Programs, Forensic Evidence, and their Evaluation by GIPSY

Serguei A. Mokhov
Concordia University
Montreal, QC, Canada
mokhov@cse.concordia.ca

Joey Paquet
Concordia University
Montreal, QC, Canada
paquet@cse.concordia.ca

Mourad Debbabi
Concordia University
Montreal, QC, Canada
debbabi@ciise.concordia.ca

ABSTRACT

Lucid programs are data-flow programs and can be visually represented as data flow graphs (DFGs) and composed visually. Forensic Lucid, a Lucid dialect, is a language to specify and reason about cyberforensics cases. It includes the encoding of the evidence (representing the context of evaluation) and the crime scene modeling in order to validate claims against the model and perform event reconstruction, potentially within large swaths of digital evidence. To aid investigators to model the scene and evaluate it, instead of typing a Forensic Lucid program, we propose to expand the design and implementation of the Lucid DFG programming onto Forensic Lucid case modeling and specification to enhance the usability of the language and the system.

Categories and Subject Descriptors

D.1.7 [**Programming Techniques**]: Visual Programming; D.2.11 [**Software Architectures**]: Domain-specific architectures; Languages—*Forensic Lucid*; D.3.2 [**Programming Languages**]: Language Classifications—*Very high-level languages; Multiparadigm languages;*; D.3.4 [**Programming Languages**]: Processors—*Compilers; Preprocessors; Run-time environments*

General Terms

Languages, Design, Security

Keywords

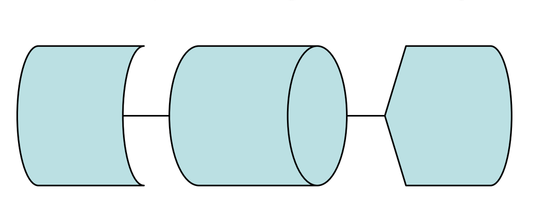
Forensic Lucid, DFG, GIPSY, forensic computing

1. OVERVIEW

Cyberforensic analysis has to do with automated or semi-automated processing of, and reasoning about, digital evidence, witness accounts, and other details from cybercrime incidents (involving computers, but not limited to them). Analysis is one of the phases in cybercrime investigation, where the other phases focus on evidence collection, preservation, chain of custody, information extraction that precede the analysis. The phases that follow the analysis are formulation of a report and potential prosecution, typically involving expert witnesses. There are quite a few techniques, tools (hardware and software), and methodologies have been developed for the mentioned phases of the process. A lot of attention has been paid to the tool development for evidence collection and preservation; a few tools have been developed to aid data "browsing" on the confiscated storage media, log files, memory, and so on. A lot less number of tools have been developed for case analysis of the data (e.g. Sleuthkit), and the existing commercial packages (e.g. Encase or FTK) are very expensive. Even less so there are case management, event modeling, and event reconstruction, especially with a solid formal theoretical base. The first formal approach to the cybercrime investigation was the finite-state automata (FSA) approach by Gladyshev et. al [5, 4]. Their approach, however, is unduly complex to use and to understand for non-theoretical-computer science or equivalent minded investigators. The aim of Forensic Lucid is to alleviate those difficulties, be sound and complete, expressive and usable, and provide even further usability improvements with the GUI to do data-flow graph-based (DFG) programming that allows translation between DFGs and the Forensic Lucid code for compilation. A previous work implemented a simpler solution for Indexical Lucid in the General Intensional Programming System (GIPSY) already [3], but requires forensic and imperative extensions. While Forensic Lucid is in the design and implementation phases, its solid base is being established in part with this work. The goal of Forensic Lucid in the cyberforensic analysis is to be able to express in a program form the encoding of the evidence, witness stories, and evidential statements, that can be tested against claims to see if there is a possible sequence or multiple sequences of events that explain a given *story*. As with the FSA, it is designed to aid investigators to avoid ad-hoc conclusions and have them look at the possible explanations the Forensic Lucid program "execution" would yield and refine the investigation, as was shown in the works [5, 4] investigators failed to analyze all the stories and their plausibility before drawing conclusions.

Related Work

There as a number of items and proposals in graph based visualization and the corresponding languages. In GIPSY, our own work in the area includes the theoretical foundation and initial practical implementation of the DFGs [17, 3]. Additionally, a part of the proposed related work on visualization and control of communication patterns and load balancing idea was to have a "3D editor" within RIPE's `DemandMonitor` that will render in 3D space the current communication patterns of a GIPSY program in execution or replay it back and allow the user visually to redistribute demands if they go off balance between workers. A kind of virtual 3D remote control with a mini expert system, an input from which can be used to teach the planning, caching, and load-balancing algorithms to perform efficiently next time a similar GIPSY application is run [7, 10]. Related work by other on visualization of load balancing, configuration, formal systems for diagrammatic modeling and visual languages and the corresponding graph systems are presented in [19, 18, 1, 2, 6, 16]. They all define some key concepts that are relevant to our visualization mechanisms within GIPSY and its corresponding General Manager Tier (GMT). We propose to build upon those works to represent the nested evidence, crime scene as a 3D DFG, and the reconstructed events flow upon evaluation. Such a feature is projected in the near future to support the previous work of the authors on intensional forensic computing, evidence modeling and encoding, and Forensic Lucid [11, 13, 12, 14, 15] and MARFL [9, 8] (where the intensional hybrid programming languages being realized within GIPSY platform to investigate the language properties and test the run-time). For that related work an example of a 2D DFG corresponding to a simple Lucid program is in Figure 1. In Figure 2 is the conceptual hierarchical nesting of the evidential statement es context elements, such as observation sequences os, their individual observations o, and the properties being observed (P, min, max, w, t) details of which are discussed in the referenced related works. These 2D conceptual visualizations are proposed to be renderable in 3D via an interactive interface to allow modeling complex crime scenes and multidimensional evidence on demand.



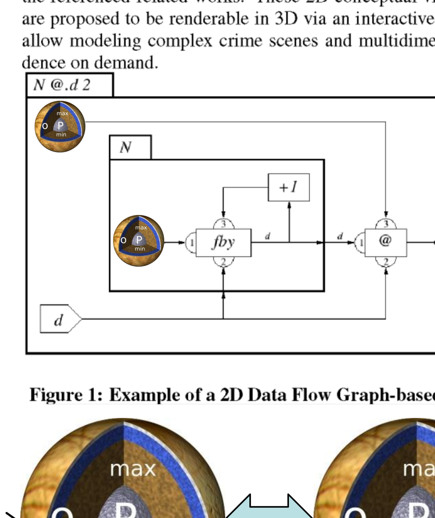
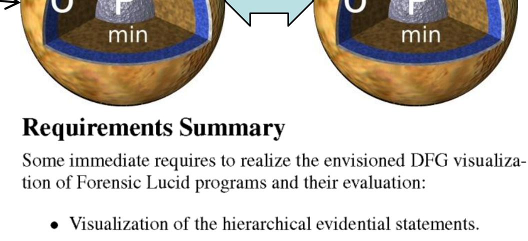


Figure 1: Example of a 2D Data Flow Graph-based Program



Requirements Summary

Some immediate requires to realize the envisioned DFG visualization of Forensic Lucid programs and their evaluation:

- Visualization of the hierarchical evidential statements.
- Placement of hybrid intensional-imperative nodes into the DFGs. Previous works did not deal with the way on how to augment the `DFGAnalyzer` and `DFGGenerator` to support hybrid GIPSY programs. This can be addressed by adding an unexpandable imperative DFG node to the graph. To make it more useful, i.e. expandable and so it's possible to generate the GIPSY code off it or reverse it.
- Java wrapper for the DFG Editor of Yimin Ding [3].
- Leveraging visualization and control of communication patterns and load balancing for the task in Euclidean space.

2. REFERENCES

[1] G. Allwein and J. Barwise, editors. *Logical reasoning with diagrams*. Oxford University Press, 1996.
[2] R. Bardohl, M. Minas, G. Taentzer, and A. Schürr. Application of graph transformation to visual languages. pages 105–180, 1999.
[3] Y. M. Ding. Bi-directional translation between data-flow graphs and Lucid programs in the GIPSY environment. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2004.
[4] P. Gladyshev. Finite state machine analysis of a blackmail investigation. *Int. J. of Digital Evidence*, 4(1), 2005.
[5] P. Gladyshev and A. Patel. Finite state machine approach to digital event reconstruction. *Digit. Investig. J.*, 2(1), 2004.
[6] N. G. Miller. *A Diagrammatic Formal System for Euclidean Geometry*. PhD thesis, Cornell University, U.S.A, 2001.
[7] S. A. Mokhov. Towards hybrid intensional programming with JLucid, Objective Lucid, and General Imperative Compiler Framework in the GIPSY. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2005.
[8] S. A. Mokhov. Encoding forensic multimedia evidence from MARF applications as Forensic Lucid expressions. In *CISSE'08*, pages 413–416. Springer, Dec. 2008.
[9] S. A. Mokhov. Towards syntax and semantics of hierarchical contexts in multimedia processing applications using MARFL. In *COMPSAC*, pages 1288–1294. IEEE CS, 2008.
[10] S. A. Mokhov. *Hybrid Intensional Computing in GIPSY: JLucid, Objective Lucid and GICF*. Lambert Academic Publishing, Mar. 2010. ISBN 978-3-8383-1198-2.
[11] S. A. Mokhov, J. Paquet, and M. Debbabi. Formally specifying operational semantics and language constructs of Forensic Lucid. In *IMF'08*, pages 197–216. GI, 2008.
[12] S. A. Mokhov, J. Paquet, and M. Debbabi. Reasoning about a simulated printer case investigation with Forensic Lucid. In *HSC'09*. SCS, 2009. To appear.
[13] S. A. Mokhov, J. Paquet, and M. Debbabi. Towards automated deduction in blackmail case analysis with Forensic Lucid. In *HSC'09*. SCS, 2009. To appear.
[14] S. A. Mokhov, J. Paquet, and M. Debbabi. Towards automatic deduction and event reconstruction using Forensic Lucid and probabilities to encode the IDS evidence. In *RAID'10*, LNCS 6307, pages 508–509. Springer, 2010.
[15] S. A. Mokhov and E. Vassev. Self-forensics through case studies of small to medium software systems. In *IMF'09*, pages 128–141. IEEE CS, 2009.
[16] OpenESB Contributors. BPEL service engine. [online], 2009. https://open-esb.dev.java.net/BPELSE.html.
[17] J. Paquet. *Scientific Intensional Programming*. PhD thesis, Department of Computer Science, Laval University, Sainte-Foy, Canada, 1999.
[18] P. C. Vinh and J. P. Bowen. On the visual representation of configuration in reconfigurable computing. *Electron. Notes Theor. Comput. Sci.*, 109:3–15, 2004.
[19] C. Zheng and J. R. Heath. Simulation and visualization of resource allocation, control, and load balancing procedures for a multiprocessor architecture. In *IASTED'06*, pages 382–387. ACTA Press, 2006.

VizSec 2010, Ottawa, ON, Canada, September 14, 2010