Reasoning About a Simulated Printer Case Investigation with Forensic Lucid[1]

Serguei A. Mokhov Joey Paquet Mourad Debbabi

Department of Computer Science and Software Engineering
Faculty of Engineering and Computer Science
Concordia University, Montréal, Québec, Canada,
`{mokhov,paquet,debbabi}@encs.concordia.ca`

ICDF2C 2011, Dublin, Ireland

[1] presented on behalf of the authors by **Andrei Soeanu**

Outline

Outline

Outline

Outline

The Problem I

- The first formal approach for cyberforensic analysis and event reconstruction appeared in two papers [GP04, Gla05] by Gladyshev et al. that relies on the finite-state automata (FSA) and their transformation and operation to model evidence, witnesses, stories told by witnesses, and their possible evaluation.
- One of the examples the papers present is the use-case for the proposed technique – the ACME Printer Case Investigation. See [GP04] for the formalization using FSA by Gladyshev and the corresponding LISP implementation.

The Problem II

- We aim at the same case to model and implement it using the new approach, which paves a way to be more friendly and usable in the actual investigator's work and serve as a basis to further development in the area.

Overview I

- In this work we model the ACME (a fictitious company name) printer case incident and make its specification in Forensic Lucid, a Lucid- and intensional-logic-based programming language for cyberforensic analysis and event reconstruction specification.
- The printer case involves a dispute between two parties that was previously solved using the finite-state automata (FSA) approach, and is now re-done in a more usable way in Forensic Lucid.

Overview II

- Our simulation is based on the said case modeling by encoding concepts like evidence and the related witness accounts as an evidential statement context in a Forensic Lucid program, which is an input to the transition function that models the possible deductions in the case.
- We then invoke the transition function (actually its reverse) with the evidential statement context to see if the evidence we encoded agrees with one's claims and then attempt to reconstruct the sequence of events that may explain the claim or disprove it.

Intensional Cyberforensics I

- Intensional Cyberforensics project
    - Cyberforensics
    - Case modeling and analysis
    - Event reconstruction
    - Language and Programming Environment
- Forensic Lucid – functional intensional forensic case programming and specification language, covering:
    - Syntax
    - Semantics
    - Compiler
    - Run-time System
    - General Intensional Programming System (GIPSY)

Intensional Cyberforensics II

- Operational aspects:
  - Operators
  - Operational Semantics
- Based on:
  - Lucid
  - Higher-Order Intensional Logic (HOIL)
  - Intensional Programming

Lucid I

- Lucid [WA85, AFJW95, AW77b, AW76, AW77a] is a dataflow intensional and functional programming language.
- In fact, it is a family of languages that are built upon intensional logic (which in turn can be understood as a multidimensional generalization of temporal logic) involving context and demand-driven parallel computation model.
- A program written in some Lucid dialect is an expression that may have subexpressions that need to be evaluated at certain *context*.

Lucid II

- Given the set of dimension $D = \{dim_i\}$ in which an expression varies, and a corresponding set of indexes or *tags* defined as placeholders over each dimension, the context is represented as a set of $<dim_i : tag_i>$ mappings and each variable in Lucid, called often a *stream*, is evaluated in that defined context that may also evolve using context operators [PMT08, Ton08, WAP05, Wan06].
- The generic version of Lucid, GIPL [Paq99], defines two basic operators `@` and `#` to navigate in the contexts (switch and query).
- The GIPL was the first generic programming language of all intensional languages, defined by the means of only two intensional operators `@` and `#`.

Lucid III

- It has been proven that other intensional programming languages of the Lucid family can be translated into the GIPL [Paq99].
- Since the Lucid family of language thrived around intensional logic that makes the notion of context explicit and central, and recently, a first class value [WAP05, Wan06, PMT08, Ton08] that can be passed around as function parameters or as return values and have a set of operators defined upon.
- We greatly draw on this notion by formalizing our evidence and the stories as a contextual specification of the incident to be tested for consistency against the incident model specification.

Lucid IV

- In our specification model we require more than just atomic context values – we need a higher-order context hierarchy to specify different level of detail of the incident and being able to navigate into the “depth” of such a context.
- A similar provision by has already been made by the author [Mok08] and earlier works of Swoboda et al. in [Swo04, SW00, SP04b, SP04a] that needs some modifications to the expressions of the cyberforensic context.
- Some other languages can be referred to as intensional even though they may not refer to themselves as such, and were born after Lucid (Lucid began in 1974).

Lucid V

- Examples include hardware-description languages (HDLs, appeared in 1977) where the notion of time (often the only "dimension", and usually progresses only forward), e.g. Verilog and VHDL.
- Another branch of newer languages for the becoming popular is aspect-oriented programming (AOP) languages, that can have a notion of context explicitly, but primarily focused on software engineering aspect of software evolution and maintainability.

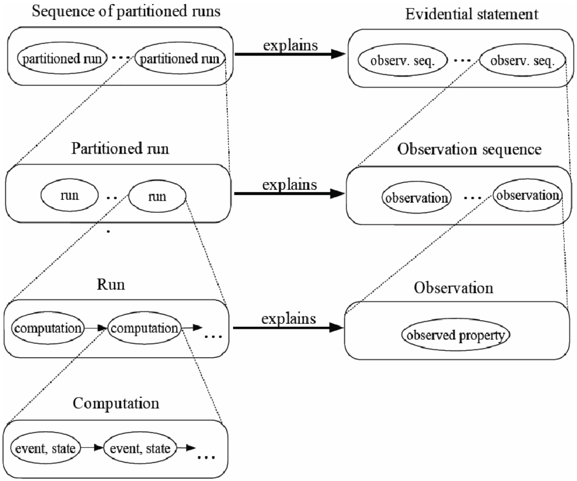
Forensic Lucid I

- A summary of the concepts and considerations in the design of the Forensic Lucid language, large portions of which were studied in the earlier work [MP08, MPD08].
- The end goal of the language design is to define its constructs to concisely express cyberforensic evidence as context of evaluations, which can be initial state of the case towards what we have actually observed (as corresponding to the final state in the Gladyshev's FSM).
- One of the evaluation engines (a topic of another paper) of the implementing system [The12] is designed to backtrace intermediate results to provide the corresponding event reconstruction path if it exists.

Forensic Lucid II

- The result of the expression in its basic form is either *true* or *false*, i.e. “guilty” or “not guilty” given the evidential evaluation context per explanation with the backtrace(s).
- There can be multiple backtraces, that correspond to the explanation of the evidence (or lack thereof).

Gladyshev's Meaning and Explanation Hierarchy



Higher Order Context

- HOCs represent essentially nested contexts, 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.

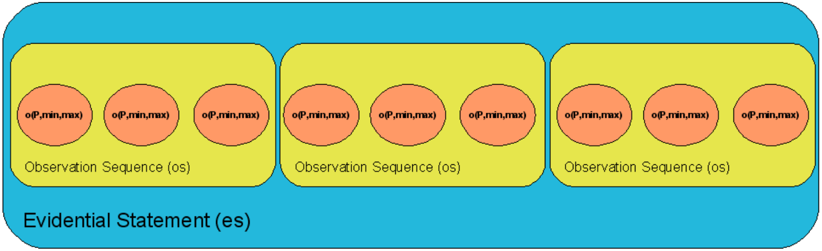


Figure: Nested Context Hierarchy Example for Digital Investigation

ACME Manufacturing Printing Case I

This is one of the cases we re-examine from the Gladyshev's FSA approach [GP04].

- *The local area network at some company called ACME Manufacturing consists of two personal computers and a networked printer.*
- *The cost of running the network is shared by its two users Alice (A) and Bob (B).*
- *Alice, however, claims that she never uses the printer and should not be paying for the printer consumables.*
- *Bob disagrees, he says that he saw Alice collecting printouts.*
- *According to the manufacturer, the printer works as follows:*

ACME Manufacturing Printing Case II

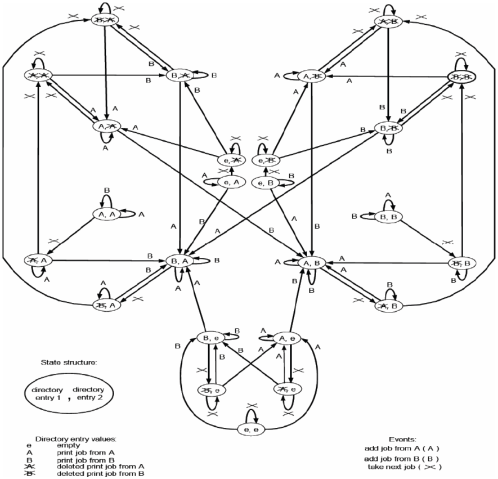
1. When a print job is received from the user, it is stored in the first unallocated directory entry of the print job directory.
2. The printing mechanism scans the print job directory from the beginning and picks the first active job.
3. After the job is printed, the corresponding directory entry is marked as “deleted”, but the name of the job owner is preserved.
4. The printer can accept only one print job from each user at a time.
5. Initially, all directory entries are empty.

ACME Manufacturing Printing Case III

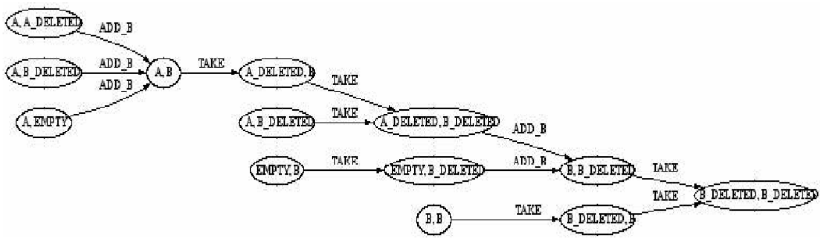
The investigator finds the current state of the printer's buffer as:

1. Job From B Deleted
2. Job From B Deleted
3. Empty
4. Empty
5. ...

Gladyshev's Printer Case State Machine



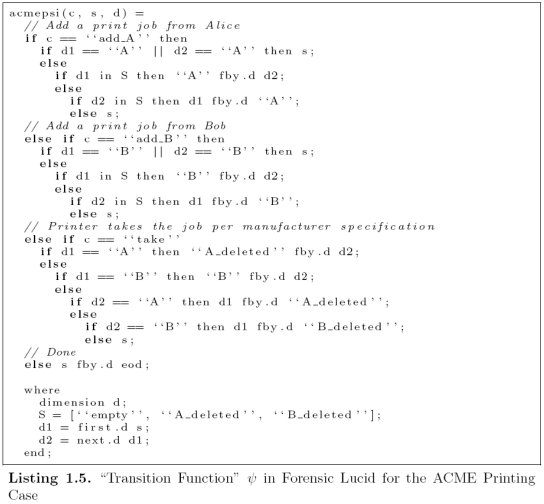
Paths Leading to $(B\_Deleted, B\_Deleted)$



```
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: Developing the Pinter Case Example 3

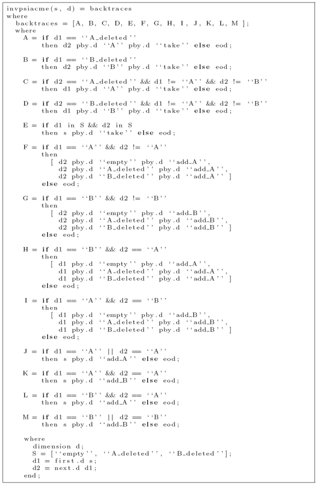
Transition Function" ψ in Forensic Lucid



```
acmepsi(c, s, d) =
  // Add a print job from Alice
  if c == ‘‘add_A’’ then
    if d1 == ‘‘A’’ || d2 == ‘‘A’’ then s;
    else
      if d1 in S then ‘‘A’’ fby.d d2;
      else
        if d2 in S then d1 fby.d ‘‘A’’;
        else s;
  // Add a print job from Bob
  else if c == ‘‘add_B’’ then
    if d1 == ‘‘B’’ || d2 == ‘‘B’’ then s;
    else
      if d1 in S then ‘‘B’’ fby.d d2;
      else
        if d2 in S then d1 fby.d ‘‘B’’;
        else s;
  // Printer takes the job per manufacturer specification
  else if c == ‘‘take’’
    if d1 == ‘‘A’’ then ‘‘A_deleted’’ fby.d d2;
    else
      if d1 == ‘‘B’’ then ‘‘B’’ fby.d d2;
      else
        if d2 == ‘‘A’’ then d1 fby.d ‘‘A_deleted’’;
        else
          if d2 == ‘‘B’’ then d1 fby.d ‘‘B_deleted’’;
          else s;
  // Done
  else s fby.d eod;

  where
    dimension d;
    S = [‘‘empty’’, ‘‘A_deleted’’, ‘‘B_deleted’’];
    d1 = first.d s;
    d2 = next.d d1;
  end;
```

Listing 1.5. "Transition Function" ψ in Forensic Lucid for the ACME Printing Case

Inverse Transition Function” Ψ^{-1} in Forensic Lucid



```
invpsiacme(s, d) = backtraces
where
  backtraces = [A, B, C, D, E, F, G, H, I, J, K, L, M ];
  where
    A = if d1 == ``A_deleted''
        then d2 pby.d ``A'' pby.d ``take'' else eod;

    B = if d1 == ``B_deleted''
        then d2 pby.d ``B'' pby.d ``take'' else eod;

    C = if d2 == ``A_deleted'' && d1 != ``A'' && d2 != ``B''
        then d1 pby.d ``A'' pby.d ``take'' else eod;

    D = if d2 == ``B_deleted'' && d1 != ``A'' && d2 != ``B''
        then d1 pby.d ``B'' pby.d ``take'' else eod;

    E = if d1 in S && d2 in S
        then s pby.d ``take'' else eod;

    F = if d1 == ``A'' && d2 != ``A''
        then
          [ d2 pby.d ``empty'' pby.d ``add_A'',
            d2 pby.d ``A_deleted'' pby.d ``add_A'',
            d2 pby.d ``B_deleted'' pby.d ``add_A'' ]
        else eod;

    G = if d1 == ``B'' && d2 != ``B''
        then
          [ d2 pby.d ``empty'' pby.d ``add_B'',
            d2 pby.d ``A_deleted'' pby.d ``add_B'',
            d2 pby.d ``B_deleted'' pby.d ``add_B'' ]
        else eod;

    H = if d1 == ``B'' && d2 == ``A''
        then
          [ d1 pby.d ``empty'' pby.d ``add_A'',
            d1 pby.d ``A_deleted'' pby.d ``add_A'',
            d1 pby.d ``B_deleted'' pby.d ``add_A'' ]
        else eod;

    I = if d1 == ``A'' && d2 == ``B''
        then
          [ d1 pby.d ``empty'' pby.d ``add_B'',
            d1 pby.d ``A_deleted'' pby.d ``add_B'',
            d1 pby.d ``B_deleted'' pby.d ``add_B'' ]
        else eod;

    J = if d1 == ``A'' || d2 == ``A''
        then s pby.d ``add_A'' else eod;

    K = if d1 == ``A'' && d2 == ``A''
        then s pby.d ``add_B'' else eod;

    L = if d1 == ``B'' && d2 == ``A''
        then s pby.d ``add_A'' else eod;

    M = if d1 == ``B'' || d2 == ``B''
        then s pby.d ``add_B'' else eod;

    where
      dimension d;
      S = [``empty'', ``A_deleted'', ``B_deleted''];
      d1 = first.d s;
      d2 = next.d d1;
    end;
```

Listing 1.6. “Inverse Transition Function” Ψ^{-1} in Forensic Lucid for the ACME Printing Case

Conclusion I

- We presented the basic overview of Forensic Lucid, its concepts, ideas, and dedicated purpose – to model, specify, and evaluation digital forensics cases.
- The process of doing so is significantly simpler and more manageable than the previously proposed FSM model and its common LISP realization. At the same time, the language is founded in more than 30 years research on correctness and soundness of programs and the corresponding mathematical foundations of the Lucid language, which is a significant factor should a Forensic Lucid-based analysis be presented in court.

Conclusion II

- We re-wrote in Forensic Lucid one of the sample cases initial modeled by Gladyshev in the FSM and Common LISP to show the specification is indeed more manageable and comprehensible than the original and fits in two pages in this paper.
- We also still realize by looking at the examples the usability aspect is still desired to be improved further for the investigators, especially when modeling ψ and Ψ^{-1}, as a potential limitation, prompting one of the future work items to address it further.

Conclusion III

- In general, the proposed practical approach in the cyberforensics field can also be used to model and evaluate normal investigation process involving crimes not necessarily associated with information technology.
- Combined with an expert system (e.g. implemented in CLIPS [Ril09]), it can also be used in training new staff in investigation techniques. The notion of hierarchical contexts as first-class values brings more understanding of the process to the investigators in cybercrime case management tools.

Future Work

- Formally prove equivalence to the FSA approach.
- Adapt/re-implement a graphical UI based on the data-flow graph tool [Din04, MPD11] to simplify Forensic Lucid programming further for not very tech-savvy investigators by making it visual. The listings provided are not very difficult to read and quite manageable to comprehend, but any visual aid is always an improvement.
- Refine the semantics of Lucx's context sets and their operators to be more sound, including Box and Range.
- Explore and exploit the notion of credibility factors of the evidence and witnesses fully.
- Release a full standard Forensic Lucid specification.

Ongoing Work: Computing Credibility Weights I

- 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.

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

Ongoing Work: Computing Credibility Weights II

- 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.

Acknowledgments

- This research work was funded by NSERC and the Faculty of Engineering and Computer Science of Concordia University, Montreal, Canada.
- We would also like to acknowledge the reviewers who took time to do a constructive quality review of this work.
- Thanks to Andrei Soeanu for the last minute accommodation to be able to present this work at ICDF2C 2011 in Dublin, on behalf of the authors.

References I

Edward A. Ashcroft, Anthony A. Faustini, Rangaswamy Jagannathan, and William W. Wadge.
Multidimensional Programming.
Oxford University Press, London, February 1995.
ISBN: 978-0195075977.

Edward A. Ashcroft and William W. Wadge.
Lucid – a formal system for writing and proving programs.
SIAM J. Comput., 5(3), 1976.

Edward A. Ashcroft and William W. Wadge.
Erratum: Lucid – a formal system for writing and proving programs.
SIAM J. Comput., 6(1):200, 1977.

References II

Edward A. Ashcroft and William W. Wadge.
Lucid, a nonprocedural language with iteration.
Communications of the ACM, 20(7):519–526, July 1977.

Yimin Ding.
Automated translation between graphical and textual representations of intensional programs in the GIPSY.
Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, June 2004.
http://newton.cs.concordia.ca/~paquet/filetransfer/publications/theses/DingYiminMSc2004.pdf.

Pavel Gladyshev.
Finite state machine analysis of a blackmail investigation.
International Journal of Digital Evidence, 4(1), 2005.

References III

Pavel Gladyshev and Ahmed Patel.
Finite state machine approach to digital event reconstruction.
Digital Investigation Journal, 2(1), 2004.

Serguei A. Mokhov.
Towards syntax and semantics of hierarchical contexts in multimedia processing applications using MARFL.
In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pages 1288–1294, Turku, Finland, July 2008. IEEE Computer Society.

References IV

Serguei A. Mokhov and Joey Paquet.
Formally specifying and proving operational aspects of Forensic Lucid in Isabelle.
Technical Report 2008-1-Ait Mohamed, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, August 2008.
In Theorem Proving in Higher Order Logics (TPHOLs2008): Emerging Trends Proceedings.

Serguei A. Mokhov, Joey Paquet, and Mourad Debbabi.
Formally specifying operational semantics and language constructs of Forensic Lucid.
In Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, and Dirk Schadt, editors, *Proceedings of the IT Incident Management and IT Forensics (IMF'08)*, LNI140, pages 197–216. GI, September 2008.

References V

Serguei A. Mokhov, Joey Paquet, and Mourad Debbabi.
On the need for data flow graph visualization of Forensic Lucid programs and forensic evidence, and their evaluation by GIPSY.
In *Proceedings of the Ninth Annual International Conference on Privacy, Security and Trust (PST), 2011*, pages 120–123. IEEE Computer Society, July 2011.
Short paper; full version online at `http://arxiv.org/abs/1009.5423`.

Joey Paquet.
Scientific Intensional Programming.
PhD thesis, Department of Computer Science, Laval University, Sainte-Foy, Canada, 1999.

References VI

Joey Paquet, Serguei A. Mokhov, and Xin Tong.
Design and implementation of context calculus in the GIPSY environment.
In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pages 1278–1283, Turku, Finland, July 2008. IEEE Computer Society.

Gary Riley.
CLIPS: A tool for building expert systems.
[online], 2007–2009.
`http://clipsrules.sourceforge.net/`, last viewed: October 2009.

References VII

Paul Swoboda and John Plaice.
An active functional intensional database.
In F. Galindo, editor, *Advances in Pervasive Computing*, pages 56–65. Springer, 2004.
LNCS 3180.

Paul Swoboda and John Plaice.
A new approach to distributed context-aware computing.
In A. Ferscha, H. Hoertner, and G. Kotsis, editors, *Advances in Pervasive Computing*. Austrian Computer Society, 2004.
ISBN 3-85403-176-9.

References VIII

Paul Swoboda and William W. Wadge.
Vmake, ISE, and IRCS: General tools for the intensionalization of software systems.
In M. Gergatsoulis and P. Rondogiannis, editors, *Intensional Programming II*. World-Scientific, 2000.

Paul Swoboda.
A Formalisation and Implementation of Distributed Intensional Programming.
PhD thesis, The University of New South Wales, Sydney, Australia, 2004.

References IX

The GIPSY Research and Development Group.
The General Intensional Programming System (GIPSY) project.
Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2002–2012.
http://newton.cs.concordia.ca/~gipsy/, last viewed February 2010.

Xin Tong.
Design and implementation of context calculus in the GIPSY.
Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, April 2008.

William W. Wadge and Edward A. Ashcroft.
Lucid, the Dataflow Programming Language.
Academic Press, London, 1985.

References X

- Kaiyu Wan.
  Lucx: Lucid Enriched with Context.
  PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2006.

- Kaiyu Wan, Vasu Alagar, and Joey Paquet.
  Lucx: Lucid enriched with context.
  In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005)*, pages 48–14. CSREA Press, June 2005.