Toward Formal Reasoning in Cyberforensic Case Investigation with Forensic Lucid

Serguei A. Mokhov

Senior Scholar, visiting Visualization and Graphics Lab, Tsinghua University, from the Department of Computer Science and Software Engineering, Concordia University, Montréal, Québec, Canada,
mokhov@cse.concordia.ca

May 8, 2012, Tsinghua University, Beijing, China

清華大學 Tsinghua University

Part I

Background Overview

Part I Outline

Part I Outline

清華大學
Tsinghua University

Research Summary I

- The research project involves, among other things, creating a credible tool, well founded in science, to manage (potentially vast amounts of) digital evidence data, as well as descriptions of non-digital evidence, and witness accounts related to computer crime (and beyond), all in one common format in order to verify claims without omitting details and helping investigators to avoid ad-hoc conclusions and perform event reconstruction if the claim agrees with the evidence collected.

Research Summary II

- Essentially, an investigator gathers evidential data from various sources, such as logs, memory analysis, disk analysis tools, network traces, IDS logs, data-mining tools, physical evidence, human witness accounts (input and encoded manually) of the events in a knowledge base like evidential statement, against which claims (e.g. of the accused or prosecution) are validated automatically.
- The implementation and mathematical details are being finalized, but the design and semantics of a language (Forensic Lucid) that would allow investigators do such things with the evidential data are already in place and some of that was published in the peer-reviewed venues.

Founding Publications I

- S. A. Mokhov, J. Paquet, and M. Debbabi. Reasoning about a simulated printer case investigation with Forensic Lucid. In P. Gladyshev, editor, *Proceedings of ICDF2C'11*. Springer, Oct. 2011. To appear, online at `http://arxiv.org/abs/0906.5181`
- S. A. Mokhov, J. Paquet, and M. Debbabi. Towards automated deduction in blackmail case analysis with Forensic Lucid. In J. S. Gauthier, editor, *Proceedings of the Huntsville Simulation Conference (HSC'09)*, pages 326–333. SCS, Oct. 2009. ISBN 978-1-61738-587-2. Online at `http://arxiv.org/abs/0906.0049`
- S. A. Mokhov, J. Paquet, and M. 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. ISBN 978-1-4577-0582-3. doi: 10.1109/PST.2011.5971973. Short paper; full version online at `http://arxiv.org/abs/1009.5423`

Founding Publications II

- S. A. Mokhov, J. Paquet, and M. Debbabi. Formally specifying operational semantics and language constructs of Forensic Lucid. In O. Göbel, S. Frings, D. Günther, J. Nedon, and D. Schadt, editors, *Proceedings of the IT Incident Management and IT Forensics (IMF'08)*, LNI140, pages 197–216. GI, Sept. 2008. ISBN 978-3-88579-234-5. Online at `http://subs.emis.de/LNI/Proceedings/Proceedings140/gi-proc-140-014.pdf`
- S. A. Mokhov and J. Paquet. Using the General Intensional Programming System (GIPSY) for evaluation of higher-order intensional logic (HOIL) expressions. In *Proceedings of SERA 2010*, pages 101–109. IEEE Computer Society, May 2010. ISBN 978-0-7695-4075-7. doi: 10.1109/SERA.2010.23. Online at `http://arxiv.org/abs/0906.3911`

清華大學
Tsinghua University

Speaker's Bio

- PhD Candidate in Computer Science, Concordia University, Montreal, Canada where he completed his bachelor's and master's degrees in Computer Science and Information Systems Security.
- Presently a Senior Scholar visiting Tsinghua University, Beijing, China, the Department of Computer Science and Technology, the Visualization and Graphics Lab with Dr. Yankui Sun.
- A part-time faculty member at the Department of Computer Science and Software Engineering and a Systems Administrator in the Faculty of Engineering and Computer Science's Network Administration Group.
- He is also an Assistant Editor of Scholarpedia, a peer-reviewed open-access encyclopedia, TPC member in several conferences and a referee for a few journals.
- Mr. Mokhov's multidisciplinary research includes diverse aspects in distributed and parallel computing, pattern recognition and data mining, computer forensics and security, computer graphics and visualization, AI and natural language processing, intensional programming, autonomic computing, and software engineering, where he has had a number of publications.
- Strong proponent and developer of open-source software, having contributed most of his research software and the results to the community.

The Problem I

- The first formal approach for cyberforensic analysis and event reconstruction appeared in two papers [13, 12] 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.
- The examples the papers present are the use-case for the proposed technique – the ACME Printer Case Investigation and a Blackmail Investigation. See [13, 12] for the formalization using FSA by Gladyshev and the corresponding LISP implementation.

The Problem II

- We first aim at the same cases 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, introduces the notion of credibility, 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 and blackmail case incidents and make their specification in Forensic Lucid, a Lucid- and intensional-logic-based programming language for cyberforensic analysis and event reconstruction specification.
- Our initial work is based on the said cases 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.

Overview II

- 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.
- The evaluation is naturally parallel and the GIPSY's run-time system supports distributed demand-driven (eductive) evaluation for scalability and efficiency reasons when needed.

Intensional Cyberforensics I

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

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.

- E_1 := it is raining **here today**
  Context: {`place`:**here**, `time`:**today**}
- E_2 := it was raining **here** *before*(**today**) = *yesterday*
- E_3 := it is going to rain *at* (altitude **here** + 500 m)
  after(**today**) = *tomorrow*
- E_1: fix **here** to **Beijing** and assume it is a *constant*. In the month of May 2012, 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 ...
```

清華大學 Tsinghua University

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	...
Beijing	F	F	T	T	T	F	F	F	T	...
Montreal	F	F	F	F	T	T	T	F	F	...
Sydney	F	T	T	T	T	T	F	F	F	...

Higher-Order Intensional Logic (HOIL) I

- Intensional programming (IP) is based on intensional (or multidimensional) logics, which, in turn, are based on natural language understanding aspects (such as time, belief, situation, and direction).
- Intensional logic adds dimensions to logical expressions; thus, a non-intensional logic can be seen as a constant or a snapshot in all possible dimensions.
- *Intensions are dimensions* at which a certain statement is true or false (or has some other than a Boolean value).
- *Intensional operators* are operators that allow us to navigate within these dimensions.

Higher-Order Intensional Logic (HOIL) II

- *Higher-order intensional logic* (HOIL) is the one that couples functional programming as that of Lucid with multidimensional dataflows that the intensional programs can query an alter through an explicitly notion of contexts as first-class values.

Higher-Order Intensional Logic (HOIL) III

- To summarize, expressions written in virtually all Lucid dialects correspond to higher-order intensional logic (HOIL) expressions with some dialect-specific instantiations.
- They all can alter the context of their evaluation given a set of operators and in some cases types of contexts, their range, and so on.
- HOIL combines functional programming and intensional logics.
- The contextual expression can be passed as parameters and returned as results of a function and constitute the multi-dimensional constraint on the Lucid expression being evaluated.

Higher-Order Intensional Logic (HOIL) IV

- The corresponding context calculus [57, 38, 49] defines a comprehensive set of context operators, most of which are set operators and the baseline operators are @ and # that allow to switch the current context or query it, respectively.
- Other operators allow to define a context space and a point in that context corresponding to the current context.
- The context can be arbitrary large in its rank.
- The identified variables of the dimension type within the context can take on any data type, e.g. an integer, or a string, during lazy binding of the resulting context to a dimension identifier.

Lucid I

- Lucid [56, 5, 4, 2, 3] 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 [38, 49, 58, 57].
- The generic version of Lucid, GIPL [36], 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 [36].
- 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 [58, 57, 38, 49] 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 [23] and earlier works of Swoboda et al. in [43, 46, 45, 44] 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 [10], that can have a notion of context explicitly, but primarily focused on software engineering aspect of software evolution and maintainability.

JLucid, Objective Lucid, and JOOIP I

- JLucid [22] was a first attempt on intensional arrays and "free Java functions" in the GIPSY. The approach used the Lucid language as the driving main computation, where Java methods were peripheral and could be invoked from the Lucid part, but not the other way around.
- This was the first instance of hybrid programming within the GIPSY. The semantics of this approach was not completely defined, plus, it was only one-sided view (Lucid-to-Java) of the problem. JLucid did not support objects of any kind, but introduced the wrapper class idea for the free Java methods and served as a precursor to Objective Lucid.

JLucid, Objective Lucid, and JOOIP II

- Objective Lucid [22] is an extension of the JLucid language that inherits all of the JLucid's features and introduced Java objects to be available for use by Lucid. Objective Lucid expanded the notion of the Java object (a collection of members of different types) to the array (a collection of members of the same type) and first introduced the dot-notation in the syntax and operational semantics in GIPSY.

JLucid, Objective Lucid, and JOOIP III

- Like in JLucid, Objective Lucid's focus was on the Lucid part being the "main" program and did not allow Java to call intensional functions or use intensional constructs from within a Java class. Objective Lucid was the first in GIPSY to introduce the more complete operational semantics of the hybrid OO intensional language.

JLucid, Objective Lucid, and JOOIP IV

- JOOIP [59] greatly complements Objective Lucid by allowing Java to call the intensional language constructs closing the gap and making JOOIP a complete hybrid OO intensional programming language within the GIPSY environment. JOOIP's semantics further refines in a greater detail the operational semantics rules of Lucid and Objective Lucid in the attempt to make them complete.

Context-Oriented Reasoning

- Evaluation of Lucid expressions
- Reasoning About Cyberforensic Investigation Cases and Forensic Lucid
  - A Lucid dialect, Forensic Lucid [28, 24, 29, 32, 26] develops a specification of a cyberincident for analysis of claims of witnesses against encoded evidential statements to see if they agree or not and if they do provide potential backtraces of event reconstruction.
- Reasoning in Hybrid OO Environment
  - JOOIP [59, 60] is offering Lucid fragments within Java code and allowing the Lucid code to reference to Java methods and variables, along with the corresponding type system extensions and providing context-aware Java objects.
- Reasoning in Autonomic Environment
  - Vassev and Paquet designed an Autonomic GIPSY [53] (AGIPSY) version of the platform with the corresponding ASSL toolset [54, 52, 51] as a research case study.

GIPSY I

- The General Intensional Programming System (GIPSY) has been built around the Lucid family of intensional programming languages that rely on the higher-order intensional logic (HOIL) to provide context-oriented multidimensional reasoning of intensional expressions.
- It executes Lucid programs following a demand-driven distributed generator-worker architecture, and is designed as a modular collection of frameworks where components related to the development (RIPE, Run-time Integrated Programming Environment), compilation (GIPC, General Intensional Programming Compiler), and execution (GEE, General Eduction Engine) of Lucid programs are separated

GIPSY II

allowing easy extension, addition, and replacement of the components.

- GIPSY provides support for hybrid programming models that couple intensional and imperative languages for a variety of needs.
- Explicit context expressions limit the scope of evaluation of math expressions (effectively a Lucid program is a mathematics or physics expression constrained by the context) in tensor physics, regular math in multiple dimensions, etc., and for cyberforensic reasoning as one of the use-cases of interest.

GIPSY III

- Thus, GIPSY is a support testbed for HOIL-based languages some of which enable such reasoning, as in formal cyberforensic case analysis with event reconstruction. This is a proposed testing and investigation platform for our *Forensic Lucid* language.
- Summary of technologies: Java, Jini (Apache River) and JMS for distributed middleware, GIPSY cluster; multi-tier architecture.

GIPC Framework

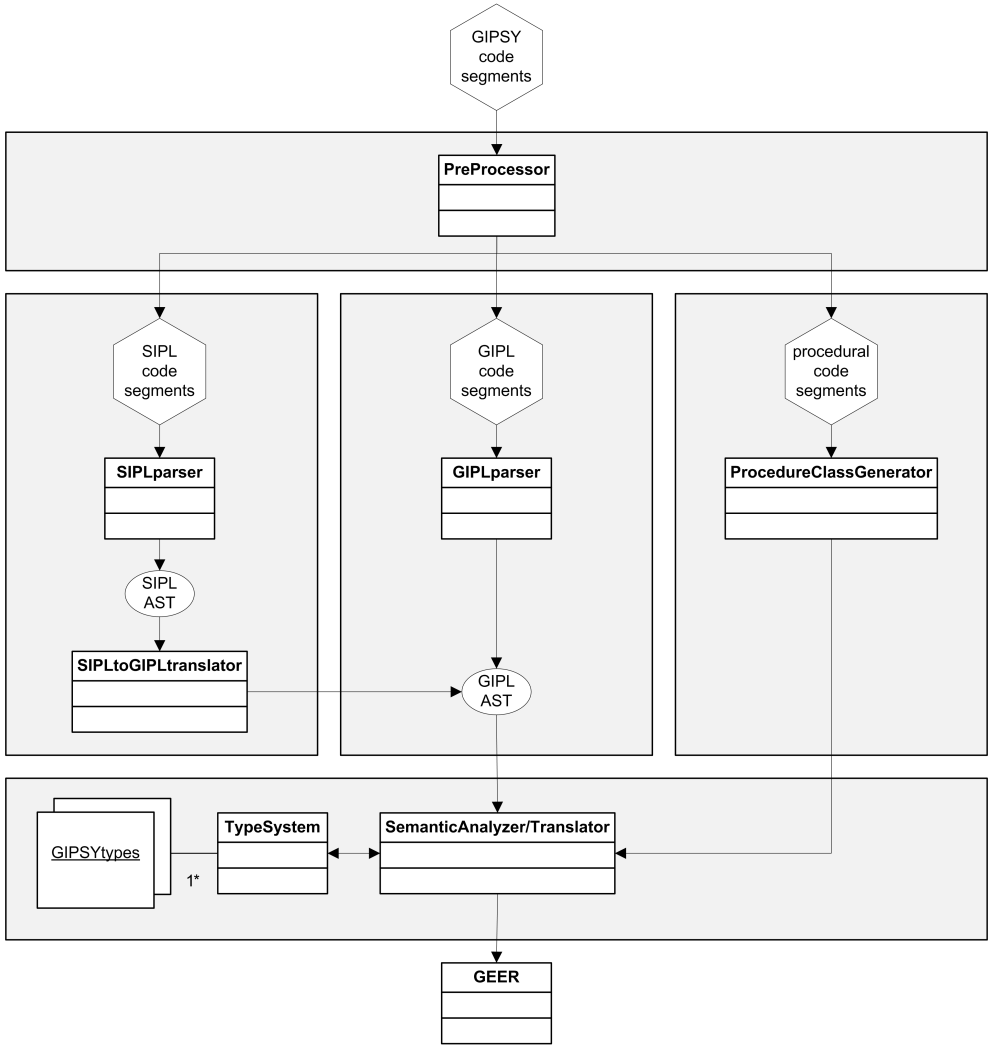


Figure: GIPC Framework [30]

GIPSY's GIPC-to-GEE GEER Flow Overview

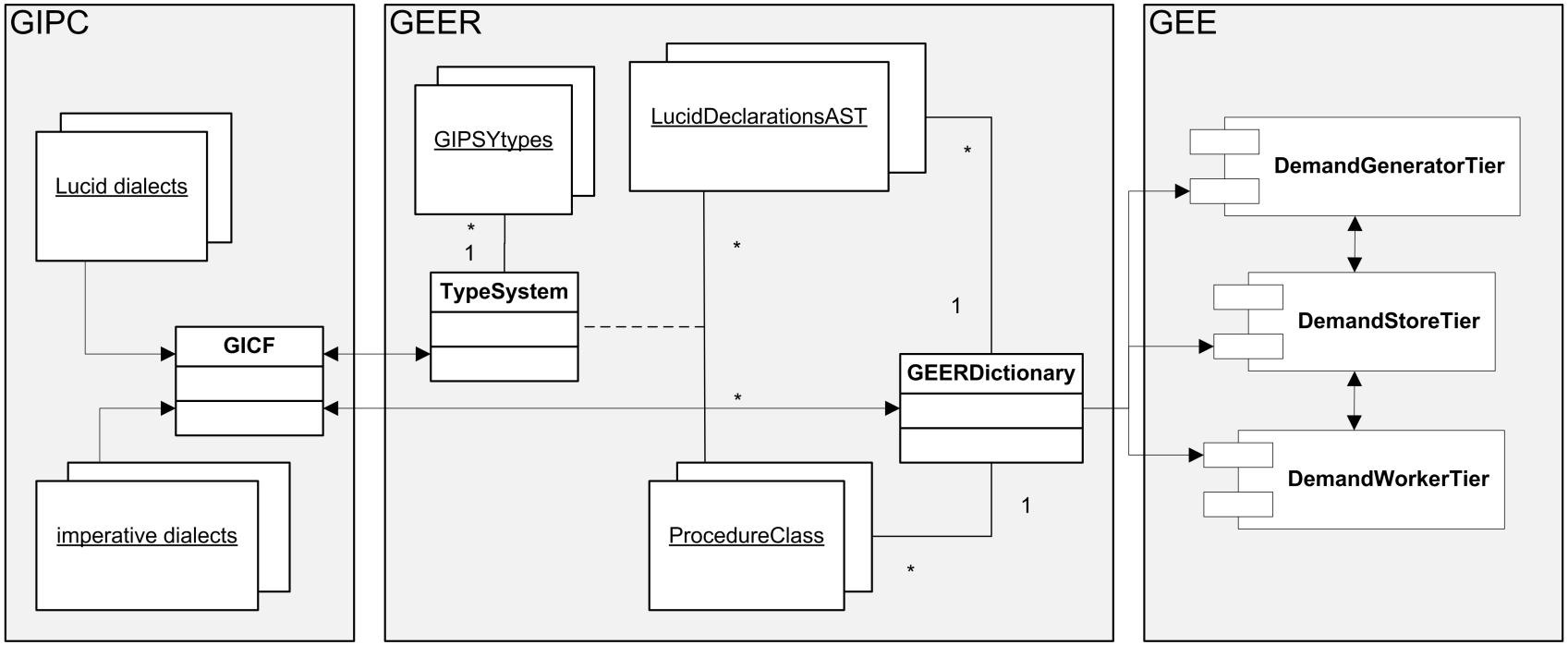


Figure: GIPSY's GIPC-to-GEE GEER Flow Overview [30]

Design of the GIPSY Node

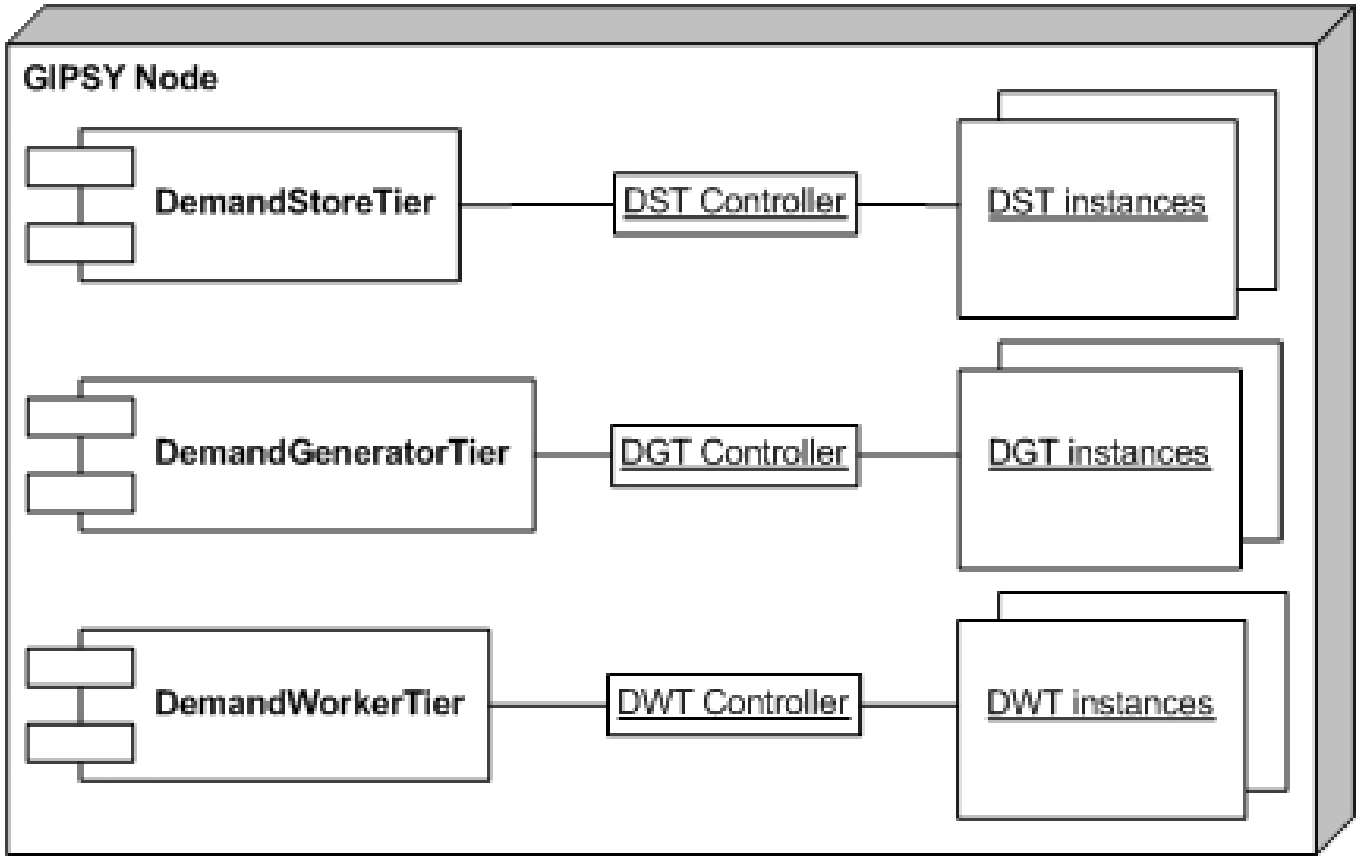


Figure: Design of the GIPSY Node [37, 16]

Part II

Forensic Lucid

清華大學
Tsinghua University

Part II Outline

Part II Outline

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 [24, 28].
- 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 [47] 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).

Features I

- We define Forensic Lucid to model the evidential statements and other expressions representing the evidence and observations as a higher-order context. An execution trace of a Forensic Lucid program would expose the possibility of the proposed claim with the events in the middle between the final observed event to the beginning of the events. Forensic Lucid aggregates the features of multiple Lucid dialects mentioned earlier needed for these tasks along with its own extensions.

Features II

- Addition of the context calculus from Lucx for operators on Lucx's context sets (union, intersection, etc.) are used to address to provide a collection of traces. Forensic Lucid inherits the properties of Lucx, Objective Lucid, JOOIP (and their comprising dialects), where the former is for the context calculus, and the latter for the arrays and structural representation of data for modeling the case data structures such as events, observations, and groupings of the related data.

Features III

- One of the basic requirements is that the complete definition of the syntax, and the operational semantics of Forensic Lucid should be compatible with the basic Lucx and GIPL, i.e. the translation rules or equivalent are to be provided when implementing the language compiler within GIPSY, and such that the GEE can execute it with minimal changes. The most difficult aspect here is, of course, the semantics of Forensic Lucid (luckily, the bulk of it is an aggregation of the semantic rules of the languages we inherit from).

Forward Tracing vs. Back-tracing I

- Naturally, the GEE makes demands in the demand-driven evaluation in the order the tree of an intentional program is traversed. Tracing of the demand requests in this case will be “forward tracing”.
- Such tracing is less useful than the mentioned back-tracing when demands are resolved, when dealing with the back-tracing in forensic investigation in an attempt to reconstruct events from the final state observations. Back-tracing is also naturally present when demands are computed and return results.

Forward Tracing vs. Back-tracing II

- The latter may not be sufficient in the forensic evaluation, so a set of reverse operators to `next`, `fby`, `asa`, etc. is needed. The development of such operators is discussed further in the syntax and semantics sections.

Context I

- We need to provide an ability to encode the stories told by the evidence and witnesses. This will constitute the context of evaluation.
- The return value of the evaluation would be a collection of backtraces, which contain the “paths of truth”.
- If a given trace contains all truths values, it’s an explanation of a story.
- If there is no such a path, i.e. the trace, there is no enough supporting evidence of the entire claim to be true.
- The context for this task for simplicity of the prototype language can be expressed as integers or strings, to which we attribute some meaning or description.

Context II

- The contexts are finite and can be navigated through in both directions of the index, potentially allowing negative tags in our tag sets of dimensions. Alternatively, our contexts can be a finite set of symbolic labels and their values that can internally be enumerated. This approach will be naturally more appropriate for humans and we have a machinery to so in Lucx's implementation in GIPSY [49, 38].

Context III

- We define streams of observations as our context, that can be a simple context or a context set. In fact, in Forensic Lucid we are defining higher-level dimensions and lower-level dimensions. The highest-level one is the *evidential statement*, which is a finite unordered set of observation sequences.
- The *observation sequence* is a finite *ordered* set of observations.
- The *observation* is an "eyewitness" of a particular property along with the duration of the observation.
- As in the FSA [12, 13], the observations are a tuples of (P, min, opt) in their generic form.

Context IV

- The observations in this form, specifically, the property P can be exploded further into Lucx's context set and further into an atomic simple context [57, 50]. Context switching between different observations is done naturally with the Lucid @ context switching operator.
- Consider some conceptual expression of a storyboard in the next slide where anything in `[ ... ]` represents a story, i.e. the context of evaluation. `foo` can be evaluated at multiple contexts (stories), producing a collection of final results (e.g. *true* or *false*) for each story as well as a collection of traces.

Intensional Storyboard Expression

```
claimA @
{
  [ final observed event, possible initial observed event ],
  [ story X ],
  [ story Y ]
}
```

- While the `[...]` notation here may be confusing with respect to `[dimension:tag]` in Lucid and more specifically in Lucx [57, 50], it is in fact a simple syntactical extension to allow higher-level groups of contexts where this syntactical sugar is later translated to the baseline context constructs.
- The tentative notation of `{[...],...,[...]}` implies a notion similar to the notion of the "context set" in [57, 50] except with the syntactical sugar mentioned earlier where we allow syntactical grouping of properties, observations, observation sequences, and evidential statements as our context sets.

- The generic observation sequence can be expanded [13] into the context stream using the *min* and *opt* values, where they will translate into index values. Thus, $obs = (A,3,0)(B,2,0)$ expands the property labels A and B into a finite stream of five indexed elements: $AAABB$. Thus, a Forensic Lucid fragment in the examples would return the third A of the $AAABB$ context stream in the observation portion of o.

```
// Give me observed property at index 2 in the observation sequence obs
o @.obs 2
where
  // Higher-level dimension in the form of (P,min,opt)
  observation o;
  // Equivalent to writing = { A, A, A, B, B };
  observation sequence obs = (A,3,0)(B,2,0);
  where
    // Properties A and B are arrays of computations
    // or any Expressions
    A = [c1,c2,c3,c4];
    B = E;
    ...
  end;
end;
```

- The property values of *A* and *B* can be anything that context calculus allows. The dimension type `observation sequence` is a finite ordered context tag set [38] that allows an integral "duration" of a given tag property.

- This may seem like we allow duplicate tag values that are unsound in the classical Lucid semantics; however, we find our way around little further in the text with the implicit index tag.
- The semantics of the arrays of computations is not a part of either GIPL or Lucx; however, the arrays are provided by JLucid and Objective Lucid. We need the notion of the arrays to evaluate multiple computations at the same context. Having an array of computations is conceptually equivalent of running an a Lucid program under the same context for each array element in a separate instance of the evaluation engine and then the results of those expressions are gathered in one ordered storage within the originating program. Arrays in Forensic Lucid are needed to represent a set of results, or *explanations* of evidential

statements, as well as denote some properties of observations. We will explore the notion of arrays in Forensic Lucid much greater detail in the near future work.

- In the FSA approach computations c_i correspond to the state q and event i that enable transition. For Forensic Lucid, we can have c_i as theoretically any Lucid expression E.

```
Observed property (context): A A A B B
        Sub-dimension index: 0 1 2 3 4

o @.obs 0 = A
o @.obs 1 = A
o @.obs 2 = A
o @.obs 3 = B
o @.obs 4 = B
```

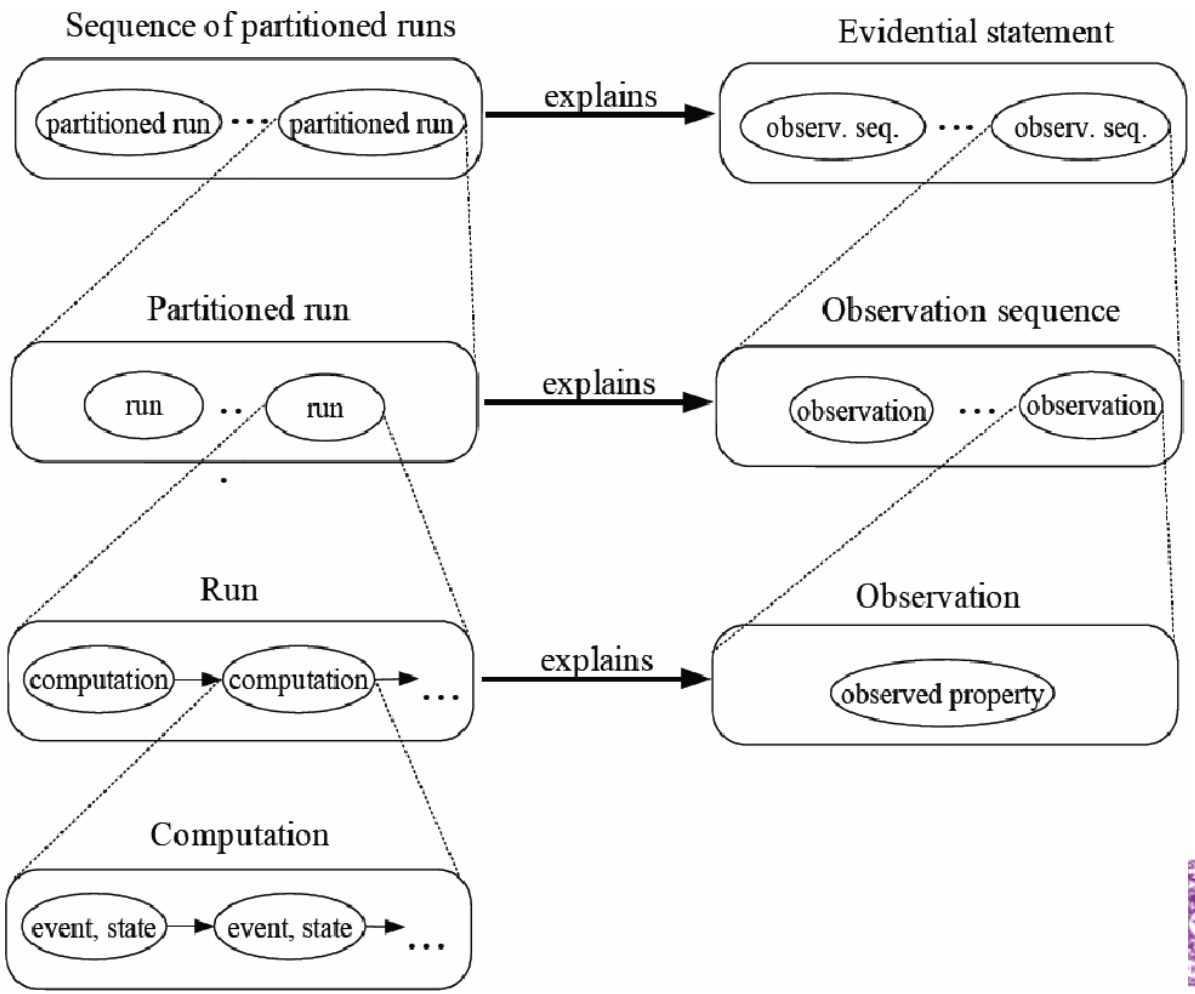
```
To get the duration/index position:

o @.obs A = 0 1 2
o @.obs B = 3 4
```

- Here we illustrate a possibility to query for the sub-dimension indices by raw property where it persists that produces a finite stream valid indices that can be used in subsequent expressions, or, alternatively by supplying the index we can get the corresponding raw property at that index. The latter feature is still under investigation of whether it is safe to expose it to Forensic Lucid programmers or make it implicit at all times at the implementation level. This is needed to remedy the

problem of “duplicate tags”: as previously mentioned, observations form the context and allow durations. This means multiple duplicate dimension tags with implied subdimension indexes should be allowed as the semantics of a traditional Lucid approaches do not allow duplicate dimension tags. It should be noted however, that the combination of the tag and its index in the stream is still unique and can be folded into the traditional Lucid semantics.

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.

Higher-Order Contexts

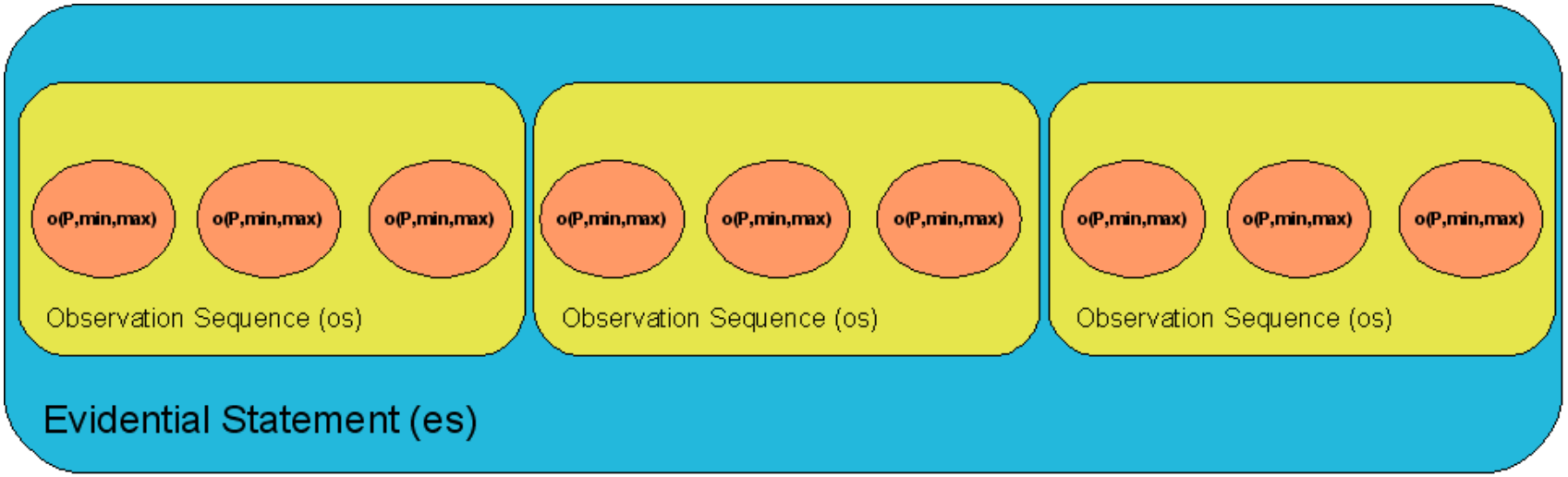


Figure: Nested Context Hierarchy Example for Digital Investigation [24, 28]

- Lucx
- Forensic Lucid
- MARFL
- iHTML

Concrete Forensic Lucid Syntax I

- The concrete syntax of the Forensic Lucid language is presented in the slides that follow. It is influenced by the productions from Lucx [58, 57], JLucid and Objective Lucid [22, 14, 21], and Indexical Lucid [36]. Some of the syntactical definitions can be, perhaps, implemented as a collection of macros.
- The `evidential statement`, `observation sequence`, and `observation` dimension types can be translated into `dimension` by some translation rules flattening them into simple contexts and context sets.

Concrete Forensic Lucid Syntax II

- The GIPSY compiler framework (GIPC) allows for the introduction of such semantic translation rules to define new language variants. We will use this feature as much as possible, though some of our syntactic constructs may have some underlying semantic details that cannot be translated into generic Lucid primitives, in which case we need to expand the existing semantics.

```
(01)    E  ::=   id
(02)         |   E(E,...,E)                          #LUCX
(03)         |   E[E,...,E](E,...,E)                 #GIPL
(04)         |   if E then E else E fi
(05)         |   # E
(06)         |   E @ E  E                            #GIPL
(07)         |   E @ E                               #LUCX
(08)         |   E where Q end;
(09)         |   [E:E,...,E:E]                       #LUCX
(10)         |   E bin-op E                          #INDEXICAL
(11)         |   un-op E                             #INDEXICAL
(12)         |   E i-bin-op E                        #INDEXICAL
(13)         |   i-un-op E                           #INDEXICAL
(14)         |   bounds
(15)         |   embed(URI, METHOD, E, E, ...)       #JLUCID
(16)         |   E[E,...,E]                          #JLUCID
(17)         |   [E,...,E]                           #JLUCID
(18)         |   E.id                                #OBJECTIVE
(19)         |   E.id(E,...,E)                       #OBJECTIVE

(20)    Q  ::=   dimension id,...,id;
(21)         |   evidential statement id,...,id [ = ES ];
(22)         |   observation sequence id,...,id [ = OS ];
(23)         |   observation id,...,id [ = O ];
(24)         |   id = E;
(25)         |   id(id,....,id) = E;                 #LUCX
(26)         |   id[id,...,id](id,....,id) = E;      #GIPL
(27)         |   E.id = E;                           #OBJECTIVE
```

```
(28)                    |    id.id,...,id(id,...,id) = E;    #OBJECTIVE
(29)                    |    QQ

(30)                ES ::=   { OS,...,OS } # evidential statement
(31)                OS ::=   { O,...,O }   # observation sequence
(32)                 O ::=   ( E, E, E )   # (property, min, opt)
                        |    $             # no-observation (Ct, 0, infinitum)
                        |    \0( E )       # zero-observation (P, 0, 0), where P = E

(33)            bin-op ::= arith-op | logical-op | bitwise-op
(34)             un-op ::= + | -

(35)          arith-op ::=  +  | - | *  | /  | %  | ^
(36)         logical-op ::=  <  | > | >= | <= | == | in | && | "||" | !
(37)         bitwise-op ::= "|" | & | ~  | !| | !&

(38)          i-bin-op ::= @ | i-bin-op-forw | i-bin-op-back | i-logic-bitwise-op | i-forensic-op

(39)      i-bin-op-forw ::= fby   |  upon  | asa   | wvr
                          |  nfby  | nupon | nasa  | nwvr

(40)      i-bin-op-back ::= pby   | rupon  | ala   | rwvr
                          |  npby  | nrupon | nala  | nrwvr

(41) i-logic-bitwise-op ::= and   | or     | xor
                          |  nand  | nor    | nxor
                          |  band  | bor    | bxor

(42)           i-un-op ::= i-bin-un-forw | i-bin-un-back | #
```

```
(43)      i-bin-un-forw ::= first   | next   | iseod
                          |  second | nnext | neg | not

(44)      i-bin-un-back ::= last    | prev   | isbod
                          |  prelast | nprev

(45)      i-forensic-op ::= combine | product | psi | invpsi

(46)             bounds ::= eod | bod | +inf | -inf
```

清華大學
Tsinghua University

```
/**
 * 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 1: The `combine` Operator

Tsinghua University

```
/**
 * 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 2: The `product` Operator

stream/index	-1	0	1	2	3	4	5	6	7	8	9	10	11
X	bod	1	2	3	4	5	6	7	8	9	10	eod	eod
Y	bod	T	F	F	T	F	F	T	T	F	T	eod	eod
X FIRST Y		1	1	1	1	1	1	1	1	1	1		
X LAST Y		10	10	10	10	10	10	10	10	10	10		
X NEXT Y			2	3	4	5	6	7	8	9	10	eod	eod
X PREV Y		bod											
X FBY Y		1	T	F	F	T	F	F	T	T	F	T	eod
X PBY Y		T	F	F	T	F	F	T	T	F	T	1	eod
X WVR Y		1			4			7	8		10		
X RWVR Y		10			8			7	4		1		
X NWVR Y			2	3		5	6			9			
X NRWVR Y			9	6		5	3			2			
X ASA Y		1	1	1	1	1	1	1	1	1	1		
X NASA Y		2	2	2	2	2	2	2	2	2	2		
X ALA Y		10	10	10	10	10	10	10	10	10	10		
X NALA Y		9	9	9	9	9	9	9	9	9	9		
X UPON Y		1	2	2	2	3	3	3	4	5	5	eod	
X RUPON Y		10	9	9	8	7	7	7	6	6	6	bod	
X NUPON Y		1	1	2	3	3	4	5	5	5	6	6	eod
X NRUPON Y		10	10	9	9	9	8	7	7	6	5	5	bod
NEG X		-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	eod	eod
NOT Y		F	T	T	F	T	T	F	F	T	F	eod	eod
X AND Y		1	0	0	1	0	0	1	1	0	1	eod	eod
X OR Y		1	2	3	5	5	6	7	9	9	11	eod	eod
X XOR Y		0	2	3	5	5	6	6	9	9	11	eod	eod

Table: Example of Application of Forensic Lucid Operators to

Transition Function I

- A transition function determines how the context of evaluation changes during computation. A general issue exists that we have to address is that the transition function ψ is problem-specific. In the FSA approach, the transition function is the labeled graph itself. In the first prototype, we follow the graph to model our Forensic Lucid equivalent.
- In general, Lucid has already basic operators to navigate and switch from one context to another, which represent the basic transition functions in themselves (the intensional operators such as @, #, `iseod`, `first`, `next`, `fby`, `wvr`, `upon`, and `asa` as well as their inverse operators). However, a specific problem being modeled requires more specific

Transition Function II

transition function than just plain intensional operators. In this case the transition function is a Forensic Lucid function where the matching state transition modeled through a sequence of intensional operators.

- A question arises a of how to explicitly model the transition function ψ and its backtrace Ψ^{-1} in the new language. A possible approach is to use predefined macros in Lucid syntax [27].

Transition Function III

- In fact, the forensic operators are just pre-defined functions that rely on traditional and inverse Lucid operators as well as context switching operators that achieve something similar to the transitions. Once modeled, it would be the GEE actually execution ψ within GIPSY. In fact, the intensional operators of Lucid represent the basic building blocks for ψ and Ψ^{-1}. We provide a first implementation of Ψ^{-1} in [28].

Operational Semantics I

- As previously mentioned, the operational semantics of Forensic Lucid for the large part is viewed as a composition of the semantic rules of Indexical Lucid, Objective Lucid, and Lucx along with the new operators and definitions. Here we list the existing combined semantic definitions to be used the new language, specifically extracts of operational semantics from GIPL [36], Objective Lucid [22], and Lucx [57] are in figures that follow. The explanation of the rules and the notation are given in great detail in the cited works and are trimmed in this article. For convenience of the reader they are recited here.

清華大學
Tsinghua University

Operational Semantics II

- The Objective Lucid semantic rules were affected and refined by some of the semantic rules of JOOIP [59].
- The new rules of the operational semantics of Forensic Lucid cover the operators primarily, including the reverse and logical stream operators as well as forensic-specific operators.
- We use the same notation as the referenced languages to maintain consistency in defining our rules.

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

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

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

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

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

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

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

$$\mathbf{E_{cT}} : \frac{\mathscr{D}, \mathscr{P} \vdash E : true \qquad \mathscr{D}, \mathscr{P} \vdash E' : v'}{\mathscr{D}, \mathscr{P} \vdash \mathtt{if}\ E\ \mathtt{then}\ E'\ \mathtt{else}\ E'' : v'}$$

$$\mathbf{E_{cF}} : \frac{\mathscr{D}, \mathscr{P} \vdash E : false \qquad \mathscr{D}, \mathscr{P} \vdash E'' : v''}{\mathscr{D}, \mathscr{P} \vdash \mathtt{if}\ E\ \mathtt{then}\ E'\ \mathtt{else}\ E'' : v''}$$

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

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

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

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

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

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

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

$$\mathbf{E_{c-vid}}: \frac{\mathscr{D},\mathscr{P}\vdash E : id \quad \mathscr{D},\mathscr{P}\vdash E' : id' \quad \mathscr{D}(id) = (\texttt{class, cid, }\underline{\texttt{cdef}}) \quad \mathscr{D}(id') = (\texttt{classv, cid.cvid, }\underline{\texttt{vdef}}) \quad \mathscr{D},\mathscr{P}\vdash\texttt{<cid.cvid>}: v}{\mathscr{D},\mathscr{P}\vdash E.E' : v}$$

$$\mathbf{E_{c-fct}}: \frac{\mathscr{D},\mathscr{P}\vdash E : id \quad \mathscr{D},\mathscr{P}\vdash E' : id' \quad \mathscr{D},\mathscr{P}\vdash E_1,\ldots,E_n : v_1,\ldots,v_n \quad \mathscr{D}(id) = (\texttt{class, cid, }\underline{\texttt{cdef}}) \quad \mathscr{D}(id') = (\texttt{classf, cid.cfid, }\underline{\texttt{fdef}}) \quad \mathscr{D},\mathscr{P}\vdash\texttt{<cid.cfid}(v_1,\ldots,v_n)\texttt{>}: v}{\mathscr{D},\mathscr{P}\vdash E.E'(E_1,\ldots,E_n) : v}$$

$$\mathbf{E_{ffid}}: \frac{\mathscr{D},\mathscr{P}\vdash E : id \quad \mathscr{D},\mathscr{P}\vdash E_1,\ldots,E_n : v_1,\ldots,v_n \quad \mathscr{D}(id) = (\texttt{freefun, ffid, }\underline{\texttt{ffdef}}) \quad \mathscr{D},\mathscr{P}\vdash\texttt{<ffid}(v_1,\ldots,v_n)\texttt{>}: v}{\mathscr{D},\mathscr{P}\vdash E(E_1,\ldots,E_n) : v}$$

$$\#\textbf{JAVA}_{\textbf{objid}} : \frac{\underline{\texttt{cdef}} = \texttt{Class cid}\,\{\ldots\}}{\mathscr{D}, \mathscr{P} \vdash \underline{\texttt{cdef}} : \mathscr{D}\dagger[\texttt{cid} \mapsto (\texttt{class}, \texttt{cid}, \underline{\texttt{cdef}})], \mathscr{P}}$$

$$\#\textbf{JAVA}_{\textbf{objvid}} : \frac{\underline{\texttt{cdef}} = \texttt{Class cid}\,\{\ldots\underline{\texttt{vdef}}\ldots\} \qquad \underline{\texttt{vdef}} = \texttt{public}\ \textit{type}\ \texttt{vid;}}{\mathscr{D}, \mathscr{P} \vdash \underline{\texttt{cdef}} : \mathscr{D}\dagger[\texttt{cid.vid} \mapsto (\texttt{classv}, \texttt{cid.vid}, \underline{\texttt{vdef}})], \mathscr{P}}$$

$$\#\textbf{JAVA}_{\textbf{objfid}} : \frac{\underline{\texttt{cdef}} = \texttt{Class cid}\,\{\ldots\underline{\texttt{fdef}}\ldots\} \qquad \underline{\texttt{fdef}} = \texttt{public}\ \textit{frttype}\ \ \texttt{fid}(\textit{fargtype}_1\ \textit{farg}}{\mathscr{D}, \mathscr{P} \vdash \underline{\texttt{cdef}} : \mathscr{D}\dagger[\texttt{cid.fid} \mapsto (\texttt{classf, cid.fid, } \underline{\texttt{fdef}}}$$

$$\#\textbf{JAVA}_{\textbf{ffid}} : \frac{\underline{\texttt{ffdef}} = \textit{frttype}\ \ \texttt{ffid}(\textit{fargtype}_1\ \textit{farg}_{id_1}, \ldots, \textit{fargtype}_n\ \textit{farg}_{id_n})}{\mathscr{D}, \mathscr{P} \vdash \underline{\texttt{ffdef}} : \mathscr{D}\dagger[\texttt{ffid} \mapsto (\texttt{freefun, ffid, } \underline{\texttt{ffdef}})], \mathscr{P}}$$

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

清華大學
Tsinghua University

$$\mathbf{E}_{\#(\mathbf{cxt})} : \frac{}{\mathscr{D}, \mathscr{P} \vdash \# : \mathscr{P}}$$

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

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

$$\mathbf{E}_{.} : \frac{\mathscr{D}, \mathscr{P} \vdash E_2 : id_2 \qquad \mathscr{D}(id_2) = (\mathtt{dim})}{\mathscr{D}, \mathscr{P} \vdash E_1.E_2 : tag(E_1 \downarrow \{id_2\})}$$

$$\mathbf{E}_{\mathbf{tuple}} : \frac{\mathscr{D}, \mathscr{P} \vdash E : id \qquad \mathscr{D} \dagger [id \mapsto (\mathtt{dim})] \qquad \mathscr{P} \dagger [id \mapsto 0] \qquad \mathscr{D}, \mathscr{P} \vdash E_i : v_i}{\mathscr{D}, \mathscr{P} \vdash \langle E_1, E_2, \ldots, E_n \rangle E : v_1 \;\; fby.id \;\; v_2 \;\; fby.id \;\; \ldots \;\; v_n \;\; fby.id \;\; \mathtt{eod}}$$

$$\mathbf{E}_{\mathbf{select}} : \frac{E = [\mathtt{d} : \mathtt{v'}] \qquad E' = \langle \mathtt{E}_1, \ldots, \mathtt{E}_n \rangle \mathtt{d} \mathscr{P}' = \mathscr{P} \dagger [d \mapsto v'] \qquad \mathscr{D}, \mathscr{P}' \vdash E' : v}{\mathscr{D}, \mathscr{P} \vdash select(E, E') : v}$$

$$\mathbf{E}_{\mathbf{at}(\mathbf{s})} : \frac{\mathscr{D}, \mathscr{P} \vdash \mathscr{C} : \{\mathscr{P}_1, \ldots, \mathscr{P}_2\} \qquad \mathscr{D}, \mathscr{P}_{i:1 \ldots m} \vdash E : v_i}{\mathscr{D}, \mathscr{P} \vdash E \mathbin{@} C : \{v_1, \ldots, v_m\}}$$

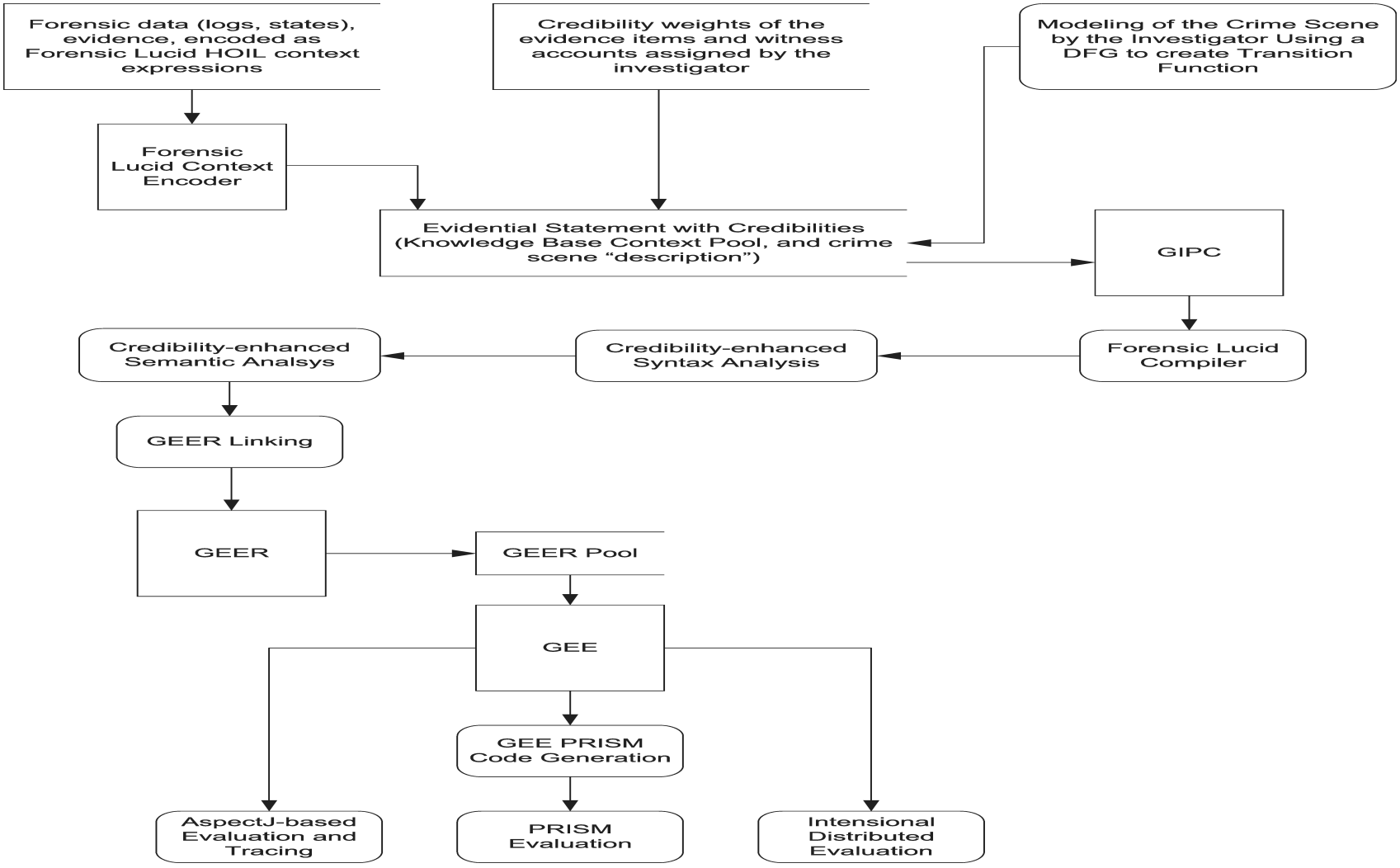
$$\mathbf{C_{box}} : \frac{\begin{array}{l}\mathscr{D},\mathscr{P} \vdash E_{d_i} : id_i \qquad \mathscr{D}(id_i) = (\mathtt{dim}) \\ \{E_1,\ldots,E_n\} = dim(\mathscr{P}_1) = \ldots = dim(\mathscr{P}_m) \\ E' = \mathtt{f}_p(\mathtt{tag}(\mathscr{P}_1),\ldots,\mathtt{tag}(\mathscr{P}_m)) \qquad \mathscr{D},\mathscr{P} \vdash E' : true\end{array}}{\mathscr{D},\mathscr{P} \vdash Box[E_1,\ldots,E_n|E'] : \{\mathscr{P}_1,\ldots,\mathscr{P}_m\}}$$

$$\mathbf{C_{set}} : \frac{\mathscr{D},\mathscr{P} \vdash E_{w:1\ldots m} : \mathscr{P}_m}{\mathscr{D},\mathscr{P} \vdash \{E_1,\ldots,E_m\} : \{\mathscr{P}_1,\ldots,\mathscr{P}_w\}}$$

$$\mathbf{C_{op}} : \frac{\mathscr{D},\mathscr{P} \vdash E : id \qquad \mathscr{D}(id) = (\mathtt{cop}, f) \qquad \mathscr{D},\mathscr{P} \vdash C_i : v_i}{\mathscr{D},\mathscr{P} \vdash E(C_1,\ldots,C_n) : f(v_1,\ldots,v_n)}$$

$$\mathbf{C_{sop}} : \frac{\mathscr{D},\mathscr{P} \vdash E : id \qquad \mathscr{D}(id) = (\mathtt{sop}, f) \qquad \mathscr{D},\mathscr{P} \vdash C_i : \{v_{i_1},\ldots,v_{i_k}\}}{\mathscr{D},\mathscr{P} \vdash E(C_1,\ldots,C_n) : f(\{v_{1_1},\ldots,v_{1_s}\},\ldots,\{v_{n_1},\ldots,v_{n_m}\})}$$

Forensic Lucid Compilation and Evaluation Flow in GIPSY



Part III

Sample Cases

Part III Outline

清華大學 Tsinghua University

ACME Manufacturing Printing Case I

This is one of the cases we re-examine from the Gladyshev's FSA approach [13].

- *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

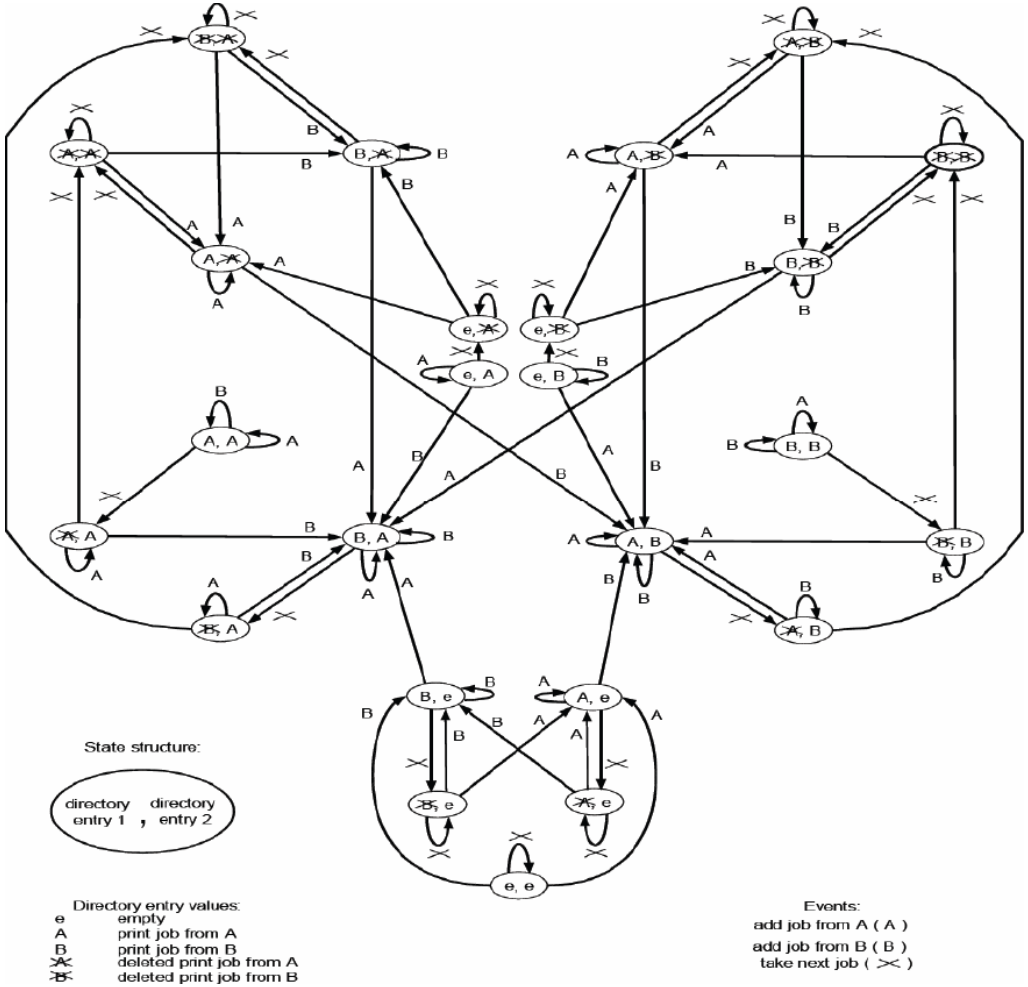
1. When a print job is received from the user, it is stored in the first unallocated directory entry of the print job directory.
2. The printing mechanism scans the print job directory from the beginning and picks the first active job.
3. After the job is printed, the corresponding directory entry is 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

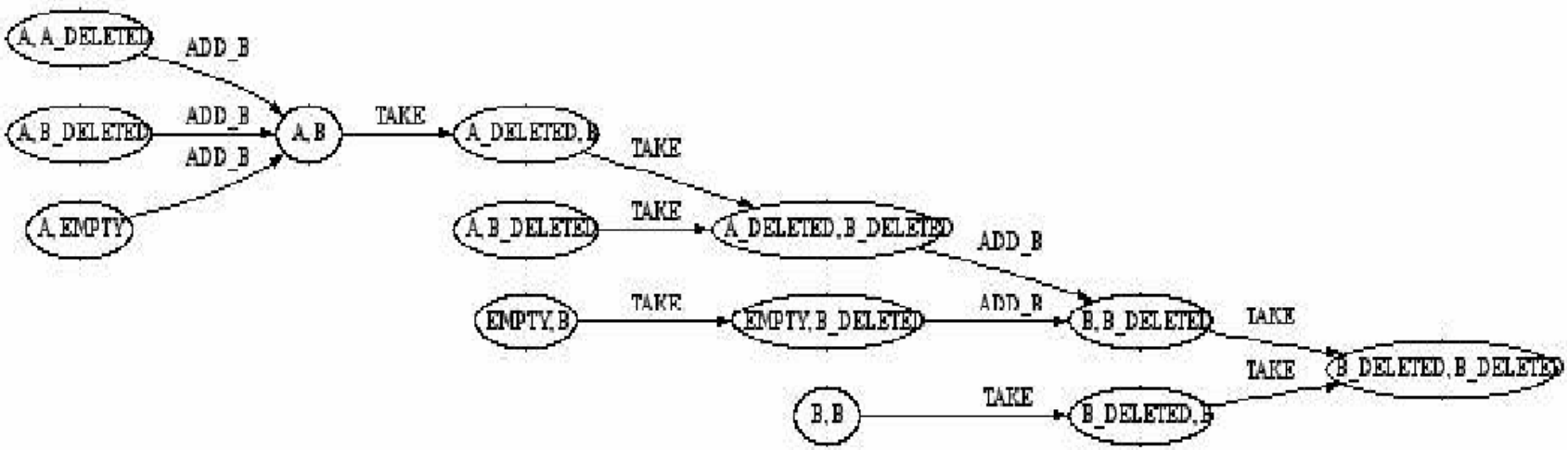
The investigator finds the current state of the printer's buffer as:

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 3: The Pinter Case “main()”

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

清華大學 Tsinghua University

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

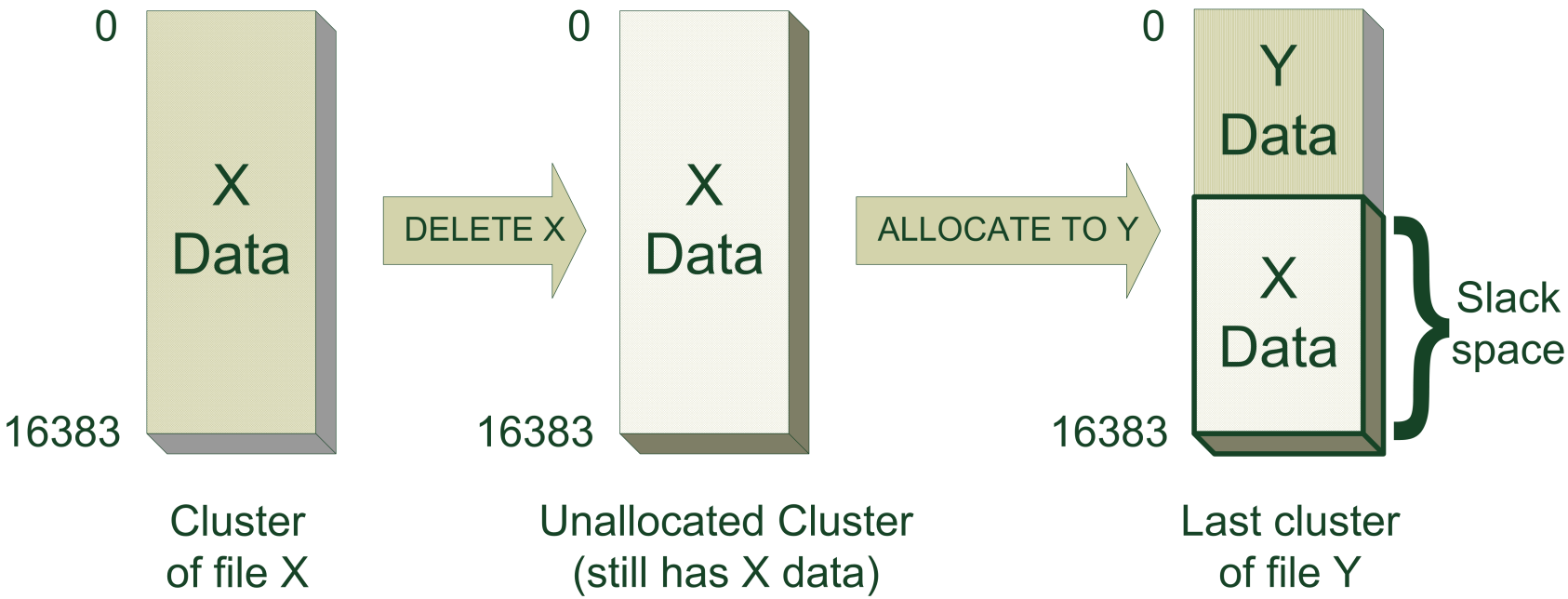


Figure: Cluster Data with the Blackmail Fragments

The case description in this section is from [12]. *A managing director of some company, Mr. C, was blackmailed. He contacted the police and handed them evidence in the form of a floppy disk that contained a letter with a number of allegations, threats, and demands. The message was known to have come from his friend Mr. A. The police officers went to interview Mr. A and found that he was on holiday abroad. They seized the computer of Mr. A and interviewed him as soon as he returned into the country. Mr. A admitted that he wrote the letter, but denied making threats and demands. He explained that, while he was on holiday, Mr. C had access to his computer. Thus, it was possible that Mr. C added the threats and demands into the letter himself to discredit Mr. A. One of the blackmail fragments was found in the slack space of another letter unconnected with the incident. When the police interviewed the*

person to whom that letter was addressed, he confirmed that he had received the letter on the day that Mr. A had gone abroad on holiday. It was concluded that Mr. A must have added the threats and demands into the letter before going on holiday, and that Mr. C could not have been involved. [12] In Figure 5 is the initial view of the incident as a diagram illustrating cluster data of the blackmail and unconnected letters [12].

Simplified model of the cluster:

possible data lengths:

0	1	2
P_L – left part	P_R – right part	

possible data values:

u – unrelated

t_1 – threats-obscured part	t_2 – threats in slack
o_1 – other data left part	o_2 – other data right part

Observed final state:

$L = 1$

$P_L = \{u, t_1, o_1\}$

$P_R = \{t_2, o_2\}$

$Q = L \times P_L \times P_R$

(u) unrelated	(t_2) threats in slack

Figure: Simplified View of the Cluster Model [12]

清華大學 Tsinghua University

In the blackmail example, the functionality of the last cluster of a file was used to determine the sequence of events and, hence, to disprove Mr. A's alibi [12]. Thus, the scope of the model is restricted to the functionality of the last cluster in the unrelated file. The last cluster model can store data objects of only three possible lengths: $L = \{0, 1, 2\}$. Zero length means that the cluster is unallocated. The length of 1 means that the cluster contains the object of the size of the unrelated letter tip. The length of 2 means that the cluster contains the object of the size of the data block with the threats. In Figure 6 is, therefore, the simplified model of the investigation [12].
The state of the last cluster can be changed by three types of events [12]:

1. Ordinary writes into the cluster:

$$W = \{(u), (t_1), (o_1), (u, t_2), (u, o_2), (t_1, t_2), (t_1, o_2), (o_1, t_2), (o_1, o_2)\}$$

2. Direct writes into the file to which the cluster is allocated (bypassing the OS):

$$W_d = \{d(u, t_2), d(u, o_2), d(o_1), d(t_1, t_2), d(t_1, o_2), d(o_1, t_2), d(o_1, o_2)\}$$

3. Deletion of the file D sets the length of the file to zero. Therefore, all writes and the deletion comprise I:

$$I = W \bigcup W_d \bigcup D$$

The final state observed by the investigators is $(1, u, t_2)$ [12]. Let O_{final} denote the observation of this state. The entire final sequence of observations is then $os_{final} = (\$, O_{final})$ [12]. The observation sequence $os_{unrelated}$ specifies that the unrelated letter was created at some time in the past, and that it was received by the person to whom it was addressed is $os_{unrelated} = (\$, O_{unrelated}, \$, (C_T, 0, 0), \$)$ where $O_{unrelated}$ denotes the observation that the "unrelated" letter tip (u) is being written into the cluster [12]. The evidential statement is then the composition of the two stories $es_{blackmail} = \{os_{final}, os_{unrelated}\}$ [12].

Mr. A's theory, encoded using the proposed notation, is $os_{Mr.A} = (\$, O_{unrelated-clean}, \$, O_{blackmail}, \$)$, where $O_{unrelated-clean}$ denotes the observation that the "unrelated" letter (u) is being written into the cluster and, at the same time, the cluster does

not contain the blackmail fragment; $O_{blackmail}$ denotes the observation that the right part of the model now contains the blackmail fragment (t_2) [12].
There are two most logically possible explanations that can be represented by a state machine [12]. See the corresponding state diagram for the blackmail case in Figure 7 [12].

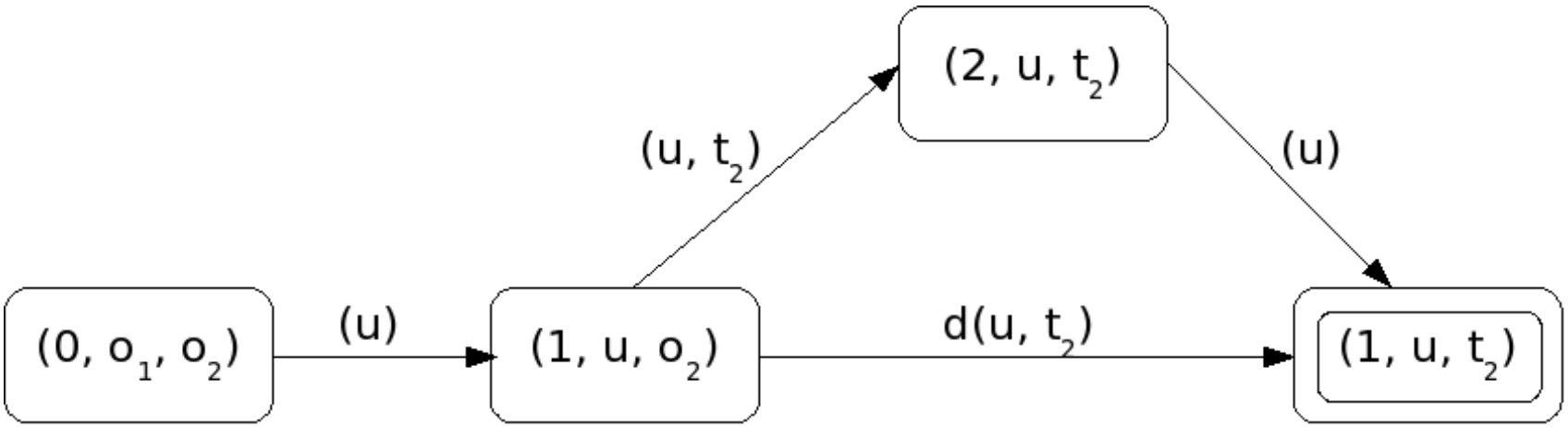


Figure: Blackmail Case State Machine

1. The first explanation [12]:
$$\ldots \xrightarrow{(u)} (1, u, o_2) \xrightarrow{(u,t_2)} (2, u, t_2) \xrightarrow{(u)} (1, u, t_2)$$
   - Finding the unrelated letter, which was written by Mr. A earlier;
   - Adding threats into the last cluster of that letter by editing it "in-place" with a suitable text editor (such as ViM [33]);
   - Restoring the unrelated letter to its original content by editing it "in-place" again [12].

> *"To understand this sequence of events, observe that certain text editors (e.g. ViM [33]) can be configured to edit text "in-place". In this mode of operation, the modified file is written back into the same disk blocks that were allocated to the original file. As a result, the user can forge the file's slack space by (1) appending the desired slack space content to the end of the file, (2) saving it, (3) reverting the file back to the original content, (4) saving it again." [12]*

2. The second explanation [12]:

$$\ldots \xrightarrow{(u)} (1, u, o_2) \xrightarrow{d(u,t_2)} (1, u, t_2)$$

   - The threats are added into the slack space of the unrelated letter by writing directly into the last cluster using, for example, a low-level disk editor [12].

清華大學 Tsinghua University

The blackmail case example of the initial implementation steps modeled in Forensic Lucid is in Listing 4. At the top of the example we construct the hierarchical context representing the evidential statement and comprising observations. The syntax is made to relate to the mathematical description of Gladyshev's FSA, but with the semantics that of Lucid. Any event property can also be mapped to a human-readable description that can be printed out in a trace. `invtans` corresponds to Ψ^{-1}; given all states, the evidential statement, and Mr. A's claim as an argument it attempts to find possible backtrace explanations within the cluster model. `trans` is ψ.

```
os_mra @ es_mra
where
  // Core context of evaluation
  evidential statement es_mra = {os_final, os_unrelated
      };

  // Mr. A's story
  observation sequence os_mra = ($, o_unrelated_clean, $
      , o_blackmail, $);
  // Crime scene description
  observation sequence os_final = ($, o_final);
  observation sequence os_unrelated = ($, o_unrelated, $
      , (Ct,0,0), $);
  observation o_final = (1, "u", "t2");
  observation o_unrelated_clean = (1, "u", "o1");

  // Corresponds to the state machine
  trans = ...
```

清華大學
Tsinghua University

```
// Event reconstruction
invtrans(Q, es_mra, o_final) = backtraces
where
  // List of all possible dimension tags
  observation Q = lengths box left_part box right_part
      ;
  // Cluster events
  observation lengths = unordered {0, 1, 2};
  // Symbolic labels map to human descriptions
  observation left_part = unordered {
      "u"  => "unrelated",
      "t1" => "threats-obscured part",
      "o1" => "other data (left part)"
  };
  observation right_part = unordered {
      "t2" => "threats in slack",
      "o2" => "other data (right part)"
  };
  backtraces = [ A, B ] @ t 5;
```

```
    where
      dimension t;
      A = o_final pby.t ``u'' pby.t (2, ``u'', ``t2'')
          pby.t (``u'', ``t2'') pby.t (1, ``u'', ``o2'')
           pby.t (0, ``o1'', ``o2'');
      B = o_final pby.t d(``u'', ``t2'') pby.t (1, ``u''
          , ``o2'') pby.t ``u'' pby.t (0, ``o1'', ``o2'')
          ;
    end;
  end;
end;
```

Listing 4: Blackmail Case Modeling in Forensic Lucid

Part IV

Concluding Remarks

清華大學
Tsinghua University

Part IV Outline

Part IV Outline

Concluding Remarks 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.

Concluding Remarks II

- We re-wrote in Forensic Lucid two of the sample cases initially 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 (when printed).
- 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.

Concluding Remarks 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 [40]), 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.

Ongoing Work I

- Formally prove equivalence to the FSA approach.
- Adapt/re-implement a graphical UI based on the data-flow graph tool [8, 31] 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.

Ongoing Work II

- Explore and exploit the notion of credibility factors of the evidence and witnesses fully in GEE.
- Include the ability of having multiple GEE engine evaluation backend plug-ins, one of which would rely on PRISM [48], and the other one on AspectJ [6];
- Release a full standard Forensic Lucid specification.
- Work with the real-world data.

Computing Credibility Weights I

- This sub-project refines the theoretical structure and formal model of the observation tuple with the credibility weight and other factors for cyberforensic analysis and event reconstruction. An earlier work suggested a mathematical theory of evidence by Dempster, Shafer and others [15, 41], where factors like credibility, trustworthiness, and the like play a role in the evaluation of mathematical expressions, which Gladyshev lacked.

Computing Credibility Weights II

- Thus, 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.

Computing Credibility Weights III

- 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.

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

Computing Credibility Weights IV

- 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.

Proposed Visualization I

- 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 as was proposed in [22].

Proposed Visualization II

- Related work by several researchers on visualization of load balancing, configuration, formal systems for diagrammatic modeling and visual languages and the corresponding graph systems are presented in [61, 55, 1, 7, 20]. They all define some key concepts that are relevant to our visualization mechanisms within GIPSY and its corresponding General Manager Tier (GMT) [18].
- We propose to build upon those works to represent the nested evidence, crime scene as a 2D or even 3D DFG, and the reconstructed events flow upon evaluation.

Proposed Visualization III

- For that related work a conceptual example of a 2D DFG corresponding to a simple Lucid program is in Figure 8. The actual current rendering of such graphs is exemplified in Figure 9 from Ding [8] in the GIPSY environment.
- These 2D conceptual visualizations are proposed to be renderable at least in 2D or in 3D via an interactive interface to allow modeling complex crime scenes and multidimensional evidence on demand. The end result could look like something expanding or “cutting out” nodes or complex-type results conceptually exemplified in Figure 10.

Canonical Example of a 2D Data Flow Graph-Based Program

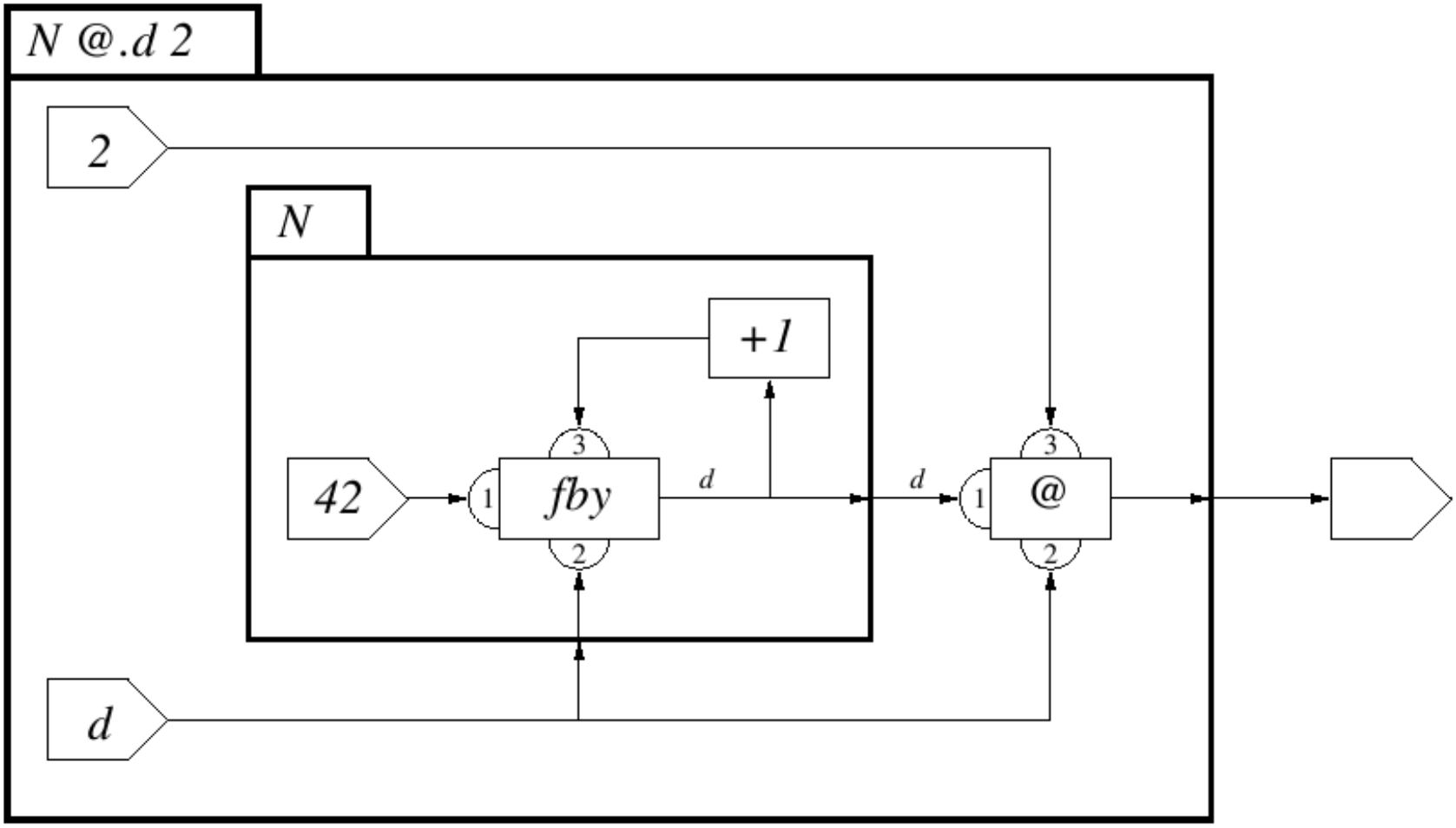


Figure: Canonical Example of a 2D Data Flow Graph-Based Program

Example of an Actual Rendered 2D Data Flow Graph-Based

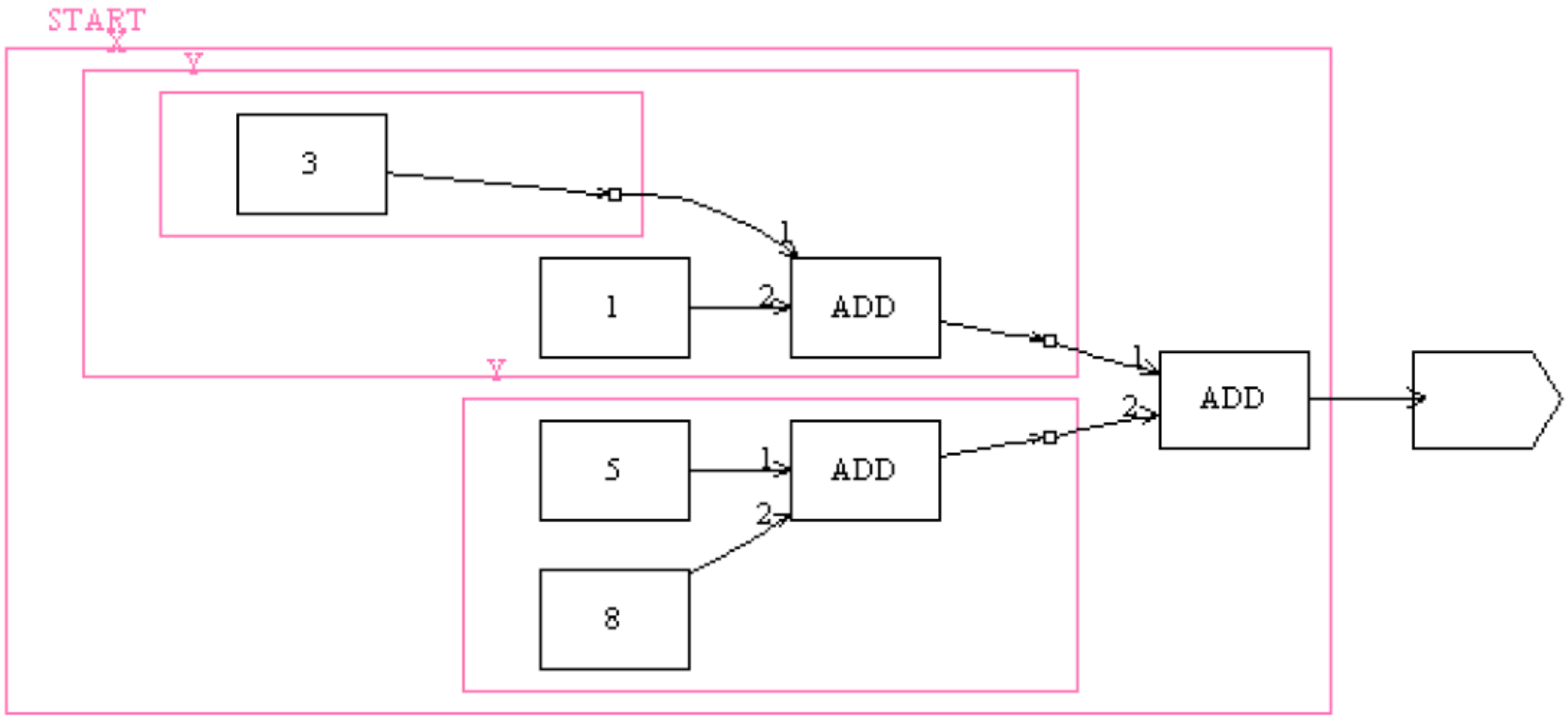


Figure: Example of an Actual Rendered 2D Data Flow Graph-Based Program with Graphviz [9]

Modified Example of a 2D Data Flow Graph-based Program

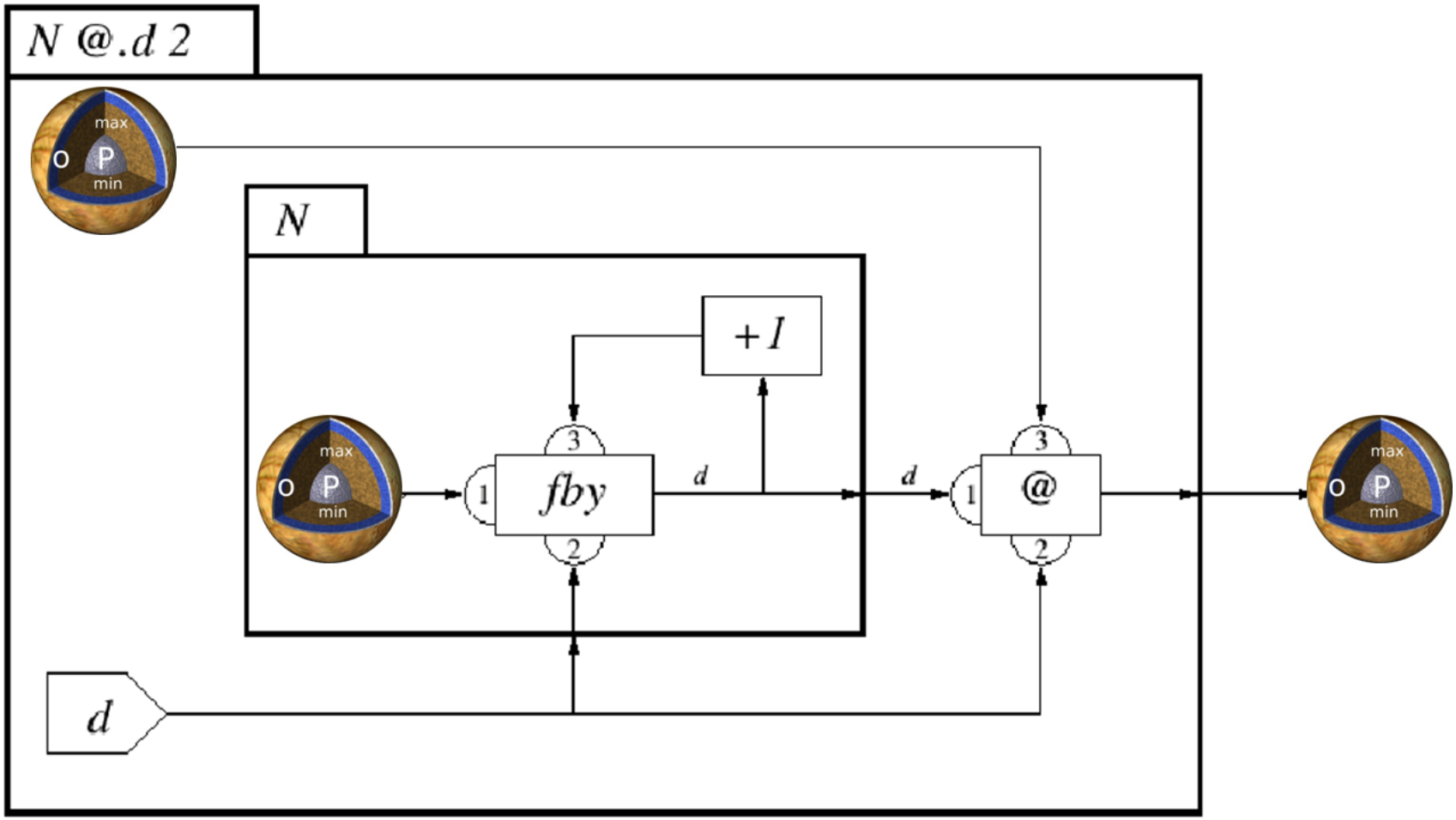


Figure: Modified Example of a 2D Data Flow Graph-based Program with 3D Elements

- In Forensic Lucid, the need is to represent visually forensic cases, evidence, and other specification components is obvious for usability and other issues.
- Placing it in 3D helps to structure the "program" (specification) and the case in 3D space can help arrange and structure the case in a virtual environment better with the evidence items encapsulated in 3D spheres akin to Russian dolls, and can be navigated in depth to any level of detail e.g. via clicking.
- The depth and complexity of operational semantics and demand-driven (eductive) execution model are better represented and comprehended visually in 3D especially when doing event reconstruction.

- Ding's implementation allows navigation from a graph to a graph by expanding more complex nodes to their definitions, e.g. more elaborate operators such *whenever* (`wvr`) or *advances upon* (`upon`), their reverse operators, forensic operators, and others.
- Some immediate requirements to realize the envisioned DFG visualization of Forensic Lucid programs and their evaluation:
  - Visualization of the hierarchical evidential statements (potentially deeply nested contexts).
  - Placement of hybrid intensional-imperative nodes into the DFGs such as mixing Java and Lucid program fragments.

- The GIPSY research and development group's previous research did not deal with aspects on how to augment the `DFGAnalyzer` and `DFGGenerator` of Ding to support hybrid GIPSY programs.
- This can be addressed by adding an "unexpandable" (one cannot click their way through its depth) 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 back we can leverage recent additions to Graphviz and GIPSY.
- The newer versions of Graphviz support additional features that are more usable for our needs at the present. Moreover, with the advent of JOOIP [60], the Java 5 ASTs are made available along with embedded Lucid fragments that can be tapped into when generating the `dot` code's AST.

- A Java-based wrapper for the DFG Editor of Ding [8] to enable its native use withing Java-based GIPSY and plug-in IDE environments like Eclipse.

- One of the goals of this work is to find the optimal technique, with soundness and completeness and formal specifications along with the ease of implementation and usability; thus we'd like to solicit opinions and insights of this work in selecting the technique or a combination of techniques, which seems a more plausible outcome. The current design allows any of the implementation to be chosen or a combination of them.

- First, the most obvious is Ding's [8] basic DFG implementation within GIPSY as it is already a part of the project and done for the two predecessor Lucid dialects GIPL and Indexical Lucid. Additionally, the modern version of Graphviz now has some integration done with Eclipse [11], so GIPSY's IDE – RIPE (Run-time Interactive Programming Environment) – may very well be the an Eclipse-based plug-in.

- Puckette came up with the PureData [39] language and its commercial offshoots, which also employ DFG-like programming with boxes and inlets and outlets of any data types graphically placed and connected as “patches” and allowing for sub-graphs and external implementations of inlets in procedural languages. Puckette’s original design was targetting signal processing for electronic music and video processing and production for interactive artistic and performative processes but has since outgrown that notion. The PureData externals allow deeper media visualizations in OpenGL, video, etc. thereby potentially enhancing the whole aspect of the process significantly.

- The BPEL (Business Process Execution Language) and its visual realization within NetBeans [42, 34] for SOA (service-orient architectures) and web services is another good model for inspiration [35, 19, 17] that has recently undergone a lot of research and development, including flows, picking structures, faults, and parallel/asynchronous and sequential activities. More importantly, BPEL notations have a backing formalism modeled upon based on Petri nets. BPEL specifications actually translate to executable Java web services code.

Acknowledgments

- Visualization and Graphics Lab, Department of Computer Science and Technology, Tsinghua University as a host.
- Dr. Yankui Sun and his students for their support.
- The audience.
- Canada-China Scholars' Exchange Program (CCSEP) scholarship.
- This research work was funded by NSERC and the Faculty of Engineering and Computer Science of Concordia University, Montreal, Canada.
- Thanks to many of the GIPSY project current and former team members for their valuable contributions, suggestions, and reviews, including Sleiman Rabah, Yi Ji, Bin Han, Aihua Wu, Emil Vassev, Xin Tong, Amir Pourteymour, and Peter Grogono.

清華大學
Tsinghua University

Questions?

Questions, suggestions, and feedback are welcome

- now,
- after the talk,
- or by email: `mokhov@cse.concordia.ca`.
- Thank you :-)

References I

[1] G. Allwein and J. Barwise, editors. *Logical reasoning with diagrams*. Oxford University Press, Inc., New York, NY, USA, 1996. ISBN 0-19-510427-7.

[2] E. A. Ashcroft and W. W. Wadge. Lucid – a formal system for writing and proving programs. *SIAM J. Comput.*, 5(3), 1976.

[3] E. A. Ashcroft and W. W. Wadge. Erratum: Lucid – a formal system for writing and proving programs. *SIAM J. Comput.*, 6(1):200, 1977.

[4] E. A. Ashcroft and W. W. Wadge. Lucid, a nonprocedural language with iteration. *Communications of the ACM*, 20(7):519–526, July 1977. ISSN 0001-0782. doi: 10.1145/359636.359715.

[5] E. A. Ashcroft, A. A. Faustini, R. Jagannathan, and W. W. Wadge. *Multidimensional Programming*. Oxford University Press, London, Feb. 1995. ISBN: 978-0195075977.

[6] AspectJ Contributors. *AspectJ: Crosscutting Objects for Better Modularity*. eclipse.org, 2007. `http://www.eclipse.org/aspectj/`.

[7] R. Bardohl, M. Minas, G. Taentzer, and A. Schürr. Application of graph transformation to visual languages. In *Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools*, volume 2, pages 105–180. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999. ISBN 981-02-4020-1.

[8] Y. 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`.

References II

[9] Y. 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, 2004.

[10] W. Du. On the relationship between AOP and intensional programming through context, July 2005. Keynote talk at the Intensional Programming Session of PLC'05.

[11] Eclipse contributors et al. Eclipse Platform. eclipse.org, 2000–2012. `http://www.eclipse.org`, last viewed April 2012.

[12] P. Gladyshev. Finite state machine analysis of a blackmail investigation. *International Journal of Digital Evidence*, 4(1), 2005.

[13] P. Gladyshev and A. Patel. Finite state machine approach to digital event reconstruction. *Digital Investigation Journal*, 2(1), 2004.

[14] P. Grogono, S. Mokhov, and J. Paquet. Towards JLucid, Lucid with embedded Java functions in the GIPSY. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005)*, pages 15–21. CSREA Press, June 2005.

[15] R. Haenni, J. Kohlas, and N. Lehmann. Probabilistic argumentation systems. Technical report, Institute of Informatics, University of Fribourg, Fribourg, Switzerland, Oct. 1999.

[16] B. Han, S. A. Mokhov, and J. Paquet. Advances in the design and implementation of a multi-tier architecture in the GIPSY environment with Java. In *Proceedings of SERA 2010*, pages 259–266. IEEE Computer Society, 2010. ISBN 978-0-7695-4075-7. doi: 10.1109/SERA.2010.40. Online at `http://arxiv.org/abs/0906.4837`.

[17] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business Process Execution Language for Web Services version 1.1. [online], IBM, Feb. 2007. `http://www.ibm.com/developerworks/library/specification/ws-bpel/`.

Tsinghua University

References III

[18] Y. Ji. Scalability evaluation of the GIPSY runtime system. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Mar. 2011.

[19] D. Koenig. Web services business process execution language (WS-BPEL 2.0): The standards landscape. Presentation, IBM Software Group, 2007.

[20] N. G. Miller. *A Diagrammatic Formal System for Euclidean Geometry*. PhD thesis, Cornell University, U.S.A, 2001.

[21] S. Mokhov and J. Paquet. Objective Lucid – first step in object-oriented intensional programming in the GIPSY. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005)*, pages 22–28. CSREA Press, June 2005.

[22] 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, Oct. 2005. ISBN 0494102934; online at `http://arxiv.org/abs/0907.2640`.

[23] S. 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. doi: 10.1109/COMPSAC.2008.206.

[24] S. A. Mokhov and J. 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, Aug. 2008. In Theorem Proving in Higher Order Logics (TPHOLs2008): Emerging Trends Proceedings.

清華大學 Tsinghua University

References IV

[25] S. A. Mokhov and J. Paquet. Using the General Intensional Programming System (GIPSY) for evaluation of higher-order intensional logic (HOIL) expressions. In *Proceedings of SERA 2010*, pages 101–109. IEEE Computer Society, May 2010. ISBN 978-0-7695-4075-7. doi: 10.1109/SERA.2010.23. Online at `http://arxiv.org/abs/0906.3911`.

[26] S. A. Mokhov and E. Vassev. Self-forensics through case studies of small to medium software systems. In *Proceedings of IMF'09*, pages 128–141. IEEE Computer Society, Sept. 2009. ISBN 978-0-7695-3807-5. doi: 10.1109/IMF.2009.19.

[27] S. A. Mokhov, J. Paquet, and M. Debbabi. Designing a language for intensional cyberforensic analysis. Unpublished, 2007.

[28] S. A. Mokhov, J. Paquet, and M. Debbabi. Formally specifying operational semantics and language constructs of Forensic Lucid. In O. Göbel, S. Frings, D. Günther, J. Nedon, and D. Schadt, editors, *Proceedings of the IT Incident Management and IT Forensics (IMF'08)*, LNI140, pages 197–216. GI, Sept. 2008. ISBN 978-3-88579-234-5. Online at `http://subs.emis.de/LNI/Proceedings/Proceedings140/gi-proc-140-014.pdf`.

[29] S. A. Mokhov, J. Paquet, and M. Debbabi. Towards automated deduction in blackmail case analysis with Forensic Lucid. In J. S. Gauthier, editor, *Proceedings of the Huntsville Simulation Conference (HSC'09)*, pages 326–333. SCS, Oct. 2009. ISBN 978-1-61738-587-2. Online at `http://arxiv.org/abs/0906.0049`.

[30] S. A. Mokhov, J. Paquet, and X. Tong. A type system for hybrid intensional-imperative programming support in GIPSY. In *Proceedings of C3S2E'09*, pages 101–107, New York, NY, USA, May 2009. ACM. ISBN 978-1-60558-401-0. doi: 10.1145/1557626.1557642.

清華大學 Tsinghua University

References V

[31] S. A. Mokhov, J. Paquet, and M. 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. ISBN 978-1-4577-0582-3. doi: 10.1109/PST.2011.5971973. Short paper; full version online at `http://arxiv.org/abs/1009.5423`.

[32] S. A. Mokhov, J. Paquet, and M. Debbabi. Reasoning about a simulated printer case investigation with Forensic Lucid. In P. Gladyshev, editor, *Proceedings of ICDF2C'11*. Springer, Oct. 2011. To appear, online at `http://arxiv.org/abs/0906.5181`.

[33] B. Moolenaar and Contributors. Vim the editor – Vi Improved. [online], 2009. `http://www.vim.org/`.

[34] NetBeans Community. NetBeans Integrated Development Environment. [online], 2004–2012. `http://www.netbeans.org`.

[35] OpenESB Contributors. BPEL service engine. [online], 2009. `https://open-esb.dev.java.net/BPELSE.html`.

[36] J. Paquet. *Scientific Intensional Programming*. PhD thesis, Department of Computer Science, Laval University, Sainte-Foy, Canada, 1999.

[37] J. Paquet. Distributed eductive execution of hybrid intensional programs. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*, pages 218–224, Seattle, Washington, USA, July 2009. IEEE Computer Society. ISBN 978-0-7695-3726-9.

[38] J. Paquet, S. A. Mokhov, and X. 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. doi: 10.1109/COMPSAC.2008.200.

References VI

[39] M. Puckette and PD Community. Pure Data. [online], 2007–2012. http://puredata.org.

[40] G. Riley. CLIPS: A tool for building expert systems. [online], 2007–2011. http://clipsrules.sourceforge.net/, last viewed May 2012.

[41] G. Shafer. *The Mathematical Theory of Evidence*. Princeton University Press, 1976.

[42] Sun Microsystems, Inc. NetBeans 6.7.1. [online], 2009–2010. http://netbeans.org/downloads/6.7.1/index.html.

[43] P. Swoboda. *A Formalisation and Implementation of Distributed Intensional Programming*. PhD thesis, The University of New South Wales, Sydney, Australia, 2004.

[44] P. Swoboda and J. Plaice. An active functional intensional database. In F. Galindo, editor, *Advances in Pervasive Computing*, pages 56–65. Springer, 2004. LNCS 3180.

[45] P. Swoboda and J. 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.

[46] P. Swoboda and W. 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.

[47] 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 April 2012.

[48] The PRISM Team. PRISM: a probabilistic model checker. [online], 2004–2012. http://www.prismmodelchecker.org/, last viewed June 2010.

清華大學 Tsinghua University

References VII

[49] X. Tong. Design and implementation of context calculus in the GIPSY. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Apr. 2008.

[50] X. Tong, J. Paquet, and S. A. Mokhov. Complete context calculus design and implementation in GIPSY. [online], 2007–2008. `http://arxiv.org/abs/1002.4392`.

[51] E. Vassev. *ASSL: Autonomic System Specification Language – A Framework for Specification and Code Generation of Autonomic Systems*. LAP Lambert Academic Publishing, Nov. 2009. ISBN: 3-838-31383-6.

[52] E. Vassev and S. A. Mokhov. An ASSL-generated architecture for autonomic systems. In *Proceedings of C3S2E'09*, pages 121–126, New York, NY, USA, May 2009. ACM. ISBN 978-1-60558-401-0. doi: 10.1145/1557626.1557645.

[53] E. Vassev and J. Paquet. Towards autonomic GIPSY. In *Proceedings of the Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems (EASE 2008)*, pages 25–34. IEEE Computer Society, 2008. ISBN 978-0-7695-3140-3. doi: 10.1109/EASe.2008.9.

[54] E. I. Vassev. *Towards a Framework for Specification and Code Generation of Autonomic Systems*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2008.

[55] 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. ISSN 1571-0661. doi: 10.1016/j.entcs.2004.02.052.

[56] W. W. Wadge and E. A. Ashcroft. *Lucid, the Dataflow Programming Language*. Academic Press, London, 1985.

[57] K. Wan. *Lucx: Lucid Enriched with Context*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2006.

Tsinghua University

References VIII

[58] K. Wan, V. Alagar, and J. 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.

[59] A. Wu, J. Paquet, and S. A. Mokhov. Object-oriented intensional programming: Intensional Java/Lucid classes. In *Proceedings of SERA 2010*, pages 158–167. IEEE Computer Society, 2010. ISBN 978-0-7695-4075-7. doi: 10.1109/SERA.2010.29. Online at: `http://arxiv.org/abs/0909.0764`.

[60] A. H. Wu. *OO-IP Hybrid Language Design and a Framework Approach to the GIPC*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2009.

[61] C. Zheng and J. R. Heath. Simulation and visualization of resource allocation, control, and load balancing procedures for a multiprocessor architecture. In *MS'06: Proceedings of the 17th IASTED international conference on Modelling and simulation*, pages 382–387, Anaheim, CA, USA, 2006. ACTA Press. ISBN 0-88986-592-2.

Tsinghua University