Short Paper: Graded Modal Types for Integrity and Confidentiality

Daniel Marshall
School of Computing
University of Kent
dm635@kent.ac.uk

Dominic Orchard
School of Computing
University of Kent
D.A.Orchard@kent.ac.uk

Abstract

Graded type systems, such as the one underlying the Granule programming language, allow various different properties of a program's behaviour to be tracked via annotating types with additional information, which we call *grades*. One example of such a property, often used as a case study in prior work on graded types, is *information flow control*, in which types are graded by a lattice of security levels allowing noninterference properties to be automatically verified and enforced. These typically focus on one particular aspect of security, however, known as *confidentiality*; public outputs are prohibited from depending on private inputs. *Integrity*, a property specifying that trusted outputs must not depend on untrusted inputs, has not been examined in this context.

This short paper aims to remedy this omission. It is well-known that confidentiality and integrity are in some sense dual properties, but simply reversing the ordering of the security lattice turns out to be unsatisfactory for the purpose of combining both kinds of property in a single system, at least in our setting. We analogize the situation to recent work on embedding both linear and uniqueness types in a graded framework, and use this framing to demonstrate that we can enforce both integrity and confidentiality alongside one another. The main idea is to add an additional flavour of modality annotated for integrity, such that the existing graded comonad for tracking confidentiality now also acts as a *relative monad* over the new modality, with rules allowing information to flow from trusted to public to private.

1 Introduction and Motivation

Information flow control aims to track the flow of information through a program when it is executed, to make sure that the program handles that information in a secure way [23]. Secure information flow (discussed in the literature since the 1970s [9, 10]) encompasses multiple aspects, with two of the most essential being *confidentiality* and *integrity*. Pfleeger's textbook *Security in Computing* [20] describes these two properties as follows. Confidentiality "ensures that assets are accessed only by authorised parties", or in other words that private information is never accessed by a program which only has public clearance. Meanwhile, integrity "means that assets can be modified only by authorised parties or only in authorised ways", such that a trusted program never depends on information from an untrusted source. The strictest desirable property in both cases is *noninterference*, which only holds if public or trusted outputs may *never* depend on private or untrusted inputs respectively [12], though this is often considered difficult to achieve in practical systems.

Much of the prior work on using graded type systems for information flow control aims to track and restrict the outputs that can be produced by a given program [1, 13, 21]. Implementations of such ideas also exist for more widely-used functional languages such as Haskell [22]. More recently, graded type systems have been designed which enforce properties based on *coeffects*, where the inputs that can be passed into a program are the focus. Such systems (with the type system underlying Granule being the one we will focus on here [19]) often make use of information flow security as a case study in how annotating types with grades can allow more properties of a program to be verified [2, 7, 11, 16]. However, these tend to only focus on confidentiality properties, which omits a host of additional properties that could be guaranteed if it were also possible to enforce integrity. Consider the following definition of a data type in Granule.[1]

```
data Patient where
  Patient
    (Int    [Private]) -- Patient ID
    (String [Private]) -- Patient name
    (Int    [Public])  -- Patient age
```

The type `a [r]` means that we have a value of type `a` wrapped inside the □ modality which must be used according to the restriction described by the grade `r`. Here, the patient's ID and name have the grade `Private`, while their age is `Public`. Now, consider a function with the type:

```
meanAge : List Patient → Int [Public]
```

that calculates the mean age of a database (here simplified to a list) of patients and returns the result at the public security level. Since ages are also `Public`, this is fine–but if we made a mistake while writing the function and used private information such as IDs instead, this would be rejected by the type checker due to the security annotations, so no information can be leaked. Similar properties also apply when storing secret data such as passwords or credit card numbers.

However, imagine a case where we are constructing a patient to add to the database, as follows:

[1] The latest release of Granule is available to download and install from https://github.com/granule-project/granule/releases.

```
addPatient : List Patient → String [Private]
           → Int [Public] → List Patient
```

Here, we have security level grades restricting who will be able to view various details once the database has been updated, but we have no way to stop some compromised code from passing in an attempt at SQL injection using a string such as `"Alice'); DROP TABLE patients;--"`, for example. If this input is treated as trustworthy, we might well encounter dramatic problems later in our program's execution.

In order to avoid this, we would want some kind of grade that carries information about the *provenance* of our data [14], so we could declare that we can only safely add patient information into our database if the string verifiably comes from a trusted location. This would also be useful if, for instance, we were encrypting private data and wanted a way to ensure that our random numbers used for encryption were reliable and had not been tainted by an untrusted source.

It is well understood that this kind of integrity property is dual in some sense to the confidentiality properties which Granule can already express [4] (though it is also known that this duality is not sufficient to cover every facet of the concept of integrity, with more complex mechanisms than a lattice model being required for some applications [8, 17]). It turns out that this duality is closely comparable to a similar duality between *linear* types (forming the basis of Granule's graded type system) and *uniqueness* types, which has been more clearly elucidated in recent work [15].

It turns out that while linear types are a restriction on what may happen in the *future* (a linear value must never be duplicated or discarded), uniqueness types are a guarantee about what has happened in the *past* (a unique value must never *have been* duplicated). We will now show how this understanding also allows us to express integrity properties in Granule, through an additional flavour of graded modality.

2 Theory and Implementation

Our approach here builds on the type system described in the original Granule paper [19], with some extra rules for the new modality carrying integrity information.

The crucial insight here is that in order to combine confidentiality (public and private) and integrity (trusted and untrusted) in a single system, we can treat "public" and "untrusted" as the *same state*, both represented by the Public grade; these both carry the same information, telling us that there is no restriction on how the data may be used in the future (it is not restricted to private usage) and we have no guarantee about how it was used in the past (it did not necessarily come from a trusted source). The Private grade behaves exactly as described in the original work [19].

We introduce a new $*$ modality (whose syntax is borrowed from the corresponding modality for uniqueness types [15], as mentioned above) to carry the Trusted grade, with the important rules for this modality's behaviour given below.

$$\frac{\Gamma \vdash t : *_{\text{Trusted}}\, A}{\Gamma \vdash \&t : \Box_{\text{Public}}\, A}\ \textsc{Return} \qquad \frac{\varnothing \vdash t : A}{\varnothing \vdash *t : *_{\text{Trusted}}\, A}\ \textsc{Nec}$$

$$\frac{\Gamma_1 \vdash t_1 : \Box_{\text{Public}}\, A \quad \Gamma_2, x : *_{\text{Trusted}}\, A \vdash t_2 : \Box_{\text{Public}}\, B}{\Gamma_1 + \Gamma_2 \vdash \text{endorse } t_1 \text{ as } x \text{ in } t_2 : \Box_{\text{Public}}\, B}\ \textsc{Bind}$$

The Return rule maps a trusted value to a public (untrusted) one, allowing the information flow for integrity to work just as we would expect. The Bind rule allows a public value to be temporarily used as trusted within the context of a larger computation; this mimics the common integrity pattern of *endorsement*, where a value is examined and then declared to be trusted to whatever extent is necessary for a particular setting [18]. The output, however, is required to be public, so that we can not leverage our temporary integrity to 'smuggle out' a trusted value outside the context of the endorsement. These rules are accompanied by a necessitation rule, allowing a value to be trusted by default if it has no dependencies.

Note that the naming and pattern of the rules suggests a structural relationship between the two modalities. The $\Box$ modality previously acted as a *graded comonad*, but now when graded by Public also acts as a *relative monad* [3] over the $*$ modality graded by Trusted, and the various required axioms for a relative monad do indeed hold here also.

The Granule implementation follows much the same pattern. Granule already possesses a *semiring graded necessity modality*, where for a preordered semiring $(\mathcal{R}, *, 1, +, 0, \sqsubseteq)$, there is a family of types $\{\Box_r A\}_{r \in \mathcal{R}}$. Granule's confidentiality tracking is based on the particular semiring $\{\text{Public}, \text{Private}\}$,[2] where $0 = \text{Private}$, $1 = \text{Public}$, $* = \vee$ and $+ = \wedge$. The implementation provides an additional $*$ modality obeying the above rules along with syntactic sugar for applying them. This allows us to rewrite the earlier example as follows:

```
addPatient : List Patient → String *Trusted
           → Int [Public] → List Patient
```

Now, only a string from a trusted source will be accepted as the name of a patient added to our database. Note that we do not need to change the definition of our data type, since the string can be converted to Private through a combination of the Return rule and the *approximation* rule that already exists in Granule's type system, which in this case allows Public information to be treated as Private since $\text{Private} \sqsubseteq \text{Public}$ is given by the ordering of the semiring.

3 Conclusion and Future Work

We have shown that simple integrity properties can be expressed in Granule alongside confidentiality, leveraging a similar duality to one that exists between linearity and uniqueness. Directions for further research could include:

[2] Note that in the current implementation the grades in this particular semiring are in fact given the names Lo and Hi, but throughout this paper we consistently use Public and Private for clarity.

- Extending Granule's capacity to track confidentiality and integrity further, going beyond lattices containing only two security levels in order to enforce more complex and fine-grained properties.
- Considering additional aspects of information flow security such as *availability*, where information which should always be available cannot depend on information that may be unavailable; the direction of information flow here is the same as for integrity, so this should be possible using the new flavour of modality.
- Borrowing the ideas currently being developed for extending Granule's uniqueness types to a more complex *ownership* model via fractional grading on the $*$ modality [5], which here may allow us to express integrity properties relating to *separation of duties* [14]; this would be a starting point for capturing further aspects of integrity that go beyond what can be described by the current lattice-based model [8].
- It would be interesting to explore how this idea connects to recent work on bridging the gap between monadic and comonadic approaches to information flow analyses [6]. Note that Granule's current confidentiality tracking is primarily comonadic in nature, but also incorporates a touch of monadic flavour through use of a `flatten` operation, which allows transformations such as $\Box_{\text{Public}}(\Box_{\text{Private}}\ A) \rightarrow \Box_{\text{Private}}\ A$; this operation is derivable from Granule's rules for pattern matching on nested modalities [19].

This paper also forms part of a larger body of work that involves uncovering a general algebraic structure underlying *guarantees* about a program which are best represented using a graded comonad that also acts as a relative monad over some functor. This work is itself ongoing.

References

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. 1999. A Core Calculus of Dependency. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Antonio, Texas, USA) *(POPL '99)*. Association for Computing Machinery, New York, NY, USA, 147–160. https://doi.org/10.1145/292540.292555

[2] Andreas Abel and Jean-Philippe Bernardy. 2020. A Unified View of Modalities in Type Systems. *Proc. ACM Program. Lang.* 4, ICFP, Article 90 (Aug 2020), 28 pages. https://doi.org/10.1145/3408972

[3] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. 2010. Monads Need Not Be Endofunctors. In *Foundations of Software Science and Computational Structures*, Luke Ong (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 297–311.

[4] Kenneth J Biba. 1975. *Integrity Considerations for Secure Computer Systems.* Technical Report MTR-3153. The Mitre Corporation.

[5] John Boyland. 2003. Checking Interference with Fractional Permissions. In *Static Analysis*, Radhia Cousot (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 55–72.

[6] Pritam Choudhury. 2022. Monadic and Comonadic Aspects of Dependency Analysis. *Proc. ACM Program. Lang.* 6, OOPSLA22, Article 172 (Oct 2022), 29 pages. https://doi.org/10.1145/3563335

[7] Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. 2021. A Graded Dependent Type System with a Usage-Aware Semantics. *Proc. ACM Program. Lang.* 5, POPL, Article 50 (Jan 2021), 32 pages. https://doi.org/10.1145/3434331

[8] David D. Clark and David R. Wilson. 1987. A Comparison of Commercial and Military Computer Security Policies. In *1987 IEEE Symposium on Security and Privacy*. 184. https://doi.org/10.1109/SP.1987.10001

[9] Ellis S Cohen. 1978. Information Transmission in Sequential Programs. *Foundations of Secure Computation* (1978), 297–335.

[10] Dorothy E. Denning and Peter J. Denning. 1977. Certification of Programs for Secure Information Flow. *Commun. ACM* 20, 7 (Jul 1977), 504–513. https://doi.org/10.1145/359636.359712

[11] Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart, and Tarmo Uustalu. 2016. Combining Effects and Coeffects via Grading. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming* (Nara, Japan) *(ICFP 2016)*. Association for Computing Machinery, New York, NY, USA, 476–489. https://doi.org/10.1145/2951913.2951939

[12] J. A. Goguen and J. Meseguer. 1982. Security Policies and Security Models. In *1982 IEEE Symposium on Security and Privacy*. 11. https://doi.org/10.1109/SP.1982.10014

[13] Nevin Heintze and Jon G. Riecke. 1998. The SLam Calculus: Programming with Secrecy and Integrity. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '98)*. Association for Computing Machinery, New York, NY, USA, 365–377. https://doi.org/10.1145/268946.268976

[14] Peng Li and Steve Zdancewic. 2005. Unifying Confidentiality and Integrity in Downgrading Policies. In *In Proc. of the LICS'05 Affiliated Workshop on Foundations of Computer Security (FCS)*. Citeseer.

[15] Daniel Marshall, Michael Vollmer, and Dominic Orchard. 2022. Linearity and Uniqueness: An Entente Cordiale. In *Programming Languages and Systems*, Ilya Sergey (Ed.). Springer International Publishing, Cham, 346–375.

[16] Benjamin Moon, Harley Eades III, and Dominic Orchard. 2021. Graded Modal Dependent Type Theory. In *Programming Languages and Systems*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 462–490.

[17] Andrew C. Myers and Barbara Liskov. 2000. Protecting Privacy Using the Decentralized Label Model. *ACM Trans. Softw. Eng. Methodol.* 9, 4 (Oct 2000), 410–442. https://doi.org/10.1145/363516.363526

[18] Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. 2006. Enforcing Robust Declassification and Qualified Robustness. *Journal of Computer Security* 14, 2 (2006), 157–196. https://doi.org/10.3233/JCS-2006-14203

[19] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative Program Reasoning with Graded Modal Types. *Proc. ACM Program. Lang.* 3, ICFP, Article 110 (Jul 2019), 30 pages. https://doi.org/10.1145/3341714

[20] Charles P. Pfleeger. 1988. *Security in Computing*. Prentice-Hall, Inc., USA.

[21] François Pottier and Vincent Simonet. 2003. Information Flow Inference for ML. *ACM Trans. Program. Lang. Syst.* 25, 1 (Jan 2003), 117–158. https://doi.org/10.1145/596980.596983

[22] Alejandro Russo. 2015. Functional Pearl: Two Can Keep a Secret, If One of Them Uses Haskell. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming* (Vancouver, BC, Canada) *(ICFP 2015)*. Association for Computing Machinery, New York, NY, USA, 280–288. https://doi.org/10.1145/2784731.2784756

[23] A. Sabelfeld and A.C. Myers. 2003. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications* 21, 1 (2003), 5–19. https://doi.org/10.1109/JSAC.2002.806121