Developing Advanced Security Concurrent Separation Logic (ASecCSL) [1]

Toby Murray
School of Computing and Information Systems, University of Melbourne
Email: toby.murray@unimelb.edu.au

Navid Abapour
Department of Computer Science, University of Mohaghegh Ardabili
Email: navidabapour@student.uma.ac.ir

Abstract

By invention of novel concurrency paradigms and computable semantics, programs with modern architectures need to be based on the logical systems that support them from different dimensions. Considering it's simplicity and powerful structure, Security Concurrent Separation Logic (SecCSL) can be an outstanding system for these types of programs. SecCSL can get used for proving properties of program, which are related to information flow. Utilizing an exclusive relational assertion semantics, discerning sensitivity of location and value, and providing a clear soundness proof are features that SecCSL circulates around them. In this paper Advanced Security Concurrent Separation Logic (ASecCSL) is presented as a collection of improvements for SecCSL, which along with injecting probability to SecCSL, it is trying to provide an adaptive attitude for quantitative information flow.

Keywords: Information Flow and Leakage, Noninterference, Probabilistic Programs

1 Introduction

Probabilistic primitives are giving us the chance to implement complex constructions in programs; so, pushing the edges of languages and logical systems into probabilistic computation frameworks, apart from creating more abilities, can significantly increase the security level of programs. In the past, by using randomized and distributional approaches, the programs had whether special probability distributions for input or there was a set of random moves for a specified input. Despite of mixing these two approaches nowadays, there are limited numbers of models that avoid specifications of systems and their focus is on concrete systems, and they ignore pure non-determinism or non-probabilistic by using frameworks like *Probabilistic Abstract Interpretation* (PAI) for producing concrete semantics with order-theoretic. Furthermore, connecting *Markov Decision Processes* and expectation transformer semantics of *probabilistic Guarded Command Language* is another way to both create outstanding flexibility in checking formal probabilistic semantics and express randomized algorithms in the program's flow. Probabilistic Type Theory can be another tool to develop new ideas about probabilistic programming e.g. for connecting formal semantic interpretation and syntactic structure.

[1] This manuscript is the report of a simple prototype development from Navid Abapour (B.Sc.) under supervision of Toby Murray. (Last Update: October, 2021)

So, according to these issues and ideas, we need a logical system to focus on security as well as nondeterminism. From security aspect, probabilistic programs can use control flow graphs with program's translations in a structured language to check the secure information flow in multiple elements. Also, clearly probabilistic noninterference and noninterference under refinement are outputs of utilizing probabilistic choice and nondeterminism in the language, but measuring the leakage of information can be a better idea to ensure the security investigation of information flow.

Security Concurrent Separation Logic (SecCSL) is a special type of logic that apart from it's strength and supporting arrays and pointers, it is significantly simple; by executing with symbolic automated, it is able to verify information flow security for a subset of C language automatically via SecC verifier. Despite of the most similar information flow logical systems, SecCSL has a relational assertion semantic for increasing flexibility in feasible automation, along with proving a clear soundness proof for the rules, which is formalized in Isabelle/HOL. The property proof mechanism behind SecCSL, is based on the timing-sesitive security, and Γ is a security typing context that SecCSL use it to track the value sensitivity, and value-dependent predicates are helping to show the location sensitivity.

On the other hand, Security Concurrent Separation Logic has a totally deterministic language semantics for programs, and there is no way for programs to e.g. use randomness. As a result, the definition of information flow security is completely strict and summarized in revealing no information at all. Thus, SecCSL has the potential to get enhanced, and with ASecCSL we demonstrate several cases of improvements for SecCSL:

(I) Initially, we add randomness to Security Concurrent Separation Logic's language semantics by applying nondeterministic models and probabilistic choice.

(II) According to small-step model of operational semantics in SecCSL, the language's semantics is turning into probabilistic type.

(III) Eventually, in order to capture the concerns about quantitative information flow in probabilistic language semantics in ASecCSL, a suitable definition is provided.

2 Adding Randomness to SecCSL

In addition to applying new changes to sequence proof rule, the probabilistic choice's rule should get specified. By showing security type with δ, infix operator $\square$ non-deterministically chooses to execute one of its two operands c_i and c_j. In fact, operator $\square$ executes c_i with probability p, and c_j with probability $1-p$.

$$\frac{\ell \vdash \{P\} c_1 \{R\} \qquad \ell \vdash \{R\} c_2 \{Q\}}{\ell \vdash \{P\}\ c_1\ {}_p\square_\delta\ c_2\ \{Q\}} \quad \textsc{Prob Choice}$$

Considering type *unit* and $supp(\ell) = \{i \mid \ell(i) \neq unit\}$, we are going to define compatibility of ℓ_1 and ℓ_2 like this:

$$\forall i \in supp(\ell_1) \cap supp(\ell_2),\ \ell_1(i) = \ell_2(i) \Longrightarrow \ell_1 \Downarrow \ell_2 \quad \ni \quad \triangle_1 \Downarrow \ell_2, (\triangle_1 \ominus \ell_2) \Downarrow \triangle_2$$

If we consider *union context* as $\ell_1 \oplus \ell_2$, then *difference context* can shown as:

$$\ell_1 \ominus \ell_2 : i \notin supp(\ell_2) \Rightarrow i \mapsto \ell_1(i)$$

Therefore, the Sequential Composition can get modified so that it can be able to support more flexibility of non-determinism:

$$\frac{\ell \vdash \{P\} c_1 \{R\} \qquad \ell_1 \vdash c_1 :\ \triangle_1 \qquad \ell \vdash \{R\} c_2 \{Q\} \qquad \ell_2 \vdash c_2 :\ \triangle_2}{\ell_1 \oplus (\ell_2 \ominus \triangle_1) \vdash c_1; c_2 : (\triangle_1 \ominus \ell_2) \oplus \triangle_2} \quad \textsc{Seq}$$

3 Modifying the Command Semantics

Denote the standard list operations of reading the first element of a list and removing the first element of a list as head and tail, respectively. Given a refiner μ, the value $head(\mu(\delta))$ is used to resolve the next choice annotated with type δ. Also, the command's vectors are shown by $\vec{c} = \langle C_0 \dots C_{n-1} \rangle$ that consist of program's thread pools.

$$\frac{head(\,\mu(\,\delta)\,) = i \qquad \mu' = \mu[\,\delta := tail(\,\mu(\,\delta)\,)\,]}{(\,\mathbf{run}\ c_1\ {}_p\square_{\,\delta}\ c_2, \mu, L, s, h) \xrightarrow{\sigma} (\,\mathbf{stop}\ c_i, \mu', L', s', h')}$$

$$\frac{<c_i, \mu, L, s, h> \longrightarrow <\vec{c}, \mu', L', s', h'>}{<\langle C_0 ... C_{n-1}\rangle, \mu, L, s, h> \longrightarrow^{i}_{\sigma(i,n)} <\langle C_0 ... C_{i-1}\ \vec{c}\ C_{i+1} ... C_{n-1}\rangle, \mu', L', s', h'>}$$

$$\frac{<c_i, \mu, L, s, h> \longrightarrow <\vec{c}, \mu', L', s', h'>}{<\langle C_0 ... C_{n-1}\rangle, \mu, L, s, h> \longrightarrow^{i}_{\frac{1}{n}} <\langle C_0 ... C_{i-1}\ \vec{c}\ C_{i+1} ... C_{n-1}\rangle, \mu', L', s', h'>}$$

$$\frac{S' = S(x \longmapsto [\![e]\!]_s)}{<\mathbf{run}\, x := e, \mu, L, s, h> \xrightarrow{<\tau>}_1 <\mathbf{stop}\, \mu', L, s', h>} \qquad \frac{[\![e]\!]_s \notin \mathrm{dom}\ (h)}{<\mathbf{run}\, x := [e], \mu, L, s, h> \xrightarrow{<\tau>}_1 \mathbf{abort}}$$

$$\frac{[\![e]\!]_s \in \mathrm{dom}\ (h) \qquad s' = s(x \longmapsto h([\![e]\!]_s))}{<\mathbf{run}\, x := [e], \mu, L, s, h> \xrightarrow{<\tau>}_1 <\mathbf{stop}\, \mu, L, s', h>} \qquad \frac{[\![e_1]\!]_s \notin \mathrm{dom}\ (h)}{<\mathbf{run}\, [e_1] := e_2, \mu, L, s, h> \xrightarrow{<\tau>}_1 \mathbf{abort}}$$

$$\frac{[\![e_1]\!]_s \in \mathrm{dom}\ (h) \qquad h' = h([\![e_1]\!]_s \longmapsto [\![e_2]\!]_s)}{<\mathbf{run}\ [e_1] := e_2, \mu, L, s, h> \xrightarrow{<\tau>}_1 <\mathbf{stop}\ \mu', L, s, h'>}$$

$$\frac{l \in L \qquad L' = L \cup \{l\}}{<\mathbf{run}\ \mathrm{lock}\, l, \mu, L, s, h> \xrightarrow{<\tau>}_1 <\mathbf{stop}\ \mu', L', s, h>}$$

$$\frac{l \notin L \qquad L' = L \cup \{l\}}{<\mathbf{run}\ \mathrm{unlock}\, l, L, s, h> \xrightarrow{<\tau>}_1 (\mathbf{stop}\ \mu, L', s, h)} \qquad \frac{<\mathbf{run}\ c_1, \mu, L, s, h> \xrightarrow{\sigma}_1 \mathbf{abort}}{<\mathbf{run}\ c_1; c_2, \mu, L, s, h> \xrightarrow{\sigma}_1 \mathbf{abort}}$$

$$\frac{<\mathbf{run}\ c_1, \mu, L, s, h> \xrightarrow{\sigma}_1 \mathbf{abort}}{<\mathbf{run}\ c_1 \parallel c_2\,, \mu, L, s, h> \xrightarrow{<1>.\sigma}_1 \mathbf{abort}}$$

$$\frac{<\mathbf{run}\ c_1, \mu, L, s, h> \xrightarrow{\sigma}_1 <\mathbf{stop}\ \mu', L', s', h'> \qquad j \in [1, n] \qquad s \vdash \mathbb{C}_j}{<\square_{i=1}^{n}\ \mathbb{C}_i \mid p_i \to (\mathbf{run}\ c_1; c_2, \mu, L, s, h)> \xrightarrow{\sigma}_{\widetilde{p}_j} <\mathbf{run}\ c_j, \mu', L', s', h'>}$$

$$\frac{<\mathbf{run}\ c_1, \mu, L, s, h> \xrightarrow{\sigma}_1 <\mathbf{run}\ c_1', \mu', L', s', h'> \qquad j \in [1, n] \qquad s \vdash \mathbb{C}_j}{<\square_{i=1}^{n}\ \mathbb{C}_i \mid p_i \to (\mathbf{run}\ c_1; c_2, \mu, L, s, h)> \xrightarrow{\sigma}_{\widetilde{p}_j} <\mathbf{run}\ c_1'; c_2, \mu', L', s', h'>}$$

$$\frac{\begin{array}{c}<\mathbf{run}\ c_1, \mu, L, s, h> \xrightarrow{\sigma}_1 <\mathbf{stop}\ \mu', L', s', h'> \qquad j \in [1, n] \qquad s \vdash \mathbb{C}_j \\ <\mathbf{run}\ c_i, \mu, L, s, h> \xrightarrow{\sigma}_{\widetilde{p}_j} <\mathbf{run}\ c_i', \mu', L', s', h'>\end{array}}{<\square_{i=1}^{n}\ \mathbb{C}_i \mid p_i \to (\mathbf{run}\ c_1 \parallel c_2, \mu, L, s, h)> \xrightarrow{<1>\ \sigma}_{\widetilde{p}_j} <\mathbf{run}\ c_j', \mu', L', s', h'>}$$

$$\frac{<\mathbf{run}\ c_1,\mu,L,s,h> \xrightarrow{\sigma}_1 <\mathbf{run}\ c_1',\mu',L',s',h'> \qquad j\in[1,n] \qquad s\vdash\mathbb{C}_j \qquad <\mathbf{run}\ c_i,\mu,L,s,h> \xrightarrow{\sigma}_{\widetilde{p}_j} <\mathbf{run}\ c_i',\mu,L',s',h'>}{\begin{array}{l}<\square_{i=1}^n\ \mathbb{C}_i \mid p_i \to (\mathbf{run}\ c_1 \parallel c_2,\mu,L,s,h)> \xrightarrow{<1>\ \sigma}_{\widetilde{p}_j} <\mathbf{run}\ c_1' \parallel c_2,\mu',L',s',h'> \\ <\square_{i=1}^n\ \mathbb{C}_i \mid p_i \to (\mathbf{run}\ c_2 \parallel c_1,\mu,L,s,h)> \xrightarrow{<1>\ \sigma}_{\widetilde{p}_j} <\mathbf{run}\ c_2 \parallel c_1',\mu',L',s',h'>\end{array}}$$

$$\frac{\text{if } s\vDash b \text{ then } c'=c_1 \text{ else } c'=c_2 \qquad j\in[1,n] \qquad s\vdash\mathbb{C}_j}{<\square_{i=1}^n\ \mathbb{C}_i \mid p_i \to (\mathbf{run}\ \text{if } b \text{ then } c_i \text{ else } c_j,\mu,L,s,h)> \xrightarrow{<\tau>}_{\widetilde{p}_j} <(\mathbf{run}\, c',\mu,L,s,h)>}$$

4 Developing the Definition of QIF for SecCSL

From non-determinism approach, a program with ability to start from initial state e and end up to e' can cover the noninterference property, if for all $e_1(l) = e_2(l)$, achieving a state e_2 can be accessible, such that the program which can start from e_2 and end up at e_2' with regards to $e_1(l) = e_2(l)'$ be able to terminate with success. More specifically, the Sabelfeld's results on bisimulation can get used to strength the confidentiality: A PER (Partial Equivalence Relation) like R is a σ-probabilistic low-bisimulation on program's commands, and $\vec{c_1}\ R\ \vec{c_2}$ iff

$$\forall\, s_1 =_\ell s_2.<\vec{c_1},\mu,L,s_1,h> \longrightarrow <\vec{c_1'},\mu',L',s_1',h'> \Longrightarrow \exists \vec{c_2'},s_2'.<\vec{c_2},\mu,L,s_2,h> \longrightarrow <\vec{c_2'},\mu',L',s_2',h'>$$

$$\text{Such that: } \sum\{p|<\vec{c_1},\mu,L,s_1,h> \longrightarrow_p <\vec{c_3},\mu,L,s,h>,\ \vec{c_3}\in[\vec{c_1'}]_R,\ \ s=_\ell s_1'\} =$$

$$\sum\{p|<\vec{c_2},\mu,L,s_2,h> \longrightarrow_p <\vec{c_3},\mu,L,s,h>,\ \vec{c_3}\in[\vec{c_2'}]_R,\ \ s=_\ell s_2'\}$$

$$\text{Hence: } \vec{c_1'}\ R\ \vec{c_2'},\ \ s_1' =_\ell s_2'$$

$$\vec{c} \text{ is } \sigma\text{-secure} \iff \vec{c}\ \sim_\ell^\sigma \vec{c}, \quad \vec{c}\approx_\ell \vec{c}_2 \iff \forall\sigma.\vec{c}_1 \sim_\ell^\sigma \vec{c}_2, \quad \vec{c} \text{ is scheduler-independent secure} \iff \vec{c}\approx_\ell \vec{c}$$

In the SecCSL, a program is secure, iff it leaks no information at all. Also, noninterference is qualitative property, and security of a probabilistic program, which is based on a logical system that limiting it self to noninterference will have critical vulnerabilities; so, creating structural relations between security and leakage is going to be highly effective against adversary's attacks. By assuming $x \in X$ and $y \in Y$, $p(x,y)$ is their joint distribution, let $\hat{A}$ denote the randomness of input/output variable A and $\hat{l'}$ as an output observation random variable. Several concepts need to get clarify before continuing, and it can be done by letting $\mathcal{X}$ to has a prior π, along with a channel ($\mathcal{X}$, $\mathcal{Y}$, $\mathcal{C}$):

$$\text{prior vulnerability} \quad V(\pi) = max_{x\in\mathcal{X}}\,\pi[x]$$

$$\text{post-prior vulnerability} \quad V(\pi,\mathcal{C}) = \sum_{y\in\mathcal{Y}} max_{x\in\mathcal{X}}\,\pi[x]\mathcal{C}[x,y]$$

$$\text{min-entropy} \quad \mathcal{H}_\infty(\pi) = -\log V(\pi)$$

$$\text{conditional min-entropy} \quad \mathcal{H}_\infty(\pi, \mathcal{C}) = -\log V(\pi, \mathcal{C})$$

$$\text{min-entropy leakage} \quad I_\infty(\pi, \mathcal{C}) = \mathcal{H}_\infty(\pi) - \mathcal{H}_\infty(\pi, \mathcal{C})$$

Now we can show secure information flow by realizing the leakage ($\mathcal{L}$) in mutual information between $\hat{l'}$ and $\hat{h}$ given the knowledge of $\hat{l}$ as a formal relation:

$$\mathcal{L}(\hat{l'}; \hat{h}|\hat{l}) = I_\infty(\pi, \mathcal{C})(\hat{l'}|l) \quad \ni \quad \mathcal{L}(\hat{l'}; \hat{h}) = \sum_x \sum_y p(x, y) \log_2 \tfrac{p(x,y)}{p(x)p(y)}$$

$$\mathcal{L}(\hat{l'}; \hat{h}|\hat{l}) = I_\infty(\pi, \mathcal{C})(\hat{l'}|\hat{l}) + I_\infty(\pi, \mathcal{C})(\hat{h}|\hat{l}) - I_\infty(\pi, \mathcal{C})(\hat{l'}, \hat{h}|\hat{l})$$

$$\mathcal{L}(\hat{l'}; \hat{h}|\hat{l}) = I_\infty(\pi, \mathcal{C})(\hat{h}|\hat{l}) - I_\infty(\pi, \mathcal{C})(\hat{h}|\hat{l'}, \hat{l})$$

References

C. Mu and D. Clark, "Quantitative Analysis of Secure Information Flow via Probabilistic Semantics," 2009 International Conference on Availability, Reliability and Security, 2009, pp. 49-57.

A. Di Pierro and H. Wiklicky, "An operational semantics for probabilistic concurrent constraint programming," Proceedings of the 1998 International Conference on Computer Languages (Cat. No.98CB36225), 1998, pp. 174-183.

F. Dahlqvist and D. Kozen, "Semantics of Higher-Order Probabilistic Programs with Conditioning," Proc. ACM Program. Lang., vol. 4, no. POPL, Dec. 2019.

D. Huang and G. Morrisett, "An Application of Computable Distributions to the Semantics of Probabilistic Programming Languages," in Programming Languages and Systems, 2016, pp. 337–363.

M. Sabok, S. Staton, D. Stein, and M. Wolman, "Probabilistic Programming Semantics for Name Generation," Proc. ACM Program. Lang., vol. 5, no. POPL, Jan. 2021.

A. Di Pierro, C. Hankin, and H. Wiklicky, "Probabilistic Semantics and Program Analysis," in Proceedings of the Formal Methods for Quantitative Aspects of Programming Languages, and 10th International Conference on School on Formal Methods for the Design of Computer, Communication and Software Systems, 2010, pp. 1–42.

Staton, S., Yang, H., Heunen, C., Kammar, O., and Wood, F.D. (2016). Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. 2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), 1-10.

D. Kozen, "Semantics of probabilistic programs," 20th Annual Symposium on Foundations of Computer Science (sfcs 1979), 1979, pp. 101-114.

K. R. O'Neill, M. R. Clarkson and S. Chong, "Information-flow security for interactive programs," 19th IEEE Computer Security Foundations Workshop (CSFW'06), 2006, pp. 12 pp.-201.

T. Amtoft and A. Banerjee, "A Semantics for Probabilistic Control-Flow Graphs," ArXiv, vol. abs/1711.02256, 2017.

F. Gretz, J.-P. Katoen, and A. McIver, "Operational versus weakest pre-expectation semantics for the probabilistic guarded command language," Performance Evaluation, vol. 73, pp. 110–132, 2014.

H. Jifeng, K. Seidel, and A. McIver, "Probabilistic models for the guarded command language," Science of Computer Programming, vol. 28, no. 2, pp. 171–192, 1997.

R. Cooper, S. Dobnik, S. Larsson, and S. Lappin, "Probabilistic Type Theory and Natural Language Semantics," Linguistic Issues in Language Technology, vol. 10, 2015.

P. Mardziel, M. S. Alvim, M. Hicks and M. R. Clarkson, "Quantifying Information Flow for Dynamic Secrets," 2014 IEEE Symposium on Security and Privacy, 2014, pp. 540-555.

M. R. Clarkson, A. C. Myers, and F. B. Schneider, "Quantifying information flow with beliefs," J. Comput. Secur., vol. 17, pp. 655–701, 2009.

Clarke, D., and Keller, B. Efficiency in Ambiguity: Two Models of Probabilistic Semantics for Natural Language, IWCS, 2015.

Y. Kawamoto, K. Chatzikokolakis, and C. Palamidessi, "On the Compositionality of Quantitative

Information Flow," ArXiv, vol. abs/1611.00455, 2017.

D. Volpano and G. Smith, "Probabilistic Noninterference in a Concurrent Language," J. Comput. Secur., vol. 7, no. 2–3, pp. 231–253, Mar. 1999.

D. Volpano and G. Smith, "Probabilistic noninterference in a concurrent language," in Proceedings of the Computer Security Foundations Workshop, Jul. 1998, pp. 34–43.

A. Sabelfeld and D. Sands, "Probabilistic noninterference for multi-threaded programs," Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13, 2000, pp. 200-214.

R. Pardo, W. Rafnsson, C. Probst, and A. Wasowski, "Privug: Quantifying Leakage using Probabilistic Programming for Privacy Risk Analysis," ArXiv, vol. abs/2011.08742, 2020.

A. Sabelfeld, "Confidentiality for Multithreaded Programs via Bisimulation," in Perspectives of System Informatics, 2003, pp. 260–273.

A. Aldini and A. Di Pierro, "A Quantitative Approach to Noninterference for Probabilistic Systems," Electronic Notes in Theoretical Computer Science, vol. 99, pp. 155–182, 2004.

G. Smith and D. Volpano, "Secure Information Flow in a Multi-Threaded Imperative Language," in Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1998, pp. 355–364.

A. Popescu, J. Hölzl, and T. Nipkow, "Formalizing Probabilistic Noninterference," in Certified Programs and Proofs, 2013, pp. 259–275.

G. Smith, "Improved typings for probabilistic noninterference in a multi-threaded language," Journal of Computer Security, vol. 14, pp. 591–623, Dec. 2006.