Modeling Security

Thomas Given-Wilson

PACE Meeting, Lyon

February 9, 2014

Introduction

This presentation is a discussion of current work and progresses via motivating examples. The syntax will mostly be based upon process calculi, particularly π-calculi and Concurrent Constraint Programming (CCP).

There two main parts to the presentation

1. Information leakage
2. Languages and models

Information Leakage

Information leakage is often measured by considering the probabilistic outputs of a process (also function or channel) given some secret information.

For example, we can represent the behaviour of a fair coin toss (no secret information) output on a channel m with a process $\mathcal{C}_m$ as follows:

$$\mathcal{C}_m \quad \stackrel{\text{def}}{=} \quad (\nu n)(\overline{n}\langle 0\rangle + \overline{n}\langle 1\rangle \mid n(x).\overline{m}\langle x\rangle) \ .$$

Clearly with fair non-deterministic choice $+$ both 0 and 1 will be output along m with 0.5 probability.

Hiding Secrets

Now consider the leakage of two processes that begin with some secret information $s \in \{0, 1\}$.

A process that leaks all the information (along a channel name m):

$$\mathcal{L}_m \quad \stackrel{\text{def}}{=} \quad \overline{m}\langle s \rangle \ .$$

and a process that leaks no information (by bitwise or'ing the secret with a fair coin):

$$\mathcal{S}_m \quad \stackrel{\text{def}}{=} \quad (\nu n)(\mathcal{C}_n \mid n(c).([s = c]\overline{m}\langle 0 \rangle \mid [s \neq c]\overline{m}\langle 1 \rangle))$$

and with the coin abstracted away to a parameter c:

$$\mathcal{S}_m(c) \quad \stackrel{\text{def}}{=} \quad [s = c]\overline{m}\langle 0 \rangle \mid [s \neq c]\overline{m}\langle 1 \rangle \ .$$

Combining Processes

It would be nice to know when processes can be safely combined, or to know what the results on leakage are of combining processes. However, this turns out to be rather complex.

Consider a process $\mathcal{B}_n(c)$ that simply outputs the result of a fair coin toss c. Neither $\mathcal{S}_m(c)$ nor $\mathcal{B}_n(c)$ leak any information about s alone, however knowing both outputs yields the secret s!

So can we reason about leakage when combining processes?

Independence of Variables

One solution that solves the previous problem (as identified by Yusuke Kawamoto) is to have independence of the functions (processes/variables). Here this would prevent the sharing of the coin c between both processes.

So consider two instances of the $\mathcal{S}_m$ function $\mathcal{S}1_{m1}$ and $\mathcal{S}2_{m2}$ as follows

$$\begin{aligned} \mathcal{S}1_{m1} &\stackrel{\text{def}}{=} (\nu n)(\mathcal{C}_n \mid n(c1).([s = c1]\overline{m1}\langle 0\rangle \mid [s \neq c1]\overline{m1}\langle 1\rangle)) \\ \mathcal{S}2_{m2} &\stackrel{\text{def}}{=} (\nu n)(\mathcal{C}_n \mid n(c2).([s = c2]\overline{m2}\langle 0\rangle \mid [s \neq c2]\overline{m2}\langle 1\rangle)) \ . \end{aligned}$$

Both do not leak information independently, and also do not leak information when combined.

External Knowledge

However, what if an adversary knew from observation when $c2 > c1$? Maybe:

1. the algorithms for generating the coins are observably different to an adversary, or
2. the algorithms for computing the outputs are different, or
3. the adversary has some other source of information...

Perhaps the most interesting to model would be 2, something like $\mathcal{S}2_{m2}$ replaced by:

$$\mathcal{S}2'_{m2} \stackrel{\text{def}}{=} (\nu n)(\mathcal{C}_n \mid n(c2).([c2 = 0]\overline{m2}\langle s\rangle \mid [c2 = 1]\overline{m2}\langle (s+1)\%2\rangle))$$

where the calculation of $(s + 1)\%2$ takes more reductions.

Weakly Equivalent is too Weak

Perhaps we can solve these kinds of problems by enforcing strong equivalence results? The difference in calculation time between $\mathcal{S}2_{m2}$ and $\mathcal{S}2'_{m2}$ could be captured by representing the calculation time as a τ reduction with $\mathcal{S}2'_{m2} \stackrel{\text{def}}{=}$

$$(\nu n)(\mathcal{C}_n \mid n(c2).([c2 = 0]\overline{m2}\langle s\rangle \mid [c2 = 1] \quad \tau. \quad \overline{m2}\langle(s+1)\%2\rangle)) \ .$$

Now we could show that strong equivalence separates (some) processes that leak information from those that don't.

About Equivalence...

While considering behavioural equivalence, alternative approaches such as high and low information can be examined. Consider that an alternative to declaring independence in the abstract manner here, is to define it by declaring that variables may not be shared between processes. The problem of the original

$$\begin{aligned} \mathcal{S}_m(c) &\stackrel{\text{def}}{=} [s = c]\overline{m}\langle 0\rangle \mid [s \neq c]\overline{m}\langle 1\rangle \\ \mathcal{B}_m(c) &\stackrel{\text{def}}{=} \overline{m}\langle c\rangle \end{aligned}$$

can be solved by declaring c a high variable. Now leaking c can be seen as an information leak.

High and Low too Strong

Unfortunately this turns out to be too strong. Consider the alternative formulation of $\mathcal{S}_m(c)$ given by

$$\mathcal{S}'_m(c) \stackrel{\text{def}}{=} [c = 0]\overline{m}\langle s\rangle \mid [c = 1]([s = 0]\overline{m}\langle c\rangle \mid [s = 1]\overline{m}\langle 0\rangle) \ .$$

This is (strongly) behaviourally equivalent to $\mathcal{S}_m(c)$ that leaks no information, but can leak the "high" variables s and c.

What About When Leakage is Reduced

There are lots of ways that combining processes can leak information, but can combining process hide information?

Consider the following two processes:

$$\mathcal{T}1_{m1}(c1) \stackrel{\text{def}}{=} [c1 = 0]\,\tau.\,\overline{m1}\langle s\rangle \mid [c1 = 1]\overline{m1}\langle (s+1)\%2\rangle$$
$$\mathcal{T}2_{m2}(c2) \stackrel{\text{def}}{=} [c2 = 0]\overline{m2}\langle s\rangle \mid [c2 = 1]\,\tau.\,\overline{m2}\langle (s+1)\%2\rangle\ .$$

Due to the silent reductions τ either one alone leaks the secret. Yet running them in parallel only leaks the secret some of the time (depending on the coins and order of reductions taken).

Leakage can be reduced further by combining all the outputs into a single result, e.g.

$$\mathcal{T}_m(c1, c2) \stackrel{\text{def}}{=} \mathcal{T}1_{m1}(c1) \mid \mathcal{T}2_{m2}(c2) \mid m1(x).m2(y).\overline{m}\langle x, y\rangle\ .$$

Leakage Summary

A summary on modeling leakage with processes:

- Composition of processes can leak information
- Weak behavioural equivalence is too weak
- Using high and low variables is too strong
- Composition of processes can hide information

Languages and Models

A different arc of research is into understanding and creating languages that can model privacy and security properties.

Constructing new languages to specifically model properties, for example spacial systems with desirable properties.

Understanding languages, their expressiveness, and their relation to each other.

Spacial Concurrent Constraint Programming (SCCP)

A development of Concurrent Constraint Programming (CCP) that includes a notion of agent spaces. Consists of processes P and constraints c with reductions of processes and a collection of constraints σ captured by:

$$\frac{}{\langle \mathtt{ask}(c) \to P, \sigma \rangle \longmapsto \langle P, \sigma \rangle} \; \sigma \models c \qquad \frac{}{\langle \mathtt{tell}(c), \sigma \rangle \longmapsto \langle \mathbf{0}, \sigma \sqcup c \rangle}$$

the SCCP extension add a process $[P]_i$ that contains the process P within the space of an agent i. Also the concept of the scope of the constraints that are within an agent space $s_i(c)$. Consider the new reduction

$$\frac{\langle P, \rho \rangle \longmapsto \langle P', \rho' \rangle}{\langle [P]_i, \sigma \rangle \longmapsto \langle [P']_i, \sigma \sqcup s_i(\rho') \rangle} \; s_i(\sigma) = \rho \; .$$

A Communication Problem

Unfortunately this language does not allow for communication since the `tell` primitive is still scoped by agent spaces.

$$\frac{\langle \texttt{tell}(c), \rho\rangle \longmapsto \langle \mathbf{0}, \rho' \sqcup c\rangle}{\langle [\texttt{tell}(c)]_i, \sigma\rangle \longmapsto \langle [\mathbf{0}]_i, \sigma \sqcup s_i(\rho' \sqcup c)\rangle} \; s_i(\sigma) = \rho \; .$$

This implies the creation of a new `send` primitive to send information to another agent, regardless of spaces/scopes.

This alone could be non-trivial to add to the language in a clean manner, but is made more complex by security concerns. . .

Accepting Messages

Simply allowing messages to be sent to an agent allows malicious agents to send bad constraints. For example, a malicious agent can simply send contradiction to another agent to render the other agent contradictory.

$$\langle [\texttt{send}(j, \bot)]_i \mid [P]_j, \sigma \rangle \Longmapsto \langle []_i \mid [P]_j, \sigma \sqcup s_j(\bot) \rangle$$

This in turn implies that an `acc`(ept) primitive may be required that allows the receiving agent to declare which other agents to accept messages from.

This could perhaps be solved with some kind of global message buffer like the constraints where messages live in transit between `send` and `acc`.

Agent Boundaries

However, this ignores agent boundaries as potential barriers to communication, and which space belonging to the receiving agent (and perhaps sending agent) is involved in the communication.

An alternative that could start addressing these is to consider agent boundaries like in the Mobile Ambient calculus, and have explicit primitives to move in and out of agent spaces...

$$\frac{}{\langle \texttt{enter}(i) \rightarrow P \mid [Q]_i, \sigma \rangle \longmapsto \langle [P \mid Q]_i, \sigma \rangle}$$

$$\frac{}{\langle [\texttt{exit}(i) \rightarrow P \mid Q]_i, \sigma \rangle \longmapsto \langle P \mid [Q]_i, \sigma \rangle} \quad .$$

Who Owns a Boundary?

However, this is still problematic as this would allow any agent to send messages/processes across any boundaries it knows. In practice, boundaries are usually controlled by one or both sides, consider:

- A private network connected to (inside?) the internet.
- A user application running on (inside?) a kernel/system space.
- A laptop connected to (inside?) a private network.

Clearly there is no simple answer.

Aside: this work is with linked with Frank Valencia's and it is a goal to try and find a logical axiomatisation for any new communication primitives.

Understanding Languages

Building new languages, particularly process calculi, is “easy” and there are many of them. Another area of research is better understanding of process calculi in general.

Here the focus is on understanding the rôle of certain properties in communication primitives; past examples include: synchronism, arity, communication-medium, spaces, types, and pattern-matching.

Some recent features include: intensionality, symmetry, and logics.

Intensionality

Intensionality is the idea that a communication primitive may have behaviour dependent upon the structure of what is being communicated. For example consider the three processes:

$$P1 \stackrel{\text{def}}{=} \overline{n}\langle a \bullet b \rangle \qquad P2 \stackrel{\text{def}}{=} \overline{n}\langle c \rangle \qquad Q \stackrel{\text{def}}{=} n(x \bullet y).Q' \qquad R \stackrel{\text{def}}{=} n(z).R'$$

where $P1$ and $P2$ are outputs of the compound $a \bullet b$ and the name c, respectively. Also Q and R are inputs of $x \bullet y$ and z, respectively. $P1$ can reduce with both Q and R:

$$\begin{aligned} & P1 \mid Q \mid R \longmapsto \{a/x, b/y\}Q' \mid R \\ \text{or} \quad & P1 \mid Q \mid R \longmapsto Q \mid \{a \bullet b/z\}R' \end{aligned}$$

however $P2$ can only reduce with R

$$P2 \mid Q \mid R \longmapsto Q \mid \{c/z\}R'$$

Expressiveness of Intensionality

Intensionality (that includes the capability to determine equality of names in interaction) turns out to be able to encode: synchronism, arity, communication-medium, and pattern-matching.

Theorem

Any asynchronous/synchronous, monadic/polyadic, data-space based/channel-based, (non-)pattern-matching language can be encoded into any intensional language. That is, intensionality alone is sufficient to encode: synchronicty, polyadicity, channel-based communication, and name-matching.

Onwards...

Other features not (yet) captured with these kinds of results. Future work can explore relations based on these:

- **Symmetry:** Such as in fusion calculus and concurrent pattern calculus (CPC). (Symmetry has been used to show separation results, mostly for CPC.)
- **Logics:** Such as in CCP style calculi and Psi calculus. (Logics have been used to show separation results for Psi calculus.)
- **Other features...**

Goal to show which calculi have greater expressiveness, and also which features provide expressiveness. Also means that equivalence in expressiveness means freedom to chose the most convenient calculus, e.g. for modeling a system/property.

Conclusions

Information leakage with process calculi:

- Process composition can leak both more or less information
- Syntactical mechanisms are insufficient (high/low names)
- Strong equivalences required

Languages and models:

- Combining spaces and communication is messy
- Spacial/hierarchical classifications do not always align
- Many features of communication primitives, their relative expressiveness is not fully understood.