Introduction

Calculus Vs. Logic Vs. Algebra

- **Calculus**
  - Branch of mathematics that studies continuously changing properties
  - Characterized by the use of infinite processes, involving passage to a limit
- **Logic**
  - Systematic study of valid inference
  - Necessitates a distinction between logical validity and truth
- **Algebra**
  - Branch of mathematics concerned with operations on sets
  - Find solutions to systems of equations
Pi Calculus

- Generalization of Calculus of Communicating Systems (CCS)
- Contains two entities
  - Processes
  - Channels
- Processes interact by synchronous rendezvous on channels
- Basic Lotos is a combination of CCS and CSP

Boxed Pi Calculus

- Based on unordered asynchronous message passing
- Turing Complete
- Includes a boxing primitive
  - Communication across box boundaries is strictly limited
- Process terms cannot be placed on the channel
- No primitives for the movement of boxes
- It is untyped

Syntax

- Names
  - All combination of letters except reserved letters
- Values and Patterns
  - Values \( u, v \)
    - \( u ::= x \)
    - \( v ::= \langle v_1 \ldots v_n \rangle \)
  - Pattern \( p \)
    - \( p ::= _{\ldots} \)
    - \( p ::= x \)
    - \( p ::= \langle p_1 \ldots p_n \rangle \)
Syntax Continued

- Process P
  - $P ::= n[P]$
  - $P ::= P | P'$
  - $P ::= x\cdot y$
  - $P ::= \exists p.P$
  - $P ::= x \cdot (p.P)$
  - $P ::= (v x) P$
- Input tag i and output tag o can be $\ast$, $\dagger$, n

Semantics

- Operational Semantics
  - describes the meaning of a programming language
  - specifying how it executes on an abstract machine
- Reduction Semantics
  - Defines the internal computation of the processes
- Labeled semantics
  - Inductively defined on process structure by a structural operational semantics (SOS)

Filtering Example

- alice[P] | linalice x | routalice x | netalice x
- $W_{1}(\_)$ def $= (v a) (a \cdot \_ | B)$
  - $B = lin x, in x | rout x, out x$
- $W_{1}$ prevents Q from accessing the network
- Assumes $P=0$ and $Q=\text{in}^* x, \text{net}^* x$
Filtering Example Continued

- $\in_{\text{alice}} | \text{alice}[P | W, (Q)]$
- $\in_{\text{alice}} | \text{alice}[(v a)[a(Q) | B]]$
- $\text{alice}[(v a)[a(Q) | B]]$
- $\text{alice}[(v a)[a(Q) | B]]$
- $\text{alice}[(v a)[a(Q) | B]]$
- $\text{alice}[(v a)[a(Q) | B]]$
- $\text{alice}[(v a)[a(Q) | B]]$
- $\text{alice}[(v a)[a(Q) | B]]$

Unidirectional-flow Wrapper

- Provides ordered asynchronous communication
- Eliminates the awkwardness of only using only unordered asynchronous communication while ensuring casual independence
- Necessitates the maintaining of an unbounded buffer

Colouring and Causal Flow

- The box pi calculus is extended to handle coloring
- Output processes are annotated with the sets of colours that record their histories
  - The sets of principles that have effected them in the past
- Very simple colouring that only captures a limited amount of causality information
Causality Types

- The type system statically captures causal flows
- A wrapper can be shown to satisfy the causal flow property simply by checking that it is well-typed

Discussion

- Wrappers impose security policies on components for which it is impractical to analyze the internal structure
- Alternative policies
  - Java style sand boxing
  - Code signing

Conclusions

- Provided techniques for proving that software wrappers enforce user specified flow constraints
- Security is solely a function of causality