Analyzing Computational Models

Michael W. Mislove

Tulane University
New Orleans, LA

LSU NSF/SFS Workshop on
Critical Infrastructure Protection
July 2, 2013

Work sponsored by AFOSR & NSF

Outline

Theory: Track A vs Track B

- Some basic background
- Some examples: two presentations from last week at Tulane
- Two more examples: Security-related work
- My own research: *Continuous random variables*

Mathematical Models

Mathematical constructs – systems – that model computational processes

Examples:

- Operational models (e.g., automata):
  - Give step-by-step representation of computational processes
  - Good for understanding how a process evolves, or finding bugs
  - Often too low-level to *prove* properties
- Denotational models (e.g., domains):
  - Give mathematical models of computational processes
  - High level – abstract away from low-level details
  - Good for proving process properties
    - *Compositional*
    - *Automated tool support (proof assistants)*

Domains and Computability

- *Church-Turing Thesis:*
  Partial recursive functions $f : \mathbb{N} \rightharpoonup \mathbb{N}$ are the computable functions.

Domains and Computability

- *Church-Turing Thesis:*
  Partial recursive functions $f : \mathbb{N} \rightharpoonup \mathbb{N}$ are the computable functions.
- Modeling partial recursives:
  - $f \leq g \quad \text{iff} \quad \operatorname{dom} f \subseteq \operatorname{dom} g \ \& \ g|_{\operatorname{dom} f} = f$

    – *Extensional order*
  - $(\mathbb{N} \rightharpoonup \mathbb{N}, \leq)$ chain complete partial order

    – $\sup C = \bigcup \{f \mid f \in C\}$
  - $f = \sup_{n \in \mathbb{N}} f|_{\{0,\ldots,n\}}$

Domains and Computability

- *Church-Turing Thesis:*
  Partial recursive functions $f\colon \mathbb{N} \rightharpoonup \mathbb{N}$ are the computable functions.
- $F\colon (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$ *Scott continuous* if $F(\sup C) = \sup F(C)$ for every chain $C \subseteq (\mathbb{N} \rightharpoonup \mathbb{N})$.
- *Example:*

$$F(f)(n) = \begin{cases} 1 & \text{if } n = 0, \\ n \cdot F(f)(n-1) & \text{if } F(f)(n-1) \text{ is defined,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

- $\mathit{Fac}\colon \mathbb{N} \rightharpoonup \mathbb{N}$ satisfies $\mathit{Fac} = \mathrm{FIX}(F) = \sup_n F^n(\emptyset)$.

Domains and Computability

- *Church-Turing Thesis:*
  Partial recursive functions $f\colon \mathbb{N} \rightharpoonup \mathbb{N}$ are the computable functions.
- *Knaster-Tarski-Scott Fixed Point Theorem:*
  Each Scott continuous selfmap $F\colon (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$ has a least fixed point, $\mathsf{FIX}(F) = \sup_n F^n(\emptyset)$.
- *Myhill-Shepherson Theorem:*
  The partial recursives are those $f \in \mathbb{N} \rightharpoonup \mathbb{N}$ satisfying $f = \mathsf{FIX}(F)$ for some Scott continuous $F\colon (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$.
- *Church-Turing-Scott Thesis:*
  The computable functions are those that are least fixed points of Scott continuous selfmaps of $(\mathbb{N} \rightharpoonup \mathbb{N})$.

Domains and Programming Languages

- *More Abstractly (Scott, 1969):*
  The lambda calculus admits a model $M = [D^\infty \to D^\infty]$ where every term is a Scott continuous selfmap of a recursively defined domain $D^\infty \simeq [D^\infty \to D^\infty]$.
  - More generally, *every* model of the lambda calculus is a reflexive object in some *cartesian closed category*.
    - Like Set - category of sets and functions
    - Only known models are categories of domains

Domains and Programming Languages

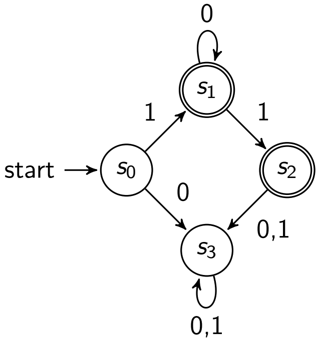
- *More Abstractly (Scott, 1969):*
  The lambda calculus admits a model $M = [D^\infty \to D^\infty]$ where every term is a Scott continuous selfmap of a recursively defined domain $D^\infty \simeq [D^\infty \to D^\infty]$.
- *Scott's Program:*
  - Data types are partially ordered structures
  - Programs are Scott continuous maps over data types
  - Cartesian closed categories of *domains* are natural denotational models

Domains and Programming Languages

- *More Abstractly (Scott, 1969):*
  The lambda calculus admits a model $M = [D^\infty \to D^\infty]$ where every term is a Scott continuous selfmap of a recursively defined domain $D^\infty \simeq [D^\infty \to D^\infty]$.
- *Scott's Program:*
  - Data types are partially ordered structures
  - Programs are Scott continuous maps over data types
  - Cartesian closed categories of *domains* are natural denotational models
- Moggi's Program:
  - Use *monads* to model *computational effects:*
    - Power domains for nondeterminism,
    - Continuations, exceptions, etc.
  - Each arises as family of algebras for an endofunctor on a category of domains.

Automata

- Automata are among the simplest computational models
  - Simple graphical representation
  - Illustrate how computations unfold
  - Easy to write (for simple processes)
  - Limited use for complicated processes
  - Not useful for proving general properties
- *Simple Example:*



Language accepted by A:

$L_A = \{10^*, 10^*1\}$

From Automata to Domains

- A - finite alphabet $\Rightarrow A^\infty = A^* \cup A^\omega$ word monoid over A.
  - – A^∞ partial order in *prefix order:*
    $s \leq t$ iff $su = t$ for some word u.
  - – $(A^\infty, \leq)$ chain complete poset
- $L_A = \{10^*, 10^*1\} \subseteq \{0,1\}^\infty$

From Automata to Domains

- A - finite alphabet $\Rightarrow A^\infty = A^* \cup A^\omega$ word monoid over A.

  – A^∞ partial order in *prefix order:*
  $s \leq t$ iff $su = t$ for some word u.

  – $(A^\infty, \leq)$ chain complete poset

- $L_A = \{10^*, 10^*1\} \subseteq \{0,1\}^\infty$

- L_A *not* closed under prefixes,...

  ... but $\downarrow L_A = \{s \in \{0,1\}^\infty \mid s \leq u \in L_A\}$ is.

- $\downarrow L_A$ also closed under sups of chains.

  So $\downarrow L_A = \downarrow 10^\omega \cup \{10^n 1 \mid n \geq 0\}$ is a domain.

  Also a *safety property* (Alpern & Schneider)

  – Can't distinguish L_A from $L_B = \{1(00)^*, 10^*1\}$

From Automata to Domains

- A - finite alphabet $\Rightarrow A^\infty = A^* \cup A^\omega$ word monoid over A.
  - A^∞ partial order in *prefix order:*
    $s \leq t$ iff $su = t$ for some word u.
  - $(A^\infty, \leq)$ chain complete poset
- $L_A = \{10^*, 10^*1\} \subseteq \{0,1\}^\infty$
- Better map: $A_0 = A \cup \{\checkmark\}$; $s \mapsto s\checkmark : L_A \to A_0^\infty$.

  Differentiates
  $\downarrow L_A = \downarrow\{10^n1\checkmark \mid n \geq 0\} \cup \downarrow\{10^n\checkmark\} \cup \downarrow 10^\omega$ from
  $\downarrow L_B = \downarrow\{10^n1\checkmark\} \cup \downarrow\{1(0^{2n})\checkmark \mid n \geq 0\} \cup \downarrow 10^\omega$.
- $L_A \mapsto \downarrow\{s\checkmark \mid s \in L_A\}$ is one-to-one on regular languages.
  - Define $s \cdot t = \begin{cases} s_1 t & \text{if } s = s_1\checkmark, s_1 \in A^*, \\ s & \text{otherwise.} \end{cases}$

From Automata to Domains

- A - finite alphabet $\Rightarrow A^{\infty} = A^{*} \cup A^{\omega}$ word monoid over A.
  - A^{∞} partial order in *prefix order:*
    $s \leq t$ iff $su = t$ for some word u.
  - $(A^{\infty}, \leq)$ chain complete poset
- $L_A = \{10^{*}, 10^{*}1\} \subseteq \{0,1\}^{\infty}$
- Better map: $A_0 = A \cup \{\checkmark\}$; $s \mapsto s\checkmark : L_A \to A_0^{\infty}$.

  Differentiates
  $\downarrow L_A = \downarrow\{10^n1\checkmark \mid n \geq 0\} \cup \downarrow\{10^n\checkmark\} \cup \downarrow 10^{\omega}$ from
  $\downarrow L_B = \downarrow\{10^n1\checkmark\} \cup \downarrow\{1(0^{2n})\checkmark \mid n \geq 0\} \cup \downarrow 10^{\omega}$.
- $L_A \mapsto \downarrow\{s\checkmark \mid s \in L_A\}$ is one-to-one on regular languages.
  - Define $s \cdot t = \begin{cases} s_1 t & \text{if } s = s_1\checkmark, s_1 \in A^{*}, \\ s & \text{otherwise.} \end{cases}$

  Supports all regular language constructs in the model

Report for MFPS, LICS and CSF

Last week, three leading theory conferences met at Tulane:

- MFPS - Mathematical Foundations of Programming Semantics
- LICS - Logic in Computer Science
- CSF - Computer Security Foundations Symposium
  - Three leading theory conferences.
  - Attracted 250 participants from US, Europe and Far East
  - Deliberately co-located to encourage interaction

Here are results from a selection of presentations:

Report from MFPS

- System T: simply typed λ-calculus $+\ \mathbb{N}$:

$$\begin{array}{rcl} \mathcal{T} & ::= & \mathbb{N} \mid \mathcal{T} \times \mathcal{T} \mid \mathcal{T} \longrightarrow \mathcal{T} \\ P & ::= & x \mid MN \mid \lambda x : \mathcal{T}.M \end{array}$$

Also includes a *recursor* R:

$$\begin{array}{rcl} R\,0\,u\,v & \to & u \\ R\,(S\,t)\,u\,v & \to & v\,(R\,t\,u\,v)\,t \end{array}$$

Report from MFPS

- System T: simply typed λ-calculus $+ \mathbb{N}$:

$$\begin{aligned} \mathcal{T} &::= \mathbb{N} \mid \mathcal{T} \times \mathcal{T} \mid \mathcal{T} \longrightarrow \mathcal{T} \\ P &::= x \mid MN \mid \lambda x : \mathcal{T}.M \end{aligned}$$

  - Devised by Gödel to prove relative consistency of arithmetic
  - Simplest typed programming language

Report from MFPS

- System T: simply typed λ-calculus $+ \mathbb{N}$:

$$\begin{array}{rcl} \mathcal{T} & ::= & \mathbb{N} \mid \mathcal{T} \times \mathcal{T} \mid \mathcal{T} \longrightarrow \mathcal{T} \\ P & ::= & x \mid MN \mid \lambda x : \mathcal{T}.M \end{array}$$

- At MFPS, Martín Escardó (Birmingham) proved the following:

  The functions $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ denotable by terms of Gödel's System T are continuous, and the functions $f : 2^{\mathbb{N}} \to \mathbb{N}$ denotable by terms of Gödel's System T are uniformly continuous *using Agda to do the proof!*

  - Agda proof assistant: Interactive system for writing and checking proofs; based on intuitionistic type theory, a foundational system for constructive mathematics developed by the Swedish logician Per Martin-Löf.

Report from LICS

- At LICS, Prakash Panangaden (McGill) used duality theory to explain Brzozowski's Algorithm (1964):

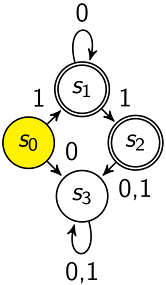
Input: DFA – $M = (S, A, s_0, F, \delta)$

  - Reverse transitions, interchange initial and final states
  - Determinize the result
  - Take the reachable states
  - Repeat

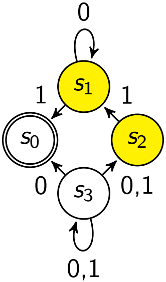
Result: The minimal DFA recognizing the same language!

- Joint work by Filippo Bonchi, Marcello Bonsangue, Helle Hvid Hansen, Prakash Panangaden, Jan Rutten and Alexandra Silva.

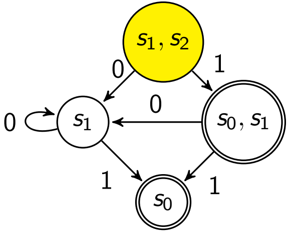
Brzozowski's Algorithm



$L_A = 10^* + 10^*1$

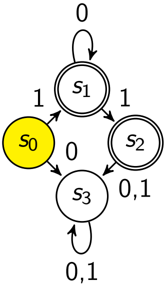


$0^*1 + 10^*1$

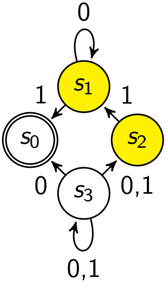


$0^*1 + 10^*1$

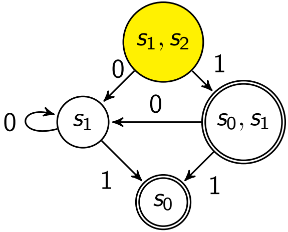
Brzozowski's Algorithm



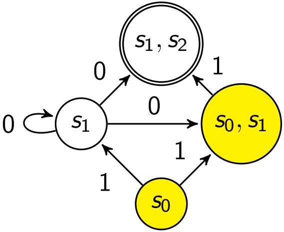
$L_A = 10^* + 10^*1$



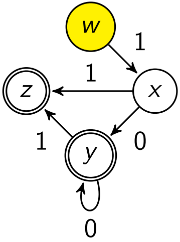
$0^*1 + 10^*1$



$0^*1 + 10^*1$



$10^* + 10^*1$



$10^* + 10^*1$

Brzozowski's Algorithm (cont'd)

- Why does this work?
  - Paths in the dual automaton are *backtracking* from the final states toward the initial state.
  - *Reachability* assures paths go all the way back to the initial state.
    - Also assures all states in the dual automaton are *observable*
- Let's make this more precise

Reachability

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"] \\
A^* \arrow[d, "\delta_A"] \\
(A^*)^A
\end{tikzcd}
$$

$\epsilon(*) = \epsilon$ and $\delta_A(w)\colon A \to A^*$ by $\delta_A(w)(a) = wa$.

Reachability

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"] \arrow[dr, "\iota"] & \\
A^* \arrow[r, "r"] \arrow[d, "\delta_A"] & S \arrow[d, "\delta"] \\
(A^*)^A \arrow[r, "r^A"'] & S^A
\end{tikzcd}$$

$$\iota(*) = s_0 \text{ and } r(w) = \begin{cases} s_0 & \text{if } w = \epsilon \\ \delta(r(u), a) & \text{if } w = ua. \end{cases}$$

$r^A \colon (A^*)^A \to S^A$ by $r^A(f) = r \circ f$.

Note: $r^A \circ \delta_A = \delta \circ r$, as is easily checked.

Reachability

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"] \arrow[dr, "\iota"] & \\
A^* \arrow[r, "r"] \arrow[d, "\delta_A"] & S \arrow[d, "\delta"] \\
(A^*)^A \arrow[r, "r^A"'] & S^A
\end{tikzcd}
$$

$\iota(*) = s_0$ and $r(w) = \begin{cases} s_0 & \text{if } w = \epsilon \\ \delta(r(u), a) & \text{if } w = ua. \end{cases}$

$r^A \colon (A^*)^A \to S^A$ by $r^A(f) = r \circ f$.

Note: $r^A \circ \delta_A = \delta \circ r$, as is easily checked.

An automaton is *reachable* if every state is reachable.

So, M is reachable iff r is a surjection.

Observability

$$\begin{tikzcd}
 & 2 \\
S \arrow[ur, "f"] \arrow[r, "o"] \arrow[d, "\delta"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
S^A \arrow[r, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

$2 = \{0,1\}$, $f(s) = 1$ iff $s \in F$,

$o(s) = \{w \mid (\exists s' \in S)\ s \xrightarrow{w} s'\}$,

$$\epsilon?(L) = \begin{cases} 1 & \text{if } \epsilon \in L, \\ 0 & \text{otherwise.} \end{cases}$$

$\beta(L)(a) = \{w \mid aw \in L\}$.

Observability

```tikzcd
 & & 2 \\
S \arrow[urr, "f"] \arrow[rr, "o"] \arrow[d, "\delta"'] & & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
S^A \arrow[rr, "o^A"'] & & (2^{A^*})^A
\end{tikzcd}
```

$$2 = \{0, 1\},\ f(s) = 1 \text{ iff } s \in F,$$

$$o(s) = \{w \mid (\exists s' \in S)\ s \xrightarrow{w} s'\},$$

$$\epsilon?(L) = \begin{cases} 1 & \text{if } \epsilon \in L, \\ 0 & \text{otherwise.} \end{cases}$$

$$\beta(L)(a) = \{w \mid aw \in L\}.$$

An automaton is *observable* if distinct states generate distinct languages.

So, M is observable iff o is an injection.

Reachability and Observability

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"] \arrow[dr, "\iota"] & & 2 \\
A^* \arrow[r, "r"] \arrow[d, "\delta_A"] & S \arrow[r, "o"] \arrow[d, "\delta"] \arrow[ur, "f"] & 2^{A^*} \arrow[u, "\epsilon?"] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, "r^A"'] & S^A \arrow[r, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

Determinization is crucial for the following:

Theorem: A deterministic automaton M accepting L is reachable iff $rev(M)$ is observable accepting $rev(L)$.

Corollary: M is minimal iff M is reachable and observable, iff r is a surjection and o is an injection.

What does all this have to do with Critical Infrastructure Protection?

- Cyberinfrastructure relies on computational components for proper functioning:

  Military, financial, transportation, utilities, information....

  All require secure command and control mechanisms.
- Show how to use *formal methods* to analyze and prove security protocols are correct.
  - Utilize process calculi (some probabilistic) and their models (usually domain-theoretic)
  - Reasoning is intricate and proofs are arcane and involved
  - Often aided by automated tools
- We'll discuss two examples:
  1. Another paper from the meeting, this time from CSF
  2. Example of *banking* using security automata of Schneider, modeled using *CSP-OZ* by Basin, Olderog and Sevinc

From CSF

At the Computer Security Foundations Symposium, Benjamin Pierce gave a talk about his new DARPA project, Crash/SAFE. Here's a rundown of SAFE:

- Clean-slate design of entire system stack:
  - Hardware
  - System software
  - Programming languages
- Support for critical security primitives at all levels (from hardware up)
  - Memory safety (avoid security breaches, e.g., buffer overflows, dangling pointers, etc.)
  - Strong dynamic typing
  - Information flow control (IFC) and access control
- Verification of key mechanisms deeply integrated into design process

From CSF

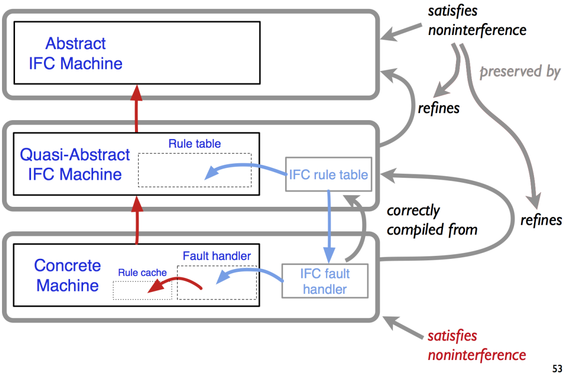
New hardware: Effective use of resources on security; remove compiler from TCB (partially); make security mechanisms available for writing low-level systems code

OS Level: "Zero-kernel OS"; no overprivileged component

Application level: Breeze – mostly functional, security-oriented PL; dynamic type- and security checks; every value annotated with an IFC label; labels public

A *crucial aspect* of the project is the use of formal methods to prove code correct using the Coq proof assistant.

Hardware Design



53

Noninterference: A machine with observation $(\Omega, |\cdot|, \sim)$ satisfies *termination-insensitive noninterference* if for any observer $o \in \Omega$ and any pair of indistinguishable initial data $\iota_1 \sim_o \iota_2$ and pair of executions $\mathit{Init}(\iota_1) \xrightarrow{t_1} *$ and $\mathit{Init}(\iota_2) \xrightarrow{t_2} *$, $|t_1|_o \sim_o |t_2|_o$.

Secure Banking System

- Bank has *Users* who have *Accounts* and who can *Check Balances* and *Transfer Funds* between accounts
- Specifications:
  - Users and Accounts specified by the sets:

    $[UserId, AccID,\ PIN,\ TN], \quad Val : \mathbb{PZ}, \quad Sum : \mathbb{PN}$
  - Support operations:
    - *login*
    - *logout*
    - *Check balance*
    - *Request transfer*
    - *Execute transfer*
    - *Abort*

Secure Banking System

- Bank has *Users* who have *Accounts* and who can *Check Balances* and *Transfer Funds* between accounts
- Specifications:
  - Users and Accounts specified by the sets:

    $[UserId, AccID,\ PIN,\ TN], \quad Val : \mathbb{P}\mathbb{Z}, \quad Sum : \mathbb{P}\mathbb{N}$
  - Support operations:
    - *login*
    - *logout*
    - *Check balance*
    - *Request transfer*
    - *Execute transfer*
    - *Abort*
- Approach (work of Basin, Olderog & Sevinc):
  - Define components as security automata
  - Translate components into *CSP* (communications) + *OZ* (data).
  - Prove security using *CSP* models.

Security Automata

- $A = (Q, S, I, \delta)$ where:
  Q – countable set of *states*
  $S \subseteq Q$ – *start states*
  I – countable set of *input symbols*
  $\delta \colon Q \times I \longrightarrow 2^Q$ *transition function*
- First devised by F. Schneider; variant of Büchi automata.

Security Automata

- $A = (Q, S, I, \delta)$ where:
  Q – countable set of *states*
  $S \subseteq Q$ – *start states*
  I – countable set of *input symbols*
  $\delta \colon Q \times I \longrightarrow 2^Q$ *transition function*
- Analysis proceeds by
  1) Writing processes as security automata
  2) Translating security automata into a *specification language*
  3) Proving correctness using a denotational model for specification language.

Security Automata

- $A = (Q, S, I, \delta)$ where:
  Q – countable set of *states*
  $S \subseteq Q$ – *start states*
  I – countable set of *input symbols*
  $\delta \colon Q \times I \longrightarrow 2^Q$ *transition function*
- Specification language:
  - Combination of *CSP* and *Z*:
    CSP – process calculus based on communication events
    Z – based on set theory and predicate logic
    – used for data, state spaces and state transformations
  - Write specifications in *CSP-OZ* and prove they are correct
    Translate everything into *CSP*
    For finite data can use FDR tool to prove correctness

CSP Basics

CSP is a process calculus in which processes are specified by the following BNF:

$$\begin{array}{rcl} P & ::= & \mathit{STOP} \mid \mathit{SKIP} \mid a \rightarrow P \mid P \sqcap Q \\ & & \mid P \Box Q \mid P \parallel_A Q \mid P \setminus A \mid X \end{array}$$

where $a \in \mathrm{Act}$, the set of (communication) actions, $A \subseteq \mathrm{Act}$, and X is a process variable.

CSP Basics

CSP is a process calculus in which processes are specified by the following BNF:

$$\begin{array}{rcl} P & ::= & STOP \mid SKIP \mid a \to P \mid P \sqcap Q \\ & & \mid P \Box Q \mid P \parallel_A Q \mid P \setminus A \mid X \end{array}$$

where $a \in \mathrm{Act}$, the set of (communication) actions, $A \subseteq \mathrm{Act}$, and X is a process variable.

Some examples:

- $(a \to P) \sqcap (b \to P)$ vs $(a \to P) \Box (b \to P)$:

$$\begin{array}{l} tr(a \to P) = \{\epsilon, a\} \cup \ \{a.t \mid t \in tr(P)\} \\ tr((a \to P) \sqcap (b \to P)) = \{\epsilon, a, b\} \cup \{a.t \mid t \in tr(P)\} \\ \qquad\qquad\qquad\qquad\qquad\qquad \cup \{b.t \mid t \in tr(P)\} \\ \qquad\qquad\qquad\qquad\quad = tr((a \to P) \Box (b \to P)) \end{array}$$

- $P \sqsubseteq_T Q$ iff $tr(P) \supseteq tr(Q)$.

CSP Basics

CSP is a process calculus in which processes are specified by the following BNF:

$$\begin{array}{rcl} P & ::= & STOP \mid SKIP \mid a \to P \mid P \sqcap Q \\ & & \mid P \Box Q \mid P \parallel_A Q \mid P \setminus A \mid X \end{array}$$

where $a \in \mathrm{Act}$, the set of (communication) actions, $A \subseteq \mathrm{Act}$, and X is a process variable.

Some examples:

- Need stronger semantics – *Failures:*

  $(a \to P) \sqcap (b \to P)$ can *refuse* a and b on the first step, but $(a \to P) \Box (b \to P)$ cannot.

  $Fail(P) = \{(t, A) \mid t \in tr(P)$ & P can refuse $a \in A$ after $t\}$
- $P \sqsubseteq_F Q$ iff $Fail(P) \supseteq Fail(Q)$.

CSP Basics

CSP is a process calculus in which processes are specified by the following BNF:

$$\begin{array}{rcl} P & ::= & \mathit{STOP} \mid \mathit{SKIP} \mid a \to P \mid P \sqcap Q \\ & & \mid P \Box Q \mid P \parallel_A Q \mid P \setminus A \mid X \end{array}$$

where $a \in \mathrm{Act}$, the set of (communication) actions, $A \subseteq \mathrm{Act}$, and X is a process variable.

- Processes can also name *channels:*

  $c.t? \to P(t)$: process that listens on channel c and when receiving an input, then acts like $P(t)$.

  $c.t! \to P$: process that sends output t on channel c and then acts like P.

Insecure Bank

- The Bank

$B ::= (login \rightarrow (Bal \,\Box\, TranReq \,\Box\, logout)) \parallel_C ExecTran$

$C = \{ExecTran\}$

$$
\begin{aligned}
login &::= \Box_{u \in uid}\; c_{login}.u? \rightarrow SKIP \\
Bal &::= \Box_{a \in AcctId}\; c_{Bal}.a? \rightarrow c_{Bal}.s_a! \rightarrow SKIP \\
TranReq &::= c_{TranReq}.a_1?.a_2?.s? \rightarrow c_{ExecTran}.a_1!.a_2!.s! \rightarrow SKIP \\
ExecTran &::= c_{ExecTran}.a_1?.a_2?.s \rightarrow \\
&\qquad\qquad (s_{a_1} := a_1 - s) \rightarrow (s_{a_2} := a_2 + s) \rightarrow SKIP \\
logout &::= c_{logout}.bye? \rightarrow STOP
\end{aligned}
$$

Insecure Bank

- The Bank

$$B ::= (login \rightarrow (Bal \;\square\; TranReq \;\square\; logout)) \parallel_C ExecTran$$
$$C = \{ExecTran\}$$

$$\begin{aligned}
login &::= \square_{u \in uid}\; c_{login}.u? \rightarrow SKIP \\
Bal &::= \square_{a \in AcctId}\; c_{Bal}.a? \rightarrow c_{Bal}.s_a! \rightarrow SKIP \\
TranReq &::= c_{TranReq}.a_1?.a_2?.s? \rightarrow c_{ExecTran}.a_1!.a_2!.s! \rightarrow SKIP \\
ExecTran &::= c_{ExecTran}.a_1?.a_2?.s \rightarrow \\
&\qquad (s_{a_1} := a_1 - s) \rightarrow (s_{a_2} := a_2 + s) \rightarrow SKIP \\
logout &::= c_{logout}.bye? \rightarrow STOP
\end{aligned}$$

- A User $\quad U ::= (login \rightarrow (Bal \;\sqcap\; TranReq \;\sqcap\; logout))$

$$\begin{aligned}
login &::= c_{login}.u! \rightarrow SKIP \\
Bal &::= c_{Bal}.a! \rightarrow c_{Bal}.s? \rightarrow SKIP \\
TranReq &::= c_{TranReq}.a_1!.a_2!.s! \rightarrow SKIP \\
logout &::= c_{logout}.bye! \rightarrow STOP
\end{aligned}$$

Insecure Bank

$InSecBank ::= U \parallel_A B, \quad A = \{login, Bal, TranReq, logout\}$

- The Bank
  $B ::= (login \rightarrow (Bal \,\Box\, TranReq \,\Box\, logout)) \parallel_C ExecTran$
- A User $\quad U ::= (login \rightarrow (Bal \sqcap TranReq \sqcap logout))$

Securing the Bank

Add a secure *SecComp* and run in parallel with *InSecBank*:

$SecBank ::= InSecBank \parallel_B SecComp$

$B = \{login, Bal, TranReq, Abort, logout, ChkPin, ChkTranReq\}$

SecComp:

- $login ::= c_{login}.u?.p? \rightarrow ChkPin$
- $ChkPin ::= c_{ChkPin}.u?.p? \rightarrow$
  $$((u, p) \in Valid \rightarrow SKIP) \,\Box\, ((u, p) \notin Valid \rightarrow Abort)$$
- $ChkAcctId ::= c_{ChkAcctId}.u?.a? \rightarrow \cdots$
- $ChkTranReq ::= c_{TranReq}.a?.n? \rightarrow \cdots$

Properties of Secure Bank

$SecBank ::= InSecBank \parallel_B SecComp$

$B = \{login, Bal, TranReq, Abort, logout, ChkPin, ChkTranReq\}$

(*) No *ExecTran* takes place before a successful *ChkTranReq*

can be shown using:

$$
\begin{array}{rcl}
P_0 & ::= & ChkTranReq.T \rightarrow P_1 \\
 & & \Box\,(\Box_{a \in D}\, a \rightarrow P_0) \\
P_1 & ::= & ExecTran.a_1?.a_2?.s? \rightarrow P_0 \\
 & & \Box\,(\Box_{a \in D}\, a \rightarrow P_0)
\end{array}
$$

$D = \{login, Bal, TranReq, Abort, logout, ChkPin, ChkTranReq\}$

Modeling Probability

- Standard model in domains is *Probabilistic Power Domain*

  $\mathit{Prob}(D)$ – Probability measures with
  $\mu \leq \nu$ iff $\mu(U) \leq \nu(U)$ ($\forall U$ open)

- Not well understood

  Structure is hard to analyze
  Adds complications of probabilistic order to order on D
  $d \mapsto \delta_d \colon D \hookrightarrow \mathit{Prob}(D)$ order-embedding

- Doesn't "play well with other monads".

Alternative: Random Variable model:

- Restricts order on probability to domain of random variable
- Separates orders, simplifies construction
- Standard approach in probability theory

Continuous Random Variables

- $f \colon (X, \mu) \to (Y, \Omega)$ *random variable*
  - (X, μ) probability space,
  - (Y, Ω) measure space
  - f is *measurable:* $f^{-1}(A)$ measurable $(\forall A \in \Omega)$
  - *Continuous* if X and Y topological spaces, f continuous and X, Y endowed with Borel σ-algebras.

Continuous Random Variables

- $f\colon (X,\mu) \to (Y,\Omega)$ *random variable*
- Assume X, Y *domains* endowed with *Scott topology:*

  U Scott open iff $U = \uparrow U = \{d \in D \mid (\exists u \in U)\, u \leq d\}$ and
  $\sup C \in U \;\Rightarrow\; U \cap C \neq \emptyset$, $\forall$ chains C

  BCD – Bounded complete domains & Scott continuous maps
  $(D, \leq)$ has sups of chains & all non-empty sets have greatest lower bounds

Continuous Random Variables

- $f\colon (X,\mu) \to (Y,\Omega)$ *random variable*
- Assume X, Y *domains* endowed with *Scott topology:*

  $CRV(X,Y) = \{(\mu, f) \mid \mu \in Prob(X), f\colon \operatorname{supp}\mu \to Y\}$

  $\operatorname{supp}\mu = \bigcap\{C \subseteq X \mid \mu(C) = 1 \ \&\ C \text{ closed}\}$.

Continuous Random Variables

- $f\colon (X, \mu) \to (Y, \Omega)$ *random variable*
- Assume X, Y *domains* endowed with *Scott topology:*

  $CRV(X, Y) = \{(\mu, f) \mid \mu \in Prob(X), f\colon \operatorname{supp}\mu \to Y\}$
- $\mathcal{C}$ - Cantor tree

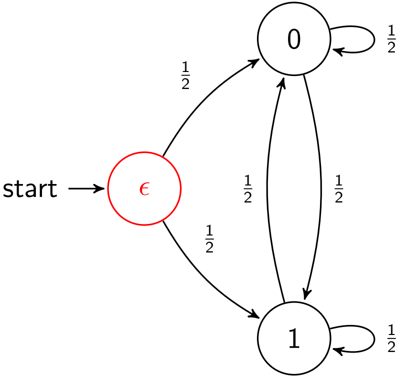
  Goubault-Larrecq & Varacca, LICS 2011:
  BCD closed under

$$\begin{array}{rcl} \Theta RV(\mathcal{C}, P) & = & \{(\mu, f) \in CRV(\mathcal{C}, P) \mid \mu \textit{ thin}\} \\ (\mu, f) \leq (\nu, g) & \text{iff} & \pi_{\operatorname{supp}\mu}(\nu) = \mu \;\&\; f \circ \pi_{\operatorname{supp}\mu} \leq g \end{array}$$

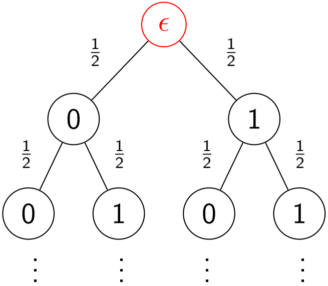
- *Goal:* Understand $\Theta RV(\mathcal{C}, P)$ construction for $P \in \mathrm{BCD}$

Motivating the Order - Automata

- A *(generative) probabilistic automaton* A has a finite set S of states, a start state $s_0 \in S$, a finite set of actions, Act, and a transition relation $\longrightarrow \subseteq S \times \mathrm{Prob}(Act \times S)$.
- Here's a simple example with one action, *flip*:



Unfolding the automaton:



Taking place on $\mathcal{C}$

Motivating the Order - Trace Distributions

- Typically, such automata are modeled by their *trace distributions* $\mu_i \in Prob(\mathcal{C})$:

  $\mu_0 = \delta_\epsilon$

  $\mu_1 = \frac{1}{2}\delta_0 + \frac{1}{2}\delta_1$

  $\mu_2 = \frac{1}{4}\delta_{00} + \frac{1}{4}\delta_{01} + \frac{1}{4}\delta_{10} + \frac{1}{4}\delta_{11}$

  $\vdots$

- Stripping away the probabilities, we have the following sets $X_i \subseteq \mathcal{C}$ on which the μ_i are *concentrated:*

  μ_0 concentrated on $X_0 = \{\epsilon\}$
  μ_1 concentrated on $X_1 = \{0, 1\}$
  μ_2 concentrated on $X_2 = \{00, 01, 10, 11\}$
  $\vdots$
  μ_∞ concentrated on $X_\infty = 2^\omega$

Motivating the Order - Trace Distributions

- Stripping away the probabilities, we have the following sets $X_i \subseteq \mathcal{C}$ on which the μ_i are *concentrated:*

  μ_0 concentrated on $X_0 = \{\epsilon\}$
  μ_1 concentrated on $X_1 = \{0, 1\}$
  μ_2 concentrated on $X_2 = \{00, 01, 10, 11\}$
  $\vdots$
  μ_∞ concentrated on $X_\infty = 2^\omega$

- Notice that the X_ns are *antichains*, and

  $X_0 \sqsubseteq_C X_1 \sqsubseteq_C X_2 \sqsubseteq_C \cdots \sqsubseteq_C X_\infty$, where

$$\begin{aligned} X \sqsubseteq_C Y \quad &\Leftrightarrow \quad X \subseteq {\downarrow} Y \ \& \ Y \subseteq {\uparrow} X \\ &\Leftrightarrow \quad \pi_X(Y) = X \end{aligned}$$

The Underlying Structure - Domains and Trees

- $2 = \{0, 1\}$

  $2^\infty = 2^* \cup 2^\omega$ is a *domain* under the prefix order.
  2^* – the finite words

  2^∞ is *coherent*

  Compact in the *Lawson topology*

  Open sets: $U = \uparrow k \setminus \uparrow F, \quad k \in 2^*, F \subseteq 2^*$ finite

The Underlying Structure - Domains and Trees

- $2 = \{0, 1\}$

  $2^\infty = 2^* \cup 2^\omega$ is a *domain* under the prefix order.
- $AC(2^\infty) = (\{X \mid \text{Lawson-compact antichain}\}, \sqsubseteq_C)$

$$\begin{aligned} X \sqsubseteq_C Y \quad &\Leftrightarrow \quad X \subseteq {\downarrow}Y \;\&\; Y \subseteq {\uparrow}X \\ &\Leftrightarrow \quad \pi_X(Y) = X \end{aligned}$$

  Subdomain of $\mathcal{P}_C(2^\infty)$.

The Underlying Structure - Domains and Trees

- $2 = \{0, 1\}$

  $2^\infty = 2^* \cup 2^\omega$ is a *domain* under the prefix order.
- $AC(2^\infty) = (\{X \mid \text{Lawson-compact antichain}\}, \sqsubseteq_C)$
- **Theorem:** $AC(2^\infty)$ is a bounded complete domain: all nonempty subsets have infima.

  $$(\emptyset \neq \mathcal{F} \subseteq AC(2^\infty)) \;\Rightarrow\; \inf \mathcal{F} = \mathrm{Max}(\bigcap_{X \in \mathcal{F}} {\downarrow} X)$$

The Underlying Structure - Domains and Trees

- $2 = \{0, 1\}$

  $2^\infty = 2^* \cup 2^\omega$ is a *domain* under the prefix order.
- $AC(2^\infty) = (\{X \mid \text{Lawson-compact antichain}\}, \sqsubseteq_C)$
- **Theorem:** $AC(2^\infty)$ is a bounded complete domain: all nonempty subsets have infima.
  Moreover, given $\{X_n\}_{n\in\mathbb{N}} \subseteq AC(2^\infty)$ directed and $X \in AC(2^\infty)$, TAE:

  (i) $X = \sup_n X_n$

  (ii) $X = \lim_n X_n$ in the Vietoris topology on $\Gamma(2^\infty)$.
- In particular, *any* $X \in AC(2^\infty)$ satisfies

  $X = \sup_n \pi_n(X) = \lim_n \pi_n(X)$, where

  $\pi_n \colon 2^\infty \to 2^{\leq n}$ is the canonical retraction.

Thin Probability Measures

- $\mu \in \mathit{Prob}(2^\infty)$ is *thin* if $\mathsf{supp}_\Lambda\, \mu \in \mathit{AC}(2^\infty)$.

  Note: $\mathsf{supp}_\Lambda\, \mu$ is in the *Lawson* topology.

- Define $\mu \leq \nu$ iff $\pi_{\mathsf{supp}_\Lambda\, \mu}(\nu) = \mu$

  Agrees with usual domain order (*qua* valuations) / functional analysis order via cones.

  $\Theta \mathit{Prob}(2^\infty) = (\{\mu \in \mathit{Prob}(2^\infty) \mid \mu \text{ thin}\}, \leq)$.

Thin Probability Measures

- $\mu \in \mathit{Prob}(2^{\infty})$ is *thin* if $\mathrm{supp}_{\Lambda}\, \mu \in \mathit{AC}(2^{\infty})$.
- **Proposition:** $(\Theta\mathit{Prob}(2^{\infty}), \leq)$ is a bounded complete domain: all nonempty subsets have infima.

$$(\emptyset \neq \mathcal{M} \subseteq \Theta\mathit{Prob}(2^{\infty}) \;\Rightarrow\; \forall \nu \in \mathcal{M},$$

$$\inf \mathcal{M} = \pi_M(\nu), \quad M = \inf_{\mu \in \mathcal{M}} \mathrm{supp}_{\Lambda}\, \mu)$$

Thin Probability Measures

- $\mu \in Prob(2^\infty)$ is *thin* if $\mathrm{supp}_\Lambda\, \mu \in AC(2^\infty)$.
- **Proposition:** $(\Theta Prob(2^\infty), \leq)$ is a bounded complete domain: all nonempty subsets have infima.

  Moreover, given $\{\mu_n\}_{n\in\mathbb{N}} \subseteq \Theta Prob(2^\infty)$ chain and $\mu \in \Theta Prob(2^\infty)$, TAE:

  (i) $\mu = \sup_n \mu_n$

  (ii) $\mu = \lim_n \mu_n$ in the weak *-topology on $\Theta Prob(2^\infty)$.
- In particular, *any* $\mu \in \Theta Prob(2^\infty)$ satisfies

  $\mu = \sup_n \pi_n(\mu) = \lim_n \pi_n(\mu)$, where

  $\pi_n \colon \mathcal{C} = 2^\infty \to 2^{\leq n} \equiv \mathcal{C}_n$ is the canonical retraction.

Adding Function Spaces

- $\mathcal{C}_n \equiv \pi_n(\mathcal{C}) \implies \mathcal{C}_n \overset{\iota_n}{\hookrightarrow} \mathcal{C} \xrightarrow{\pi_n}\!\!\!\!\rightarrow \mathcal{C}_n$
  $P \in \mathrm{BCD} \implies$

  $f \mapsto f \circ \pi_n \colon [\mathcal{C}_n \longrightarrow P] \hookrightarrow [\mathcal{C} \longrightarrow P]$ &

  $g \mapsto g \circ \iota_n \colon [\mathcal{C} \longrightarrow P] \twoheadrightarrow [\mathcal{C}_n \longrightarrow P]$.

Adding Function Spaces

- $\mathcal{C}_n \equiv \pi_n(\mathcal{C}) \implies \mathcal{C}_n \overset{\iota_n}{\hookrightarrow} \mathcal{C} \xrightarrow{\pi_n}\!\!\!\twoheadrightarrow \mathcal{C}_n$
  $P \in \mathrm{BCD} \implies$

  $f \mapsto f \circ \pi_n \colon [\mathcal{C}_n \longrightarrow P] \hookrightarrow [\mathcal{C} \longrightarrow P]$ &

  $g \mapsto g \circ \iota_n \colon [\mathcal{C} \longrightarrow P] \twoheadrightarrow [\mathcal{C}_n \longrightarrow P]$.
- $P \in \mathrm{BCD} \implies [\mathcal{C} \longrightarrow P] \in \mathrm{BCD}$:

$$[\mathcal{C} \longrightarrow P] \simeq \lim_n [\mathcal{C}_n \longrightarrow P] \simeq \lim_n P^{\mathcal{C}_n}.$$

Adding Function Spaces

- $P \in \mathrm{BCD} \implies [\mathcal{C} \longrightarrow P] \in \mathrm{BCD}$:

$$[\mathcal{C} \longrightarrow P] \simeq \lim_n [\mathcal{C}_n \longrightarrow P] \simeq \lim_n P^{\mathcal{C}_n}.$$

Defining the Model

- $\Theta Prob(\mathcal{C}) \times [\mathcal{C} \longrightarrow P] \in \mathrm{BCD}$ if $P \in \mathrm{BCD}$.

Adding Function Spaces

- $P \in \mathrm{BCD} \implies [\mathcal{C} \longrightarrow P] \in \mathrm{BCD}$:

  $$[\mathcal{C} \longrightarrow P] \simeq \lim_n [\mathcal{C}_n \longrightarrow P] \simeq \lim_n P^{\mathcal{C}_n}.$$

Defining the Model

- $\Theta\mathit{Prob}(\mathcal{C}) \times [\mathcal{C} \longrightarrow P] \in \mathrm{BCD}$ if $P \in \mathrm{BCD}$.
- $\Theta\mathit{RV}(2^\infty, P) = \{(\mu, f) \mid \mu \in \Theta\mathit{Prob}(A^\infty), f\colon \operatorname{supp}\mu \longrightarrow P\}$

  – retract of $\Theta\mathit{Prob}(\mathcal{C}) \times [\mathcal{C} \longrightarrow P]$:

  $(\mu, f) \mapsto (\pi_Y(\mu), f \circ \pi_Y)$ is the projection

  where $Y = \operatorname{supp}\mu$.

Possible Applications

- Allow *modular* construction to add probability to existing models:
  - Lynch, et al.'s Timed I/O Automata
  - *CSP* models
- Part of program to bring *information theory* to computational models
  - Have results about entropy and channel capacity using domain theory
  - Could apply to programs as channels
- Potential application to quantum information

The End

Thank You!