A progress report: DaViz and DaProof

Hans-Dieter A. Hiep

Master Student
(Vrije [Universiteit) van Amsterdam]

July 11, 2017

Outline of this talk

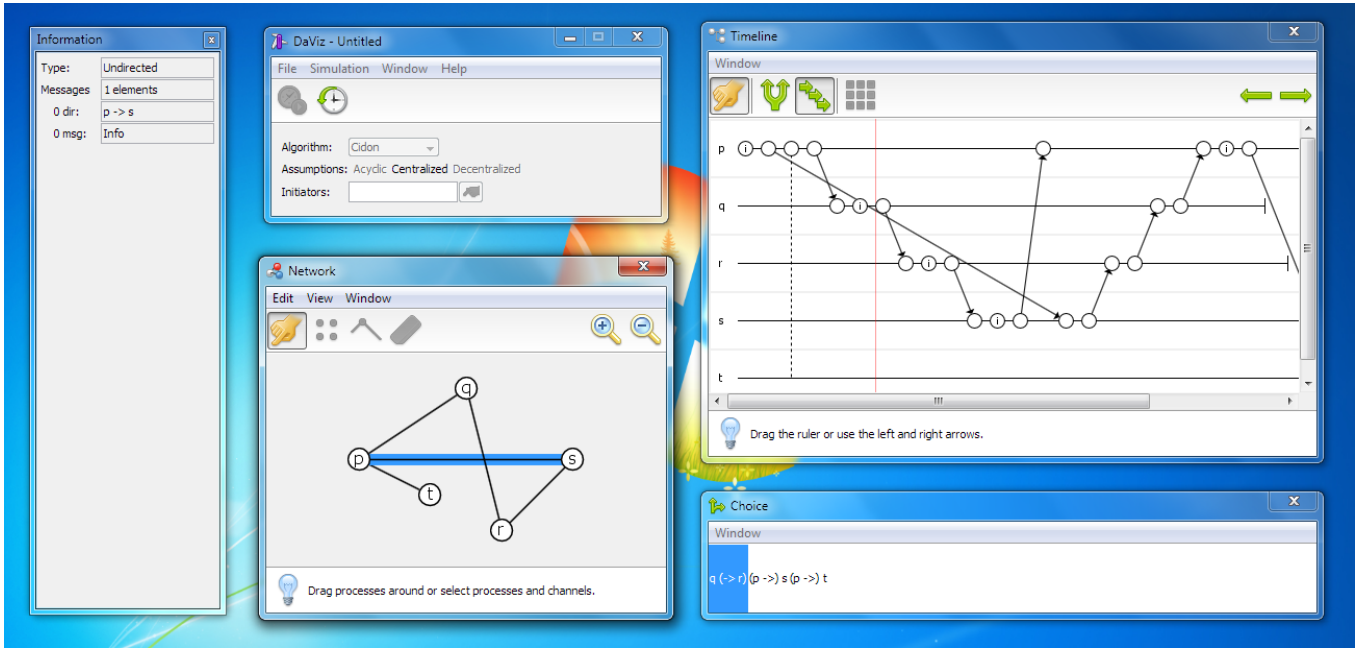
1. Part 1: DaViz
   - Motivation
   - Demonstration
2. Part 2: DaProof
   - Monoids
   - Traces
   - Corresponding events
   - Operational semantics
   - Outlook

Goals of this talk

1. Demonstrate DaViz
2. Explain free partially commutative monoids
3. Show DaProof's operational semantics

DaViz

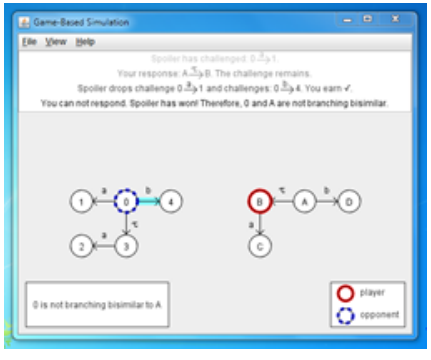
A project to design and develop a system that simulates distributed algorithms and visualizes them in a straight-forward and interactive manner, to promote experimentation.



Predecessor: Bisimulation Games

Bisimulation Games (supervised by Jeroen Keiren, April 2016)

- Motto: "playing games $\Rightarrow$ intuitive understanding"
- **Reuse** an implemention of *(branching) bisimulation* in Haskell
- Ported Haskell code to **Frege**, a Haskell-for-the-JVM
- Visualization and interaction implemented as a GUI
  - Using AWT/Swing framework



DaViz: a replicated design

DaViz (supervised by Wan Fokkink)

- Motto: "exploration by user $\Rightarrow$ intuitive understanding"
- Long-term goal: a tool for teaching
- Less tedious than to work out examples by hand
- Replicated design: Java for GUI and Haskell for simulation
- Implements $\sim$5 distributed algorithms

DaViz: demonstration

D
E
M
O

DaProof

DaProof (supervised by Alban Ponse)

- A study of the semantics of DaViz
- Motivated by two concerns:
  1. Useful decomposition of distributed algorithms
  2. Discussion regarding Cidon's distributed depth-first search
- Here given from the bottom up
- Formulations in Coq proof assistant

Outline

Motto:

abstract	concrete
equational specification	representation
algebra	realization
operational semantics	simulation

DaProof is abstract. DaViz is concrete.

Monoids

Definition

A *monoid* $\mathbb{M}(S, \cdot, 1)$ is:

$$\left.\begin{array}{ll} \text{set} & S \\ \text{binary operation} & \cdot : S \times S \to S \\ \text{unit} & 1 : S \end{array}\right\} \text{signature}$$

such that:

$$\left.\begin{array}{lll} \text{associativity} & \forall s, t, u \in S. & (s \cdot t) \cdot u = s \cdot (t \cdot u) \\ \text{identity} & \forall s \in S. & 1 \cdot s = s = s \cdot 1 \end{array}\right\} \text{equational spec.}$$

Any structure which admits these two axioms,
also admits all monoid theorems.

Free monoids

We now define monoids freely generated by a carrier set C.

Definition

Let $\mathscr{T}(C)$ be the smallest set:

- generated by C, i.e. $C \subseteq \mathscr{T}(C)$
- closed under binary operation $\cdot : \mathscr{T}(C) \times \mathscr{T}(C) \to \mathscr{T}(C)$
- closed under unit $1 : \mathscr{T}(C)$

Question: is this a monoid? No, axioms do not hold.

- E.g. $x \cdot 1 \neq x$ or $(x \cdot y) \cdot z \neq x \cdot (y \cdot z)$
- Since terms are freely generated by the signature

Free monoids

Definition

Let $\cong$ be the smallest congruence relation on $\mathscr{T}(C)$:

- that is associative $(x \cdot y) \cdot z \cong x \cdot (y \cdot z)$
- that respects identity $1 \cdot x \cong x \cong x \cdot 1$

(Recall: congurence is equivalence and respects algebraic structure)
We now identify all elements related by the equivalence,
using the following quotient:

Definition

Let C^* be the *free monoid* that is $\mathscr{T}(C)\backslash\cong$.

We call C a *generator* of C^*. What are the elements of C^*?

- Precisely all equivalence classes $[x] = \{y \in \mathscr{T}(C) \mid y \cong x\}$

Free monoids

Why **free** monoid?

- Informally, it is the simplest structure that captures the algebraic definition of monoid
- For every monoid $\mathbb{M}(S, \cdot, 1)$ there is a unique homomorphism from S^* to $\mathbb{M}(S, \cdot, 1)$
- Captures application on arbitrary monoid
- Equivalence of two elements in a free monoid implies equivalence in every monoid

How to decide equivalence?

- Find a unique representative $\hat{x}$ in $[x]$ for each equivalence class
- Find a normalization that maps every element in $[x]$ to $\hat{x}$

Free monoids

Can we do better than quotients?

- A concrete representation of free monoids
- That consists only of the representatives

Definition

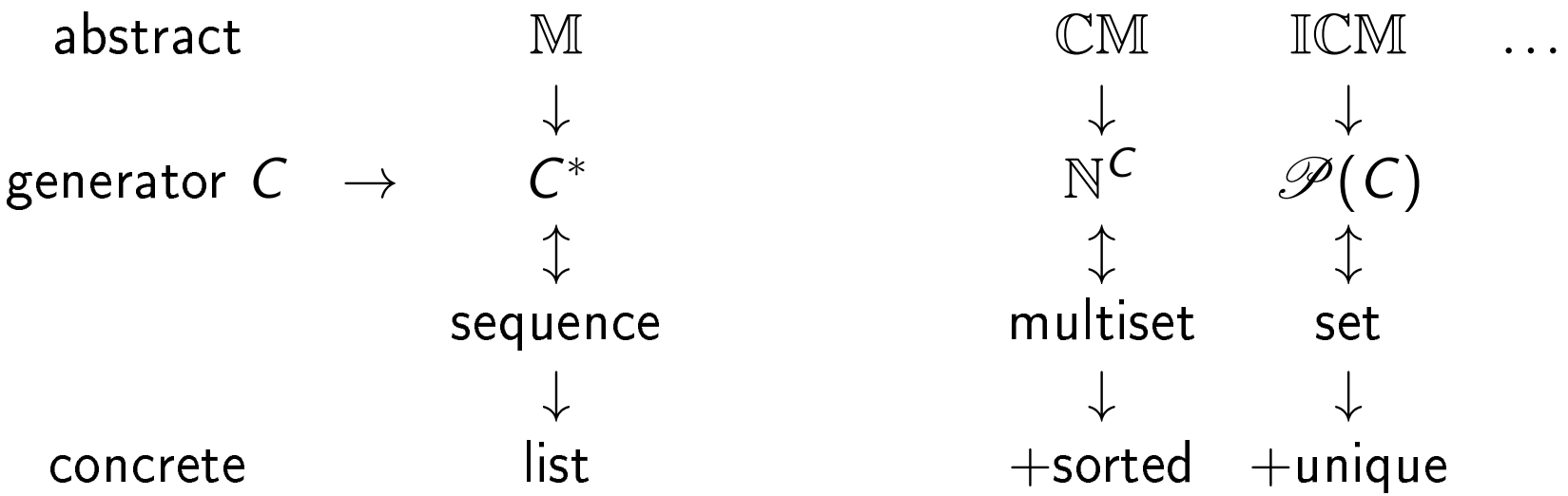
Let S^* be the smallest set such that:

- the empty list $\varepsilon \in S^*$
- given list $\Gamma \in S^*$ and $x \in S$, the prefix $x\Gamma \in S^*$

Questions:

- Representation of unit?
- Representation of elements of generator?
- Representation of binary operator?

Monoid overview

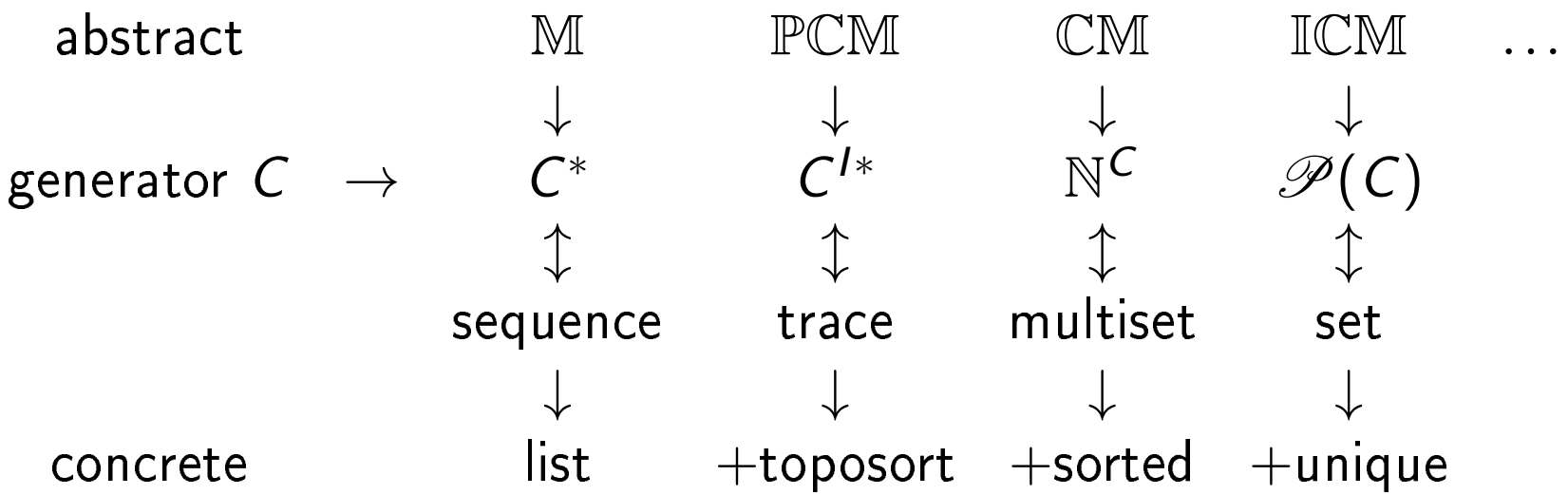


Monoid overview

Recap:

1. Given an algebraic definition (signature, equations)
2. Fix elements of the definition
3. Term language freely generated by signature
4. Congruence generated by specification
5. Quotient term language by congruence
6. Elect representatives of equivalence classes
7. Determine term language consisting only of representatives

Monoid overview



Partially commutative monoids

Definition

An *independence relation* $I \subseteq S \times S$ is an irreflexive, symmetric relation over S.

Two events that happen *concurrently* are independent.
We will now look at *computations*,
i.e. permutations of concurrent events.

Partially commutative monoids

1. Give an algebraic definition (signature, equations)

Definition

A *partially commutative monoid* $\mathbb{PCM}(S, I, \cdot, 1)$ is:

monoid	$\mathbb{M}(S, \cdot, 1)$
independence relation	$I : \mathscr{P}(S \times S)$

such that:

partial commutation	$\forall (s,t) \in I. \quad s \cdot t = t \cdot s$

Free partially commutative monoids

We now define PCM freely generated by a carrier set C.

2. Fix elements of the definition

Unlike before, we also fix $I \subset C \times C$.

3. Term language freely generated by signature

The set of terms $\mathscr{T}(C)$ remains the same.

4. Congruence generated by specification

Definition

Let $x \cong_I y$ be the smallest congruence relation on $\mathscr{T}(C)$:

- that is associative $(x \cdot y) \cdot z \cong_I x \cdot (y \cdot z)$
- that respects identity $1 \cdot x \cong_I x \cong_I x \cdot 1$
- that commutes $x \cdot y \cong_I y \cdot x$ if $(x, y) \in I$

Free partially commutative monoids

5. Quotient term language by congruence

Definition

Let C^{I*} be the *free partially commutative monoid* that is $\mathscr{T}(C)\backslash \cong_I$.

Question: What are the elements of C^{I*}? Equivalence classes (called *traces*).

6. Elect representatives of equivalence classes

We have two canonical choices:

- **lexicographic normal form**
- Foata normal form

Free partially commutative monoids

Let normalization assume a total order $<$ between elements in C.
Let $\ll$ be a lexicographic order induced by $<$.

Definition

Let $\varphi : C^* \to C^{I*}$ be the canonical injection, $\varphi : \hat{x} \mapsto [x]$.
We call $\hat{x}$ a linearization of $[x]$.

Definition

A lexicographic normal form of $t \in C^{I*}$ is the lexicographically smallest element $x \in C^*$ such that $\varphi(x) = t$.

Algorithm: swap adjacent elements $b \cdot a$ to $a \cdot b$ if $a < b$ and $(a, b) \in I$, until no longer possible.

Free partially commutative monoids

Example

Suppose we have the traces:

$$[b \quad a \quad d \quad c \quad b \quad a]$$

and

$$[a \quad b \quad d \quad a \quad c \quad b]$$

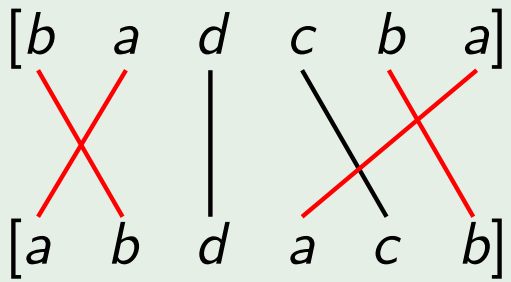
Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$, et cetera.
This trace is in lexicographic normal form.

Free partially commutative monoids

Example

Suppose we have the traces:

and



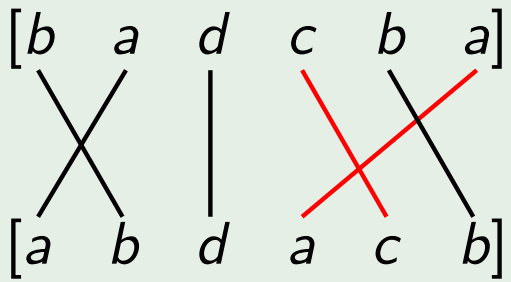
Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$, et cetera.
This trace is in lexicographic normal form.

Free partially commutative monoids

Example

Suppose we have the traces:

and



Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$, et cetera.
This trace is in lexicographic normal form.

Free partially commutative monoids

Example

Suppose we have the traces:

$$[b \quad a \quad d \quad c \quad b \quad a]$$

and

$$[a \quad b \quad d \quad a \quad c \quad b]$$

Let $(a, b) \in I$ and $(c, a) \in I$.
These are trace equivalent.
Let $a < b < c < d$, et cetera.
This trace is in lexicographic normal form.

Free partially commutative monoids

7. Determine term language consisting only of representatives

Concretely, FPCM correspond to lists that are topologically sorted.

- Free partially commutative monoids are called *trace monoids*.
- If I maximal subset of $C \times C$, then PCM is a CM.

Recap:

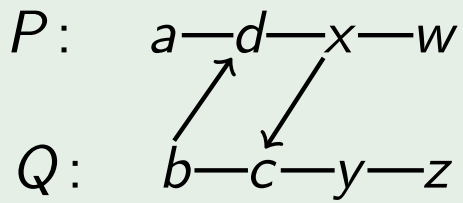
- FPCM is used to model concurrency
- FPCM captures all possible operations on a PCM, i.e. a unique homomorphism from S^{I*} to every $\mathbb{PCM}(S, I, \cdot, 1)$

Partially commutative monoid

An example of a partially commutative monoid.

Example

Consider the following computation:



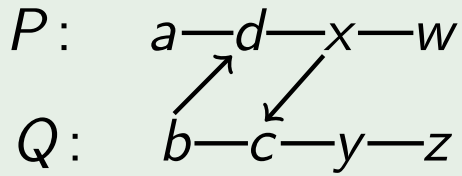
Example linearizations:

$$[a \quad b \quad d \quad x \quad c \quad w \quad y \quad z]$$

$$[a \quad b \quad d \quad x \quad w \quad c \quad y \quad z]$$

Lamport's logical clock

Example



Computing logical clock values (cf. vector clocks):

$$\left[\begin{pmatrix}1\\0\end{pmatrix} \quad \begin{pmatrix}0\\1\end{pmatrix} \quad \begin{pmatrix}2\\1\end{pmatrix} \quad \begin{pmatrix}3\\1\end{pmatrix} \quad \begin{pmatrix}3\\4\end{pmatrix} \quad \begin{pmatrix}4\\1\end{pmatrix} \quad \begin{pmatrix}3\\5\end{pmatrix} \quad \begin{pmatrix}3\\6\end{pmatrix}\right]$$
$$a \quad b \quad d \quad x \quad c \quad w \quad y \quad z$$

Every CM is PCM, so "message count" $(\mathbb{N}, +, 0)$ is PCM.
Also the "unreceived messages" monoid is PCM.

Corresponding messages

A process p sends a message m to another process q.

Definition

The set of events $\mathbb{E}$ is:

- a send event $(p \xrightarrow{m} q) \in \mathbb{E}$,
- a receive event $(q \xleftarrow{m} p) \in \mathbb{E}$.

We informally "know" the notion of corresponding events.
Assumptions:

Lossless channels Every send corresponds to precisely one receive.

Unbounded buffers The no. of messages in transit is unbounded.

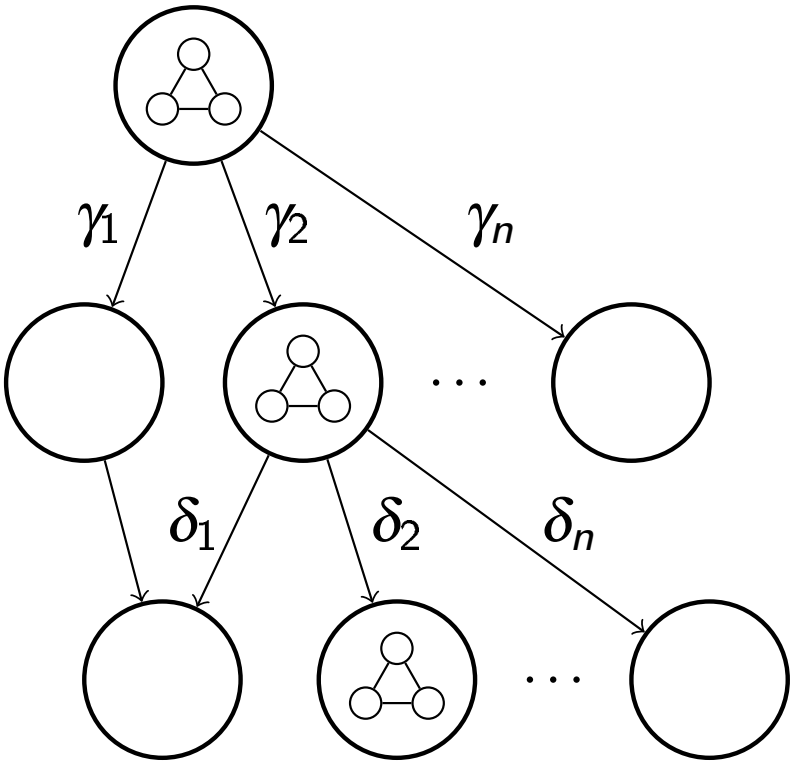
Queuing channels The channel acts as a FIFO queue.

Operational semantics: overview

Overall strategy:

1. Abstract over network:
   - nodes, process identifiers V
   - symmetric edge relation $E \subseteq V \times V$
   - connected
2. Abstract over algorithm:
   - message space μ,
   - state space σ
3. Realize algorithm by giving process description
   - Map from state space to local effect
4. Model executions by unraveling process descriptions

Operational semantics: big picture



- directed acyclic graph
- potentially infinite
- rooted in initial configuration
- each transition labeled by event
- path from root to node is a partially commutative monoid

Process descriptions: effects

Process descriptions are local. Applied uniformly to all nodes.

Definition

The set of *local effects* $\mathbb{L}$ is given by:

1. a *send* effect $S : \mu \times V \times \sigma \to \mathbb{L}$
2. a *receive* effect $R : (\mu \times V \to \sigma) \to \mathbb{L}$
3. an *internal* effect $I : \mathscr{P}(\sigma) \to \mathbb{L}$.

Instead of $I(\emptyset)$ we write $\checkmark$, a *termination* effect.

Definition

By $P : \sigma \to \mathbb{L}$ we denote a *process description*.

Process descriptions: configurations

Configurations are global. A configuration $C = (\kappa, \lambda)$ where

Channel labeling $\kappa : E \to \mu^*$ maps channel to traveling messages.

Process labeling $\lambda : V \to \sigma$ maps node to its current local state.

update local state
$$\lambda[p := s](x) \stackrel{\text{def}}{=} \begin{cases} s & x = p \\ \lambda(x) & \text{otherwise} \end{cases}$$

append message
$$\kappa[p \stackrel{m}{\to} q](x) \stackrel{\text{def}}{=} \begin{cases} \kappa(x) @ (m \# \varepsilon) & (p, q) = x \\ \kappa(x) & \text{otherwise} \end{cases}$$

test message
$$\kappa[q \stackrel{m}{\leftarrow} p]? \Leftrightarrow \exists xs!.\, \kappa(p, q) = m \# xs$$

$\curvearrowright$ receive message
$$\kappa[q \leftarrow p](x) \stackrel{\text{def}}{=} \begin{cases} xs & (p, q) = x \\ \kappa(x) & \text{otherwise} \end{cases}$$

Process descriptions: configurations

Configurations are global. A configuration $C = (\kappa, \lambda)$ where

Channel labeling $\kappa : E \to \mu^*$ maps channel to traveling messages.

Process labeling $\lambda : V \to \sigma$ maps node to its current local state.

update local state
$$\lambda[p := s](x) \stackrel{\text{def}}{=} \begin{cases} s & x = p \\ \lambda(x) & \text{otherwise} \end{cases}$$

append message
$$\kappa[p \stackrel{m}{\to} q](x) \stackrel{\text{def}}{=} \begin{cases} \kappa(x) @ (m \# \varepsilon) & (p, q) = x \\ \kappa(x) & \text{otherwise} \end{cases}$$

test message
$$\kappa[q \stackrel{m}{\leftarrow} p]? \Leftrightarrow \exists xs!.\, \kappa(p, q) = m \# xs$$

↷ receive message
$$\kappa[q \leftarrow p](x) \stackrel{\text{def}}{=} \begin{cases} xs & (p, q) = x \\ \kappa(x) & \text{otherwise} \end{cases}$$

Process descriptions: rules

$$\frac{\Delta \vdash (\kappa, \lambda) \quad P(\lambda(p)) = I(X) \quad s \in X}{\Delta \vdash (\kappa, \lambda[p := s])}\ I$$

$$\frac{\Delta \vdash (\kappa, \lambda) \quad P(\lambda(p)) = S(m, q, s)}{\Delta, (p \xrightarrow{m} q) \vdash (\kappa[p \xrightarrow{m} q], \lambda[p := s])}\ S$$

$$\frac{\Delta \vdash (\kappa, \lambda) \quad P(\lambda(p)) = R(f) \quad \kappa[p \xleftarrow{m} q]? \quad f(m, q) = s}{\Delta, (p \xleftarrow{m} q) \vdash (\kappa[p \leftarrow q], \lambda[p := s])}\ R$$

Process descriptions: rules

$$\frac{\Delta \vdash (\kappa, \lambda) \quad P(\lambda(p)) = I(X) \quad s \in X}{\Delta \vdash (\kappa, \lambda[p := s])}\ I$$

- Δ is a trace consisting of events $\mathbb{E}$
- non-deterministic choice of next state $s \in X$
- does not apply if node terminates, $P(\lambda(p)) = \checkmark = I(\emptyset)$
- updates local state $\lambda[p := s]$

Process descriptions: rules

$$\frac{\Delta \vdash (\kappa, \lambda) \quad P(\lambda(p)) = S(m, q, s)}{\Delta, (p \xrightarrow{m} q) \vdash (\kappa[p \xrightarrow{m} q], \lambda[p := s])}\ S$$

- record send event $p \xrightarrow{m} q$
- updates channel state $\kappa[p \xrightarrow{m} q]$, local state $\lambda[p := s]$

Process descriptions: rules

$$\frac{\Delta \vdash (\kappa, \lambda) \quad P(\lambda(p)) = R(f) \quad \kappa[p \overset{m}{\leftarrow} q]? \quad f(m, q) = s}{\Delta, (p \overset{m}{\leftarrow} q) \vdash (\kappa[p \leftarrow q], \lambda[p := s])}\ R$$

- condition: $\kappa[q \overset{m}{\leftarrow} p]?$ implies channel state $m\#xs$
- state $f(m, q) = s$ depends on received message and sender
- record receive event $p \overset{m}{\leftarrow} q$
- updates channel state $\kappa[p \leftarrow q]$, local state $\lambda[p := s]$

Example: Cidon's algorithm

Cidon's distributed depth-first search.

$$\mu \stackrel{\text{def}}{=} \{\text{Token}, \text{Info}\}$$

$$\sigma_{el} \stackrel{\text{def}}{=} \{\text{Received}(p), \text{Replied}(p), \text{Undefined}, \text{Initiator} \mid p \in V\}$$

$$\sigma \stackrel{\text{def}}{=} \{0,1\} \times \sigma_{el} \times (V \uplus \{u\}) \times 2^V \times 2^V$$

Process description is too large to show here
Algorithm adds axiom: initial configuration
Use the tool to explore executions

Outlook: proving properties

Observe that in the judgement $\Delta \vdash (\kappa, \lambda)$, κ is determined by Δ, viz. "unreceived messages" PCM.

With slight adaption of definitions, Δ also determines λ.

Let $\Delta \vdash \phi$ denote the judgement that ϕ holds for (κ, λ).

1. Convergence of system $\Rightarrow$ termination $\Rightarrow$ induction principle
2. Show properties $\Delta \vdash \phi$ by induction

Outlook: decomposing processes

Premise: algorithms can be decomposed à-la process algebra
Premise: composition of algorithms preserve correctness properties

Example

Given two process descriptions: $\langle \mu_1, \sigma_1, P_1 \rangle$ and $\langle \mu_2, \sigma_2, P_2 \rangle$.

- sequential composition $P_1; P_2$ by $\mu = \mu_1 \uplus \mu_2$ and $\sigma = \sigma_1 \cup \sigma_2$
  - does P_1 have any dangling messages?
- parallel composition $P_1 || P_2$ by $\mu = \mu_1 \uplus \mu_2$ and $\sigma = \sigma_1 \times \sigma_2$
  - two modes of termination: both or just one?

Outlook: decomposing processes

Example

Given a process description: $\langle \mu_1, \sigma_1, P_1 \rangle$

- let P_2 depend on any other process, then embedding P_1 into P_2
  - for example, basic algorithms & control algorithms
- let P_2 depend on terminal state σ_X, functional composition $P_1 \circ P_2$
  - which states are relevant? fan-in or fan-out?

Thank you for your attention. Questions?

Cidon's DDFS

$$P(\top, s, \bot, F, I) = I(\{\top, s, \lceil x \rceil, F \backslash x, I \backslash x\} \mid x \in F)$$
where $s \in \{\text{Initiator}, \text{Received}(p)\}$ and $F \neq \emptyset$

$$P(\top, s, \lceil x \rceil, F, I) = S(\text{Info}, i, (\top, s, \lceil x \rceil, F, I \backslash i))$$
where $s \in \{\text{Initiator}, \text{Received}(p)\}$ and $i \in I$ and $I \neq \emptyset$

$$P(\top, s, \lceil x \rceil, F, \emptyset) = S(\text{Token}, x, (\bot, s, \lceil x \rceil, F, \emptyset)$$
where $s \in \{\text{Initiator}, \text{Received}(p)\}$

$$P(\top, s, \bot, \emptyset, I) = S(\text{Info}, i, (\top, s, \bot, \emptyset, I \backslash i))$$
where $s = \text{Received}(p)$ and $i \in I$ and $I \neq \emptyset$

$$P(\top, s, \bot, \emptyset, \emptyset) = S(\text{Token}, p, (\bot, s', \bot, \emptyset, \emptyset))$$
where $s = \text{Received}(p)$ and $s' = \text{Replied}(p)$

$$P(\top, \text{Initiator}, \bot, \emptyset, \emptyset) = \checkmark$$

$$P(\bot, \text{Replied}(p), \bot, \emptyset, \emptyset) = \checkmark$$

Cidon's DDFS

$$P(\perp, \text{Undefined}, \perp, F, I) = R(f)$$

where
$$f(\text{Token}, p) = (\top, \text{Received}(p), \perp, F \setminus p, I \setminus p)$$
$$f(\text{Info}, p) = (\perp, \text{Undefined}, \perp, F \setminus p, I)$$

$$P(\perp, s, \lceil x \rceil, F, I) = R(f)$$

where $s \in \{\text{Received}(p), \text{Initiator}\}$

and
$$f(m, x) = (\top, s, \perp, F, I)$$
$$f(m, q) = (\perp, s, \perp, F \setminus q, I)$$

$$P(\perp, s, \perp, F, I) = R(f)$$

where $s \in \{\text{Received}(p), \text{Initiator}\}$

and
$$f(\text{Token}, q) = (\top, s, \perp, F, I)$$
$$f(\text{Info}, q) = (\perp, s, \perp, F \setminus q, I)$$