

# 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

*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.*

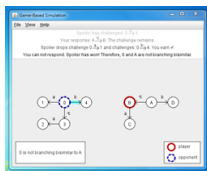
The screenshot displays the DaViz software interface, which is used for simulating and visualizing distributed algorithms. The interface is divided into several panels:

- Information:** A panel on the left showing simulation parameters: Type: Undirected, Messages: elements, 0 dir: p->s, and 0 msg: info.
- DaViz - Untitled:** A central panel for simulation configuration. It includes a menu (File, Simulation, Window, Help), a refresh button, and settings for Algorithm (Cdsn), Assumptions (Acyclic, Centralized, Decentralized), and an Initiators field.
- Network:** A panel showing a network topology with five nodes (0, 1, 2, 3, 4) and edges. A blue highlight is on the edge between nodes 2 and 3. A tooltip at the bottom says "Drag processes around or select processes and channels."
- Timeline:** A panel on the right showing a timeline visualization. The vertical axis is labeled with nodes p, q, r, s, t. The horizontal axis represents time. A red vertical line indicates the current simulation time. A tooltip at the bottom says "Drag the ruler or use the left and right arrows."
- Choice:** A panel at the bottom right showing a choice of actions:  $s \leftarrow r \mid p \rightarrow q \mid s \leftarrow p \rightarrow t$ .

# Predecessor: Bisimulation Games

## Bisimulation Games (supervised by Jeroen Keiren, April 2016)

- Motto: “playing games  $\Rightarrow$  intuitive understanding”
- **Reuse** an implementation 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

D  
E  
M  
O

## 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:

set	$S$	} signature
binary operation	$\cdot : S \times S \rightarrow S$	
unit	$1 : S$	

such that:

associativity	$\forall s, t, u \in S. (s \cdot t) \cdot u = s \cdot (t \cdot u)$	} equational spec.
identity	$\forall s \in S. 1 \cdot s = s = s \cdot 1$	

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

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

## Definition

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

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

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  $\mathcal{T}(C)$ :

- 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: congruence 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  $\mathcal{T}(C) \setminus \cong$ .

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

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

## 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

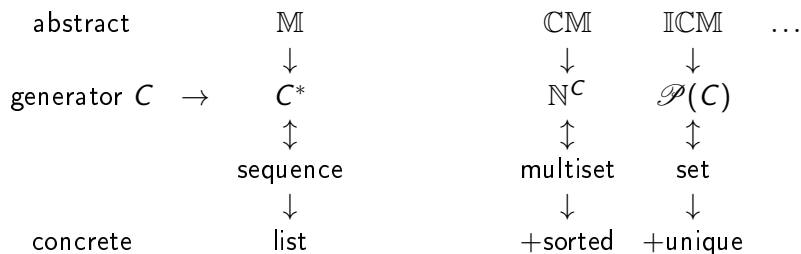
Let  $S^*$  be the smallest set such that:

- 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



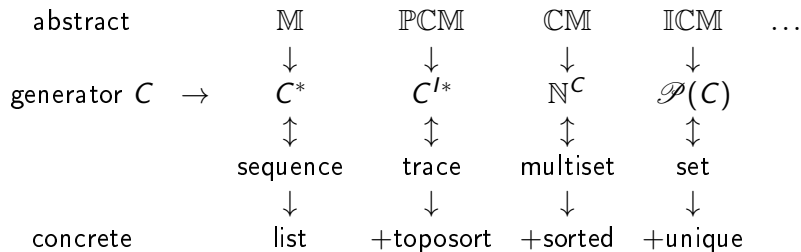
# Monoid overview

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*  $\text{PCM}(S, I, \cdot, 1)$  is:

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

such that:

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

# 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  $\mathcal{T}(C)$  remains the same.

4. Congruence generated by specification

## Definition

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

- 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  $\mathcal{T}(C) \setminus \cong_I$ .

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  $\varphi : C^* \rightarrow C^{I*}$  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:

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

and

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

Let  $(a, b) \in I$  and  $(c, a) \in I$ .

These are trace equivalent.

Let  $a < b < c < d$ , et cetera.

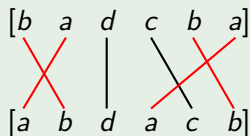
This trace is in lexicographic normal form.

# 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.

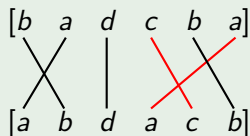


# Free partially commutative monoids

## 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:

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

and

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

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:

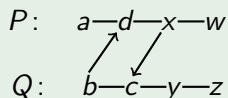
- FPCM is used to model concurrency
- FPCM captures all possible operations on a PCM, i.e. a unique homomorphism from  $S^{I*}$  to every  $\text{PCM}(S, I, \cdot, 1)$

# Partially commutative monoid

An example of a partially commutative monoid.

## Example

Consider the following computation:

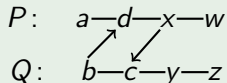


Example linearizations:

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

# Lamport's logical clock

## Example



Computing logical clock values (cf. vector clocks):

$$\begin{bmatrix} (1) & (0) & (2) & (3) & (3) & (4) & (3) & (3) \\ (0) & (1) & (1) & (1) & (4) & (1) & (5) & (6) \\ a & b & d & x & c & w & y & z \end{bmatrix}$$

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.

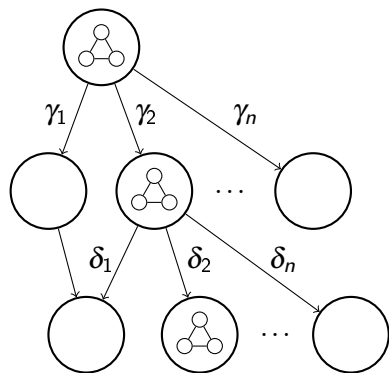
**Queuing channels** The channel acts as a FIFO queue.

# 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  $\mu$ ,
  - state space  $\sigma$
- 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  $S : \mu \times V \times \sigma \rightarrow \mathbb{L}$
- 2 a *receive* effect  $R : (\mu \times V \rightarrow \sigma) \rightarrow \mathbb{L}$
- 3 an *internal* effect  $I : \mathcal{P}(\sigma) \rightarrow \mathbb{L}$ .

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

## Definition

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

# Process descriptions: configurations

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

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

**Process labeling**  $\lambda : V \rightarrow \sigma$  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	$\kappa[p \xrightarrow{m} q](x) \stackrel{\text{def}}{=} \begin{cases} \kappa(x) @ (m \# \varepsilon) & (p, q) = x \\ \kappa(x) & \text{otherwise} \end{cases}$
test message	$\kappa[q \xleftarrow{m} p]? \Leftrightarrow \exists xs!. \kappa(p, q) = m \# xs$
$\curvearrowright$ 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**  $\kappa : E \rightarrow \mu^*$  maps channel to traveling messages.

**Process labeling**  $\lambda : V \rightarrow \sigma$  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	$\kappa[p \xrightarrow{m} q](x) \stackrel{\text{def}}{=} \begin{cases} \kappa(x) @ (m \# \varepsilon) & (p, q) = x \\ \kappa(x) & \text{otherwise} \end{cases}$
test message	$\kappa[q \xleftarrow{m} p]? \Leftrightarrow \exists \mathbf{xS}!. \kappa(p, q) = m \# \mathbf{xS}$
$\curvearrowright$ receive message	$\kappa[q \leftarrow p](x) \stackrel{\text{def}}{=} \begin{cases} \mathbf{xS} & (p, q) = x \\ \kappa(x) & \text{otherwise} \end{cases}$

# 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

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

- $\Delta$  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

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

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

## Example: Cidon's algorithm

Cidon's distributed depth-first search.

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

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

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

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  $\Delta \vdash (\kappa, \lambda)$ ,  $\kappa$  is determined by  $\Delta$ , viz. “unreceived messages” PCM.

With slight adaption of definitions,  $\Delta$  also determines  $\lambda$ .

Let  $\Delta \vdash \phi$  denote the judgement that  $\phi$  holds for  $(\kappa, \lambda)$ .

- 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  $\sigma_X$ , functional composition  $P_1 \circ P_2$ 
  - which states are relevant? fan-in or fan-out?

Thank you for your attention. Questions?

$$P(\top, s, \perp, F, I) = I(\{\top, s, [x], F \setminus x, I \setminus x\} \mid x \in F)$$

where  $s \in \{\text{Initiator}, \text{Received}(p)\}$  and  $F \neq \emptyset$

$$P(\top, s, [x], F, I) = S(\text{Info}, i, (\top, s, [x], F, I \setminus i))$$

where  $s \in \{\text{Initiator}, \text{Received}(p)\}$  and  $i \in I$  and  $I \neq \emptyset$

$$P(\top, s, [x], F, \emptyset) = S(\text{Token}, x, (\perp, s, [x], F, \emptyset))$$

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

$$P(\top, s, \perp, \emptyset, I) = S(\text{Info}, i, (\top, s, \perp, \emptyset, I \setminus i))$$

where  $s = \text{Received}(p)$  and  $i \in I$  and  $I \neq \emptyset$

$$P(\top, s, \perp, \emptyset, \emptyset) = S(\text{Token}, p, (\perp, s', \perp, \emptyset, \emptyset))$$

where  $s = \text{Received}(p)$  and  $s' = \text{Replied}(p)$

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

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

$$\begin{aligned}
 & P(\perp, \text{Undefined}, \perp, F, I) = R(f) \\
 \text{where} \quad & f(\text{Token}, p) = (\top, \text{Received}(p), \perp, F \setminus p, I \setminus p) \\
 & f(\text{Info}, p) = (\perp, \text{Undefined}, \perp, F \setminus p, I) \\
 & P(\perp, s, \lceil x \rceil, F, I) = R(f) \\
 \text{where } s \in & \{\text{Received}(p), \text{Initiator}\} \\
 \text{and} \quad & f(m, x) = (\top, s, \perp, F, I) \\
 & f(m, q) = (\perp, s, \perp, F \setminus q, I) \\
 & P(\perp, s, \perp, F, I) = R(f) \\
 \text{where } s \in & \{\text{Received}(p), \text{Initiator}\} \\
 \text{and} \quad & f(\text{Token}, q) = (\top, s, \perp, F, I) \\
 & f(\text{Info}, q) = (\perp, s, \perp, F \setminus q, I)
 \end{aligned}$$