Weak Models of Distributed Computing, with Connections to Modal Logic

Lauri Hella[1], Matti Järvisalo[2], Antti Kuusisto[1], Juhana Laurinharju[2], Tuomo Lempiäinen[2], Kerkko Luosto[2], Jukka Suomela[2], and Jonni Virtema[1]

1 University of Tampere
2 University of Helsinki

Weak Models of Distributed Computing

- Introduce novel complexity classes for distributed computing.
- Each class is a collection of *graph problems* that can be solved in a variant of the *port-numbering model*.
- We give a *complete classification* of the computational powers of the classes.

Weak Models of Distributed Computing

- Introduce novel complexity classes for distributed computing.
- Each class is a collection of *graph problems* that can be solved in a variant of the *port-numbering model*.
- We give a *complete classification* of the computational powers of the classes.
- We also establish a natural connection between the classes and variants of *modal logic*.

Deterministic Distributed Algorithms

- A graph $G = (V, E)$ presents a distributed system.
- Each node v runs the same state machine $\mathcal{M}$. Nodes know only their degree; no unique identifiers.
- Computation proceeds in sychronous steps;

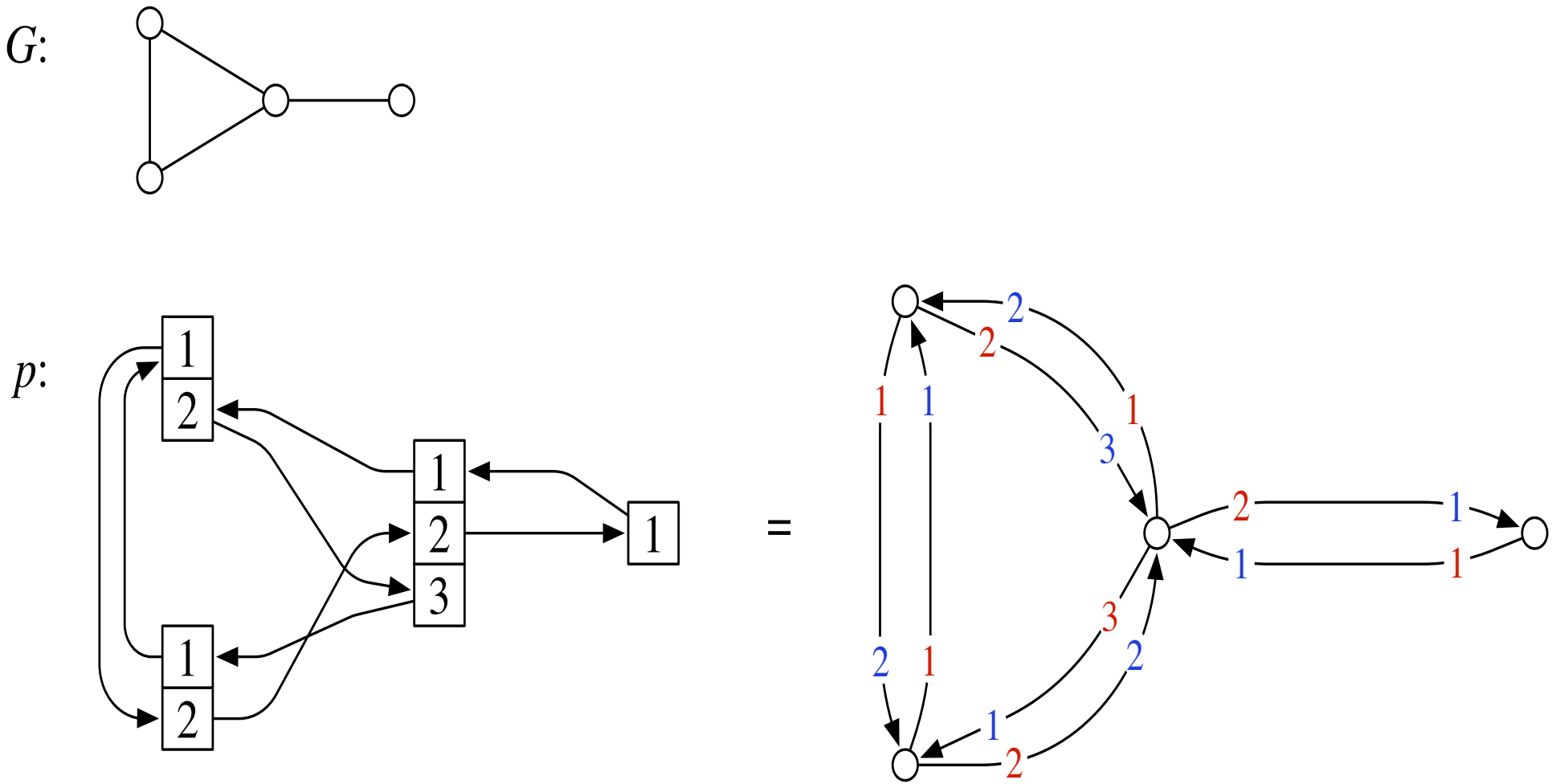
Deterministic Distributed Algorithms

- A graph $G = (V, E)$ presents a distributed system.
- Each node v runs the same state machine $\mathcal{M}$. Nodes know only their degree; no unique identifiers.
- Computation proceeds in sychronous steps;
- In *one time step*, each machine
  - sends messages to its neighbours,
  - receives messages from its neighbours,
  - updates its state based on the received messages.
- If the new state is a stopping state, the machine halts.
- There is no limit on message size.

Port numbering

- A node of degree k sends messages through k output ports and receives messages trough k input ports.
- Both input ports and output ports are numbered with $1, 2, ..., k$.
- Port in $G = (V, E)$ is a pair (v, i) s.t. $1 \leq i \leq degree(v)$.
- Port numbering: bijection p mapping ports to ports of neighbouring nodes
- $p((v, i)) = (u, j)$: message sent by v to port (v, i) is received by u from port (u, j)

Port numbering



Consistent port numbering

- All neighbours u and v use the same ports in both directions: if $p((v,i)) = (u,j)$, then $p((u,j)) = (v,i)$.

Graph problems

- Graph problem is a function Π that associates with each graph $G = (V, E)$ a set $\Pi(G)$ of solutions.
- Solution is a mapping $S\colon V \to Y$ for a fixed finite set Y.

Graph problems

- Graph problem is a function Π that associates with each graph $G = (V, E)$ a set $\Pi(G)$ of solutions.
- Solution is a mapping $S\colon V \to Y$ for a fixed finite set Y.
- Finding a vertex cover: $Y = \{0, 1\}$ and $\Pi(G)$ is the set of functions $S\colon V \to Y$ s.t. $\{v \in V \mid S(v) = 1\}$ is a vertex cover of G.

Solving graph problems

- A sequence $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$ of algorithms solves Π in time $T : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ if for any G of maximum degree at most Δ and any port numbering p of G,
  - $\mathcal{A}_\Delta$ stops in time $T(\Delta, |V|)$ on input (G, p),
  - The output of $\mathcal{A}_\Delta$ on (G, p) is in $\Pi(G)$.

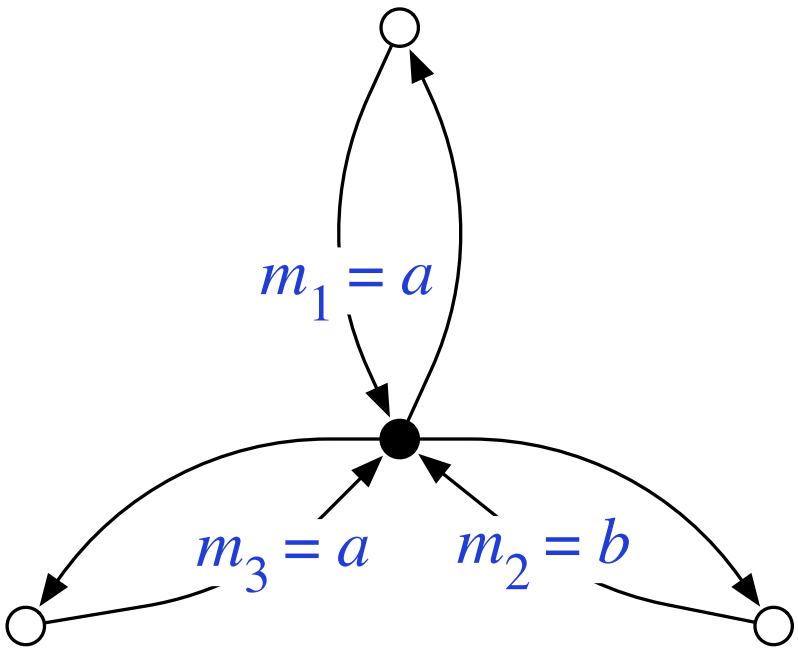
Solving graph problems

- A sequence $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$ of algorithms solves Π in time $T : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ if for any G of maximum degree at most Δ and any port numbering p of G,
  - $\mathcal{A}_\Delta$ stops in time $T(\Delta, |V|)$ on input (G, p),
  - The output of $\mathcal{A}_\Delta$ on (G, p) is in $\Pi(G)$.
- $\mathcal{A}$ solves Π assuming *consistency*, if the above holds for all consistent port numberings p.
- $\mathcal{A}$ solves Π in *constant time*, if T does not depend on $|V|$.

Algorithm Classes

- Vector: all distributed algorithms.
- Multiset: nodes receive a multiset of messages.
- Set: nodes receive messages as sets.
- Broadcast: nodes send the same message to all neighbours.

Vector vs. Multiset vs. Set



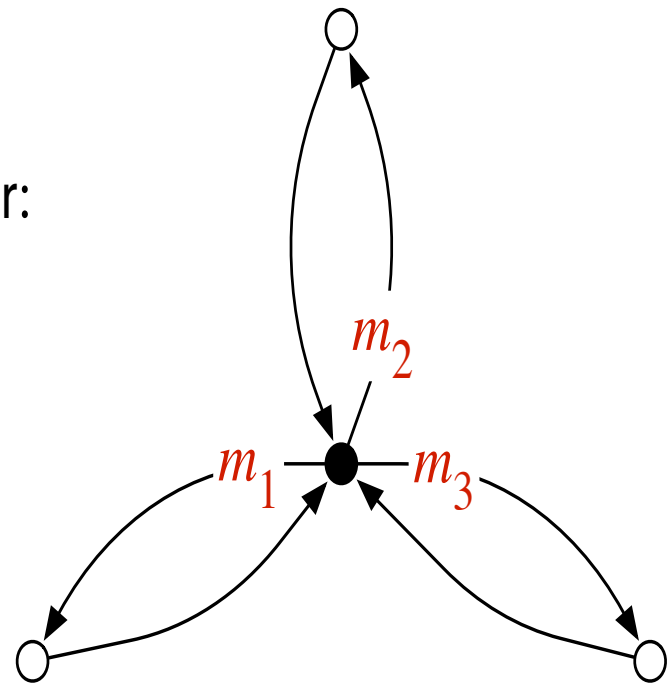
Vector: received (a, b, a)

Multiset: received $\{a, a, b\}$

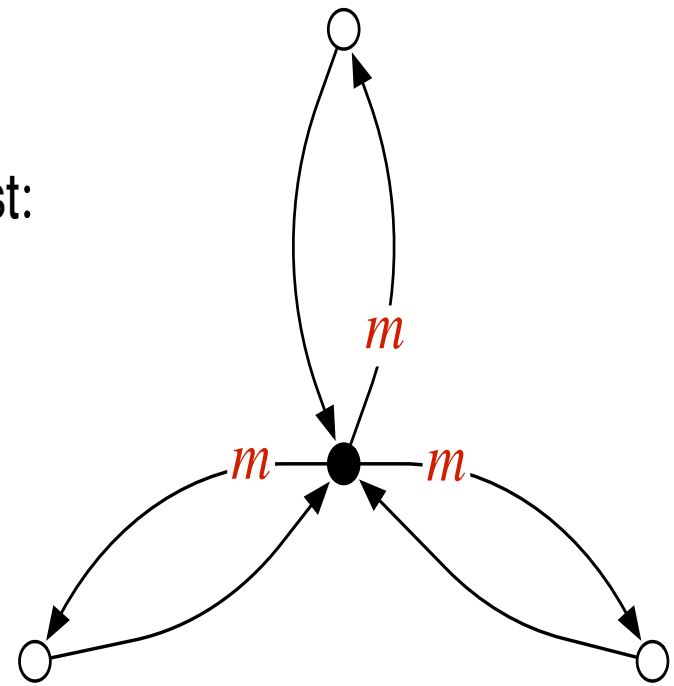
Set: received $\{a, b\}$

Vector vs. Broadcast

Vector:



Broadcast:



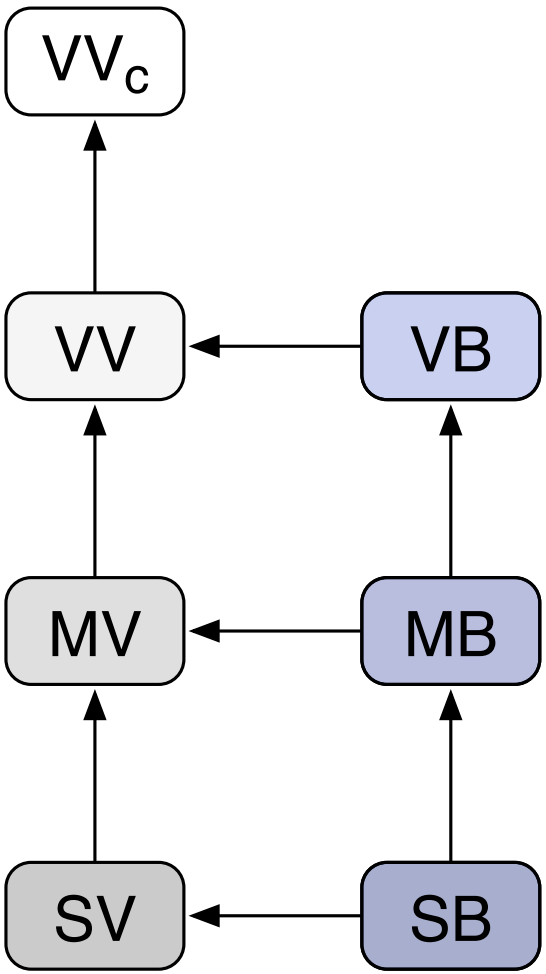
Complexity Classes

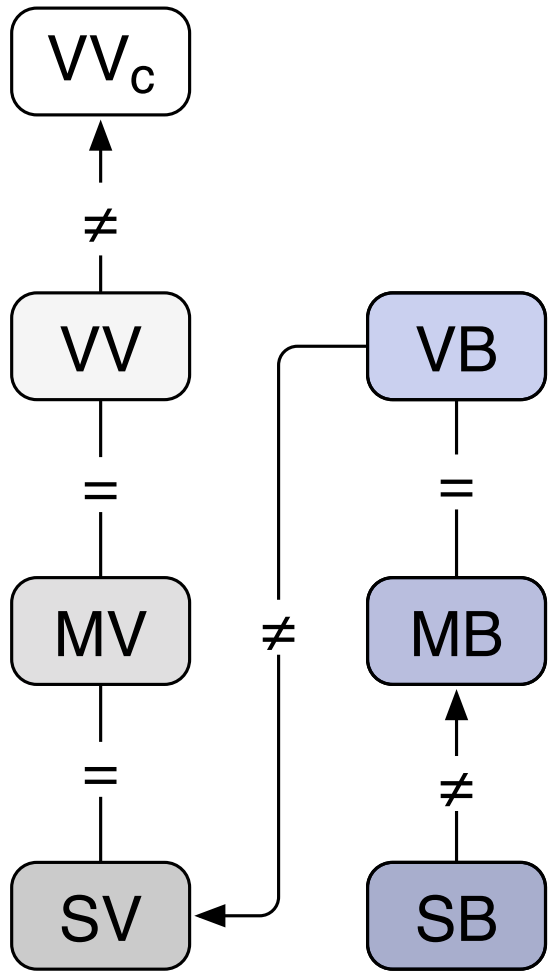
- $\mathsf{VV_c}$: some algorithm sequence $\mathcal{A}$ in Vector solves Π assuming consistency of port numbering.
- VV: some $\mathcal{A}$ in Vector solves Π.
- MV: some $\mathcal{A}$ in Multiset solves Π.
- SV: some $\mathcal{A}$ in Set solves Π.
- VB: some $\mathcal{A}$ in Broadcast that solves Π.
- MB: some $\mathcal{A}$ in the intesection of Multiset and Broadcast solves Π.
- SB: some $\mathcal{A}$ in the intersection of Set and Broadcast solves Π.

Constant time classes

- $\mathsf{VV_c}(1)$: some $\mathcal{A}$ in Vector solves Π in constant time assuming consistency of port numbering.
- $\mathsf{VV}(1)$: some algorithm $\mathcal{A}$ in Vector solves Π in constant time.
- Similarly $\mathsf{MV}(1)$, $\mathit{SV}(1)$, $\mathit{VB}(1)$, $\mathit{MB}(1)$ and $\mathsf{SB}(1)$.

The Classes are Linearly Ordered by Containment





SV = MV

Theorem
SV = MV.

An MV algorithm is simulated by an SV algorithm.

Multiplicity of incoming messages can be accessed by sending back and forth—together with the original message—information about the port numbers to which each message was sent.

Any node v connected to distinct nodes u and w sends messages to u and w via output ports with different port numbers, so ultimately u and w will send a different message back to v.

MV = VV

Theorem
MV = VV.

Define a linear ordering of messages. This defines a lexicographic order over full message histories received by any port.

Ordering of message histories is used in order to recover information about received vectors of messages by looking at received multisets.

Characterizing the Complexity Classes with Modal Logic

For each class constant time class C there is a modal logic $\mathrm{ML}(C)$ such that there is a *canonical one-to-one correspondence* between algorithms in C and formulae of $\mathrm{ML}(C)$.

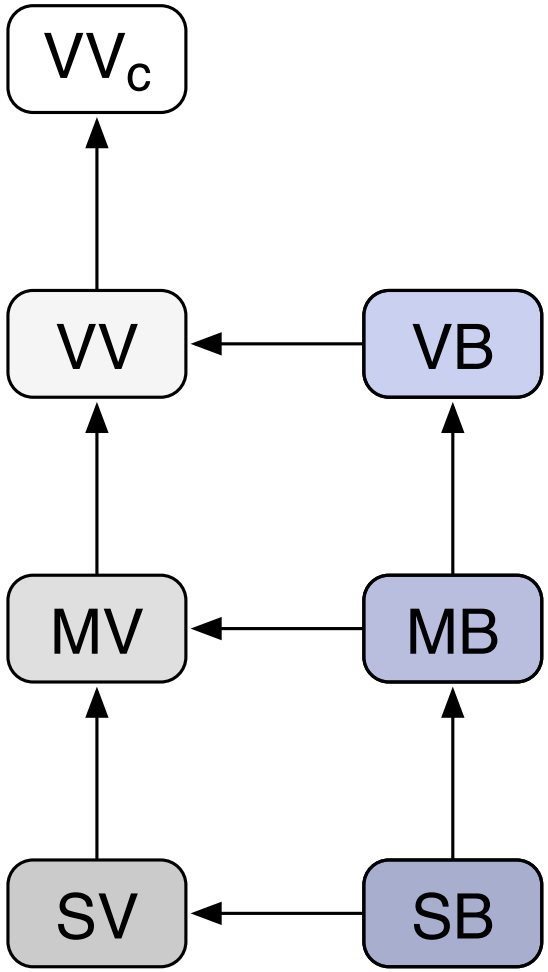
Therefore $\mathrm{ML}(C)$ is a complete specification language for C.

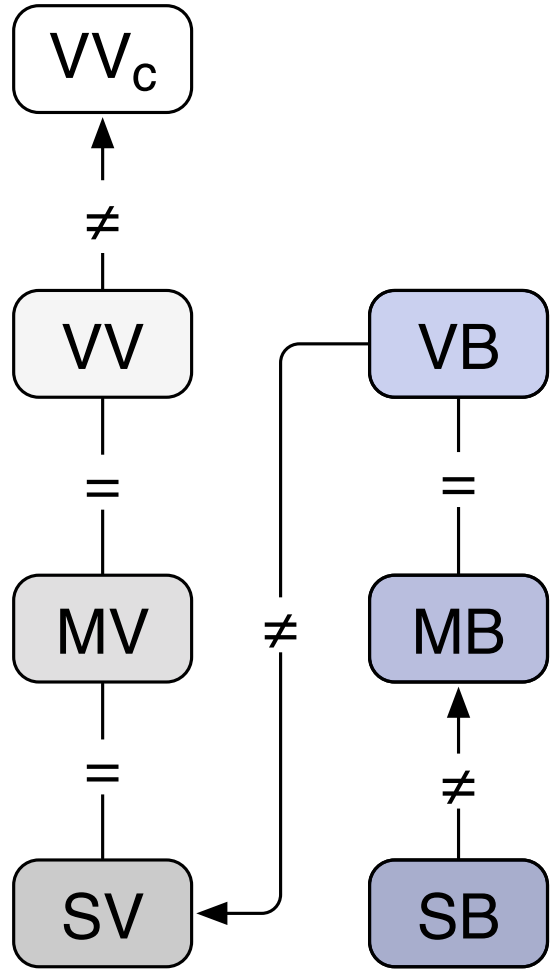
Modal Logic

$\varphi := q_n \mid \varphi \wedge \psi \mid \neg\varphi \mid \langle i, j \rangle \varphi$, where q_n are *proposition symbols*.

$$(G, p), v \models q_n \text{ iff } degree(v) = n.$$
$$(G, p), v \models \langle i, j \rangle \varphi \text{ iff } (G, p), u \models \varphi \text{ for some node } u \text{ such that } p(u, j) = (v, i).$$

We use tools of modal logic in the proofs separating the classes.


VV_c
VV
VB
MV
MB
SV
SB


VV_c
≠
VV
VB
=
=
MV
≠
MB
=
≠
SV
SB