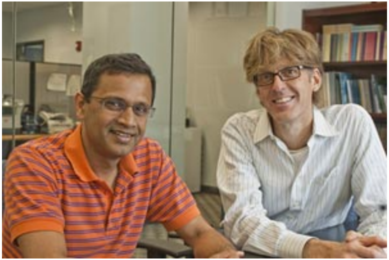
Eventual Consistency for CRDTs



Radha Jagadeesan James Riely

DePaul University
Chicago, USA

ESOP 2018

CRDTs?

CRDTs?

C = blah blah
R = mumble

DT = *Data Type*

Data Type

- "An *abstract data type* defines a class of abstract objects which is completely characterized by the operations available on those objects." (Liskov/Zilles 1974)

Data Type

- "An *abstract data type* defines a class of abstract objects which is completely characterized by the operations available on those objects." (Liskov/Zilles 1974)
- Eg, binary set with operations:
  - `+0`, `+1`: add
  - `-0`, `-1`: remove
  - ✗`0`, ✗`1`: membership query returning false
  - ✓`0`, ✓`1`: membership query returning true

Data Type

- "An *abstract data type* defines a class of abstract objects which is completely characterized by the operations available on those objects." (Liskov/Zilles 1974)
- Eg, binary set with operations:
  - `+0`, `+1`: add
  - `-0`, `-1`: remove
  - `✗0`, `✗1`: membership query returning false
  - `✓0`, `✓1`: membership query returning true
- Sequential interface:
  - ☺ `+0-0✗0`
  - ☹ `+0-0✓0`

Data Type

- "An *abstract data type* defines a class of abstract objects which is completely characterized by the operations available on those objects." (Liskov/Zilles 1974)
- Eg, list with operations:
  - `put(0)`, `put(1)`, `put(2)`, ...: add to end
  - `q=[]`, `q=[0]`, `q=[0,1]`, ...: query returning list contents

Data Type

- "An *abstract data type* defines a class of abstract objects which is completely characterized by the operations available on those objects." (Liskov/Zilles 1974)
- Eg, list with operations:
  - `put(0)`, `put(1)`, `put(2)`, …: add to end
  - `q=[]`, `q=[0]`, `q=[0,1]`, …: query returning list contents
- Sequential interface:
  - ☺ `put(0) put(1) q=[0,1]`
  - ☹ `put(0) put(1) q=[1,0]`

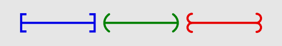
Data Type

- "An *abstract data type* defines a class of abstract objects which is completely characterized by the operations available on those objects." (Liskov/Zilles 1974)
- Eg, list with operations:
  - `put(0)`, `put(1)`, `put(2)`, ...: add to end
  - `q=[]`, `q=[0]`, `q=[0,1]`, ...: query returning list contents
- Sequential interface:
  - ☺ `put(0) put(1) q=[0,1]`
  - ☹ `put(0) put(1) q=[1,0]`
- ADT: contract between *implementor* and *client*
  - Implementor and client take turns

What about concurrent clients?

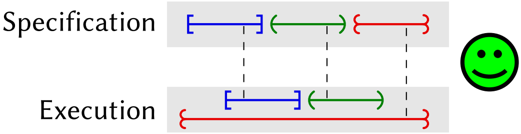
Linearizability (Herlihy/Wing 1990)

- "Each method call should appear to take effect instantaneously at some moment between its invocation and response." (Herlihy/Shavit 2008)

Specification 

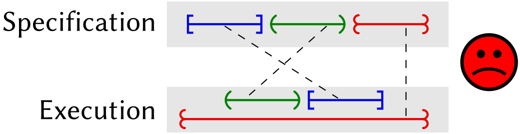
Linearizability (Herlihy/Wing 1990)

- "Each method call should appear to take effect instantaneously at some moment between its invocation and response."
(Herlihy/Shavit 2008)



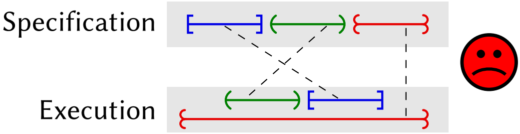
Linearizability (Herlihy/Wing 1990)

- "Each method call should appear to take effect instantaneously at some moment between its invocation and response." (Herlihy/Shavit 2008)



Linearizability (Herlihy/Wing 1990)

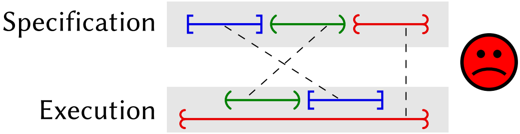
- "Each method call should appear to take effect instantaneously at some moment between its invocation and response." (Herlihy/Shavit 2008)



- Client wins!
  - Compositional (Herlihy/Wing 1990)
  - Exactly characterizes programmer view (for coordinating clients) (Filipovic/O'Hearn/Rinetzky/Yang 2010)

Linearizability (Herlihy/Wing 1990)

- "Each method call should appear to take effect instantaneously at some moment between its invocation and response."
  (Herlihy/Shavit 2008)



- Client wins!
  - Compositional (Herlihy/Wing 1990)
  - Exactly characterizes programmer view (for coordinating clients)
    (Filipovic/O'Hearn/Rinetzky/Yang 2010)
- Implementor loses!
  - Intrinsically inefficient (Dwork/Herlihy/Waarts 1997)
    See also: CAP theorem (Gilbert/Lynch 2002)

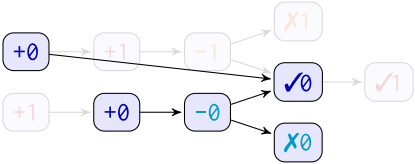
High performance?

High performance? ☹

R = Replicated

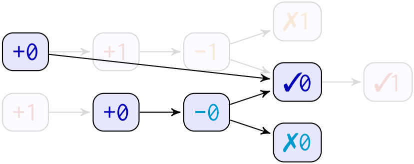
Replicated Sets: Add-Wins Set

- Specification of query:
  - ✗0 if every +0 followed by −0
  - ✓0 if some +0 not followed by −0
- Example Execution:



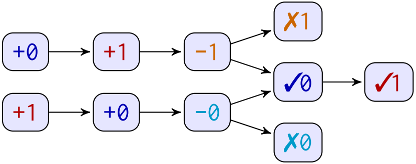
Replicated Sets: Add-Wins Set

- Specification of query:
  - ✗0 if every +0 followed by −0
  - ✓0 if some +0 not followed by −0
- Example Execution:



Replicated Sets: Add-Wins Set

- Specification of query:
  - ✗0 if every +0 followed by −0
  - ✓0 if some +0 not followed by −0
- Example Execution:

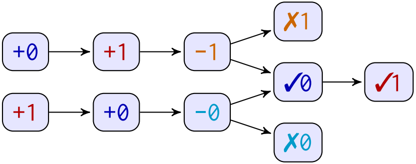


☺ High-performance implementation

Not linearizable: No interleaving satisfies both ✓0 and ✓1

Replicated Sets: Add-Wins Set

- Specification of query:
  - ✗0 if every +0 followed by −0
  - ✓0 if some +0 not followed by −0
- Example Execution:



☺ High-performance implementation

Not linearizable: No interleaving satisfies both ✓0 and ✓1

☺ Strong Eventual Consistency (SEC)

Replicas that see same updates give same answers

Replicated Sets: Amazon Dynamo (?)

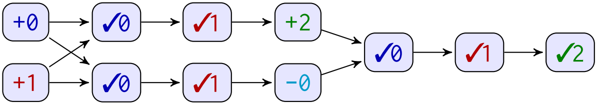
- Specification:

  …sloppy quorum…vector clock…

- ☺ High-performance implementation
- ☺ Strong Eventual Consistency (SEC)

  Replicas that see same updates give same answers

- Example: (Bieniusa/Zawirski/Preguiça/Shapiro/Baquero/Balegas/Duarte 2012)



Replicated Sets: Amazon Dynamo (?)

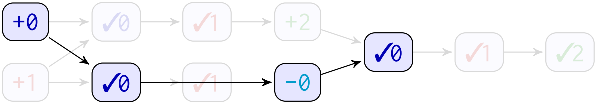
- Specification:

  …sloppy quorum…vector clock…

- High-performance implementation
- Strong Eventual Consistency (SEC)

  Replicas that see same updates give same answers

- Example: (Bieniusa/Zawirski/Preguiça/Shapiro/Baquero/Balegas/Duarte 2012)



- Is this a set?

  +0✓0-0✓0 is not a set execution

Replicated Sets: Amazon Dynamo (?)

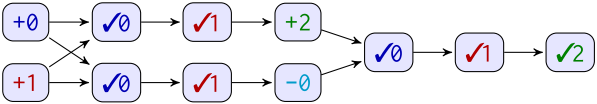
- Specification:

  …sloppy quorum…vector clock…

- ☺ High-performance implementation
- ☺ Strong Eventual Consistency (SEC)

  Replicas that see same updates give same answers

- Example: (Bieniusa/Zawirski/Preguiça/Shapiro/Baquero/Balegas/Duarte 2012)



- ☹ Is this a set?

  +0✓0-0✓0 is not a set execution

- ☺ No, it's a Multi-Value Register (Shapiro 2011, MSR Talk)

Replicated Sets: Amazon Dynamo (?)

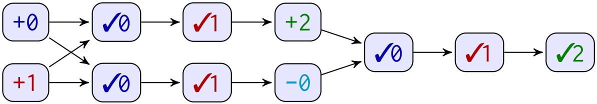
- Specification:

  …sloppy quorum…vector clock…

- ☺ High-performance implementation
- ☺ Strong Eventual Consistency (SEC)

  Replicas that see same updates give same answers

- Example: (Bieniusa/Zawirski/Preguiça/Shapiro/Baquero/Balegas/Duarte 2012)



- ☹ Is this a set?

  +0✓0-0✓0 is not a set execution

- ☺ No, it's a Multi-Value Register (Shapiro 2011, MSR Talk)
- ☹ But SEC does not explain this

State Of Play

- Correctness Criterion: Strong Eventual Consistency (SEC)
  - ☺ Add-Wins Set Example
  - ☺ Amazon Dynamo Example
- ☹ But Add-Wins is more set-like

State Of Play

- Correctness Criterion: Strong Eventual Consistency (SEC)
  - ☺ Add-Wins Set Example
  - ☺ Amazon Dynamo Example
- ☹ But Add-Wins is more set-like
- Correctness: sequential vs replicated

Idea	Sequential	Replicated
Safety Termination		

State Of Play

- Correctness Criterion: Strong Eventual Consistency (SEC)
  - ☺ Add-Wins Set Example
  - ☺ Amazon Dynamo Example
- ☹ But Add-Wins is more set-like
- Correctness: sequential vs replicated

Idea	Sequential	Replicated
Safety	Partial correctness	
Termination	Total correctness	

State Of Play

- Correctness Criterion: Strong Eventual Consistency (SEC)
  - Add-Wins Set Example
  - Amazon Dynamo Example
- But Add-Wins is more set-like
- Correctness: sequential vs replicated

Idea	Sequential	Replicated
Safety	Partial correctness	
Termination	Total correctness	*Convergence* = SEC

State Of Play

- Correctness Criterion: Strong Eventual Consistency (SEC)
  - ☺ Add-Wins Set Example
  - ☺ Amazon Dynamo Example
- ☹ But Add-Wins is more set-like
- Correctness: sequential vs replicated

Idea	Sequential	Replicated
Safety	Partial correctness	???
Termination	Total correctness	*Convergence* = SEC

State Of Play

- Correctness Criterion: Strong Eventual Consistency (SEC)
  - ☺ Add-Wins Set Example
  - ☺ Amazon Dynamo Example
- ☹ But Add-Wins is more set-like
- Correctness: sequential vs replicated

Idea	Sequential	Replicated
Safety	Partial correctness	???
Termination	Total correctness	*Convergence* = SEC

- This paper: What is a good notion of safety?

Safety?

C = Conflict-free

CRDTs (Shapiro/Preguiça/Baquero/Zawirski 2011)

- Conflict-free, operationally defined = either
  - Convergent, State-based
  - Commutative, Operation-based

CRDTs (Shapiro/Preguiça/Baquero/Zawirski 2011)

- Conflict-free, operationally defined = either
  - Convergent, State-based
  - Commutative, Operation-based
- ☺ Sufficient to establish SEC

CRDTs (Shapiro/Preguiça/Baquero/Zawirski 2011)

- Conflict-free, operationally defined = either
  - Convergent, State-based
  - Commutative, Operation-based

☺ Sufficient to establish SEC

😐 Examples also appear to satisfy safety (in some sense)

CRDTs (Shapiro/Preguiça/Baquero/Zawirski 2011)

- Conflict-free, operationally defined = either
  - Convergent, State-based
  - Commutative, Operation-based
- Sufficient to establish SEC
- Examples also appear to satisfy safety (in some sense)
- Correctness defined using *concurrent* spec

> *"It is infinitely easier and more intuitive for us humans to specify how abstract data structures behave in a sequential setting, where there are no interleavings. Thus, the standard approach to arguing the safety properties of a concurrent data structure is to specify the structure's properties sequentially, and find a way to map its concurrent executions to these 'correct' sequential ones."* *(Shavit 2011)*

CRDTs (Shapiro/Preguiça/Baquero/Zawirski 2011)

- Conflict-free, operationally defined = either
  - Convergent, State-based
  - Commutative, Operation-based

- Sufficient to establish SEC
- Examples also appear to satisfy safety (in some sense)
- Correctness defined using *concurrent* spec

> *"It is infinitely easier and more intuitive for us humans to specify how abstract data structures behave in a sequential setting, where there are no interleavings. Thus, the standard approach to arguing the safety properties of a concurrent data structure is to specify the structure's properties sequentially, and find a way to map its concurrent executions to these 'correct' sequential ones."* (Shavit 2011)

- This paper:

An extensional notion of safety for CRDTs
appealing only to the *sequential* spec

This talk: From Linearizability to CRDTs in 5 relaxations

Relaxations:

- Real time: Distributed system
- Order after an accessor: Update serializability
- Order between independent updates: Preserved Program Order
- Linearize labels, not events: Punning
- Quotient specification by observational equivalence: Stuttering

This talk: From Linearizability to CRDTs in 5 relaxations

Evidence that definition is the “right” one (in paper)

- ☺ Simulation-based characterization
  - ☺ *Most General CRDT*, expressed as Labelled Transition System
  - ☺ Compositionality and Substitutivity results
  - ☺ Validation of CRDT Graph built using CRDT sets
- ☺ Corner cases
  - ☺ Updates to one replica only $\Rightarrow$ linearizable
  - ☺ Permutation equivalence in spec $\Rightarrow \ldots$
- ☺ Validates all known CRDTs
  - ☺ Add-Wins Set (Shapiro/Preguiça/Baquero/Zawirski 2011)
  - ☺ Collaborative Text-Editing Protocol (Attiya/Burckhardt/Gotsman/Morrison/Yang/Zawirski)

This talk: From Linearizability to CRDTs in 5 relaxations

Evidence that definition is the “right” one (in paper)

- Simulation-based characterization
  - *Most General CRDT*, expressed as Labelled Transition System
  - Compositionality and Substitutivity results
  - Validation of CRDT Graph built using CRDT sets
- Corner cases
  - Updates to one replica only $\Rightarrow$ linearizable
  - Permutation equivalence in spec $\Rightarrow \ldots$
- Validates all known CRDTs
  - Add-Wins Set (Shapiro/Preguiça/Baquero/Zawirski 2011)
  - Collaborative Text-Editing Protocol (Attiya/Burckhardt/Gotsman/Morrison/Yang/Zawirski)
- Validates *every possible* CRDTs
  - Def of CRDT does not mention sequential spec
  - Our def = proposal for meaning of CRDT

Components of Safety

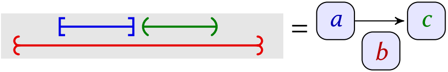
- *Linearization*: response must be consistent with some spec string
  - List replica that sees `put(0)` and `put(2)` may respond
    - ☺ `q=[0,2]`
    - ☺ `q=[2,0]`
    - ☹ `q=[]`
    - ☹ `q=[1,0,2]`
- *Monotonicity*: responses evolve sensibly
  - List replica in state `q=[0,2]`, may evolve to
    - ☺ `q=[0,1,2]`, due to arrival of `put(1)`
    - ☹ `q=[2,0]`, no support for delete or reorder

Relaxation 1: Order in Distributed Systems

- v is *valid for* Σ if …

  Σ = specification = set of strings of labels
  v = execution = Labeled Partial Order (LPO)
  Order of LPO = non-overlapping method calls (real time)

- Example:

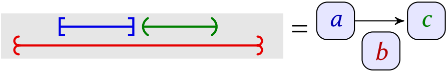


Relaxation 1: Order in Distributed Systems

- v is *valid for* Σ if …

  $C(v)$ = downclosed sets of v (i.e., *cuts*) (Chandy/Lamport 1985)
  Σ = specification = set of strings of labels
  v = execution = Labeled Partial Order (LPO)
  Order of LPO = non-overlapping method calls (real time)

- Example:



$$Cuts = \{a\},\ \{a, c\},\ \{b\},\ \{a, b\},\ \{a, b, c\}$$

$$Frontiers = \{a\},\ \{\quad c\},\ \{b\},\ \{a, b\},\ \{\quad b, c\}$$ (Maximal elements)

Relaxation 1: Order in Distributed Systems

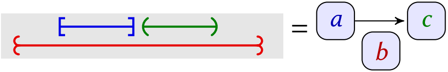
- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

  $C(v)$ = downclosed sets of v (i.e., *cuts*) (Chandy/Lamport 1985)

  Σ = specification = set of strings of labels

  v = execution = Labeled Partial Order (LPO)

  Order of LPO = non-overlapping method calls (real time)
- Example:



$\textit{Cuts} = \{a\},\ \{a, c\},\ \{b\},\ \{a, b\},\ \{a, b, c\}$

$\textit{Frontiers} = \{a\},\ \{\quad c\},\ \{b\},\ \{a, b\},\ \{\quad b, c\}$ (Maximal elements)

For specification abc, f maps cuts to subsequences of abc

Relaxation 1: Order in Distributed Systems

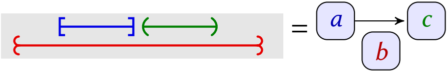
- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

  $C(v)$ = downclosed sets of v (i.e., *cuts*) (Chandy/Lamport 1985)

  Σ = specification = set of strings of labels

  v = execution = Labeled Partial Order (LPO)

  Order of LPO = non-overlapping method calls (real time)
- Example:



  $\textit{Cuts} = \{a\},\ \{a, c\},\ \{b\},\ \{a, b\},\ \{a, b, c\}$

  $\textit{Frontiers} = \{a\},\ \{\ \ c\},\ \{b\},\ \{a, b\},\ \{\ \ b, c\}$ (Maximal elements)

  For specification abc, f maps cuts to subsequences of abc
- Replicated system: No global clock

Relaxation 1: Order in Distributed Systems

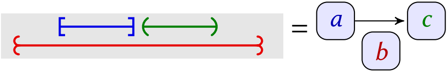
- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

  $C(v)$ = downclosed sets of v (i.e., *cuts*) (Chandy/Lamport 1985)

  Σ = specification = set of strings of labels

  v = execution = Labeled Partial Order (LPO)

  Order of LPO = *per-replica visibility*
- Example:



$$\textit{Cuts} = \{a\},\ \{a, c\},\ \{b\},\ \{a, b\},\ \{a, b, c\}$$

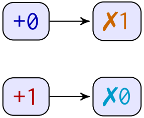
$$\textit{Frontiers} = \{a\},\ \{\quad c\},\ \{b\},\ \{a, b\},\ \{\quad b, c\}$$ (Maximal elements)

  For specification abc, f maps cuts to subsequences of abc
- Replicated system: No global clock

Relaxation 2: Update Serializability

- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

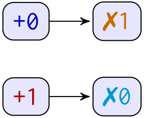
$C(v)$ = downclosed sets



Relaxation 2: Update Serializability

- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v)$ = downclosed sets



- Cannot linearize ✗0 and ✗1 together

Relaxation 2: Update Serializability

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathsf{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathsf{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathsf{seq}} f(q)$

$C(v)$ = *pointed* downclosed sets

$$\boxed{\texttt{+0}} \longrightarrow \boxed{\texttt{✗1}}$$

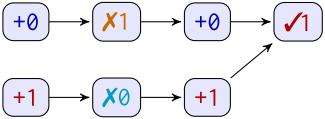
$$\boxed{\texttt{+1}} \longrightarrow \boxed{\texttt{✗0}}$$

- Cannot linearize ✗0 and ✗1 together

Relaxation 2: Update Serializability

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathsf{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathsf{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$C(v)$ = *pointed* downclosed sets

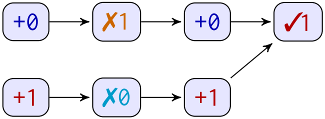


- Cannot linearize ✗0 and ✗1 together
- When linearizing ✓1, must not include both ✗0 and ✗1

Relaxation 2: Update Serializability

- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v)$ = *pointed update*-downclosed sets

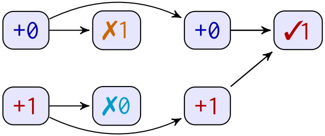


- Cannot linearize ✗0 and ✗1 together
- When linearizing ✓1, must not include both ✗0 and ✗1

Relaxation 2: Update Serializability

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathsf{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathsf{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$C(v)$ = *pointed update*-downclosed sets

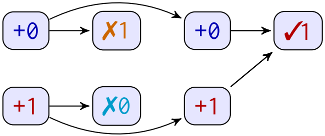


- Cannot linearize ✗0 and ✗1 together
- When linearizing ✓1, must not include both ✗0 and ✗1

Relaxation 2: Update Serializability

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathrm{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathrm{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$C(v)$ = *pointed update*-downclosed sets

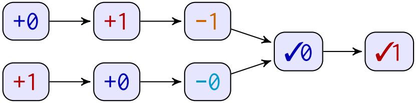


- Cannot linearize ✗0 and ✗1 together
- When linearizing ✓1, must not include both ✗0 and ✗1
- State of prior art (Burckhardt/Leijen/Fähndrich/Sagiv 2012)
  Cf. Update serializability: global order for updates
  (Hansdah/Patnaik 1986, Garcia-Molina and Wiederhold 1982)

Relaxation 3: Preserved Program Order

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathsf{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathsf{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathsf{seq}} f(q)$

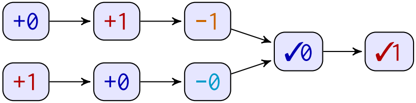
$C(v)$ = pointed update-downclosed sets



Relaxation 3: Preserved Program Order

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathrm{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathrm{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$C(v)$ = pointed update-downclosed sets

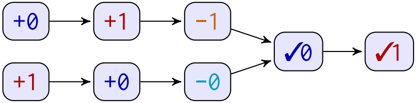


- Cannot linearize -1 and -0 together

Relaxation 3: Preserved Program Order

- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v)$ = pointed *dependent*-update-downclosed sets

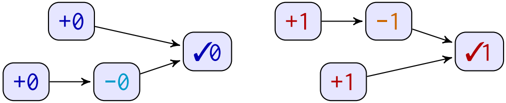


- Cannot linearize -1 and -0 together

Relaxation 3: Preserved Program Order

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathsf{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathsf{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$C(v)$ = pointed *dependent*-update-downclosed sets

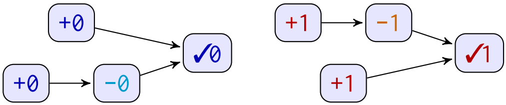


- Cannot linearize -1 and -0 together

Relaxation 3: Preserved Program Order

- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v)$ = pointed *dependent*-update-downclosed sets



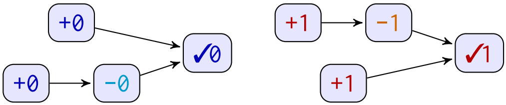
- Cannot linearize -1 and -0 together
- Cf. Preserved Program Order in relaxed memory models

(Higham/Kawash 2000, Alglave 2012).

Relaxation 3: Preserved Program Order

- v is *valid for* Σ if there exists a map $f : C(v) \to \text{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\text{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v)$ = pointed *dependent*-update-downclosed sets

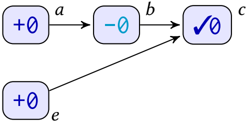


- Cannot linearize -1 and -0 together
- Cf. Preserved Program Order in relaxed memory models (Higham/Kawash 2000, Alglave 2012).
- Independency is a property of the *specification*

Relaxation 4: Puns

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathsf{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathsf{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathsf{seq}} f(q)$

  $C(v)$ = pointed dependent-update-downclosed sets

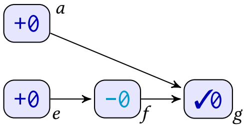


- Linearization must have $\texttt{-0}^b$ before $\texttt{+0}^e$

Relaxation 4: Puns

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathsf{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathsf{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathsf{seq}} f(q)$

$C(v)$ = pointed dependent-update-downclosed sets

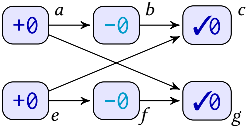


- Linearization must have $\texttt{-0}^b$ before $\texttt{+0}^e$
- Linearization must have $\texttt{-0}^f$ before $\texttt{+0}^a$

Relaxation 4: Puns

- v is *valid for* Σ if there exists a map $f : C(v) \to \mathrm{events}(v)^*$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$ and $\mathrm{labels}(f(p)) \in \Sigma$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$C(v)$ = pointed dependent-update-downclosed sets

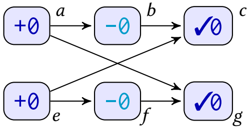


- Linearization must have $\texttt{-0}^b$ before $\texttt{+0}^e$
- Linearization must have $\texttt{-0}^f$ before $\texttt{+0}^a$
- Must linearize actions/labels, not events

Relaxation 4: Puns

- v is *valid for* Σ if there exists a map $f : \mathcal{C}(v) \to \Sigma$ s.t.
  - $\forall p \in \mathcal{C}(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in \mathcal{C}(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$\mathcal{C}(v)$ = pointed dependent-update-downclosed sets

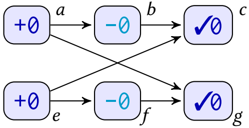


- Linearization must have $\texttt{-0}^b$ before $\texttt{+0}^e$
- Linearization must have $\texttt{-0}^f$ before $\texttt{+0}^a$
- Must linearize actions/labels, not events

Relaxation 4: Puns

- v is *valid for* Σ if there exists a map $f : \mathcal{C}(v) \to \Sigma$ s.t.
  - $\forall p \in \mathcal{C}(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in \mathcal{C}(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$\mathcal{C}(v)$ = pointed dependent-update-downclosed sets



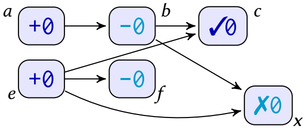
- Linearization must have -0^b before $+0^e$
- Linearization must have -0^f before $+0^a$
- Must linearize actions/labels, not events

$+0^a$, $+0^e$	:	$+0$
-0^b, -0^f	:	$+0-0$
$✓0^c$, $✓0^g$	:	$+0-0+0✓0$

Relaxation 4: A Bad Joke

- v is *valid for* Σ if there exists a map $f : C(v) \to \Sigma$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

  $C(v)$ = pointed dependent-update-downclosed sets



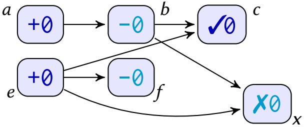
- Update order +0-0+0-0 with subsequences:

✓0^c	:	+0-0+0✓0	(✗0^c requires -0 *between* the +0s)
✗0^x	:	+0+0-0✗0	(✗0^x requires -0 *after* the +0s)

Relaxation 4: A Bad Joke

- v is *valid for* Σ if there exists a map $f : C(v) \to \Sigma$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v) =$ pointed dependent-update-downclosed sets



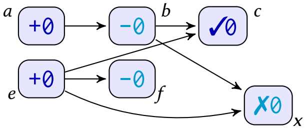
- Update order `+0-0+0-0` with subsequences:

✓0^c	:	`+0-0+0✓0`	(✗0^c requires `-0` *between* the `+0`s)
✗0^x	:	`+0+0-0✗0`	(✗0^x requires `-0` *after* the `+0`s)

Relaxation 4: A Bad Joke

- v is *valid for* Σ if there exists a map $f : C(v) \to \Sigma$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v) =$ pointed dependent-update-downclosed sets, for accessors
all dependent-update-downclosed sets, for updates



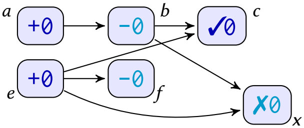
- Update order `+0-0+0-0` with subsequences:

$✓0^c$	:	`+0-0+0✓0`	($✗0^c$ requires `-0` *between* the `+0`s)
$✗0^x$	:	`+0+0-0✗0`	($✗0^x$ requires `-0` *after* the `+0`s)

Relaxation 4: A Bad Joke

- v is *valid for* Σ if there exists a map $f : C(v) \to \Sigma$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\mathrm{seq}} f(q)$

$C(v) =$ pointed dependent-update-downclosed sets, for accessors
all dependent-update-downclosed sets, for updates



- Update order +0-0+0-0 with subsequences:

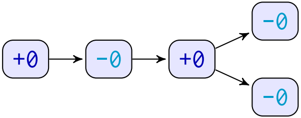
✓0[c]	:	+0-0+0✓0	(✗0[c] requires -0 *between* the +0s)
✗0[x]	:	+0+0-0✗0	(✗0[x] requires -0 *after* the +0s)

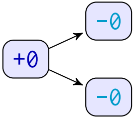
- Execution disallowed by monotonicity
  {+0[a], -0[b], +0[e]} cannot be linearized to satisfy both ✓0[c] and ✗0[x]

Relaxation 5: Observationally Equivalent Specifications

- v is *valid for* Σ if there exists a map $f : \mathcal{C}(v) \to \Sigma$ s.t.
  - $\forall p \in \mathcal{C}(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in \mathcal{C}(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$\mathcal{C}(v) =$ pointed dependent-update-downclosed sets, for accessors
all dependent-update-downclosed sets, for updates

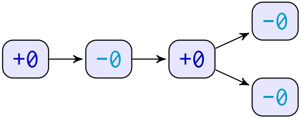


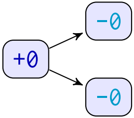


Relaxation 5: Observationally Equivalent Specifications

- v is *valid for* Σ if there exists a map $f : C(v) \to \Sigma$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{seq}} f(q)$

$C(v) =$ pointed dependent-update-downclosed sets, for accessors
all dependent-update-downclosed sets, for updates



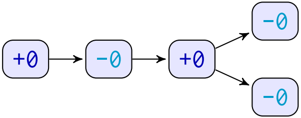


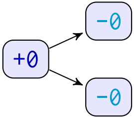
- Should this linearize to `+0-0-0+0-0+0-0-0`, or
  `+0-0+0-0-0+0-0-0`?

Relaxation 5: Observationally Equivalent Specifications

- v is *valid for* Σ if there exists a map $f : C(v) \to \Sigma$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{obs}} f(q)$

$C(v) =$ pointed dependent-update-downclosed sets, for accessors
all dependent-update-downclosed sets, for updates



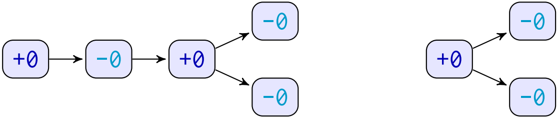


- Should this linearize to `+0-0-0+0-0+0-0-0`, or
  `+0-0+0-0-0+0-0-0`?
- These are *observationally equivalent*
  Cf. stuttering equivalence (Brookes 96)

Relaxation 5: Observationally Equivalent Specifications

- v is *valid for* Σ if there exists a map $f : C(v) \to \Sigma$ s.t.
  - $\forall p \in C(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in C(v).\ p \subseteq q$ implies $f(p) \leq_{\text{obs}} f(q)$

$C(v) =$ pointed dependent-update-downclosed sets, for accessors
all dependent-update-downclosed sets, for updates



- Should this linearize to `+0-0-0+0-0+0-0-0`, or
  `+0-0+0-0-0+0-0-0`?
- These are *observationally equivalent*
  Cf. stuttering equivalence (Brookes 96)
- Observational subsequence is a property of the *specification*

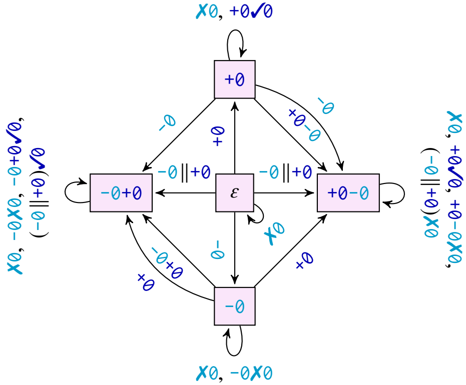
Safety: Summary

- v is *valid for* Σ if there exists a map $f : \mathcal{C}(v) \to \Sigma$ s.t.
  - $\forall p \in \mathcal{C}(v).\ p$ linearizes to $f(p)$
  - $\forall p, q \in \mathcal{C}(v).\ p \subseteq q$ implies $f(p) \leq_{\text{obs}} f(q)$

$\mathcal{C}(v) =$ pointed dependent-update-downclosed sets, for accessors
all dependent-update-downclosed sets, for updates

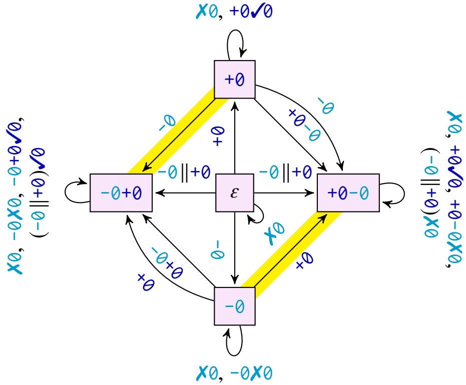
- Relaxations from linearizability:
  - Real time: Distributed system
  - Order after an accessor: Update serializability
  - Order between independent updates: Preserved Program Order
  - Linearize labels, not events: Punning
  - Quotient specification by observational equivalence: Stuttering

The Most General CRDT



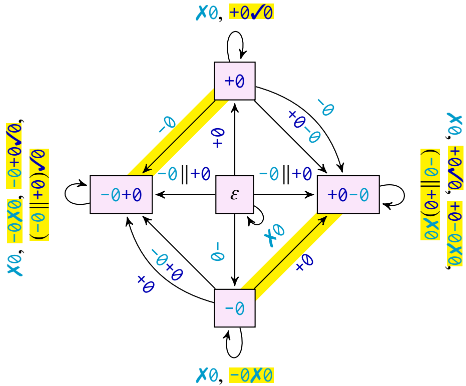
- What is the programmer model?
  Interacting with any CRDT implementation, for any specification
- Example for Set, with single +∅ and −∅
  LTS with labels = LPOs showing client history
  Maximal elements = new client actions

The Most General CRDT



- Contrast with linearizability
  - Updates may come out of order

The Most General CRDT



- Contrast with linearizability
  - Updates may come out of order
  - Accessors don't cause change of state

This talk: Definition of *safe* execution for CRDTs

In paper:

- ☺ Simulation-based characterization
  - ☺ *Most General CRDT*, expressed as Labelled Transition System
  - ☺ Compositionality and Substitutivity results
  - ☺ Validation of CRDT Graph built using CRDT sets
- ☺ Corner cases
  - ☺ Updates to one replica only $\Rightarrow$ linearizable
  - ☺ Permutation equivalence in spec $\Rightarrow \ldots$
- ☺ Validates all known CRDTs
  - ☺ Add-Wins Set (Shapiro/Preguiça/Baquero/Zawirski 2011)
  - ☺ Collaborative Text-Editing Protocol (Attiya/Burckhardt/Gotsman/Morrison/Yang/Zawirski)
- ☹ Validates *every possible* CRDTs
  - ☹ Def of CRDT does not mention sequential spec
  - ☺ Our def = proposal for meaning of CRDT