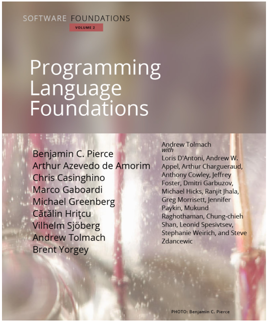
Coq, Chapar, and Coq Again

Hengfeng Wei

ICS, NJU

April 23, 2019

The Coq Proof Assistant



"Software Foundations"

Chapar: Certified Causally Consistent Distributed Key-Value Stores

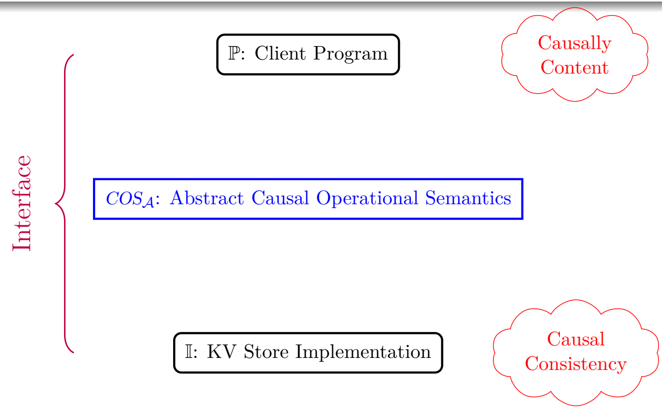
Mohsen Lesani Christian J. Bell Adam Chlipala
Massachusetts Institute of Technology, USA
{lesani, cjbell, adamc}@mit.edu

Chapar@POPL'2016

"A framework for modular verification of causal consistency for replicated key-value store implementations and their client programs."

Definition (Causally Content)

A client program is causally content if it avoids assertion failures when executed with $COS_{\mathcal{A}}$.



Definition (Causally Consistent)

A KV store impl. is causally consistent if it satisfies $COS_{\mathcal{A}}$.

Interface

$\mathbb{P}$: Client Program

$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics

$\mathbb{I}$: KV Store Implementation

$\mathbb{P}$:

get, put

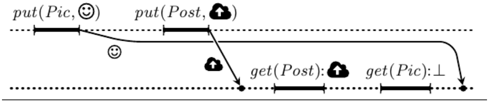
$COS_{\mathcal{A}}$:

get, put, update

$\mathbb{I}$:

get, put, guard, update

$\mathbb{P}$: Client Program



Program 1 (p_1): Uploading a photo and posting a status

$0 \rightarrow$ **Alice**

$put(Pic, ☺)$; ▷ uploads a new photo

$put(Post, ☁)$ ▷ announces it to her friends

$1 \rightarrow$ **Bob**

$post \leftarrow get(Post)$; ▷ checks Alice's post

$photo \leftarrow get(Pic)$; ▷ then loads her photo

$assert(post = ☁ \Rightarrow photo \neq \perp)$

$$assert(post \neq \perp \implies photo \neq \perp)$$

$\mathbb{P}$: Client Program

$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics

Abstract:

without referring to the details of specific implementations;

do not involving message passing.

Operational:

labelled transition system

executable (like TLA+)

Causal:

explicitly track happens-before dependencies

$$W_{\mathcal{A}} \ : \ N \rightarrow (S \times D \times U \times A \times M)$$

$c \ : C$	Clock
$d \ : D = \mathcal{P}(N \times C)$	Dependencies
$u \ : U = (K \times V \times D)^*$	Updates
$a \ : A = N \rightarrow C$	Applied
$m : M = K \rightarrow (V \times N \times C \times D)$	Store

PUT

$$\frac{\begin{array}{cc} u' = u \mathbin{+\!\!+} [(k,v,d)] & a' = a[n \mapsto a(n)+1] \\ m' = m[k \mapsto (v, n, |u'|, \emptyset)] & d' = d \cup \{(n, |u'|)\} \end{array}}{\begin{array}{c} W_{\mathcal{A}}[n \mapsto (put(k,v); s, d, u, a, m)] \\ \xrightarrow{n,\ |u'|\ \triangleright\ put(k,v)}_{\mathcal{A}} \\ W_{\mathcal{A}}[n \mapsto (s, d', u', a', m')] \end{array}}$$

GET

$$\frac{\begin{array}{c} m(k) = (v, n'', c'', d'') \\ d' = \begin{cases} d \cup \{(n'', c'')\} \cup d'' & \text{if } n'' \neq n_0 \\ d & \text{otherwise} \end{cases} \end{array}}{\begin{array}{c} W_{\mathcal{A}}[n \mapsto (x \leftarrow get(k); s, d, u, a, m)] \\ \xrightarrow{n'',\ c'',\ n\ \triangleright\ get(k):\ v}_{\mathcal{A}} \\ W_{\mathcal{A}}[n \mapsto (s[x := v], d', u, a, m)] \end{array}}$$

UPDATE

$$\frac{\begin{array}{cc} a_1(n_2) < |u_2| & u_2[\![a_1(n_2)]\!] = (k, v, d) \\ \bigwedge_{(n,c) \in d} c \leq a_1(n) & a_1' = a_1[n_2 \mapsto a_1(n_2) + 1] \\ \multicolumn{2}{c}{m_1' = m_1[k \mapsto (v, n_2, a_1'(n_2), d)]} \end{array}}{\begin{array}{c} W_{\mathcal{A}}[n_1 \mapsto (s_1, d_1, u_1, a_1, m_1)][n_2 \mapsto (s_2, d_2, u_2, a_2, m_2)] \\ \xrightarrow{n_2,\ a_1'(n_2),\ n_1\ \triangleright\ update(k,v)}_{\mathcal{A}} \\ W_{\mathcal{A}}[n_1 \mapsto (s_1, d_1, u_1, a_1', m_1')][n_2 \mapsto (s_2, d_2, u_2, a_2, m_2)] \end{array}}$$

ASSERTFAIL

$$\frac{}{C[n \mapsto (assertfail, d, u, a, m)] \xrightarrow{assertfail}_{\mathcal{A}} C[n \mapsto (skip, d, u, a, m)]}$$

GET

$$\frac{m(k) = (v, n'', c'', d'') \quad d' = \begin{cases} d \cup \{(n'', c'')\} \cup d'' & \text{if } n'' \neq n_0 \\ d & \text{otherwise} \end{cases}}{W_{\mathcal{A}}[n \mapsto (x \leftarrow get(k); s, d, u, a, m)] \xrightarrow{n'', c'', n \triangleright get(k) : v}_{\mathcal{A}} W_{\mathcal{A}}[n \mapsto (s[x := v], d', u, a, m)]}$$

$$W_{\mathcal{A}}[n \mapsto (x \leftarrow get(k); s, d, u, a, m)]$$

$$m(k) = (v, n'', c'', d'')$$

$$\xrightarrow{n'', c'', n \triangleright get(k) : v}_{\mathcal{A}}$$

$$d' = \begin{cases} d \cup \{(n'', c'')\} \cup d'' & \text{if } n'' \neq n_0 \\ d & \text{otherwise} \end{cases}$$

$$W_{\mathcal{A}}[n \mapsto (s[x := v], d', u, a, m)]$$

$\mathbb{P}$: Client Program

Causally Content

Verified Model Checker

$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics

Program 1 (p_1): Uploading a photo and posting a status

$0 \rightarrow$	**Alice**
$put(Pic, ☺)$;	▷ uploads a new photo
$put(Post, ☁)$	▷ announces it to her friends
$1 \rightarrow$	**Bob**
$post \leftarrow get(Post)$;	▷ checks Alice's post
$photo \leftarrow get(Pic)$;	▷ then loads her photo
$assert(post = ☁ \Rightarrow photo \neq \bot)$	

Put

$$\frac{u' = u +\!\!+ [(k, v, d)] \qquad a' = a[n \mapsto a(n) + 1] \qquad m' = m[k \mapsto (v, n, |u'|, \emptyset)] \qquad d' = d \cup \{(n, |u'|)\}}{W_{\mathcal{A}}[n \mapsto (put(k,v); s, d, u, a, m)] \xrightarrow{n,\ |u'|\ \triangleright\ put(k,v)}_{\mathcal{A}} W_{\mathcal{A}}[n \mapsto (s, d', u', a', m')]}$$

Get

$$\frac{m(k) = (v, n'', c'', d'') \qquad d' = \begin{cases} d \cup \{(n'', c'')\} \cup d'' & \text{if } n'' \neq n_0 \\ d & \text{otherwise} \end{cases}}{W_{\mathcal{A}}[n \mapsto (x \leftarrow get(k); s, d, u, a, m)] \xrightarrow{n'',\ c'',\ n\ \triangleright\ get(k):\ v}_{\mathcal{A}} W_{\mathcal{A}}[n \mapsto (s[x := v], d', u, a, m)]}$$

Update

$$\frac{a_1(n_2) < |u_2| \qquad u_2[\![a_1(n_2)]\!] = (k, v, d) \qquad \bigwedge_{(n,c) \in d} c \leq a_1(n) \qquad a_1' = a_1[n_2 \mapsto a_1(n_2) + 1] \qquad m_1' = m_1[k \mapsto (v, n_2, a_1'(n_2), d)]}{W_{\mathcal{A}}[n_1 \mapsto (s_1, d_1, u_1, a_1, m_1)][n_2 \mapsto (s_2, d_2, u_2, a_2, m_2)] \xrightarrow{n_2,\ a_1'(n_2),\ n_1\ \triangleright\ update(k,v)}_{\mathcal{A}} W_{\mathcal{A}}[n_1 \mapsto (s_1, d_1, u_1, a_1', m_1')][n_2 \mapsto (s_2, d_2, u_2, a_2, m_2)]}$$

AssertFail

$$\frac{}{C[n \mapsto (assertfail, d, u, a, m)] \xrightarrow{assertfail}_{\mathcal{A}} C[n \mapsto (skip, d, u, a, m)]}$$

$\mathbb{P}$: Client Program

Causally Content

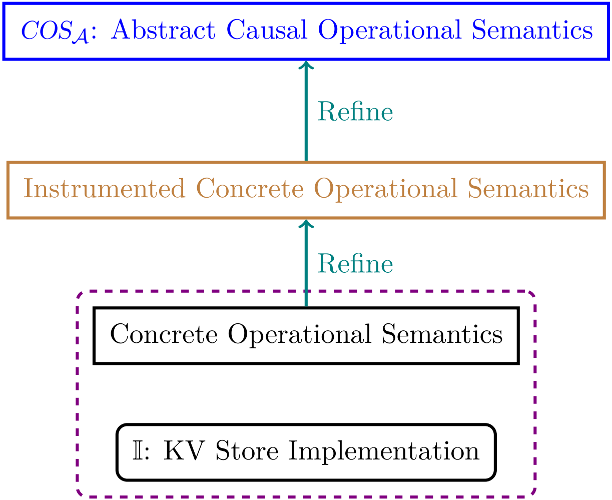
Verified Model Checker

Interface

$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics

$\mathbb{I}$: KV Store Implementation

Causal Consistency


$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics
Refine
Instrumented Concrete Operational Semantics
Refine
Concrete Operational Semantics
$\mathbb{I}$: KV Store Implementation

Concrete Operational Semantics

$\mathbb{I}$: KV Store Implementation

$$W_{\mathcal{C}} \; : \; H \times T$$

h	$: H = N \to (S \times \text{State}(V))$	Hosts
t	$: T = \mathcal{P}(M)$	Transit
m	$: M = N \times K \times V \times \text{Update}(V)$	Message
σ	$: \text{State}(V)$	Alg State
u	$: \text{Update}(V)$	Alg Update

Put

$$\frac{\mathsf{put}(V, n, \sigma, k, v) \rightsquigarrow^* (\sigma', u) \quad t' = t \cup \{(n', k, v, u) \mid n' \in N \setminus \{n\}\}}{(h[n \mapsto (put(k, v); s, \sigma)], t) \xrightarrow{n \triangleright\, put(k,v)}_{\mathcal{C}(\mathbb{I})} (h[n \mapsto (s, \sigma')], t')}$$

Get

$$\frac{\mathsf{get}(V, n, \sigma, k) \rightsquigarrow^* (v, \sigma')}{(h[n \mapsto (x \leftarrow get(k); s, \sigma)], t) \xrightarrow{n \triangleright\, get(k):\, v}_{\mathcal{C}(\mathbb{I})} (h[n \mapsto (s[x := v], \sigma')], t)}$$

Update

$$\frac{\mathsf{guard}(V, n, \sigma, k, v, u) \rightsquigarrow^* true \quad \mathsf{update}(V, n, \sigma, k, v, u) \rightsquigarrow^* \sigma'}{(h[n \mapsto (s, \sigma)], t \cup \{(n, k, v, u)\}) \xrightarrow{n \triangleright\, update(k,v)}_{\mathcal{C}(\mathbb{I})} (h[n \mapsto (s, \sigma')], t)}$$

AssertFail

$$\frac{}{(h[n \mapsto (assertfail, \sigma)], t) \xrightarrow{assertfail}_{\mathcal{C}(\mathbb{I})} (h[n \mapsto (skip, \sigma)], t)}$$

$$\frac{\text{GET} \quad \mathsf{get}(V, n, \sigma, k) \rightsquigarrow^* (v, \sigma')}{(h[n \mapsto (x \leftarrow get(k); s, \sigma)], t) \xrightarrow{n \triangleright\, get(k):\, v}_{\mathcal{C}(\mathbb{I})} \big(h[n \mapsto (s[x := v], \sigma')], t\big)}$$

Parametric on the implementation $\mathbb{I}$

$$\text{get}(V, n, \sigma, k) \rightsquigarrow^* (v, \sigma)$$

$$\xrightarrow{n \,\triangleright\, get(k):v}_{\mathcal{C}(\mathbb{I})}$$

Operational: Model the executions of the implementation $\mathbb{I}$

$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics

Instrumented Concrete Operational Semantics

Concrete Operational Semantics

Parametric on the implementation $\mathbb{I}$

$\rightarrow_{\mathcal{I}(\mathbb{I})}$ is similar to non-instrumented $\rightarrow_{\mathcal{C}(\mathbb{I})}$

"Uniquely identify put *operations to track causal dependencies between them."*

$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics

Well-Reception ↑ Refine

Instrumented Concrete Operational Semantics

Concrete Operational Semantics

Theorem (Sufficiency of Well-Reception)

Every well-receptive implementation is causally consistent.

Definition (Well-Reception)

An implementation is *well-receptive* iff there exists a *function* Rec for the implementation such that the four conditions InitCond, StepCond, CauseCond, and SeqCond are satisfied.

$$
\begin{aligned}
&\mathsf{WellRec}(\mathbb{I}) \triangleq \\
&\quad \mathit{let}\ (\mathsf{State}, \_, \_, \_, \_, \_, \_) = \mathbb{I}\ \mathit{in} \\
&\quad \exists \mathsf{Rec}\colon (\mathsf{State}(IV), N) \to C\colon \\
&\quad \mathit{let}\ \mathsf{Rec}'(W, n', n) = \\
&\qquad \mathit{let}\ (H[n' \mapsto (\_, \sigma, \_)], \_) = W\ \mathit{in} \\
&\qquad \mathsf{Rec}(\sigma, n)\ \mathit{in} \\
&\quad \mathsf{InitCond}(\mathbb{I}, \mathsf{Rec}') \wedge \mathsf{StepCond}(\mathbb{I}, \mathsf{Rec}') \wedge \mathsf{CauseCond}(\mathbb{I}, \mathsf{Rec}') \\
&\quad \wedge\ \mathsf{SeqCond}(\mathbb{I})
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{InitCond}(\mathbb{I}, \mathsf{Rec}') \triangleq \forall p, n, n'\colon \\
&\quad \mathsf{Rec}'(W_{\mathcal{I}0}(p), n, n') = 0
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{StepCond}(\mathbb{I}, \mathsf{Rec}') \triangleq \forall p, h_{\mathcal{I}}, W_{\mathcal{I}}, l_{\mathcal{I}}, W'_{\mathcal{I}}\colon \\
&\quad (W_{\mathcal{I}0}(p) \xrightarrow{h_{\mathcal{I}}}{}^{*}_{\mathcal{I}(\mathbb{I})} W_{\mathcal{I}} \ \wedge\ W_{\mathcal{I}} \xrightarrow{l_{\mathcal{I}}}_{\mathcal{I}(\mathbb{I})} W'_{\mathcal{I}}) \Rightarrow \\
&\quad \begin{cases}
\textsc{Case}\ l_{\mathcal{I}} = n, \_ \triangleright put(\_, \_, \_)\colon \_, \_ \\
\quad \mathsf{Rec}'(W'_{\mathcal{I}}, n, n) = \mathsf{Rec}'(W_{\mathcal{I}}, n, n) + 1 \wedge \\
\quad \forall n'\colon n' \neq n \Rightarrow \mathsf{Rec}'(W'_{\mathcal{I}}, n, n') = \mathsf{Rec}'(W_{\mathcal{I}}, n, n') \\
\textsc{Case}\ l_{\mathcal{I}} = n, \_ \triangleright get(\_, \_)\colon \_, \_ \\
\quad \forall n'\colon \mathsf{Rec}'(W'_{\mathcal{I}}, n, n') = \mathsf{Rec}'(W_{\mathcal{I}}, n, n') \\
\textsc{Case}\ l_{\mathcal{I}} = n', c', n \triangleright update(\_, \_, \_, \_)\colon \_ \\
\quad \mathsf{Rec}'(W_{\mathcal{I}}, n, n') + 1 = c' \wedge \\
\quad \mathsf{Rec}'(W'_{\mathcal{I}}, n, n') = \mathsf{Rec}'(W_{\mathcal{I}}, n, n') + 1 \wedge \\
\quad \forall n''\colon n'' \neq n' \Rightarrow \mathsf{Rec}'(W'_{\mathcal{I}}, n, n'') = \mathsf{Rec}'(W_{\mathcal{I}}, n, n'')
\end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{CauseCond}(\mathbb{I}, \mathsf{Rec}') \triangleq \forall p, h_{\mathcal{I}}, W_{\mathcal{I}}, l_{\mathcal{I}}, W'_{\mathcal{I}}, l''_{\mathcal{I}}\colon \\
&\quad (W_{\mathcal{I}0}(p) \xrightarrow{h_{\mathcal{I}}}{}^{*}_{\mathcal{I}(\mathbb{I})} W_{\mathcal{I}} \ \wedge\ W_{\mathcal{I}} \xrightarrow{l_{\mathcal{I}}}_{\mathcal{I}(\mathbb{I})} W'_{\mathcal{I}} \\
&\quad\ \wedge\ \mathsf{LIsUpdate}(l_{\mathcal{I}}) \wedge \\
&\quad\ \mathit{let}\ \_, \_, n \triangleright update(\_, \_, \_, m)\colon \_ = l_{\mathcal{I}} \\
&\qquad (\_, \_, \_, \_, \_, \_, l'_{\mathcal{I}}) = m\ \mathit{in} \\
&\quad\ \mathsf{LIsPut}(l''_{\mathcal{I}}) \ \wedge\ l''_{\mathcal{I}} \curvearrowright_{h_{\mathcal{I}}} l'_{\mathcal{I}}) \Rightarrow \\
&\quad \mathit{let}\ n'', c'' \triangleright put(\_, \_, \_)\colon \_, \_ = l''_{\mathcal{I}}\ \mathit{in} \\
&\quad \mathsf{Rec}'(W_{\mathcal{I}}, n, n'') \geq c''
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{SeqCond}(\mathbb{I}) \triangleq \forall p, h_{\mathcal{C}}, W_{\mathcal{C}}\colon \\
&\quad W_{\mathcal{C}0}(p) \xrightarrow{h_{\mathcal{C}}}{}^{*}_{\mathcal{C}(\mathbb{I})} W_{\mathcal{C}} \ \Rightarrow\ \exists W'_{\mathcal{S}}\colon W_{\mathcal{S}0} \xrightarrow{\mathsf{Eff}(h_{\mathcal{C}})}{}^{*}_{\mathcal{S}} W'_{\mathcal{S}}
\end{aligned}
$$

Causal memory: definitions, implementation, and programming

Mustaque Ahamad[1], Gil Neiger[2], James E. Burns[1,**], Prince Kohli[1], Phillip W. Hutto[3,***]

[1] College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA
[2] Software Technology Lab, Intel Corporation, JF3-206, 2111 N.E. 25th Avenue, Hillsboro, OR 97124-5961, USA
[3] 609 Virginia Avenue NE, Atlanta, GA 30306, USA

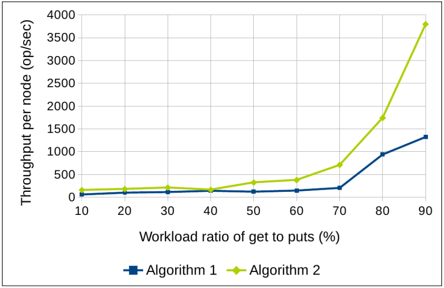
DC'1995

Stronger Semantics for Low-Latency Geo-Replicated Storage

Wyatt Lloyd[⋆], Michael J. Freedman[⋆], Michael Kaminsky[†], and David G. Andersen[‡]
[⋆]Princeton University, [†]Intel Labs, [‡]Carnegie Mellon University

NSDI'2013

OCaml


Throughput per node (op/sec)
4000
3500
3000
2500
2000
1500
1000
500
0
10
20
30
40
50
60
70
80
90
Workload ratio of get to puts (%)
Algorithm 1
Algorithm 2

Chapar: Certified Causally Consistent Distributed Key-Value Stores

Mohsen Lesani Christian J. Bell Adam Chlipala
Massachusetts Institute of Technology, USA
{lesani, cjbell, adamc}@mit.edu

Chapar@POPL'2016

Chapar for CC variants

Causal Consistency: Beyond Memory

Matthieu Perrin   Achour Mostefaoui   Claude Jard
LINA – University of Nantes, Nantes, France
[firstname.lastname]@univ-nantes.fr

PPoPP'2016

Chapar for Op-based CRDT

Verifying Strong Eventual Consistency in Distributed Systems

VICTOR B. F. GOMES, University of Cambridge, UK
MARTIN KLEPPMANN, University of Cambridge, UK
DOMINIC P. MULLIGAN, University of Cambridge, UK
ALASTAIR R. BERESFORD, University of Cambridge, UK

OOPSLA'2017

Chapar for State-based CRDT

Formal Specification and Verification of CRDTs

Peter Zeller, Annette Bieniusa, and Arnd Poetzsch-Heffter

University of Kaiserslautern, Germany
{p_zeller,bieniusa,poetzsch}@cs.uni-kl.de

FORTE'2014

Coq for the equivalence
between Op-based CRDT and State-based CRDT

A comprehensive study of Convergent and Commutative Replicated Data Types

Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski

TR'2011

Coq for the "$vis + ar$" framework

syncope: Automatic Enforcement of Distributed Consistency Guarantees

Kiarash Rahmani
Purdue University, rahmank@purdue.edu

Gowtham Kaki
Purdue

Suresh Jagannathan
Purdue University

TR'2017


THANK YOU