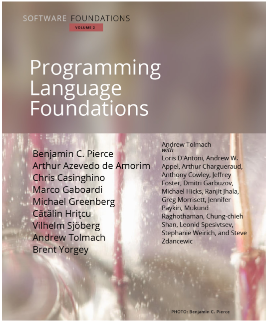
Coq, Chapar, and Coq Again

Hengfeng Wei

ICS, NJU

April 28, 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

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."

$\mathbb{P}$: Client Program

$\mathbb{I}$: KV Store Implementation

$\mathbb{P}$: Client Program

Causally Content

$\mathbb{I}$: KV Store Implementation

Causal Consistency

$\mathbb{P}$: Client Program

Causally Content

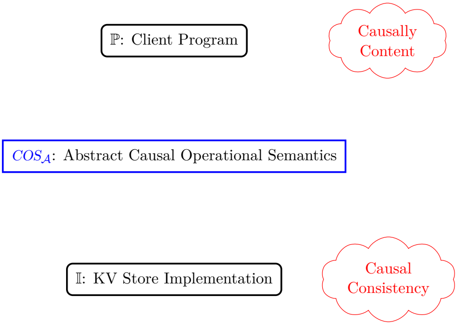
$COS_{\mathcal{A}}$: Abstract Causal Operational Semantics

$\mathbb{I}$: KV Store Implementation

Causal Consistency

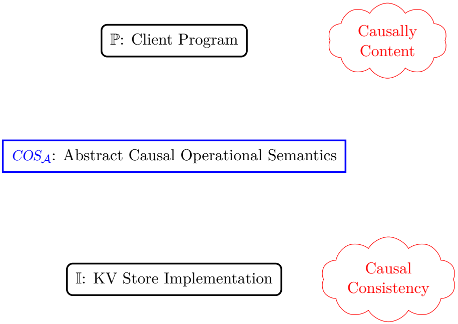
Definition (Causally Content)

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



Definition (Causally Content)

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

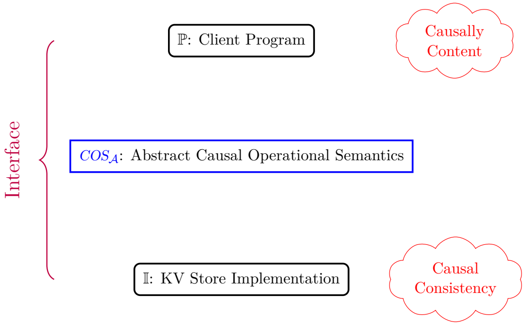


Definition (Causally Consistent)

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

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

Interface

$\mathbb{P}$: Client Program

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

$\mathbb{I}$: KV Store Implementation

$\mathbb{P}$:

get, put

$COS_{\mathcal{A}}$:

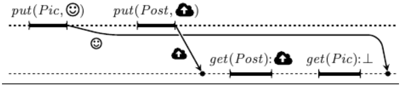
get, put, update

$\mathbb{I}$:

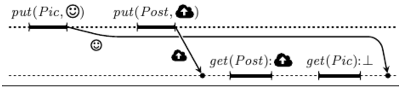
get, put, guard, update

$\mathbb{P}$: Client Program

$\mathbb{P}$: Client Program



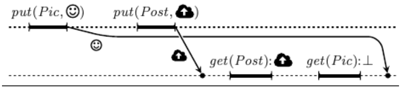
$\mathbb{P}$: Client Program



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

$\mathbb{P}$: 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

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

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

$\mathbb{P}$: Client Program

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

$\mathbb{P}$: Client Program

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

Abstract:

without referring to the details of specific implementations;

do not involving message passing.

$\mathbb{P}$: 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+)

$\mathbb{P}$: 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/maintain happens-before dependencies

PUT

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

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

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

$c \ : C$ Clock

$c \; : C$ Clock

$a \; : A = N \to C$ Applied

$c \ : C$	Clock
$a \ : A = N \to C$	Applied
$d \ : D = \mathcal{P}(N \times C)$	Dependencies

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

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


THANK YOU