Consistency Types for Replicated Data
in a Higher-order Distributed Programming Language

XIN ZHAO PHILIPP HALLER

```
def numberOfSeats(event: ID): Int = {
  fastRead(event + "num_seats")
}

def book(event: ID) = {
  val remaining: Int = numberOfSeats(event)
  if (remaining >= event.orderNum)
    paymentProcess(event)
  else display("Out of seats!")
}
```

Available read from a single replica

Result of available but inconsistent operation!

Consistent state update!

Problem: update of consistent state based on inconsistent information!

Solution

```
def numberOfSeats(event: ID): Int@con = {
  consistentRead(event + "num_seats")
}

def book(event: ID) = {
  val remaining: Int@con = numberOfSeats(event)
  if (remaining >= event.orderNum)
    paymentProcess(event)
  else display("Out of seats!")
}
```

Must strengthen consistency!

Highlights:

- A ***higher-order static consistency-typed*** language with replicated data types
- Supports shared data among ***multiple clients***
- ***Statically enforces non-interference*** between data types with weaker consistency and data types with stronger consistency
- Proofs of ***type soundness***, ***non-interference*** and the ***consistency properties***

ℓ	::=	loc | con | oac | ava	label
t	::=	$x \mid v \mid t[\ell] \mid t \bigoplus t \mid t \text{ op } t \mid t\ t \mid \text{if } x \text{ then } \{ s \} \text{ else } \{ t \}$	
		$\mid \text{ref}_\ell(t, id) \mid \text{await}(id) \mid !t \mid t := t$	
		$\mid \text{FlexRead}_\ell(t) \mid \text{FlexWrite}_\ell(t, t)$	terms
r	::=	d | true | false | $(\lambda^\ell x : \tau.\ t)$ | unit | o	raw value
id	::=	(ℓ, n) where $n \in \mathbb{N}$	identifier
v	::=	$r_\ell \mid \text{duplicated}(t)$	labeled value
τ	::=	$\text{Bool}_\ell \mid \text{Unit}_\ell \mid \text{Lat}_\ell \mid \text{Ref}_\ell\ \tau \mid \tau \xrightarrow{\ell}_\ell \tau$	types
$\bigoplus$	::=	$\vee \mid \wedge$	meet and join
op	::=	$\preceq \mid \prec$	order operations

Figure 6 Syntax of CTRD core language