

Generalizing the Correctness of Transactional Memory

Rachid Guerraoui Thomas A. Henzinger
Michał Kapałka Vasu Singh

EPFL

parametrized opacity

correctness condition for TMs

Concurrency for dummies

Transactional Memory (TM)

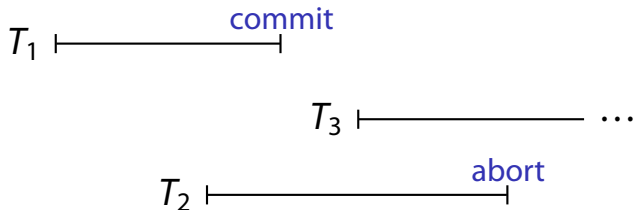
Thread 1

```
atomic {  
    acc1 . credit(5)  
    acc2 . debit(5)  
}
```

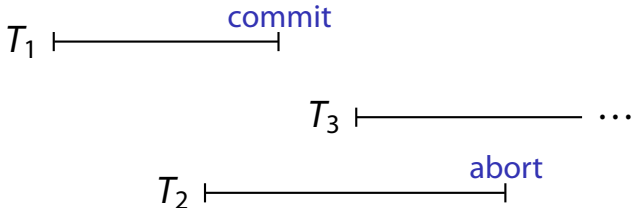
Thread 2

```
atomic {  
    sum = acc1 . balance()  
    sum += acc2 . balance()  
}
```

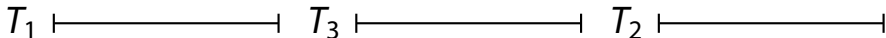
When Is a TM correct?



When Is a TM correct?



"looks like"



A Correctness Condition for TMs

Opacity:

- All transactions – as if instantaneous
- Aborted transactions – never visible
- Every transaction observes consistent state

Thread 1

```
atomic {  
    v = x.read()  
    y.write(1)  
}
```

Thread 2

```
v = x.read()  
y.write(2)
```

?

strong atomicity

weak atomicity

[Martin et al. 2006]

strong atomicity

SGLA

[Menon et al. 2008]

(S)SS

[Spear et al. 2008]

...

weak atomicity

[Martin et al. 2006]

strong atomicity

SGLA

[Menon et al. 2008]

(S)SS

[Spear et al. 2008]

...

weak atomicity

[Martin et al. 2006]

Thread 1

atomic {

`x . write(1)`

`x . write(2)`

}

Thread 2

`x . read` → 0 or 2

correct

Thread 1

atomic {

`x.write(1)`

`x.write(2)`

}

Thread 2

`x.read` → **1**

incorrect

Thread 1

```
atomic {  
  x.write(1)  
  
  x.read → 1  
}
```

Thread 2

```
x.write(2)
```

correct

Thread 1

```
atomic {  
    x.write(1)  
  
    x.read → 2  
}
```

Thread 2

```
x.write(2)
```

incorrect

Thread 1

```
atomic {  
    x.write(1)  
    y.write(1)  
}
```

Thread 2

```
x.read → 1
```

correct

Thread 1

```
atomic {  
  x.write(1)  
  y.write(1)  
}
```

Thread 2

y.read → 0

correct

Thread 1

```
atomic {  
  x.write(1)  
  y.write(1)  
}
```

Thread 2

```
x.read → 1  
y.read → 0
```

?

Thread 1

```
atomic {  
  x.write(1)  
  y.write(1)  
}
```

Thread 2

```
x.read → 1  
y.read → 0
```

Milo et al.: **correct**

Larus & Rajwar: **incorrect**

Thread 1

```
atomic {  
  x.write(1)  
  y.write(1)  
}
```

Thread 2

```
x.read → 1  
y.read → 0
```

it depends...

Thread 1

Thread 2

Thread 3

Thread 4

atomic {

`x.write(1)` `x.write(2)`

}

`x.read` → 1

`x.read` → **2**

`x.read` → 2

`x.read` → **1**

it depends...

Shared Memory

Thread 1

`x.write(1)`

`y.write(1)`

Thread 2

`x.read` → 1

`y.read` → 0

Thread 3

`y.read` → 5

Thread 1

```
atomic {  
  x.write(1)  
  y.write(1)  
}
```

Thread 2

```
x.read → 1  
y.read → 0
```

it depends...

Thread 1

Thread 2

Thread 3

Thread 4

atomic {

`x.write(1)`

`x.write(2)`

}

`x.read` → 1

`x.read` → **2**

`x.read` → 2

`x.read` → **1**

it depends...

Goal: capture the intuition behind strong atomicity
→ **parametrized opacity**

opacity

for transactions

memory model

for non-transactional operations

Goal: capture the intuition behind strong atomicity
→ **parametrized opacity**

opacity

for transactions

memory model

for non-transactional operations

Question: is parametrized opacity expensive?

Challenges...

1

Memory model \rightarrow *M*

2

Opacity

+

Memory model *M*

Parametrized Opacity

History H , memory model $M = (R, \tau)$

H ensures **opacity parametrized by M** if:

Parametrized Opacity

History H , memory model $M = (R, \tau)$

H ensures **opacity parametrized by M** if:

$$1 \quad H \rightarrow \tau(H)$$

Parametrized Opacity

History H , memory model $M = (R, \tau)$

H ensures **opacity parametrized by M** if:

1 $H \rightarrow \tau(H)$

2 $\exists <_{\text{trans}}$ – total order of transactions

Parametrized Opacity

History H , memory model $M = (R, \tau)$

H ensures **opacity parametrized by M** if:

- 1 $H \rightarrow \tau(H)$
- 2 $\exists <_{\text{trans}}$ – total order of transactions
- 3 $\exists \prec_{\text{op}}$ – determined by M (per-thread)

Parametrized Opacity

History H , memory model $M = (R, \tau)$

H ensures **opacity parametrized by M** if:

- 1 $H \rightarrow \tau(H)$
- 2 $\exists <_{\text{trans}}$ – total order of transactions
- 3 $\exists \prec_{\text{op}}$ – determined by M (per-thread)
- 4 \prec_{rt} – real-time order


Real-Time Order

Thread 1

```
x . write(1)  
atomic {  
    y . write(2)  
}
```

Thread 2

```
atomic {  
    x . read()  
}  
y . read()
```



Parametrized Opacity

History H , memory model $M = (R, \tau)$

H ensures **opacity parametrized by M** if:

- 1 $H \rightarrow \tau(H)$
- 2 $\exists \prec_{\text{trans}}$ – total order of transactions
- 3 $\exists \prec_{\text{op}}$ – determined by M (per-thread)
- 4 \prec_{rt} – real-time order
- 5 For every thread, there exists a **sequential history** that respects the above relations and is **legal**.

Thread 1

```
atomic {  
  x.write(1)  
  y.write(1)  
}
```

Thread 2

```
x.read → 1  
y.read → 0
```

Milo et al.: **correct**

Larus & Rajwar: **incorrect**

parametrized opacity: both are right!

Question:

is parametrized opacity expensive to implement?

Question:

is parametrized opacity expensive to implement?

- Most memory models: instrumentation of non-tx code necessary
- All memory models: need CAS to write to memory
- ...