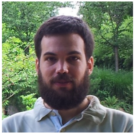
Abstract Disjointness and Abstract Atomicity

Pedro da Rocha Pinto

Thomas Dinsdale-Young

Philippa Gardner

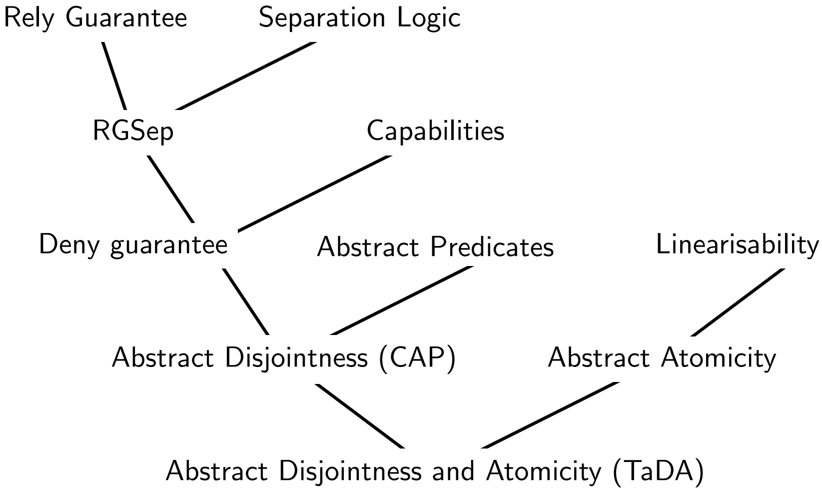
Abstract Disjointness and Abstract Atomicity








Abstract Disjointness and Abstract Atomicity



Implementation of a Counter

```
function read(x) {
  r := [x];
  return r;
}
```

// Supports concurrent reads,
// increments and weak increments

```
function incr(x) {
  do {
    r := [x];
    b := CAS(x, r, r + 1);
  } while (b = 0);
}
```

// Supports concurrent reads
// and increments

```
function wkincr(x) {
  r := [x];
  [x] := r + 1;
}
```

// Supports concurrent reads
// but not concurrent increments

Sequential Specification

Abstract predicate $\mathsf{C}(\mathtt{x}, n)$ describes a counter with value $n \in \mathbb{N}$ at address x.

$$\{\mathsf{C}(\mathtt{x}, n)\}\ \mathtt{read}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n) \land \mathsf{ret} = n\}$$
$$\{\mathsf{C}(\mathtt{x}, n)\}\ \mathtt{incr}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n+1)\}$$
$$\{\mathsf{C}(\mathtt{x}, n)\}\ \mathtt{wkincr}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n+1)\}$$

Pros: The specification captures sequential behaviour.

Cons: The specification does not support concurrency.

Abstract Disjoint Concurrency

The disjoint concurrency rule from concurrent separation logic:

$$\frac{\{P_1\}\ \mathbb{C}_1\ \{Q_1\} \quad \{P_2\}\ \mathbb{C}_2\ \{Q_2\}}{\{P_1 * P_2\}\ \mathbb{C}_1 \parallel \mathbb{C}_2\ \{Q_1 * Q_2\}}$$

Abstract Disjoint Concurrency

The disjoint concurrency rule from concurrent separation logic:

$$\frac{\{P_1\}\ \mathbb{C}_1\ \{Q_1\} \quad \{P_2\}\ \mathbb{C}_2\ \{Q_2\}}{\{P_1 * P_2\}\ \mathbb{C}_1 \parallel \mathbb{C}_2\ \{Q_1 * Q_2\}}$$

But the counter cannot be replicated:

$\mathsf{C}(\mathtt{x}, n) \implies \mathsf{C}(\mathtt{x}, n) * \mathsf{C}(\mathtt{x}, n)$ does not hold.

Abstract Disjoint Specification

Extend the counter predicate to include permissions:

$$\mathsf{C}(\mathtt{x}, n, \pi_1 + \pi_2) \iff \mathsf{C}(\mathtt{x}, n, \pi_1) * \mathsf{C}(\mathtt{x}, n, \pi_2)$$

with $\pi_1, \pi_2 \in (0, 1]$

A concurrent specification:

$$\left\{\mathsf{C}(\mathtt{x}, n, \pi)\right\}\ \mathtt{read}(\mathtt{x})\ \left\{\mathsf{C}(\mathtt{x}, n, \pi) \land \mathsf{ret} = n\right\}$$
$$\left\{\mathsf{C}(\mathtt{x}, n, 1)\right\}\ \mathtt{incr}(\mathtt{x})\ \left\{\mathsf{C}(\mathtt{x}, n+1, 1)\right\}$$
$$\left\{\mathsf{C}(\mathtt{x}, n, 1)\right\}\ \mathtt{wkincr}(\mathtt{x})\ \left\{\mathsf{C}(\mathtt{x}, n+1, 1)\right\}$$

Pros: The specification allows concurrent reads.

Cons: The specification enforces sequential increments.

Abstract Disjoint Specification

Adapt counter predicate to split counter values:

$$\mathsf{C}(\mathtt{x}, n_1 + n_2, \pi_1 + \pi_2) \iff \mathsf{C}(\mathtt{x}, n_1, \pi_1) * \mathsf{C}(\mathtt{x}, n_2, \pi_2)$$

Another concurrent specification:

$$\begin{gathered}
\{\mathsf{C}(\mathtt{x}, n, \pi)\}\ \mathtt{read}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n, \pi) \wedge \mathsf{ret} \geq n\} \\
\{\mathsf{C}(\mathtt{x}, n, 1)\}\ \mathtt{read}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n, 1) \wedge \mathsf{ret} = n\} \\
\{\mathsf{C}(\mathtt{x}, n, \pi)\}\ \mathtt{incr}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n+1, \pi)\} \\
\{\mathsf{C}(\mathtt{x}, n, 1)\}\ \mathtt{wkincr}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n+1, 1)\}
\end{gathered}$$

Pros: The specification allows concurrent reads and increments.

Cons: The specification enforces sequential weak increments. The specification does not enforce that the reads increase in value.

Client Program: Ticket Lock

The following ticket lock cannot be verified without atomic counter operations:

```
function lock(z) {
  t := incr(z.ticket);          // Get the ticket
  do {
    n := read(z.next);
  } while (n < t);              // Loop until it is its turn
}

function unlock(z) {
  wkincr(z.next);               // Move to the next number
}
```

Abstract Atomicity

Can sequential specifications be used as atomic specifications?

$$\{\mathsf{C}(\mathtt{x}, n)\}\ \mathtt{read}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n) \land \mathsf{ret} = n\}$$
$$\{\mathsf{C}(\mathtt{x}, n)\}\ \mathtt{incr}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n+1)\}$$
$$\{\mathsf{C}(\mathtt{x}, n)\}\ \mathtt{wkincr}(\mathtt{x})\ \{\mathsf{C}(\mathtt{x}, n+1)\}$$

The answer is negative. The weak increment is not atomic when other increments occur.

Abstract Atomic Specification

Introduce atomic triples:

$$\mathbb{\forall} n.\, \langle \mathsf{C}(\mathtt{x}, n) \rangle\ \mathtt{read}(\mathtt{x})\ \langle \mathsf{C}(\mathtt{x}, n) \wedge \mathsf{ret} = n \rangle$$
$$\mathbb{\forall} n.\, \langle \mathsf{C}(\mathtt{x}, n) \rangle\ \mathtt{incr}(\mathtt{x})\ \langle \mathsf{C}(\mathtt{x}, n+1) \rangle$$
$$\langle \mathsf{C}(\mathtt{x}, n) \rangle\ \mathtt{wkincr}(\mathtt{x})\ \langle \mathsf{C}(\mathtt{x}, n+1) \rangle$$

Pros:
The specification allows concurrent reads and increments.
The specification allows concurrent reads with one weak increment.
The specifications are given relative to an abstraction, hence independent of the other operations.

Specifying a Client Program: Ticket Lock

A ticket lock specification:

$$\vdash \left\{\mathsf{Locked}(\mathtt{x})\right\}\ \mathtt{unlock}(\mathtt{x})\ \left\{\mathsf{emp}\right\}$$

$$\vdash \left\{\mathsf{isLock}(\mathtt{x})\right\}\ \mathtt{lock}(\mathtt{x})\ \left\{\mathsf{isLock}(\mathtt{x}) * \mathsf{Locked}(\mathtt{x})\right\}$$

$$\mathsf{isLock}(x) \iff \mathsf{isLock}(x) * \mathsf{isLock}(x)$$

$$\mathsf{Locked}(x) * \mathsf{Locked}(x) \implies \mathsf{false}$$

Pros: A specification of ticket lock whose implementation is based on the atomic counter commands.

Cons: The specification is not atomic. For this, we need helping (on-going).

Proving the Implementation

We use TaDA, a logic for time and data abstraction.

Abstract Predicate Interpretation

$$\mathsf{C}(x, n) \quad \triangleq \quad \exists c.\, \mathbf{Counter}_c(x, n) * [\mathrm{G}]_c$$

Transition System

$$\mathrm{G} : \forall n, m \in \mathbb{N}.\, n \rightsquigarrow m$$

Region Interpretation

$$I(\mathbf{Counter}_c(x, n)) \triangleq \quad x \mapsto n$$

Proving the Implementation: wkincr

$\langle \; \mathsf{C}(\mathtt{x}, n) \; \rangle$

abstract; quantify c

$\langle \; \mathbf{Counter}_c(\mathtt{x}, n) * [\mathrm{G}]_c \; \rangle$

$\langle \; \mathbf{Counter}_c(\mathtt{x}, n+1) * [\mathrm{G}]_c \; \rangle$

$\langle \; \mathsf{C}(\mathtt{x}, n+1) \; \rangle$

Proving the Implementation: wkincr

$\langle\ \mathsf{C}(\mathtt{x}, n)\ \rangle$

abstract; quantify c

$\langle\ \mathbf{Counter}_c(\mathtt{x}, n) * [\mathrm{G}]_c\ \rangle$

make atomic

$c : n \rightsquigarrow n+1 \vdash$

$\{\ \exists n.\, \mathbf{Counter}_c(\mathtt{x}, n) * c \Mapsto \blacklozenge\ \}$

$\{\ c \Mapsto (\mathtt{r}, \mathtt{r}+1)\ \}$

$\langle\ \mathbf{Counter}_c(\mathtt{x}, n+1) * [\mathrm{G}]_c\ \rangle$

$\langle\ \mathsf{C}(\mathtt{x}, n+1)\ \rangle$

Proving the Implementation: wkincr

abstract; quantify c

make atomic

$$\langle\ \mathsf{C}(\mathtt{x}, n)\ \rangle$$

$$\langle\ \mathbf{Counter}_c(\mathtt{x}, n) * [\mathrm{G}]_c\ \rangle$$

$$c : n \rightsquigarrow n + 1 \vdash$$

$$\{\ \exists n.\, \mathbf{Counter}_c(\mathtt{x}, n) * c \Mapsto \blacklozenge\ \}$$

$$\mathtt{r} := [\mathtt{x}];$$

$$\{\ \mathbf{Counter}_c(\mathtt{x}, \mathtt{r}) * c \Mapsto \blacklozenge\ \}$$

$$\{\ c \Mapsto (\mathtt{r}, \mathtt{r} + 1)\ \}$$

$$\langle\ \mathbf{Counter}_c(\mathtt{x}, n + 1) * [\mathrm{G}]_c\ \rangle$$

$$\langle\ \mathsf{C}(\mathtt{x}, n + 1)\ \rangle$$

Proving the Implementation: wkincr

$\langle\ \mathsf{C}(\mathtt{x}, n)\ \rangle$

abstract; quantify c

$\langle\ \mathbf{Counter}_c(\mathtt{x}, n) * [\mathrm{G}]_c\ \rangle$

make atomic

$c : n \rightsquigarrow n+1 \vdash$

$\{\ \exists n.\, \mathbf{Counter}_c(\mathtt{x}, n) * c \Rrightarrow \blacklozenge\ \}$

$\mathtt{r} := [\mathtt{x}];$

$\{\ \mathbf{Counter}_c(\mathtt{x}, \mathtt{r}) * c \Rrightarrow \blacklozenge\ \}$

update region

$\langle\ \mathtt{x} \mapsto \mathtt{r}\ \rangle$

$[\mathtt{x}] := \mathtt{r} + 1;$

$\langle\ \mathtt{x} \mapsto \mathtt{r} + 1\ \rangle$

$\{\ c \Rrightarrow (\mathtt{r}, \mathtt{r} + 1)\ \}$

$\langle\ \mathbf{Counter}_c(\mathtt{x}, n+1) * [\mathrm{G}]_c\ \rangle$

$\langle\ \mathsf{C}(\mathtt{x}, n+1)\ \rangle$

Conclusions

In TaDA, we have

- Introduced atomic triples.
- Combined atomic and non-atomic specifications.
- Provided a notion of atomicity which depends on the level of abstraction at which we view the code.
- Looked at other examples: e.g. lock, MCAS library, deque.
- Started the investigation of helping.

Future: Abstract Disjointness and Abstract Atomicity