Abstracting Change

The Next 100 Separation Logics

Matthew Parkinson
Microsoft Research - Cambridge
[Ishtiaq, O’Hearn POPL 2001]
[O’Hearn, Reynolds, Yang CSL 2001]
[Reynolds LICS 2002]

SEPARATION
Separation logic’s secret sauce

• Assertion about partial states
  – Intuitively, they are about two things
    • Access rights
    • Values in memory

• Specifications contain everything you need
  - tight interpretation
Framing

Access to x.f
x.f contains 3

x.f \mapsto 3

x.f := 5

x.f \mapsto 5

\begin{align*}
x.f &\mapsto 3 \\
y.f &\mapsto 3
\end{align*}

\begin{align*}
x.f &:= 5 \\
x.f &\mapsto 5 \\
y.f &\mapsto 3
\end{align*}
Separation logic’s secret sauce

- Heap
  \[ \text{Loc} \rightarrow \text{Val} \]

- Separating conjunction
  \[ P \ast R \]
  Disjoint access rights

- Frame rule
  \[
  \frac{\{ P \} \mathcal{C} \{ Q \}}{\{ P \ast R \} \mathcal{C} \{ Q \ast R \}}
  \]
Advantages

• Clean reasoning
  – Say what changes, not what doesn’t

• Matches informal reasoning and specification:
  – ArrayList.Add
    • Adds an object to the end of the ArrayList.
  – Stack.Pop
    • Removes and returns the object at the top of the Stack.
Concurrency

[O’Hearn Concur’04]

• This style of reasoning naturally extends to disjoint concurrency:

\[
\{ P_1 \} C_1 \{ Q_1 \} \\
\{ P_2 \} C_2 \{ Q_2 \} \\
\underbrace{\{ P_1 * P_2 \} C_1 || C_2 \{ Q_1 * Q_2 \}}
\]
HIDING FROM SEPARATION

[O’Hearn, Reynolds, Yang POPL 2004]
Example: Memory manager

Client view

\[
\begin{align*}
    r &:= \text{alloc()} & r &\mapsto _ & r+1 &\mapsto _ \\
    & & r &\mapsto _ & r+1 &\mapsto _ & \text{dispose}(r)
\end{align*}
\]
Example: Memory manager

Module view

Free list

r := alloc()

Free list

r \rightarrow \_ \quad r+1 \rightarrow \_

Free list

dispose(r)

Free list

Free list
Hypothetical frame rule

• Modular procedure rule

\[
\Gamma \vdash \{ P_1 \ast R \} \ C_1 \ { Q_1 \ast R } \\
\vdots \\
\{ P_1 \} f_1 \ { Q_1 } \ldots \vdash \{ P \} \ C \ { Q } \\
\vdash \{ P \ast R \} \ \text{let } f_1 = C_1 ; \ldots \text{ in } C \ { Q \ast R } 
\]

• Hypothetical frame rule

\[
\Gamma \vdash \{ P \} \ C \ { Q } \\
\Gamma \ast R \vdash \{ P \ast R \} \ C \ { Q \ast R } 
\]
Multiple instances

- Single internal abstraction – R, Free list
- What about dynamically allocated modules?
- There isn’t a single thing to hide
BEING OPAQUE

[Parkinson and Bierman 2005]
Example: Memory Manager

\[ r := \text{malloc}(n) \]

\[ \text{free}(r) \]

How much memory should be returned

Free only deallocates blocks provided by malloc
Example: Memory Manager

\[ r := \text{malloc}(n) \]

\[ \text{free}(r) \]

\[ \text{Mblock}(r,n) \]

\[ r+n-1 \rightarrow _{\text{Mblock}} \]

\[ r+n-1 \rightarrow _{\text{Mblock}} \]
Implementation

Mblock(r,n)

Client cannot know this!

r-1 \rightarrow r+n
Abstract Predicates

- **Approach**
  - Treat abstract predicates as uninterpreted in client
  - Specify details of predicate inside module

- **Rule:**

\[
\frac{\Delta; \Gamma \vdash \{ P \} \land \{ Q \}}{\Delta, a(x_1, \ldots, x_n) = R; \Gamma \vdash \{ P \} \land \{ Q \}} \quad a \notin \text{dom}(\Delta)
\]
Fine grained abstraction: Finite Map

• Specification
  – \{ \text{emp} \} \ r := \text{create}(n) \ \{ \text{elem}(r,0,\_)*...*\ \text{elem}(r,n-1,\_)} \}
  – \{ \text{elem}(r, x, \_)} \ \text{update}(r,x,v) \ \{ \text{elem}(r, x, v)} \}
  – \{ \text{elem}(r, x, v)} \ y := \text{read}(r,x) \ \{ \text{elem}(r, x, v) \ \ast \ y=v \}

• Possible implementations
  – Array
  – List
  – ...

High-level separation corresponds to low-level separation.

High-level separation does not correspond to low-level separation.
NOBODY LIKES TO SHARE
Memory Manager

- All blocks of memory have pointer to next physical block of memory
- Grey blocks allocated; white unallocated
- Block has flag, black for allocated.
- Find free blocks by traversing next pointers.
Problem

What is Mblock predicate?
• Reachable from start of arena?
• Prevents block being removed from list?
• Pointer points to just passed the end of block?

Real problem
• Free list and Mblock are not separated
• Need to express protocols/restrictions on shared state

This is why we need 700 separation logics?
COPING WITH CHANGE

[Jones: TOPLAS 1983]
[Vafeiadis, Parkinson CONCUR 2007] [Feng, Ferreira, Shao ESOP 2007]
Example: Rely-Guarantee

Simple example

```java
while(true) { y:=x; if x<y then fail() }
||
while(true) { t := x; x:=t + 1; }
```
Example: Rely-Guarantee

Simple example

while(true) { y:=x; if x<y then fail() } (A)
||
while(true) { t := x; x:=t + 1; }

Properties for (A)
• Environment only increases x
• Process doesn’t change x
Rely-Guarantee

Simple example

\[
\text{while(true) } \{ \text{y:=x; if x<y then fail() } \}
\]

\[
\text{while(true) } \{ \text{t:=x; x:=t + 1; } \} \quad \text{(B)}
\]

Properties for (B)

- Environment doesn’t change x
- Process only increases x
Rely-guarantee

• Use relations on states express interference
  – Rely: what the environment can do
  – Guarantee: what the process can do

• Assertions about shared state must be stable

\[
\begin{align*}
G_2 & \subseteq R_1 \\
G_1 & \subseteq R_2 \\
R_1, G_1 & \vdash \{ P_1 \} C_1 \{ Q_1 \} \\
R_2, G_2 & \vdash \{ P_2 \} C_2 \{ Q_2 \} \\
R_1 \cap R_2, G_1 \cup G_2 & \vdash \{ P_1 \land P_2 \} C_1 \| C_2 \{ Q_1 \land Q_2 \}
\end{align*}
\]
RGSep/SAGL

• Combine separation logic with rely-guarantee
  – local and shared state
  – Separating conjunction, *,
    • splits local state; and
    • shares shared state.
  – Specify rely-guarantee to shared state
    • Can change footprint
RGSep parallel rule

\[ G_2 \subseteq R_1 \]
\[ G_1 \subseteq R_2 \]
\[ R_1, G_1 \vdash \{ P_1 \} C_1 \{ Q_1 \} \]
\[ R_2, G_2 \vdash \{ P_2 \} C_2 \{ Q_2 \} \]
\[ R_1 \cap R_2, G_1 \cup G_2 \vdash \{ P_1 \ast P_2 \} C_1 \parallel C_2 \{ Q_1 \ast Q_2 \} \]

Deals with sharing of state
Abstracting change?

• Context is the problem
  – Predicates will need to mention shared state, but they can’t mention rely and guarantee.

• Separation logic gives abilities to do changes
  – Extend this concept with rely-guarantee
EMBRACING CHANGE

[Dodds, Feng, Parkinson, Vafeiadis ESOP2009]
[Dinsdale-Young, Dodds, Gardner, Parkinson, Vafeiadis ECOOP 2010]
State change assertions

- **Read permission**
  \( x \mapsto n \)

- **Increment permission**
  \( \exists k. x \mapsto k \) 
  \( n \leq k \)
Deny-guarantee

Permissions structure

Action → deny, z

Combination

• 0 is the unit; and

• Combine denies with denies by addition lifting 1. (Same for guarantees.)

where z ∈ (0,1)
Parallel Rule

\[
\{ P_1 \} C_1 \{ Q_1 \} \\
\{ P_2 \} C_2 \{ Q_2 \} \\
\underline{\{ P_1 \ast P_2 \} C_1 \| C_2 \{ Q_1 \ast Q_2 \}}
\]
A Lock Module

\[
\text{locked}(x, P) \Rightarrow \text{false}
\]
Spin lock

Locked(x, P)

x ↦ 1

UNLOCK

LOCK = UNLOCK * P * x ↦ 0 ↼ x ↦ 1

UNLOCK = x ↦ 1 ↼ x ↦ 0 * UNLOCK * P
Spin lock

Locked(x,P)

isLock (x,P)

\[ x \mapsto 0 \ast P \ast UNLOCK \]

\[ \forall x \mapsto 1 \]

LOCK

guar
Ticketed Lock

Locked(x)

x.serving $\mapsto$ s * true

IncAt(s)
Ticketed Lock

Locked(x)

isLock (x)

s \leq t
* x.ticket \mapsto t
* x.serving \mapsto s

IncAt(t)

NEXT
guar

guar

Locked(x)
Concurrent Abstract Predicates

• Describe
  – shared state
  – local abilities

• Must be Self Stable
  – Immune to interference

• Other examples
  – Channels (Safe Futures [Talk by Dodds at POPL])
  – Sets (List based and B-Trees [See talk later])
DENY EVERYTHING?
Thanks

- Gavin Bierman
- Richard Bornat
- **Peter O’Hearn**
- Mike Dodds
- Thomas Dinsdale-Young
- Xinyu Feng
- Philippa Gardner
- Alexey Gotsman

- Tony Hoare
- Neel Krishnaswami
- Viktor Vafeiadis
- John Wickerson
...
Related Work

• Hypervisor/VCC
• OCap – [Feng, Shao, Guo, Dong VSTTE2008]
• Refinement calculus – [Turon, Wand]
• Context Logic - Gardner et al.