

# Contextual Equivalence and the CIU-Theorem

---

Ben Lippmeier

University of New South Wales

FP-Syd 2012/3/15



# Hoisting Bindings

---

`(\v1. let v2 = x2  
in x3)`  $\xrightarrow{\text{rewrite}}$  `let v2 = x2  
(\v1. x3)`

provided: `v1 \notin fv(x2)`

# Common Sub-Expression Elimination

---

`let v1 = x1 in`  
`let v2 = x1 in`  
`x3` rewrite → `let v1 = x1 in`  
`x3 [v1/v2]`

# Equivalence

---

- “After optimisation, the program should give the same result”

# Equivalence

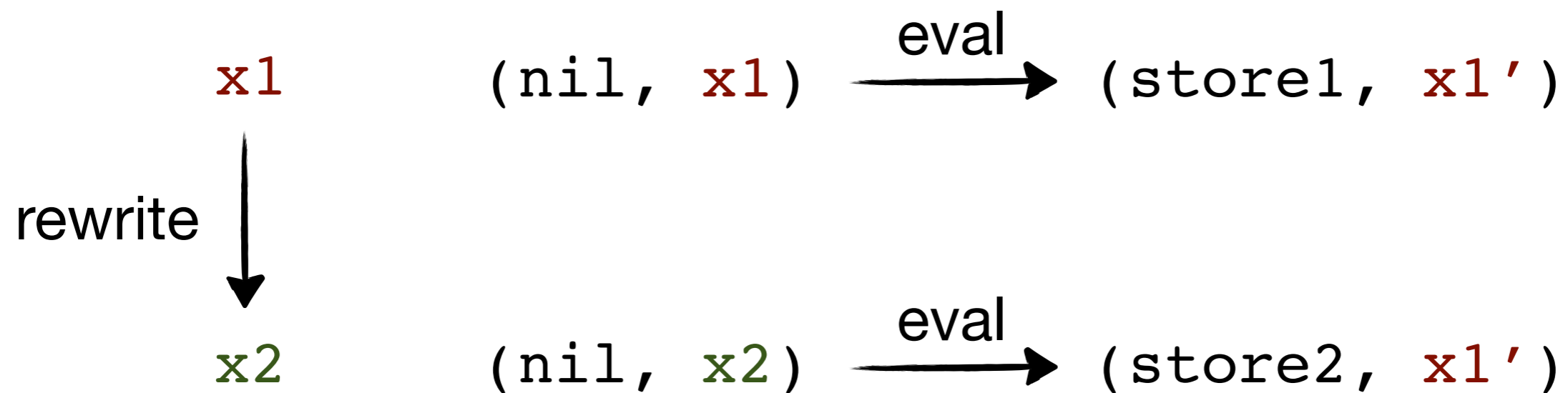
---

- “After optimisation, the program should give the same result”
- What do we mean by *result*, given that optimisations can reduce the amount of allocation?

# Equivalence

---

- “After optimisation, the program should give the same result”
- What do we mean by *result*, given that optimisations can reduce the amount of allocation?



# No evaluation under lambdas

---

```
(\v1. let v2 = 2 + 3
      in v1 + v2)
```

rewrite  
→

```
let v2 = 2 + 3 in
(\v1. v1 + v2 )
```

eval  
→

```
let v2 = 5 in
(\v1. v1 + v2 )
```

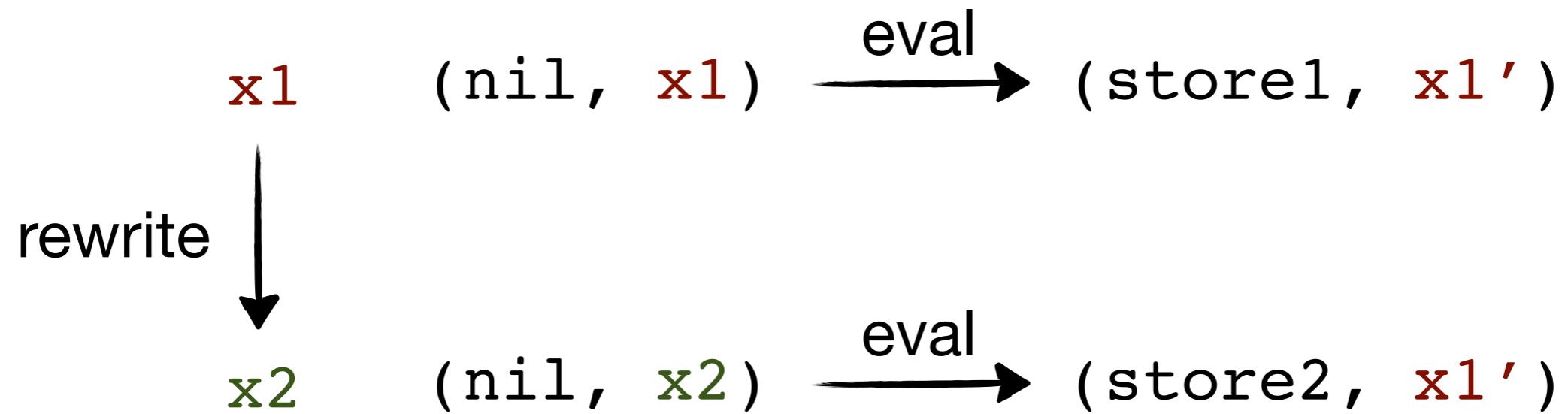
eval  
→

```
(\v1. v1 + 5 )
```



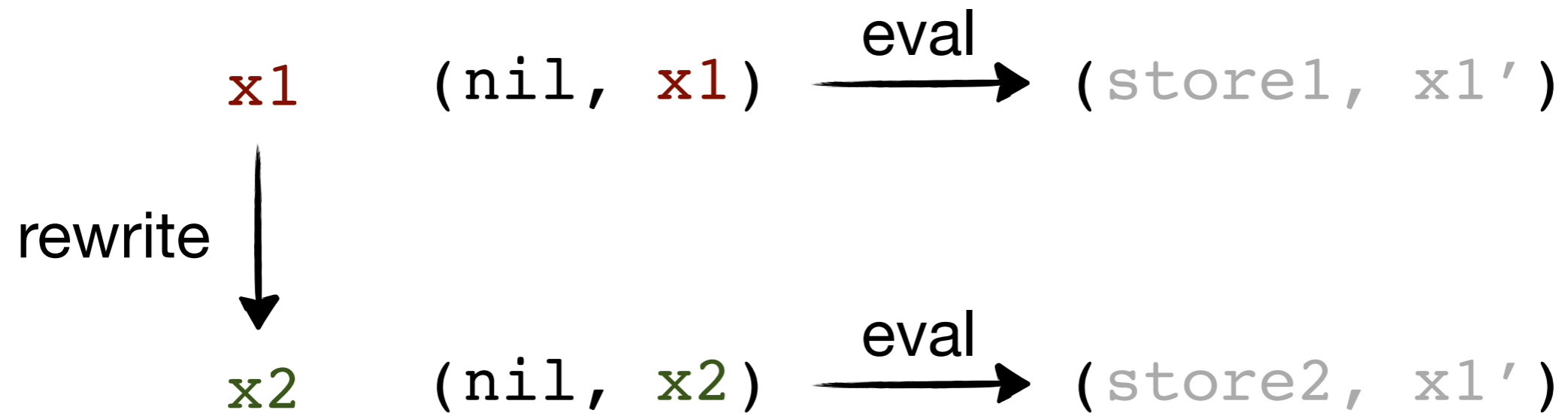
# Only observe termination

---



# Only observe termination

---



# Only observe termination

---




# Contextual Equivalence

---

```
(\v. if x3  
    then let v1 = blah in x1  
    else x4) x5
```

rewrite

```
(\v. if x3  
    then let v1 = blah in x2  
    else x4) x5
```



# Contextual Equivalence

---

`map (\v. f x3 (g x1 v)) ys`

rewrite

`map (\v. f x3 (g x2 v)) ys`

# Contextual Equivalence

---

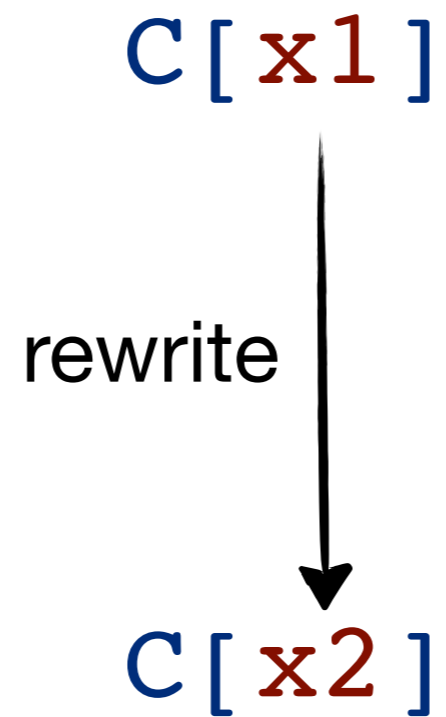
`map (\v. f x3 (g x1 v)) ys`

rewrite

`map (\v. f x3 (g x2 v)) ys`

# Contextual Equivalence

---



# Contextual Equivalence

---

$$x1 \equiv_{\text{ctx}} x2$$

forall C. TERM C[x1] <=> TERM C[x2]



# Contextual Equivalence

---

$(\backslash v1. \text{let } v2 = x2 \text{ in } x3) \xrightarrow{\text{rewrite}} \text{let } v2 = x2 (\backslash v1. x3)$

$x1 \equiv_{\text{ctx}} x2$

$\text{forall } C. \text{TERM } C[x1] \Leftrightarrow \text{TERM } C[x2]$

# Contextual Equivalence

---

$$(\backslash v1. \text{let } v2 = x2 \text{ in } x3) \equiv_{\text{ctx}} \text{let } v2 = x2 \text{ in } (\backslash v1. x3)$$
$$x1 \equiv_{\text{ctx}} x2$$
$$\text{forall } C. \text{TERM } C[x1] \Leftrightarrow \text{TERM } C[x2]$$

# Contextual Equivalence

---

$(\backslash v1. \text{let } v2 = x2 \text{ in } x3) \equiv_{\text{ctx}} \text{let } v2 = x2 (\backslash v1. x3)$

$x1 \equiv_{\text{ctx}} x2$

**forall**  $C.$  TERM  $C[x1]$   $\Leftrightarrow$  TERM  $C[x2]$

hmmmmm

# Contextual Equivalence

---

`map (\v. f x3 (g x1 v)) ys`

`map (\v. f x3 (g (v + v) v)) ys`

`map (\v. f x3 (g (v + v) v)) ys`

`C`

`C[x1]`

# Closing Substitutions

---

$$(\lambda x. x + x) 5$$

$$(\lambda x. 2 * x) 5$$

# Closing Substitutions

---

$(\lambda x. x + x) 5$

$(\lambda x. 2 * x) 5$

$C[x + x]$

# Closing Substitutions

---

$$(\lambda x. x + x) 5$$

$$5 + 5$$

$$10$$

$$(\lambda x. 2 * x) 5$$

$$2 * 5$$

$$10$$

# Closing Substitutions

---

```
let f = \z.  
    let v1 = z + 1 in  
    let v2 = 2 * z in  
    g (v1 + v2)  
in f 5
```

```
let f = \z.  
    let v2 = 2 * z in  
    let v1 = z + 1 in  
    g (v1 + v2)  
in f 5
```



# Closing Substitutions

---

```
C[let v1 = z + 1 in  
  let v2 = 2 * z in  
  g (v1 + v2)]
```

```
C[let v2 = 2 * z in  
  let v1 = z + 1 in  
  g (v1 + v2)]
```

# Closing Substitutions

---

```
let f = \z.  
  let v1 = z + 1 in  
  let v2 = 2 * z in  
  g (v1 + v2)  
in f 5
```

```
let f = \z.  
  let v2 = 2 * z in  
  let v1 = z + 1 in  
  g (v1 + v2)  
in f 5
```

```
let v1 = 5 + 1 in  
let v2 = 2 * 5 in  
g (v1 + v2)
```

```
let v2 = 2 * 5 in  
let v1 = 5 + 1 in  
g (v1 + v2)
```

# Closing Substitutions

---

```
let f = \z.  
  let v1 = z + 1 in  
  let v2 = 2 * z in  
  g (v1 + v2)  
in f 100
```

```
let f = \z.  
  let v2 = 2 * z in  
  let v1 = z + 1 in  
  g (v1 + v2)  
in f 100
```

```
let v1 = 100 + 1 in  
let v2 = 2 * 100 in  
g (v1 + v2)
```

```
let v2 = 2 * 100 in  
let v1 = 100 + 1 in  
g (v1 + v2)
```

# Closing Substitutions

---

```
let f = \z.  
  let v1 = z + 1 in  
  let v2 = 2 * z in  
  g (v1 + v2)  
in f (100 * 90)
```

```
let f = \z.  
  let v2 = 2 * z in  
  let v1 = z + 1 in  
  g (v1 + v2)  
in f (100 * 90)
```

```
let v1 = 9000 + 1 in  
let v2 = 2 * 9000 in  
g (v1 + v2)
```

```
let v2 = 2 * 9000 in  
let v1 = 9000 + 1 in  
g (v1 + v2)
```

# Closing Substitutions

---

```
let f = \z.  
  let v1 = z + 1 in  
  let v2 = 2 * z in  
  g (v1 + v2)  
in f (bar "hello")
```

```
let f = \z.  
  let v2 = 2 * z in  
  let v1 = z + 1 in  
  g (v1 + v2)  
in f (bar "hello")
```

```
let v1 = ?? + 1 in  
let v2 = 2 * ?? in  
g (v1 + v2)
```

```
let v2 = 2 * ?? in  
let v1 = ?? + 1 in  
g (v1 + v2)
```

# Contextual Equivalence (again)

---

$$x1 \equiv_{\text{ctx}} x2$$

forall C. TERM C[x1] <=> TERM C[x2]

# CIU-Equivalence

---

$$x1 \equiv_{ctx} x2$$

$$\text{forall } C. \text{ TERM } C[x1] \\ \Leftrightarrow \text{ TERM } C[x2]$$

$$x1 \equiv_{ciu} x2$$

$$\text{forall } C \sigma. \text{ TERM } C[\sigma x1] \\ \Leftrightarrow \text{ TERM } C[\sigma x2]$$

# Closed Instantiation of Use-Equivalence

---

$$x1 \equiv_{\text{ctx}} x2$$

$$\text{forall } C. \text{ TERM } C[x1] \\ \Leftrightarrow \text{ TERM } C[x2]$$

$$x1 \equiv_{\text{ciu}} x2$$

$$\text{forall } C \sigma. \text{ TERM } C[\sigma x1] \\ \Leftrightarrow \text{ TERM } C[\sigma x2]$$



# The CIU-Theorem

---

Contextual Equivalence  
and CIU-Equivalence coincide

Proved true for all lambda languages  
with uniform semantics!

# Uniform Semantics

---

- Single Step Reduction is Deterministic
- Reduction is preserved by value substitution
- If one expression reduces to another and the first terminates then so does the second.
- ... a few others

# Uniform Semantics

---

- Reduction is preserved by value substitution
- Implies that reduction does not look deep within an AST node to decide what to do.

**if True then x2 else x3**  $\Rightarrow$  **x2**

**(\v. x1) x2**  $\Rightarrow$  **x1[x2/v]**

# References

---

- Reasoning about Programs with Effects  
Ian Mason, Carolyn Talcott, 1990-1997
- Operational Reasoning for Functions with Local State  
Andrew Pitts and Ian Stark, 1998