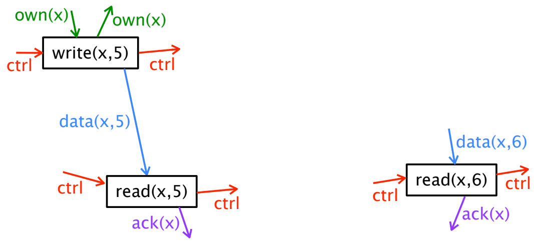
Separation Logic and Graphical Models

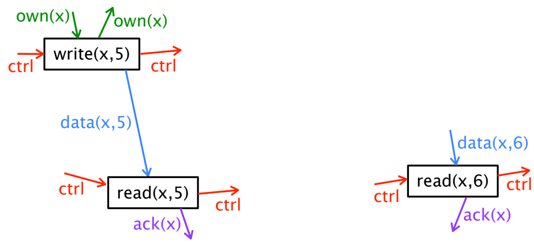
John Wickerson and Tony Hoare



Semantics Lunch, 25th October 2010

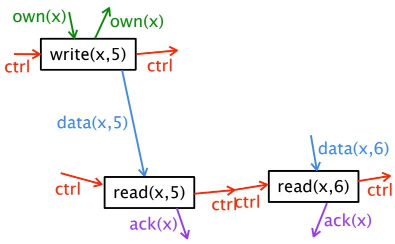
Trace composition

Problem: Composition is non-deterministic.



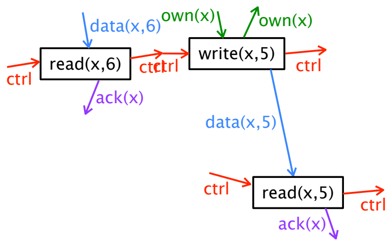
Trace composition

Problem: Composition is non-deterministic.



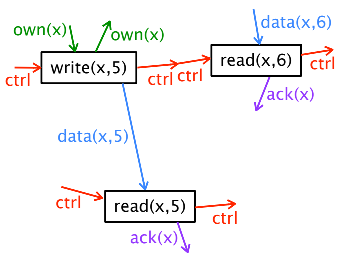
Trace composition

Problem: Composition is non-deterministic.



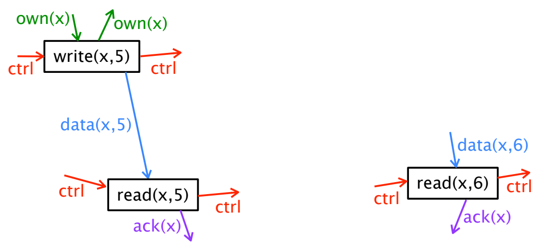
Trace composition

Problem: Composition is non-deterministic.



Trace composition

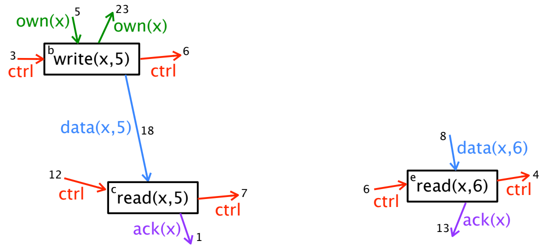
Problem: Composition is non-deterministic.



Trace composition

Problem: Composition is non-deterministic.

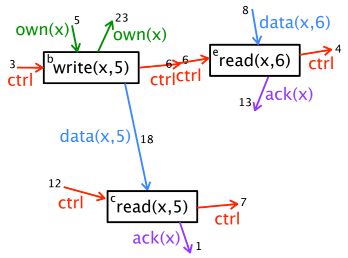
Fix: Give nodes and arrows unique identities.



Trace composition

Problem: Composition is non-deterministic.

Fix: Give nodes and arrows unique identities.



Trace representation

A trace is a 6-tuple:

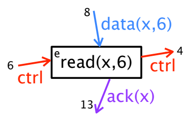
- set of nodes, $N \in \mathbb{P}_{\text{fin}}(\text{Node})$
- set of arrows, $A \in \mathbb{P}_{\text{fin}}(\text{Arrow})$
- node labelling, $NL \in N \to \text{NodeLabel}$
- arrow labelling, $AL \in A \to \text{ArrowLabel}$
- head map, $H \in A \rightharpoonup N$
- tail map, $T \in A \rightharpoonup N$

We require that $\text{dom}(H) \cup \text{dom}(T) = A$ and we forbid cycles

Trace representation

A trace is a 6-tuple:

- set of nodes, $N =$ {e}
- set of arrows, $A =$ {4, 6, 8, 13}
- node labelling, $NL =$ {e↦read(x,6)}
- arrow labelling, $AL =$ {4↦ctrl, 6↦ctrl, 8↦data(x,6), 13↦ack(x)}
- head map, $H =$ {6↦e, 8↦e}
- tail map, $T =$ {4↦e, 13↦e}



Trace disjointness

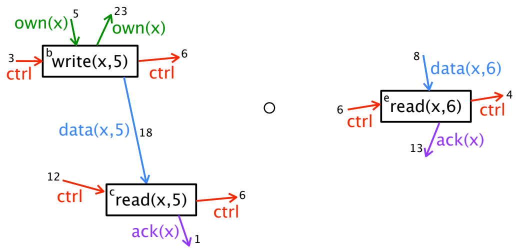
Composition is defined iff the operands are 'disjoint', which means:

1. there are no common nodes
2. any common arrows have the same label
3. any common arrows can be connected
   (i.e. dangle out of one trace and into the other)
4. composition would not introduce a cycle

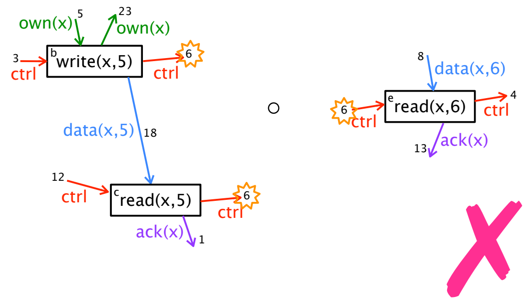
Trace composition

$t_1 \circ t_2 =$
  if t_1 and t_2 are disjoint **then**
    let $(N_1, A_1, NL_1, AL_1, H_1, T_1) = t_1$
    and $(N_2, A_2, NL_2, AL_2, H_2, T_2) = t_2$
    in $(N_1 \cup N_2, A_1 \cup A_2, NL_1 \cup NL_2,$
      $AL_1 \cup AL_2, H_1 \cup H_2, T_1 \cup T_2)$
  else undefined

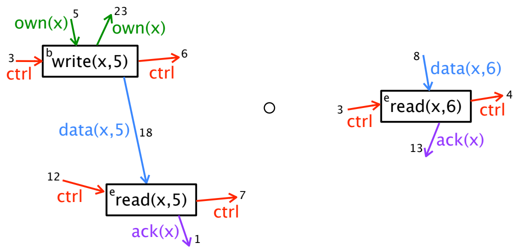
Quiz (1 of 4)



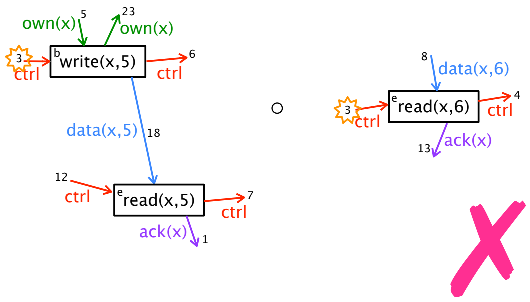
Quiz (1 of 4)



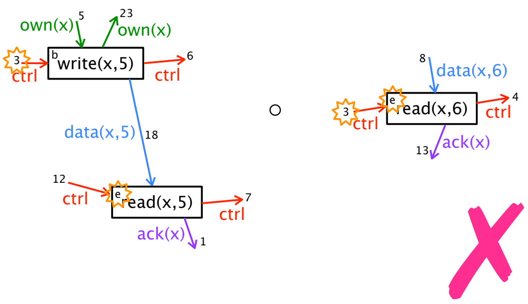
Quiz (2 of 4)



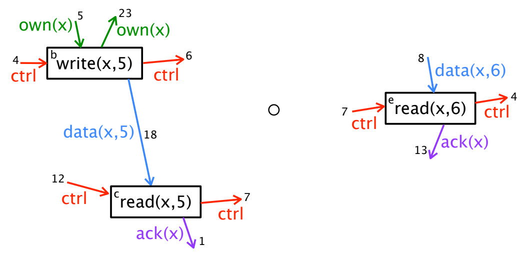
Quiz (2 of 4)



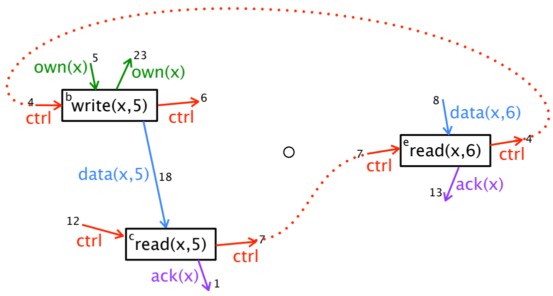
Quiz (2 of 4)



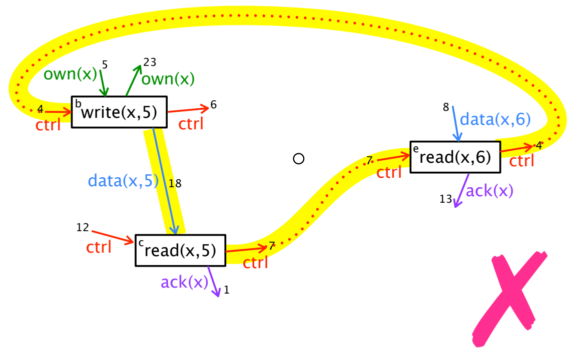
Quiz (3 of 4)



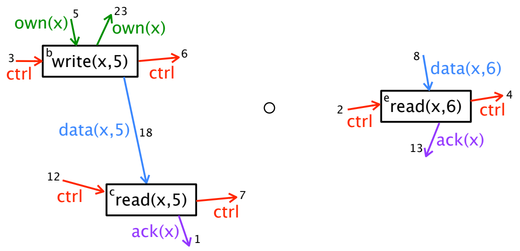
Quiz (3 of 4)



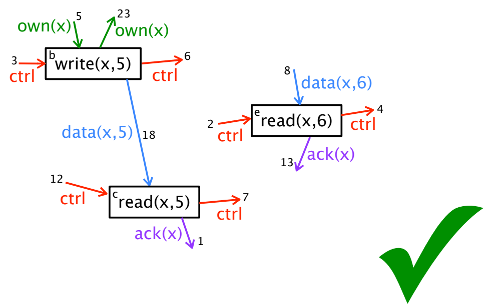
Quiz (3 of 4)



Quiz (4 of 4)



Quiz (4 of 4)



Properties of composition

The composition operator:

- is a partial binary operator of type $\mathrm{Trace} \times \mathrm{Trace} \rightharpoonup \mathrm{Trace}$
- is commutative and associative
- has unit $u = (\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing)$
- is cancellative (that is: if $t_1 \circ t_3$ and $t_2 \circ t_3$ are defined and equal, then $t_1 = t_2$)

So... $(\mathrm{Trace}, \circ, u)$ is a **separation algebra**

Models of separation logic

Heap model: $(\mathrm{Heap}, \circ, u)$

where a heap is a partial mapping from memory addresses to values, $\circ$ composes heaps that have disjoint domains, and u is the empty heap

Trace model: $(\mathrm{Trace}, \circ, u)$

where $\circ$ composes disjoint traces, and u is the empty trace

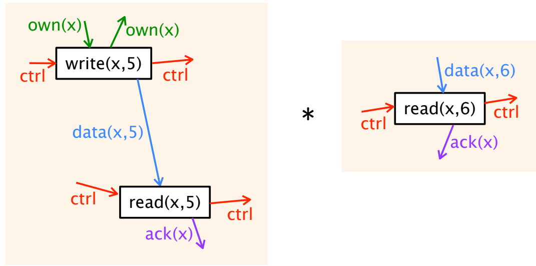
There are several others.

The '$*$' operator

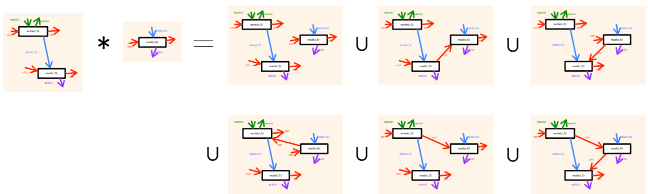
We lift $\circ$ to <u>sets</u>:

$$P * Q \stackrel{\text{def}}{=} \{t \mid \exists t_1 \in P.\, \exists t_2 \in Q.\, t = t_1 \circ t_2\}$$

'$*$' in pictures



'$*$' in pictures



The '$*$' operator

We lift $\circ$ to <u>sets</u>:

$$P * Q \stackrel{\text{def}}{=} \{t \mid \exists t_1 \in P.\, \exists t_2 \in Q.\, t = t_1 \circ t_2\}$$

In the heap model, P and Q are sets of heaps; i.e., assertions about the heap.

In our model, P and Q are sets of traces; i.e., <u>**programs**</u>.

Denotational semantics of an imperative language with concurrency

Command language

$C ::= \texttt{acq}(l)$	acquiring a lock
$\mid \texttt{rel}(l)$	releasing a lock
$\mid \texttt{lock}\ l\ \texttt{in}\ C$	lock declaration
$\mid \texttt{read}(x, v)$	reading a specific value
$\mid \texttt{write}(x, v)$	writing a specific value
$\mid \texttt{var}\ x\ \texttt{in}\ C$	variable declaration
$\mid \texttt{skip}$	empty command
$\mid \Sigma i \in I.\, C_i$	non-deterministic choice
$\mid C^*$	non-deterministic looping
$\mid C\,;C$	sequential composition
$\mid C \parallel C$	parallel composition

Meaning of commands

$[\![C]\!] : \Gamma \times \Gamma \rightarrow \mathbb{P}(\text{Trace})$

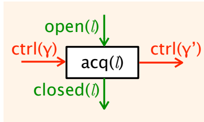
where Γ is a set of 'control arrow identifiers'

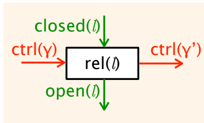
$[\![C]\!]\,(\gamma, \gamma') =$ a set of traces that have:

one incoming control arrow, labelled ctrl(γ), and

one outgoing control arrow, labelled ctrl(γ')

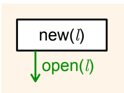
Denotational semantics (1)

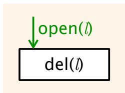
$[\![\texttt{acq}\ l]\!]\,(\gamma, \gamma') =$ 

$[\![\texttt{rel}\ l]\!]\,(\gamma, \gamma') =$ 

Denotational semantics (2)

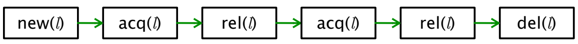
$[\![\texttt{lock}\ l\ \texttt{in}\ C]\!]\,(\gamma, \gamma') =$

(

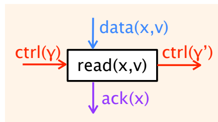
$* [\![C]\!]\,(\gamma, \gamma') *$ 

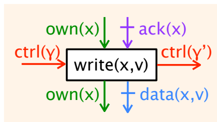
)

$\cap\ hide(l)$



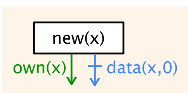
Denotational semantics (3)

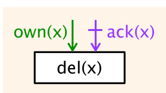
$[\![\texttt{read}(x, v)]\!]\,(\gamma, \gamma') =$ 

$[\![\texttt{write}(x, v)]\!]\,(\gamma, \gamma') =$ 

Denotational semantics (4)

$[\![\texttt{var}\ x\ \texttt{in}\ C]\!]\,(\gamma, \gamma') =$

(

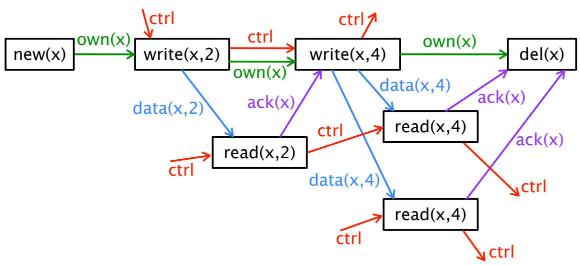
$* [\![C]\!]\,(\gamma, \gamma') *$ 

)

$\cap\ hide(x) \cap triangle(x)$

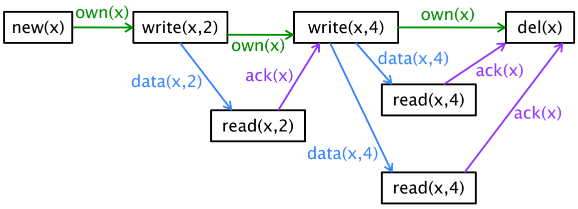
The triangle property

$triangle(x)$ holds for traces such as this one:

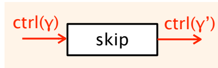


The triangle property

$triangle(x)$ holds for traces such as this one:



Denotational semantics (5)

$\llbracket \mathtt{skip} \rrbracket (\gamma, \gamma') =$ 

$$\llbracket \Sigma i \in I.\, C_i \rrbracket (\gamma, \gamma') = \bigcup i \in I.\, \llbracket C_i \rrbracket (\gamma, \gamma')$$

Denotational semantics (6)

$$\llbracket C_1 \,;\, C_2 \rrbracket = \llbracket C_1 \rrbracket \,;\, \llbracket C_2 \rrbracket$$

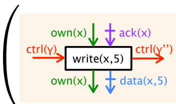
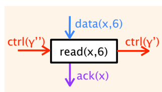
where $F \,;\, G = \lambda(\gamma, \gamma').$
$\bigcup \gamma'' \notin \{\gamma, \gamma'\}.\, (F(\gamma, \gamma'') * G(\gamma'', \gamma')) \cap hide(\gamma'')$

$$\llbracket C^* \rrbracket (\gamma, \gamma') = \bigcup k \geq 0.\, \llbracket C \rrbracket^k (\gamma, \gamma')$$

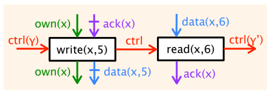
where $F^0 = \llbracket \mathtt{skip} \rrbracket$ and $F^{k+1} = F \,;\, F^k$

Example

$\llbracket \mathtt{write}(x, 5) \,;\, \mathtt{read}(x, 6) \rrbracket (\gamma, \gamma')$

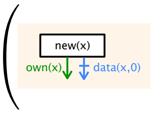
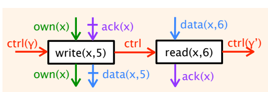
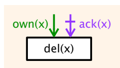
$= \bigcup \gamma'' \notin \{\gamma, \gamma'\}.\Bigg($  $*$  $\Bigg)$

$\cap\, hide(\gamma'')$

$=$ 

Example

$\llbracket \mathtt{var}\ x\ \mathtt{in}\ \{\mathtt{write}(x, 5) \,;\, \mathtt{read}(x, 6)\} \rrbracket (\gamma, \gamma')$

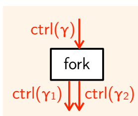
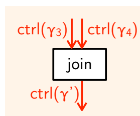
$= \Bigg($  $*$  $*$  $\Bigg)$

$\cap\, hide(x) \cap triangle(x)$

$= \varnothing$

Denotational semantics (7)

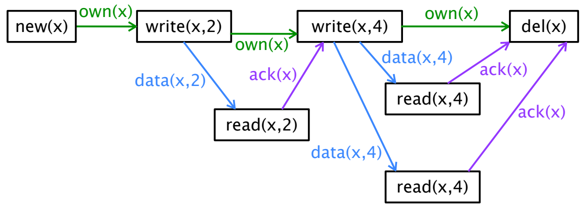
$\llbracket C_1 \parallel C_2 \rrbracket (\gamma, \gamma') = \bigcup \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\} \# \{\gamma, \gamma'\}.$

$\Bigg($  $* \llbracket C_1 \rrbracket (\gamma_1, \gamma_3) * \llbracket C_2 \rrbracket (\gamma_2, \gamma_4) *$  $\Bigg)$

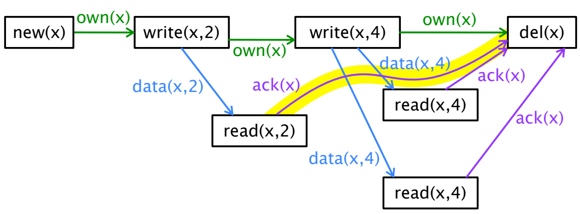
$\cap\, hide(\gamma_1, \gamma_2, \gamma_3, \gamma_4)$

Future directions

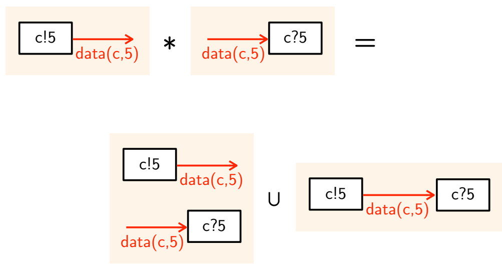
A model of weak memory?



A model of weak memory?



A model of processes?



The End