Functional In-place Update with Layered Datatype Sharing

Michal Konečný
LFCS, University of Edinburgh

Mobile
Resource
Guarantees

Introduction

- *LFPL* (Linear Functional Programming Language) [Hofmann 1999]
- *Camelot* Mobile Resource Guarantees (MRG)

Introduction

- *LFPL* (Linear Functional Programming Language) [Hofmann 1999]
- *Camelot* Mobile Resource Guarantees (MRG)
- functional language $\Rightarrow$ neat reasoning about programs
- explicit destruction $\Rightarrow$ explicit space reuse
- resource type $\Diamond$ $\Rightarrow$ explicit in-place update

✔

Functional in-place update

In ML:

$$\begin{aligned} \mathit{append}\ x\ y = \ & \text{match } x \text{ with} \\ & \quad \text{Nil} \to y \\ & \quad \mid \text{Cons}(h, t) \to \text{Cons}(h, \mathit{append}\ t\ y) \end{aligned}$$

$$h : A, t : L(A) \vdash \text{Cons}(h, t) : L(A)$$

Functional in-place update

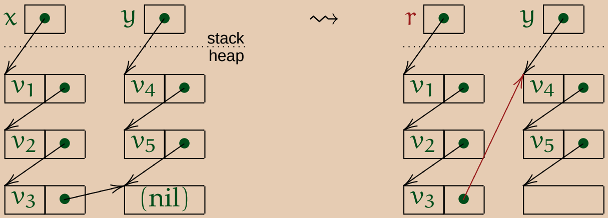
In LFPL:

$$
\begin{aligned}
\mathit{append}\ x\ y = \ & \text{match } x \text{ with} \\
& \quad \text{Nil} \ \ @d \ \ \to y \\
& \quad \mid \text{Cons}(h, t) \ \ @d \ \ \to \text{Cons}(h, \mathit{append}\ t\ y) @d
\end{aligned}
$$

$$h : A, t : L(A), d : \Diamond \vdash \text{Cons}(h, t) @d : L(A)$$

elements of $\Diamond$: units of heap space/heap locations

Heap representation:



Functional in-place update

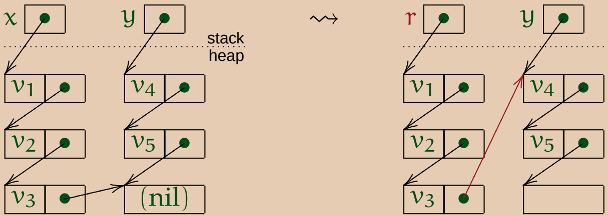
In Camelot [MRG]: $\mathsf{Cons}(h,t) = \mathsf{Cons}(h,t)@\mathit{new}()$

$$
\begin{aligned}
\mathit{append}\ x\ y = \ & \mathsf{match}\ x\ \mathsf{with} \\
& \mathsf{Nil} \qquad\qquad \rightarrow y \\
& \mid \mathsf{Cons}(h,t)\ @\_ \quad \rightarrow \mathsf{Cons}(h, \mathit{append}\ t\ y)
\end{aligned}
$$

$$h : A, t : \mathsf{L}(A), d : \Diamond \vdash \mathsf{Cons}(h,t)@d : \mathsf{L}(A)$$

elements of $\Diamond$: units of heap space/heap locations

Heap representation:



✔

Introduction (continued)

- How to forbid destroying live heap cells ?

Introduction (continued)

- How to forbid destroying live heap cells ?
- *affine linear typing* does the job
  BUT forbids sharing on the heap

Introduction (continued)

- How to forbid destroying live heap cells ?
- *affine linear typing* does the job
  BUT forbids sharing on the heap
- developing less restrictive typings for LFPL:
  - *usage aspects* (Aspinall & Hofmann 2002)
  - *explicit sharing* (Atkey 2002)
  - *layered datatype sharing* (K 2002)

Introduction (continued)

- How to forbid destroying live heap cells ?
- *affine linear typing* does the job
  BUT forbids sharing on the heap
- developing less restrictive typings for LFPL:
  - *usage aspects* (Aspinall & Hofmann 2002)
  - *explicit sharing* (Atkey 2002)
  - *layered datatype sharing* (K 2002)
- Scope: first-order, full recursion, arbitrary recursive datatypes

✔

Non-linear typings

- Recognize non-destructive use of data
- Allow some safe sharing on the heap

Non-linear typings

- Recognize non-destructive use of data
- Allow some safe sharing on the heap

Typing judgement: $\Gamma \vdash e : A; \phi$

ϕ assertions about heap

– before, during and after evaluation

– mainly: separation and preservation

– $\phi \in \Phi_{\Gamma,A}$ where $\Phi_{\Gamma,A}$ is *finite*

Non-linear typings

- Recognize non-destructive use of data
- Allow some safe sharing on the heap

Typing judgement: $\Gamma \vdash e : A; \phi$

ϕ assertions about heap – before, during and after evaluation

– mainly: separation and preservation

– $\phi \in \Phi_{\Gamma,A}$ where $\Phi_{\Gamma,A}$ is *finite*

- Usage aspects: ϕ talks about *whole* values (of variables)
- Here: ϕ talks about *portions* of these values

✔

Datatype portions (layers)

$$\text{type ilT} = \text{N} \mid \text{C of Int} * \text{ilT}$$
$$\text{type illT} = \text{NL} \mid \text{CL of ilT} * \text{illT}$$

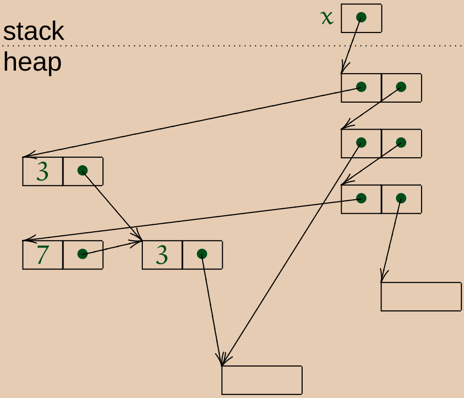
$$[\![x]\!] = [[3,3],[],[7,3]]$$
$$x : \text{illT}$$

✔

Datatype portions (layers)

$$\text{type ilT} = \text{N} \mid \text{C of Int} * \text{ilT}$$

$$\text{type illT} = \text{NL} \mid \text{CL of ilT} * \text{illT}$$



$$[\![x]\!] = [[3,3],[],[7,3]]$$

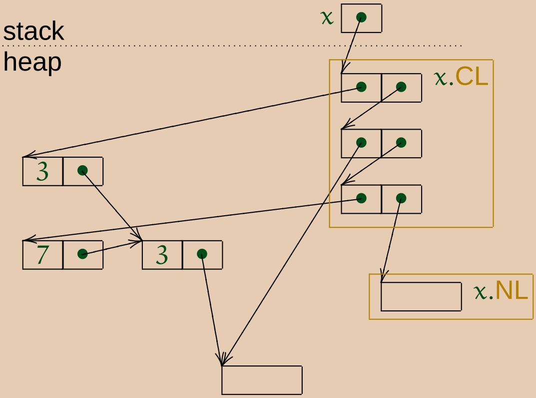
$$x : \text{illT}$$

✔

Datatype portions (layers)

$$\text{type ilT} = \text{N} \mid \text{C of Int} * \text{ilT}$$

$$\text{type illT} = \text{NL} \mid \text{CL of ilT} * \text{illT}$$



$$[\![x]\!] = [[3,3],[],[7,3]]$$

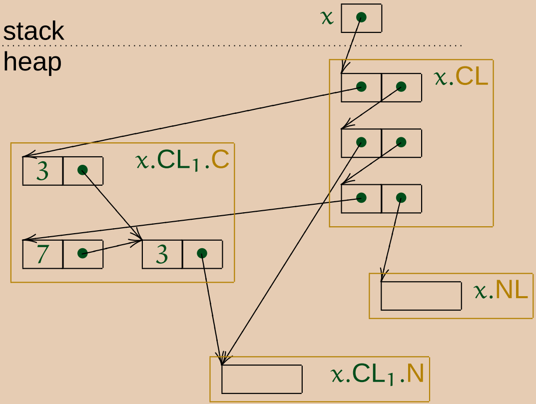
$$x : \text{illT}$$

✔

Datatype portions (layers)

$$\text{type ilT} = \text{N} \mid \text{C of Int} * \text{ilT}$$

$$\text{type illT} = \text{NL} \mid \text{CL of ilT} * \text{illT}$$



$$[\![x]\!] = [[3,3],[],[7,3]]$$

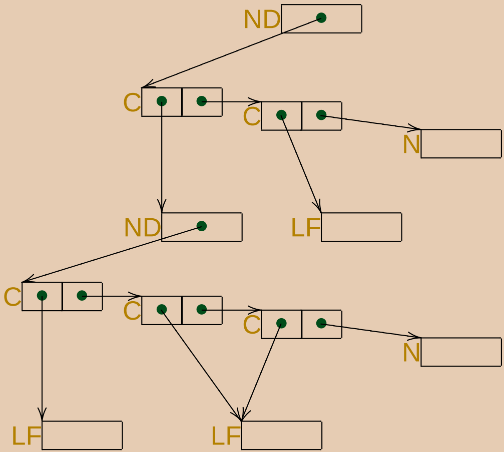
$$x : \text{illT}$$

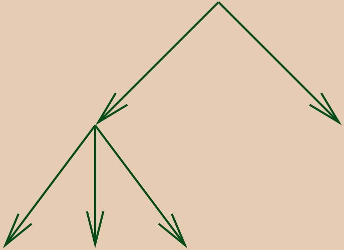
✔

Portions in mutually recursive datatypes

type treeT = LF | ND of forestT

type forestT = N | C of treeT * forestT





✔

Basic Separation Assertions

notation	name
$P1 \otimes P2$	P1 separated from P2

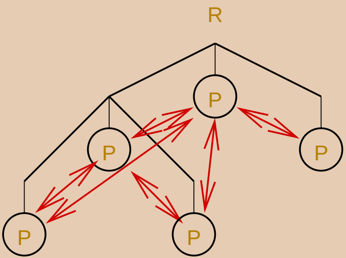
Basic Separation Assertions

notation	name
$P1 \otimes P2$	P1 separated from P2
$\otimes P / R$	P separated *along* R
$\otimes P \curlywedge R$	P separated *across* R

Basic Separation Assertions

notation	name
P1 $\otimes$ P2	P1 separated from P2
$\otimes$P/R	P separated *along* R
$\otimes$P$\curlywedge$R	P separated *across* R

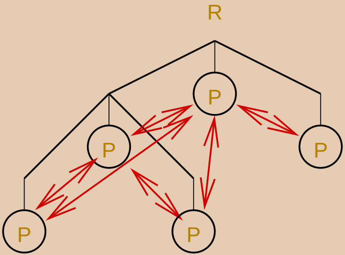
$\otimes$P/R



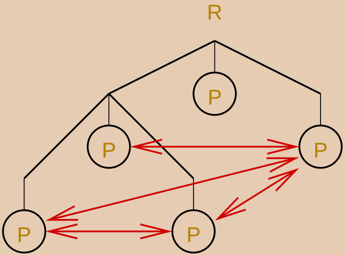
Basic Separation Assertions

notation	name
$P1 \otimes P2$	P1 separated from P2
$\otimes P / R$	P separated *along* R
$\otimes P \curlywedge R$	P separated *across* R

$\otimes P / R$



$\otimes P \curlywedge R$



Example typing judgement

list1 : ill, list2 : ill ⊢ *append* list1 list2 : ill

pre-condition:	{list1.CL ⊗ list2.CL}
portion containment:	CL ⊆ {list1.CL, list2.CL} NL ⊆ {list2.NL} CL1.C ⊆ {list1.CL1.C, list2.CL1.C} CL1.N ⊆ {list1.CL1.N, list2.CL1.N}
separation rely-guaratees:	⊗ CL1.C / ill ⇐ {list1.CL1.C ⊗ list2.CL1.C, ⊗ list1.CL1.C / ill, ⊗ list2.CL1.C / ill } ⊗ CL1.N / ill ⇐ {list1.CL1.N ⊗ list2.CL1.N, ⊗ list1.CL1.N / ill, ⊗ list2.CL1.N / ill }
not-preserved portions:	{list1.CL}

✔

Assertion inference

- No recursion $\Longrightarrow$ straight-forward bottom-up inference
- BUT rules are *hairy looking* (complex assertions), e.g.:

[CONS]

$$\frac{\begin{array}{ccc} & B_h = A[\tfrac{\delta}{\rho}] \quad B_t = L^{[\zeta]}(A[\tfrac{\delta'}{\rho'}]) & \\ E = L^{[\zeta' \subseteq \{\zeta_d\}]}(E_h) \cup E_t & E_h \in E_Y(B_h) & E_t \in E_Y(B_t) \end{array}}{\begin{array}{l} h : B_h, t : B_t, d : \Diamond^{[\zeta_d]}; \\ \{\zeta_d\} \otimes (N_D(B_h) \cup N_D(B_t)) \vdash \\ \mathsf{Cons}(h, t)@d \\ : E; \\ \{\zeta_d\}; \\ G \end{array}}$$

$h : B_h, t : B_t, d : \Diamond^{[\zeta_d]};$

$\{\zeta_d\} \otimes (N_D(B_h) \cup N_D(B_t)) \vdash$ — *separation pre-condition*

$\mathsf{Cons}(h, t)@d$

$: E;$ — *containment guarantees*

$\{\zeta_d\};$ — *destroyed portions*

G — *separation rely-guarantees*

Assertion inference

- No recursion $\Longrightarrow$ straight-forward bottom-up inference
- BUT rules are *hairy looking* (complex assertions), e.g.:

[CONS]

$$\frac{\begin{array}{ccc} & B_h = A[\tfrac{\delta}{\rho}] \quad B_t = L^{[\zeta]}(A[\tfrac{\delta'}{\rho'}]) & \\ E = L^{[\zeta' \subseteq \{\zeta_d\}]}(E_h) \cup E_t & E_h \in E_Y(B_h) & E_t \in E_Y(B_t) \end{array}}{\begin{array}{l} h : B_h, t : B_t, d : \Diamond^{[\zeta_d]}; \\ \{\zeta_d\} \otimes (N_D(B_h) \cup N_D(B_t)) \vdash \\ \mathsf{Cons}(h, t)@d \\ : E; \\ \{\zeta_d\}; \\ G \end{array}}$$

$\{\zeta_d\} \otimes (N_D(B_h) \cup N_D(B_t)) \vdash$	*separation pre-condition*
$: E;$	*containment guarantees*
$\{\zeta_d\};$	*destroyed portions*
G	*separation rely-guarantees*

- With recursion: iterative inference

✔

Iterative inference

1. Assume ideal assertions for all functions $\Gamma \vdash e : A; \top$

pre-condition:	$\emptyset$
portion containment:	$\emptyset$
separation rely-guaratees:	$\emptyset$
not-preserved portions:	$\emptyset$

Iterative inference

1. Assume ideal assertions for all functions
2. Infer assertions for function bodies

$\Gamma \vdash e : A; \top$

$\Gamma \vdash e : A; \phi_1$

$\Gamma \vdash e : A; \phi_2,\ \phi_2 \implies \phi_1$

pre-condition:	∅		
portion containment:	CL	⊆	{list1.CL, list2.CL}
	NL	⊆	{list2.NL}
	CL1.C	⊆	{list1.CL1.C, list2.CL1.C}
	CL1.N	⊆	{list1.CL1.N, list2.CL1.N}
separation rely-guaratees:	⊗ CL1.C / ill	⇐	{⊗ list2.CL1.C / ill}
	⊗ CL1.N / ill	⇐	{⊗ list2.CL1.N / ill}
not-preserved portions:	{list1.CL}		

Iterative inference

1. Assume ideal assertions for all functions
2. Infer assertions for function bodies
3. Adjust assertions of functions

$$\Gamma \vdash e : A; \top$$
$$\Gamma \vdash e : A; \phi_1$$
$$\Gamma \vdash e : A; \phi_2,\ \phi_2 \implies \phi_1$$

pre-condition:	∅		
portion containment:	CL	⊆	{list1.CL, list2.CL}
	NL	⊆	{list2.NL}
	CL1.C	⊆	{list1.CL1.C, list2.CL1.C}
	CL1.N	⊆	{list1.CL1.N, list2.CL1.N}
separation rely-guaratees:	⊗ CL1.C / ill	⇐	{⊗ list2.CL1.C / ill}
	⊗ CL1.N / ill	⇐	{⊗ list2.CL1.N / ill}
not-preserved portions:	{list1.CL}		

Iterative inference

1. Assume ideal assertions for all functions
2. Infer assertions for function bodies
3. Adjust assertions of functions
4. Repeat 2 and 3 until fixpoint

$$\Gamma \vdash e : A; \top$$
$$\Gamma \vdash e : A; \phi_1$$
$$\Gamma \vdash e : A; \phi_2,\ \phi_2 \implies \phi_1$$
$$\ldots$$
$$\Gamma \vdash e : A; \phi_n,\ \phi_n \iff \phi_{n-}$$

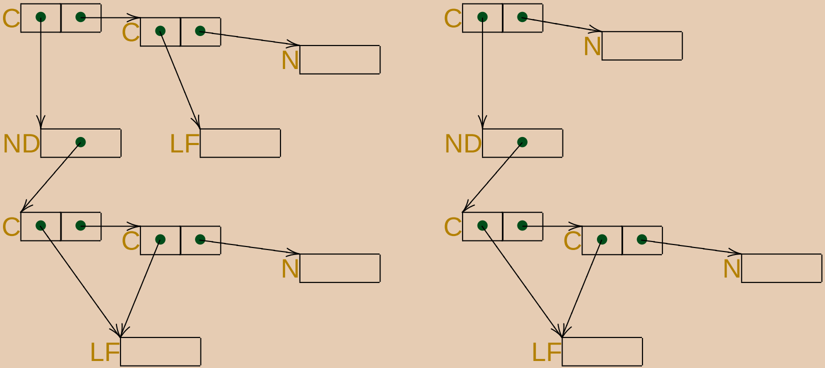
pre-condition:	{list1.CL ⊗ list2.CL}		
portion containment:	CL ⊆ {list1.CL, list2.CL}		
	NL ⊆ {list2.NL}		
	CL1.C ⊆ {list1.CL1.C, list2.CL1.C}		
	CL1.N ⊆ {list1.CL1.N, list2.CL1.N}		
separation rely-guaratees:	⊗ CL1.C / ill	⇐	{list1.CL1.C ⊗ list2.CL1.C, ⊗ list1.CL1.C / ill, ⊗ list2.CL1.C / ill }
	⊗ CL1.N / ill	⇐	{list1.CL1.N ⊗ list2.CL1.N, ⊗ list1.CL1.N / ill, ⊗ list2.CL1.N / ill }
not-preserved portions:	{list1.CL}		

✔

Forest append assertions

f1 : forestT, f2 : forestT ⊢ *forest_append* f1 f2 : forestT

pre-condition: { f1.C ⊗ f2.C,
f1.N ⊗ f2.N,
⊗ f1.C λ forestT,
⊗ f1.N λ forestT }

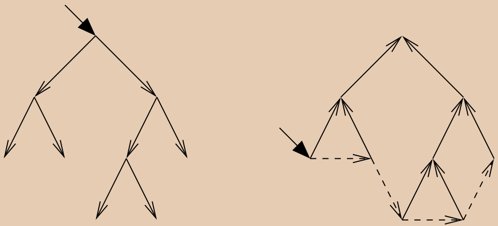


✔

Pathlist assertions

tree : bt ⊢ *pathlist* tree : ill

pre-condition:	{ ⊗ tree.LF λ btree, ⊗ tree.ND λ btree }
portion containment:	CL ⊆ {tree.LF} NL ⊆ ∅ CL1.C ⊆ {tree.ND} CL1.N ⊆ ∅
separation rely-guaratees:	⊗ CL1.C / ill ⇐ {⊥} ⊗ CL1.N / ill ⇐ {⊥}
not-preserved portions:	{tree.LF, tree.ND}



✔

Conclusion

- powerful and feasible static analysis of functional in-place update
- allows sharing among read-only layers
- assertions of quadratic size
- implemented for LFPL and most of Camelot

Conclusion

- powerful and feasible static analysis of functional in-place update
- allows sharing among read-only layers
- assertions of quadratic size
- implemented for LFPL and most of Camelot
- Future: – Infer explicit deallocation for Camelot

Conclusion

- powerful and feasible static analysis of functional in-place update
- allows sharing among read-only layers
- assertions of quadratic size
- implemented for LFPL and most of Camelot
- Future:
  - Infer explicit deallocation for Camelot
  - Enlarge scope: polymorphism, limited higher order

Conclusion

- powerful and feasible static analysis of functional in-place update
- allows sharing among read-only layers
- assertions of quadratic size
- implemented for LFPL and most of Camelot
- Future:
  - Infer explicit deallocation for Camelot
  - Enlarge scope: polymorphism, limited higher order
  - Infer formal proofs of non-interference

Conclusion

- powerful and feasible static analysis of functional in-place update
- allows sharing among read-only layers
- assertions of quadratic size
- implemented for LFPL and most of Camelot
- Future:
  - Infer explicit deallocation for Camelot
  - Enlarge scope: polymorphism, limited higher order
  - Infer formal proofs of non-interference
  - MRG:
    - infer resource-usage assertions for Camelot pr
    - and generate their proofs
    - need also non-interference assertions and thei

✔