Techniques for Formal Modelling and Verification on Dynamic Memory Allocators

Bin FANG[1,2]

[1] East China Normal University, Shanghai, China
[2] IRIF, University Paris Diderot and CNRS, Paris, France

Phd defense, Shanghai 11/09/2018

Contents

1. **Dynamic memory allocators (DMAs)**
    1. Importance and challenges
    2. Diverse design tactics
    3. Informal properties

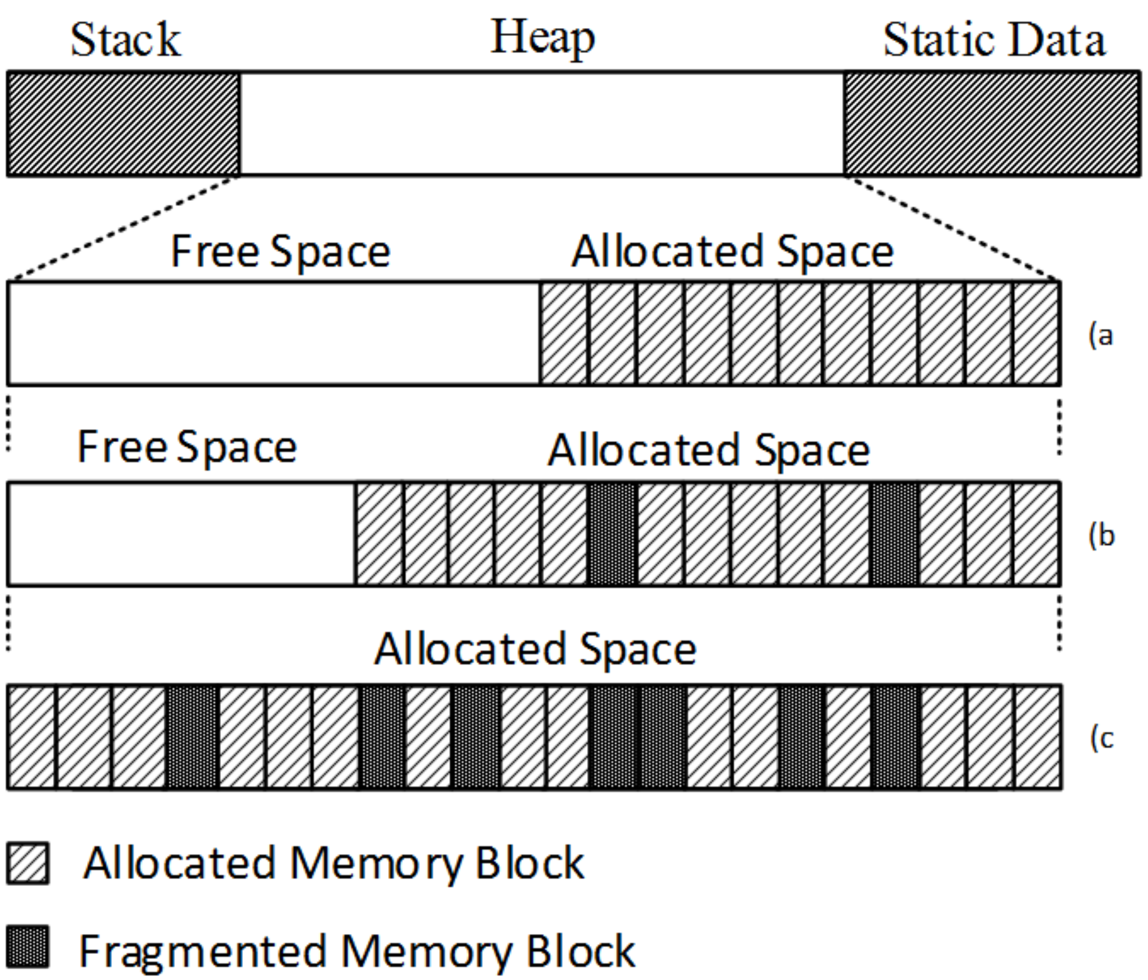
2. **Top-down formal modelling of DMAs**
    1. Specification using Event-B
    2. Modular and stepwise refinement

3. **Algorithmic verification by static analysis**
    1. Separation logic fragment **SLMA**
    2. Logic based abstractions
    3. Static analysis based on abstract interpretation

4. **Conclusion and perspectives**

Dynamic Memory Allocators

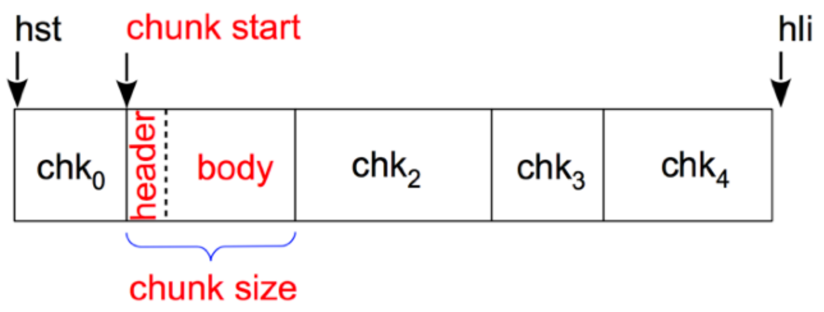
- Operating system, e.g., RTOS
- Programming language library
- Diverse features



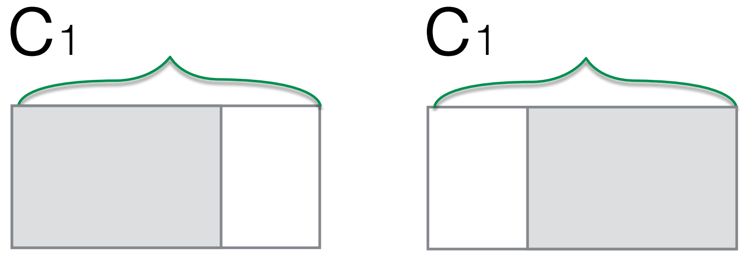

```
void init(); //initialization
bool free(void* p); //deallocation
void* alloc(size_t sz); //allocation
void* realloc(void* p, size_t sz); //change size of p
```

Design tactics

- heap list: singly / doubly linked list (SLL,DLL)
- fit policy: first fit, best fit, next fit
- splitting
- defragmentation strategy (coalescing policy)
- free chunks management (free list, eg., SLL, DLL)
- …



heap list



Design tactics

- heap list: singly / doubly linked list (SLL,DLL)

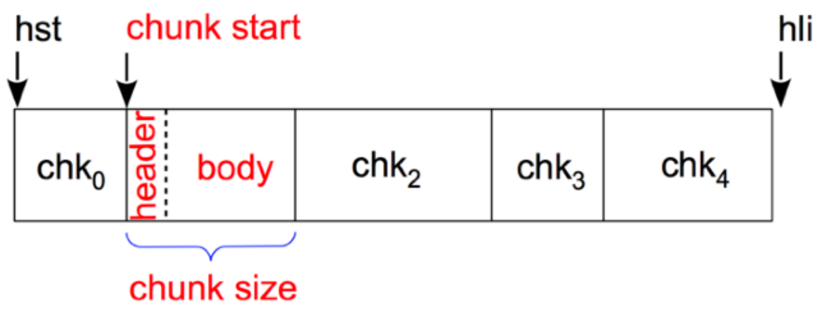
- fit policy: first fit, best fit, next fit

- splitting

- defragmentation strategy (coalescing policy)

- free chunks management (free list, eg., SLL, DLL)

- …

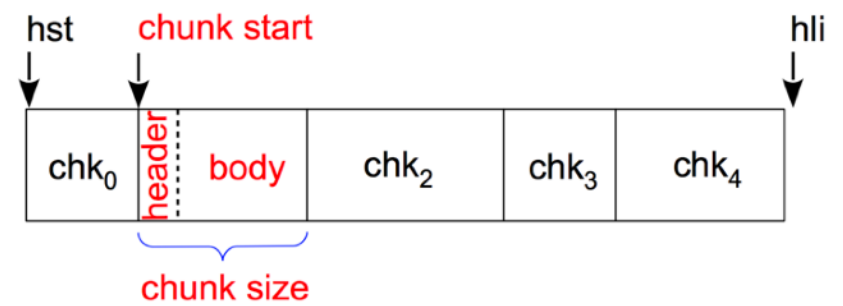


heap list

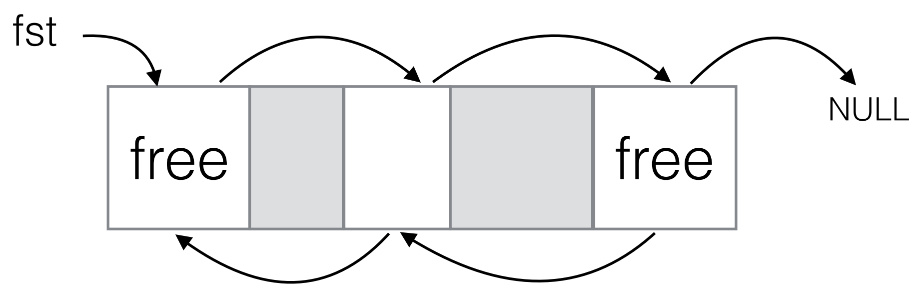
- *eager coalescing*
- *lazy coalescing*
- *no coalescing*

Design tactics

- heap list: singly / doubly linked list (SLL,DLL)
- fit policy: first fit, best fit, next fit
- splitting
- defragmentation strategy (coalescing policy)
- free chunks management (free list, eg., SLL, DLL)
- …



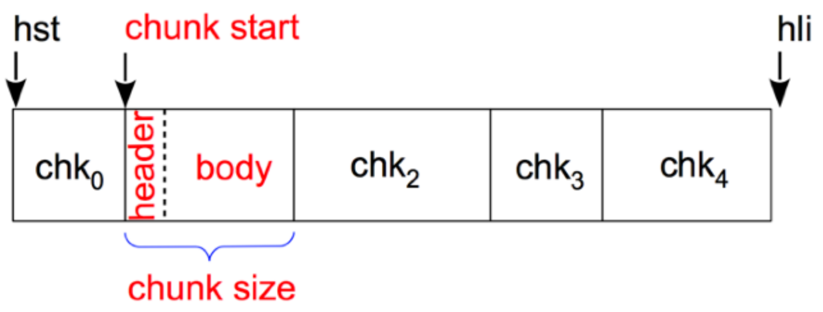
heap list



free list

Properties

- no memory leak
- no overlapped chunks
- adjacent free chunks
- shape of heap/free list: cyclic, acyclic
- sorting of free list: address sorted/unsorted

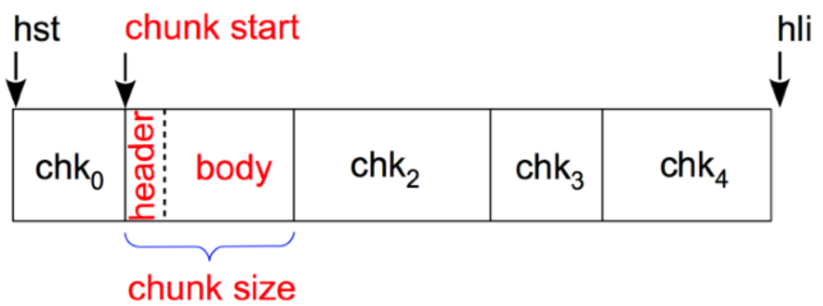


heap list

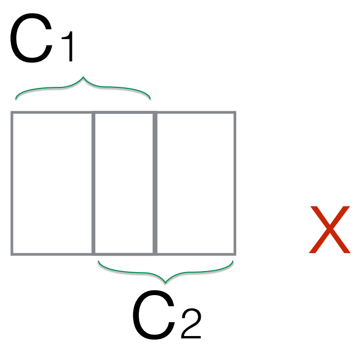
Dynamic Memory Allocators

Properties

- no memory leak
- no overlapped chunks
- adjacent free chunks
- shape of heap/free list: cyclic, acyclic
- sorting of free list: address sorted/unsorted

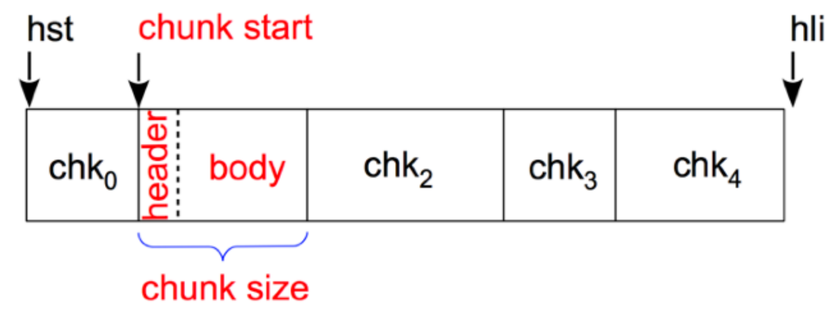


heap list

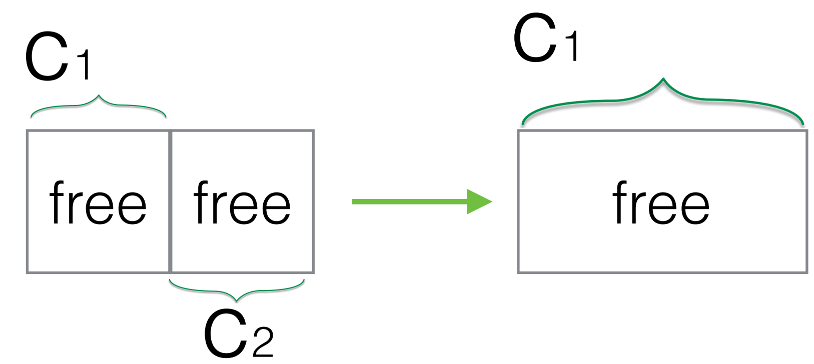


Properties

- no memory leak
- no overlapped chunks
- adjacent free chunks
- shape of heap/free list: cyclic, acyclic
- sorting of free list: address sorted/unsorted



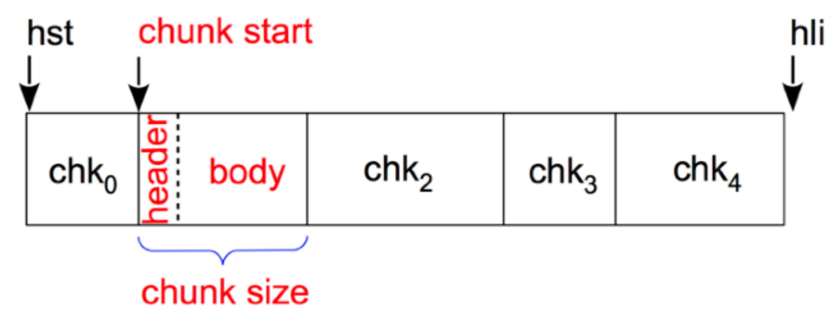
heap list



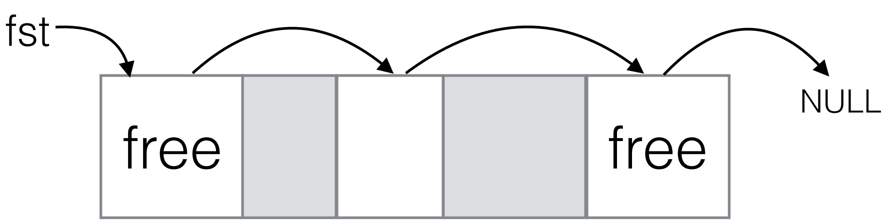
Dynamic Memory Allocators

Properties

- no memory leak
- no overlapped chunks
- adjacent free chunks
- shape of free list: cyclic, acyclic
- sorting of free list: address sorted/unsorted



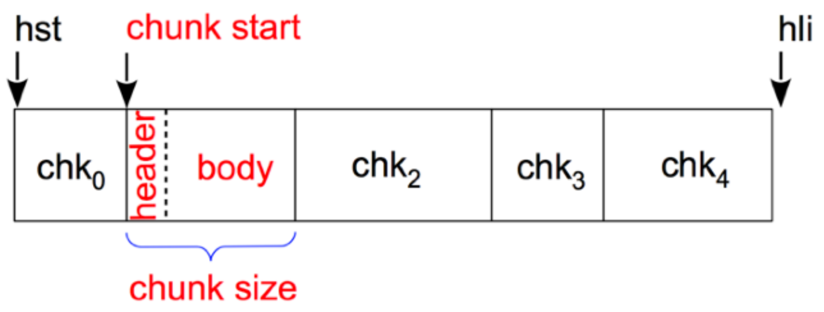
heap list



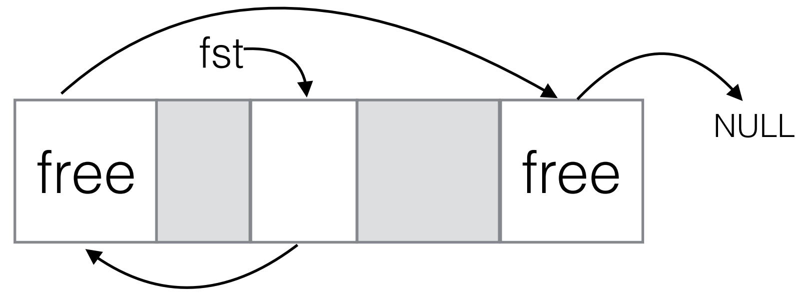
Properties

- no memory leak
- no overlapped chunks
- adjacent free chunks
- shape of free list: cyclic, acyclic
- sorting of free list: address sorted/unsorted

….



heap list



Each DMA has a set of tactics and properties

1. How to find a way to formalize?

2. How to design an abstract domain?

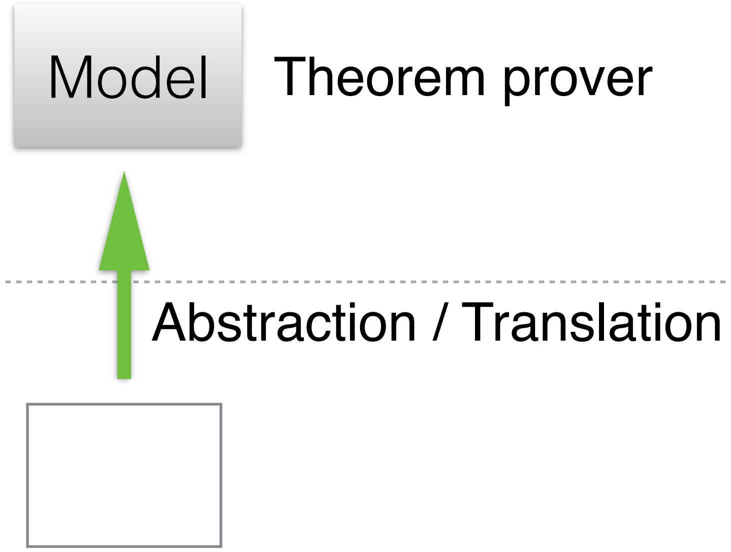
that apply to a large class of free-list DMAs, e.g.,

- IBM allocator: no heap-list, first-fit
- Kernighan&Ritchie alloc: eager coalescing, cyclic free-list, address sorted
- Lea's alloc: acyclic doubly linked free-list, unsorted, best-fit

PART I:
Formal modelling based on refinement

Procedure of Formalization

1. construct formal model (abstraction)
2. specify properties
3. auto-generate proof obligations
4. auto / interactive proof

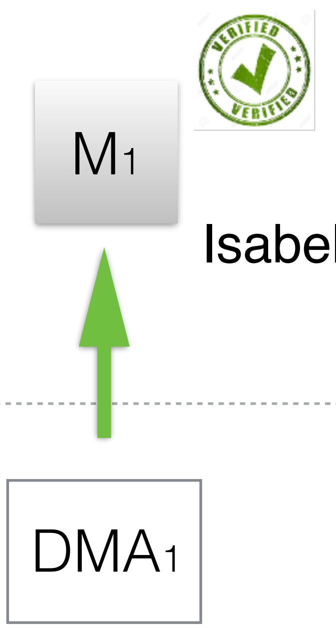


requirements: software design / algorithm

Different specification languages

[Tuch et al. 07]

abstract DMA formal model
+*properties*

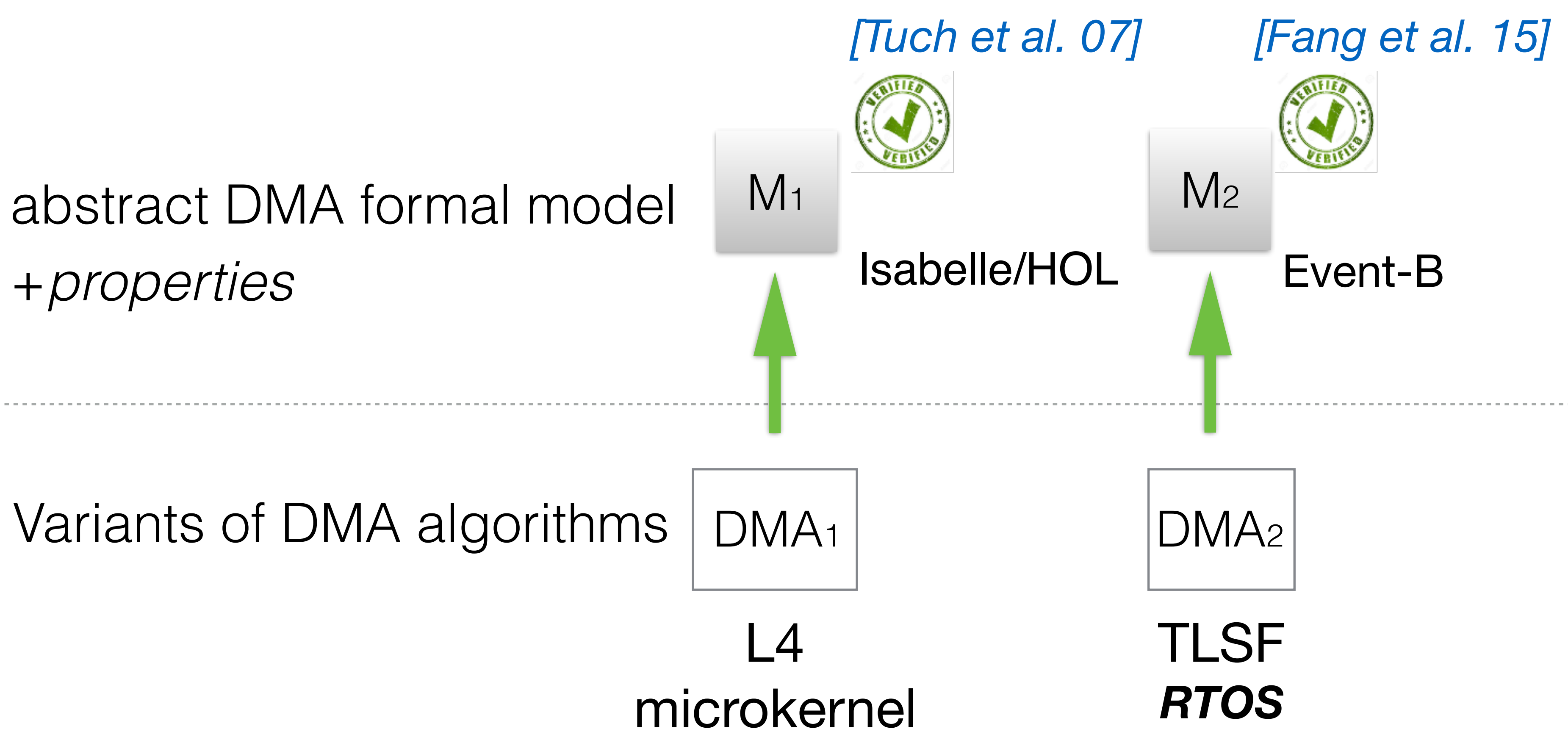


Isabelle/HOL

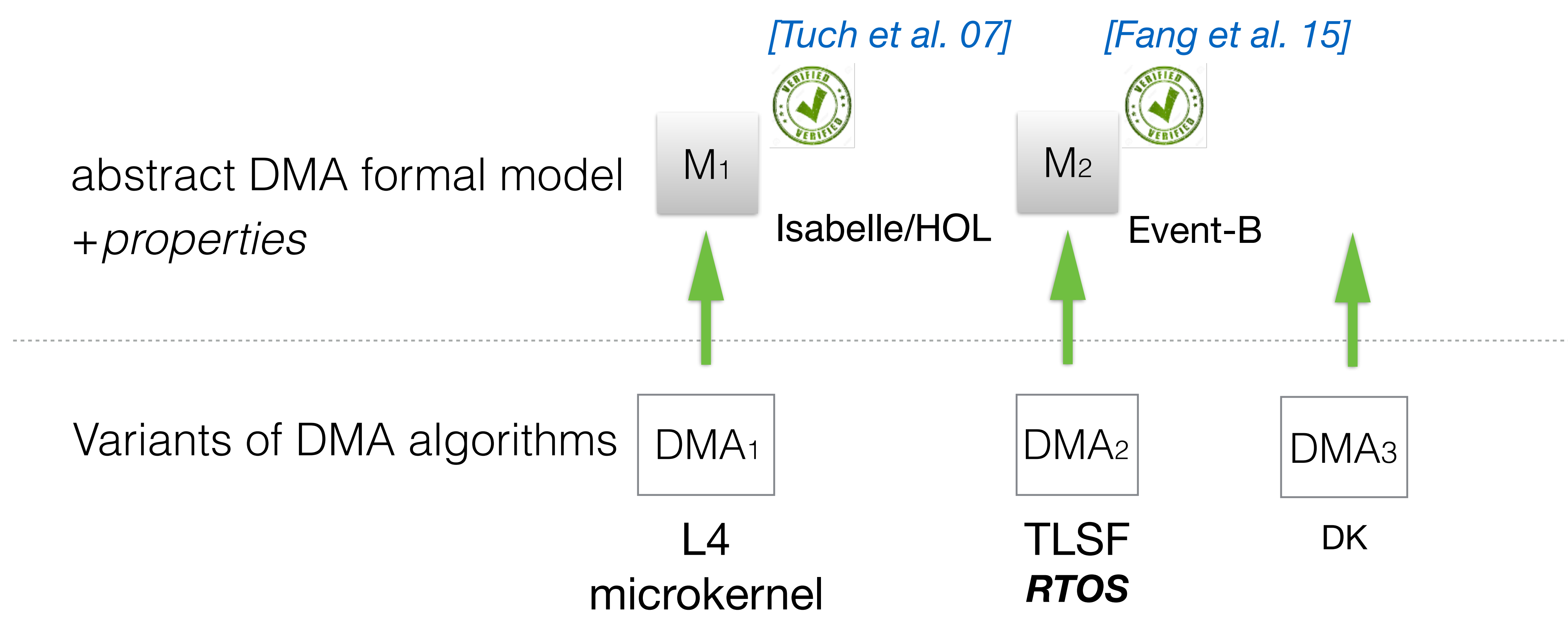
Variants of DMA algorithms

L4
microkernel

Different specification languages

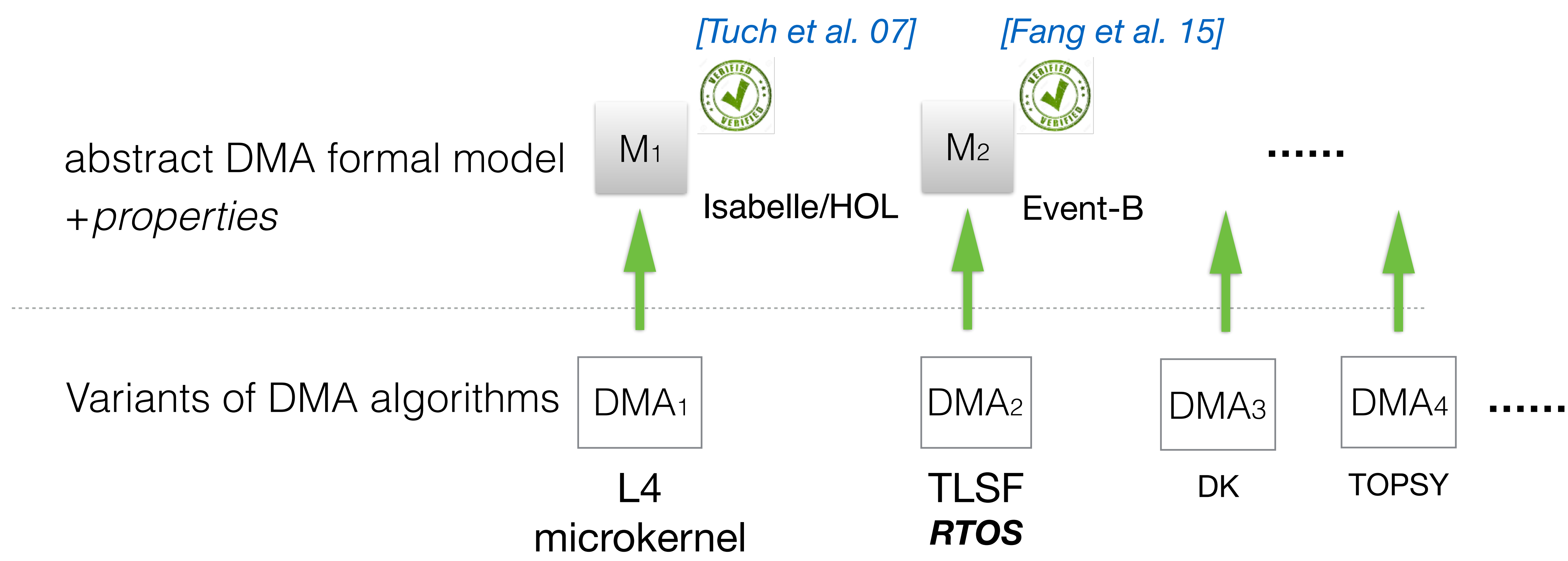


Different specification languages



Formalization of Dynamic Memory Allocators

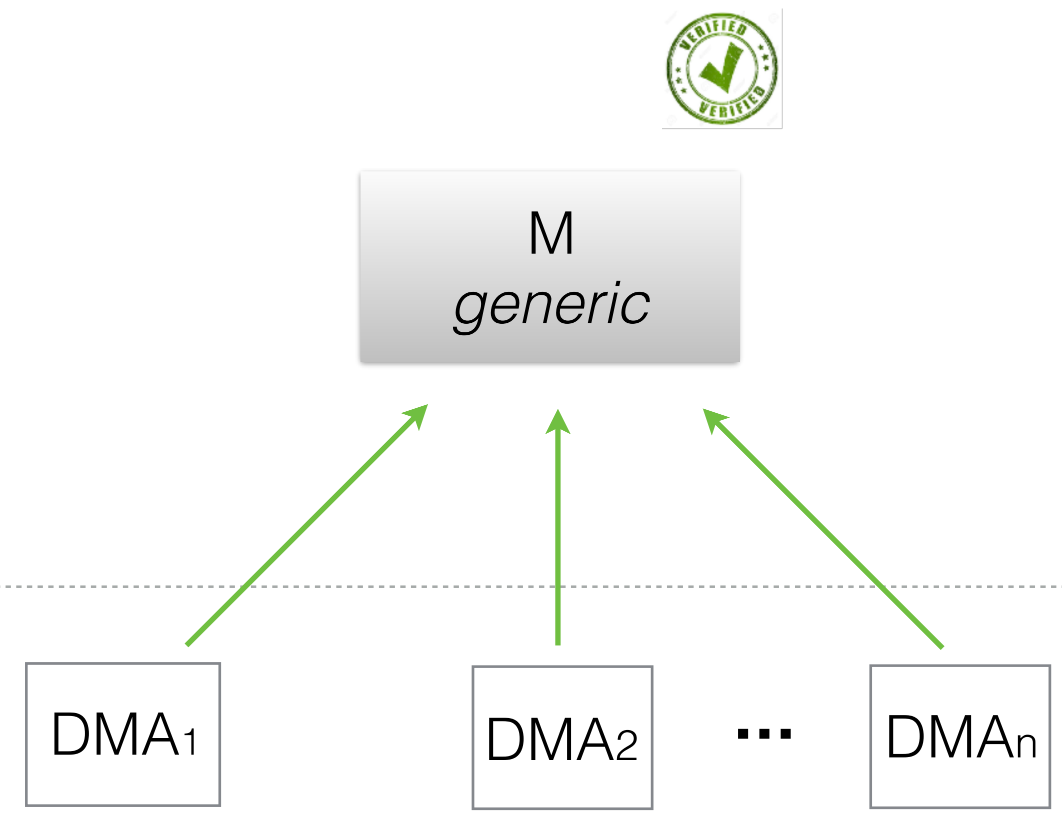
Different specification languages



A generic framework of formalization

abstract DMA formal model
+*properties*

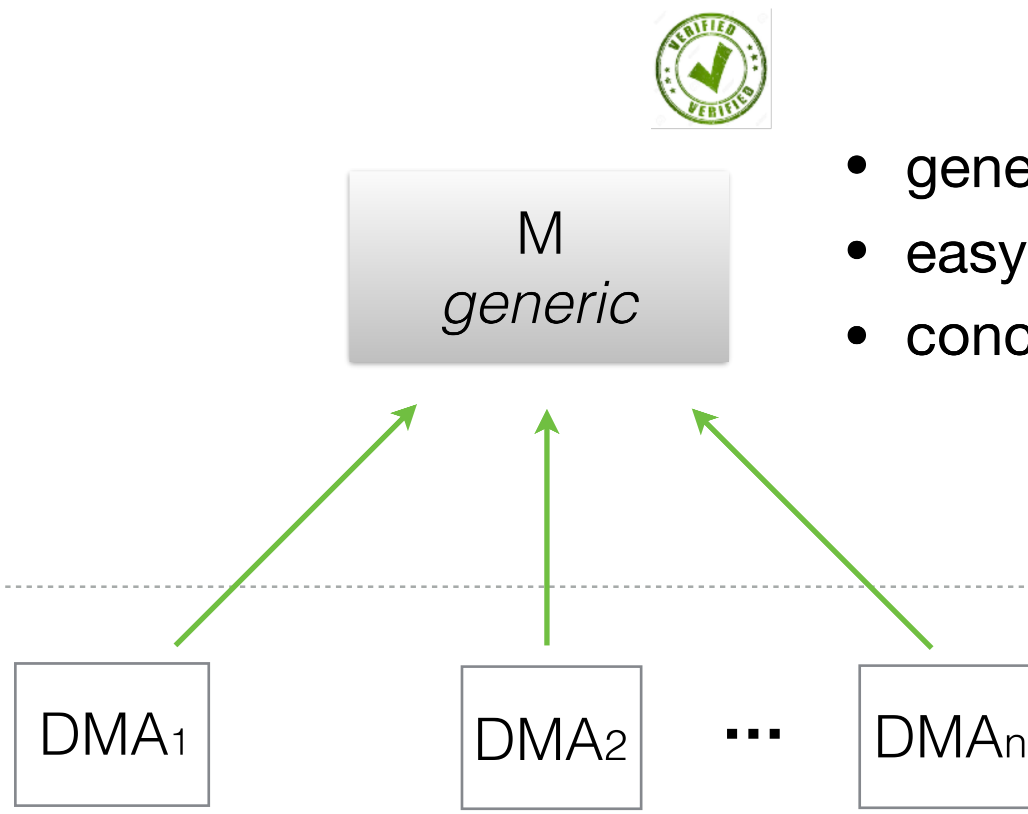
Variants of DMA algorithms



A generic framework of formalization

abstract DMA formal model
+*properties*

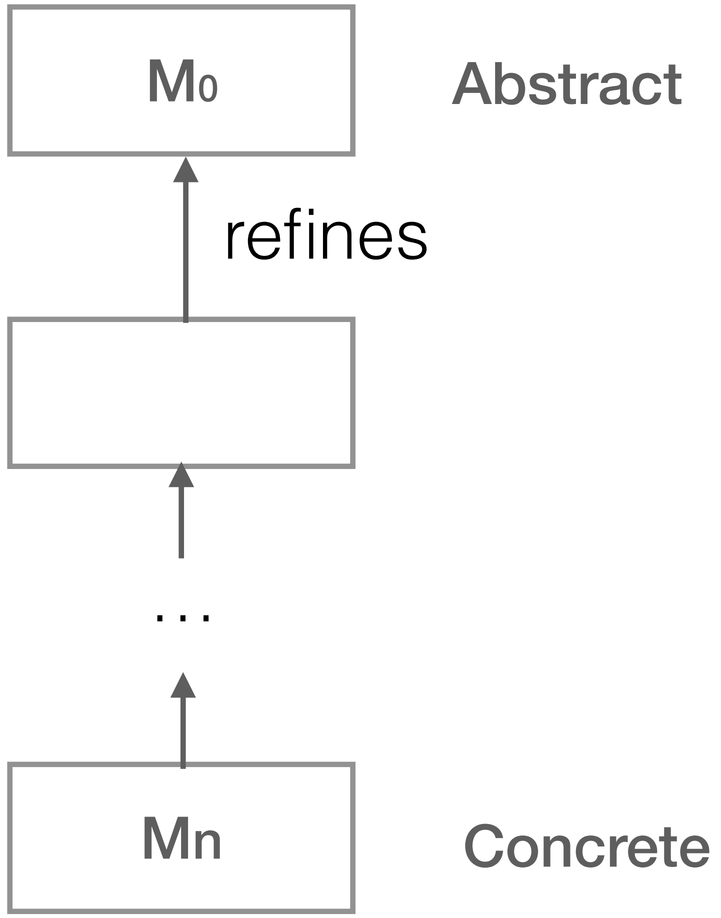
Variants of DMA algorithms



- generic
- easy to extend
- concreteness

Strategy of formalization

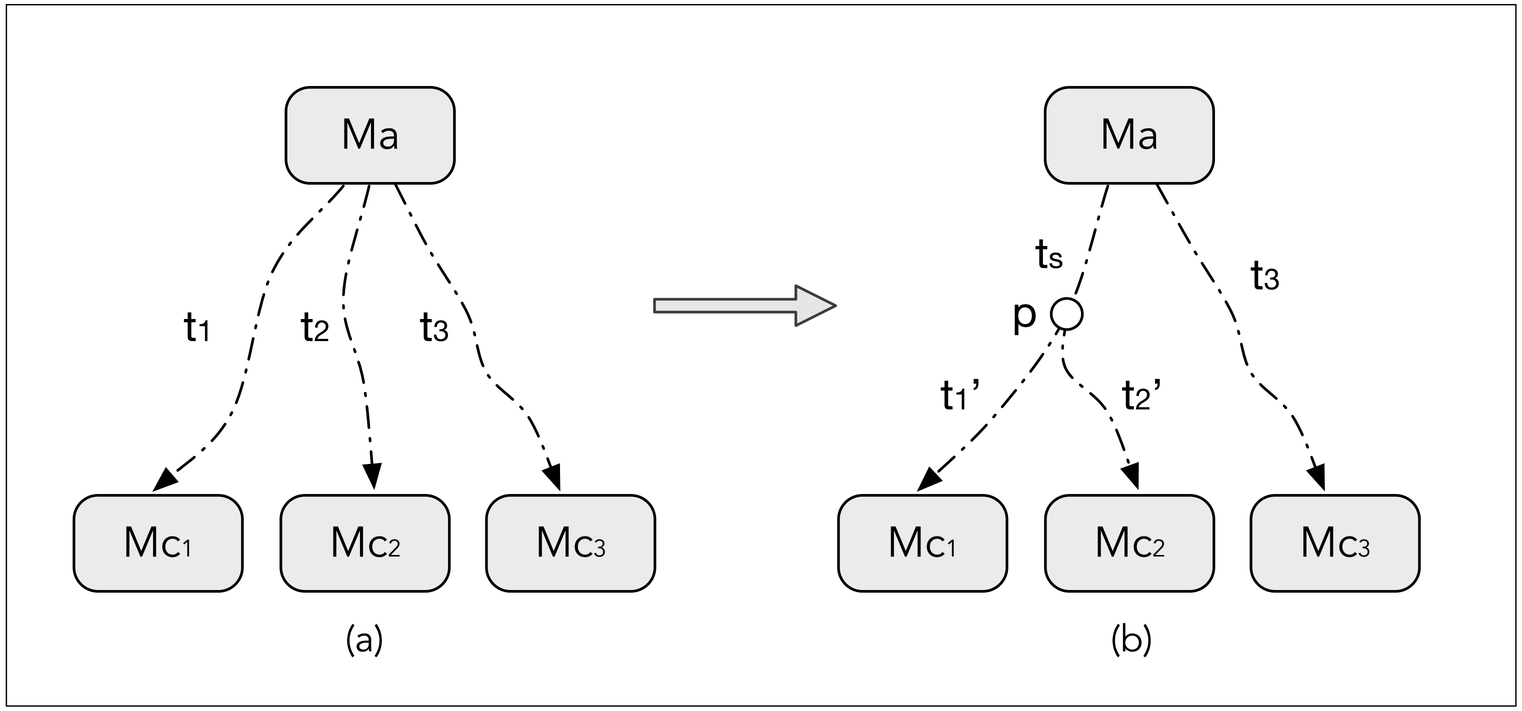
1. Event-based state transition system (Event-B modelling notation *[Tuch et al. 07]*)
2. Stepwise refinement (top-down)



TLSF *[Fang et al. 15]*

Strategy of formalization

1. Event-base state transition system
   (Event-B modelling notation *[Tuch et al. 07]*)
2. Stepwise refinement (top-down)
3. Modular formalisation



Formalization steps

1. Most abstract model (common interface)

Case study	heap list: linked	heap list: split	heap list: defrg.	free list: shape	free list: sort	fit
IBM [4]	addr, $\rightarrow$	–	–	–	–	F
DL-small [15]	size, $\rightarrow$	–	–	–	–	F
TOPSY [9]	size, $\rightarrow$	end	lazy	–	–	F
DKFF [14]	size, $\rightarrow$	start	early	A, $\rightarrow$	yes	F
DKBF [14]	size, $\rightarrow$	start	early	A, $\rightarrow$	yes	B
LA [3]	size, $\rightarrow$	start	early	A, $\rightarrow$	yes	F
DKNF [14]	size, $\rightarrow$	start	early	A, $\rightarrow$	yes	N
KR [12]	size, $\rightarrow$	start	early	C, $\rightarrow$	yes	N
DKBT [14]	size, $\leftrightarrow$	start	early	A, $\leftrightarrow$	no	B
DL-list [15]	size, $\leftrightarrow$	start	early	A, $\leftrightarrow$	no	B
TLSF [19]	size, $\leftrightarrow$	start	early	A, $\leftrightarrow$	no	B

case studies

1

```
void  init();
bool  free(void* p);
void* alloc(size_t sz);
void* realloc(void* p, size_t sz);
```

interface for clients

Formalization steps

1. Most abstract model (common interface)

1 *abstract model*

Case study	*heap list*			*free list*		*fit*
	linked	*split*	*defrg.*	*shape*	*sort*	
IBM [4]	addr, →	–	–	–	–	F
DL-small [15]	size, →	–	–	–	–	F
TOPSY [9]	size, →	end	lazy	–	–	F
DKFF [14]	size, →	start	early	A, →	yes	F
DKBF [14]	size, →	start	early	A, →	yes	B
LA [3]	size, →	start	early	A, →	yes	F
DKNF [14]	size, →	start	early	A, →	yes	N
KR [12]	size, →	start	early	C, →	yes	N
DKBT [14]	size, ↔	start	early	A, ↔	no	B
DL-list [15]	size, ↔	start	early	A, ↔	no	B
TLSF [19]	size, ↔	start	early	A, ↔	no	B

case studies

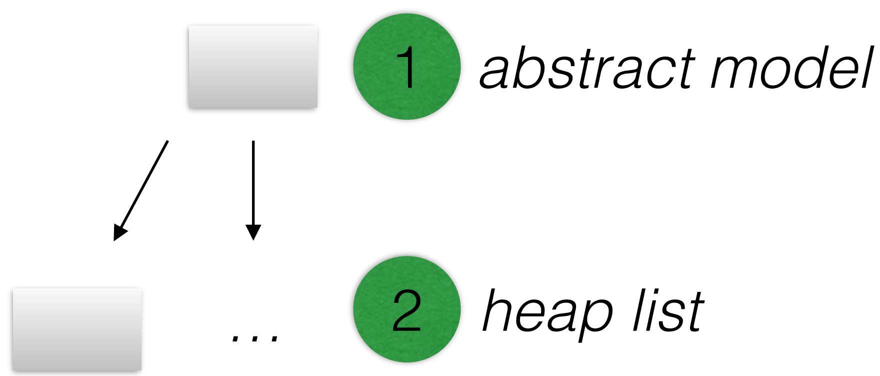
1

```
void   init();
bool   free(void* p);
void*  alloc(size_t sz);
void*  realloc(void* p, size_t sz);
```

interface for clients

Formalization steps

1. Most abstract model (common interface)
2. heap list types



2

	heap list			*free list*		
Case study	*linked*	*split*	*defrg.*	*shape*	*sort*	*fit*
IBM [4]	addr, →	–	–	–	–	F
DL-small [15]	size, →	–	–	–	–	F
TOPSY [9]	size, →	end	lazy	–	–	F
DKFF [14]	size, →	start	early	A, →	yes	F
DKBF [14]	size, →	start	early	A, →	yes	B
LA [3]	size, →	start	early	A, →	yes	F
DKNF [14]	size, →	start	early	A, →	yes	N
KR [12]	size, →	start	early	C, →	yes	N
DKBT [14]	size, ↔	start	early	A, ↔	no	B
DL-list [15]	size, ↔	start	early	A, ↔	no	B
TLSF [19]	size, ↔	start	early	A, ↔	no	B

case studies

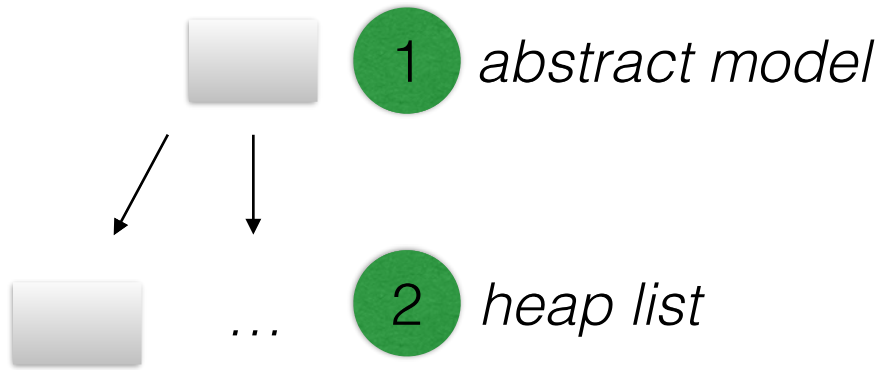
Formalization steps

1. Most abstract model (common interface)
2. heap list types

2

Case study	heap list			free list		fit
	linked	*split*	*defrg.*	*shape*	*sort*	*fit*
IBM [4]	addr, →	–	–	–	–	F
DL-small [15]	size, →	–	–	–	–	F
TOPSY [9]	size, →	end	lazy	–	–	F
DKFF [14]	size, →	start	early	A, →	yes	F
DKBF [14]	size, →	start	early	A, →	yes	B
LA [3]	size, →	start	early	A, →	yes	F
DKNF [14]	size, →	start	early	A, →	yes	N
KR [12]	size, →	start	early	C, →	yes	N
DKBT [14]	size, ↔	start	early	A, ↔	no	B
DL-list [15]	size, ↔	start	early	A, ↔	no	B
TLSF [19]	size, ↔	start	early	A, ↔	no	B

case studies



Abstract allocation

Alloc
- status updating

1

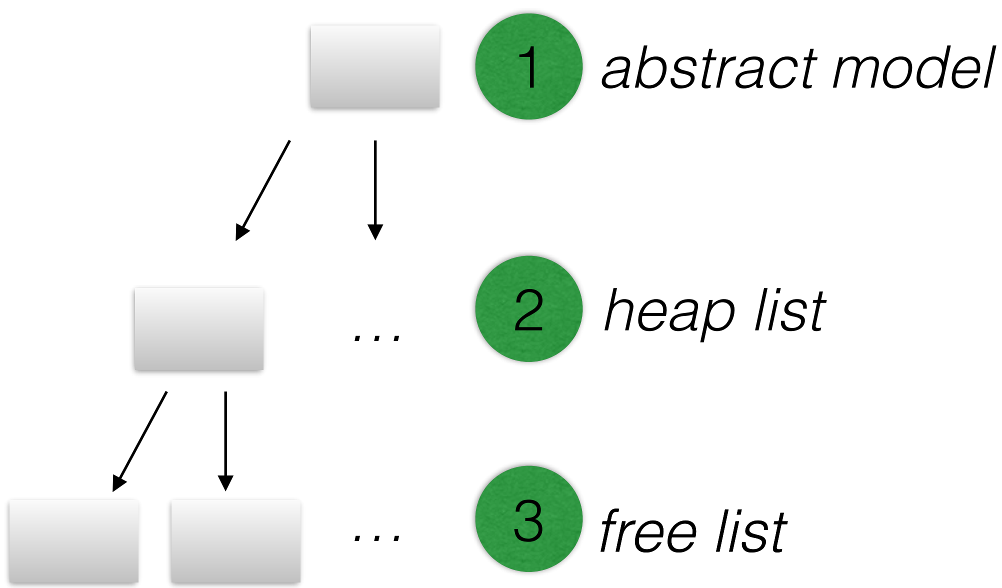
Refined allocation

Alloc
- searching
- splitting
- …

2

Formalization steps

1. Most abstract model (common interface)
2. heap list types
3. Free list types



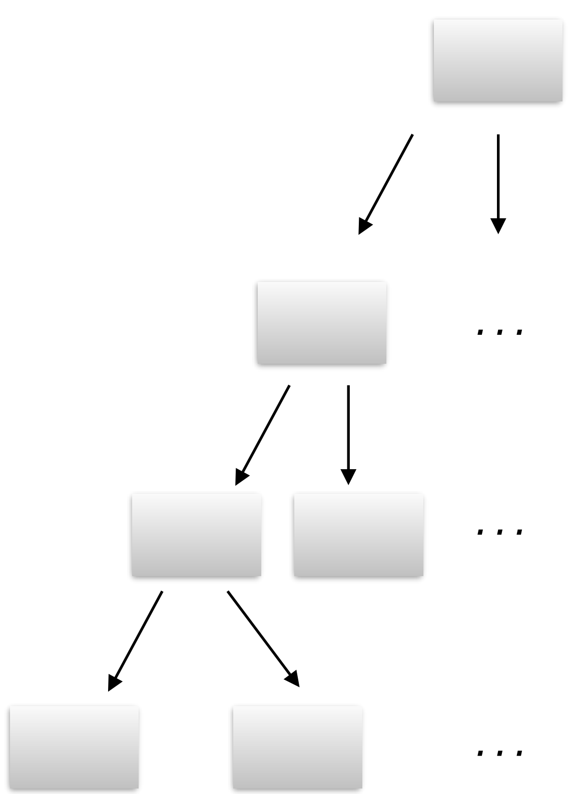
	heap list (2)			free list (3)		
Case study	*linked*	*split*	*defrg.*	*shape*	*sort*	*fit*
IBM [4]	addr, →	–	–	–	–	F
DL-small [15]	size, →	–	–	–	–	F
TOPSY [9]	size, →	end	lazy	–	–	F
DKFF [14]	size, →	start	early	A, →	yes	F
DKBF [14]	size, →	start	early	A, →	yes	B
LA [3]	size, →	start	early	A, →	yes	F
DKNF [14]	size, →	start	early	A, →	yes	N
KR [12]	size, →	start	early	C, →	yes	N
DKBT [14]	size, ↔	start	early	A, ↔	no	B
DL-list [15]	size, ↔	start	early	A, ↔	no	B
TLSF [19]	size, ↔	start	early	A, ↔	no	B

case studies

3

Formalization steps

1. Most abstract model (common interface)
2. heap list types
3. Free list types
4. Fit policies



1 *abstract model*

2 *heap list*

3 *free list*

4 *fit policy*

	2			3		4
	heap list			*free list*		
Case study	*linked*	*split*	*defrg.*	*shape*	*sort*	*fit*
IBM [4]	addr, →	–	–	–	–	F
DL-small [15]	size, →	–	–	–	–	F
TOPSY [9]	size, →	end	lazy	–	–	F
DKFF [14]	size, →	start	early	A, →	yes	F
DKBF [14]	size, →	start	early	A, →	yes	B
LA [3]	size, →	start	early	A, →	yes	F
DKNF [14]	size, →	start	early	A, →	yes	N
KR [12]	size, →	start	early	C, →	yes	N
DKBT [14]	size, ↔	start	early	A, ↔	no	B
DL-list [15]	size, ↔	start	early	A, ↔	no	B
TLSF [19]	size, ↔	start	early	A, ↔	no	B

case studies

Refined allocation

Alloc
- searching
- splitting
- …

3

Refined allocation

Alloc
- **searching**
- splitting
- …

4

Formalization of Dynamic Memory Allocators

Hierarchy of models

1. Extensible hierarchy
2. Clear refinement principles
3. Covers diverse DMAs

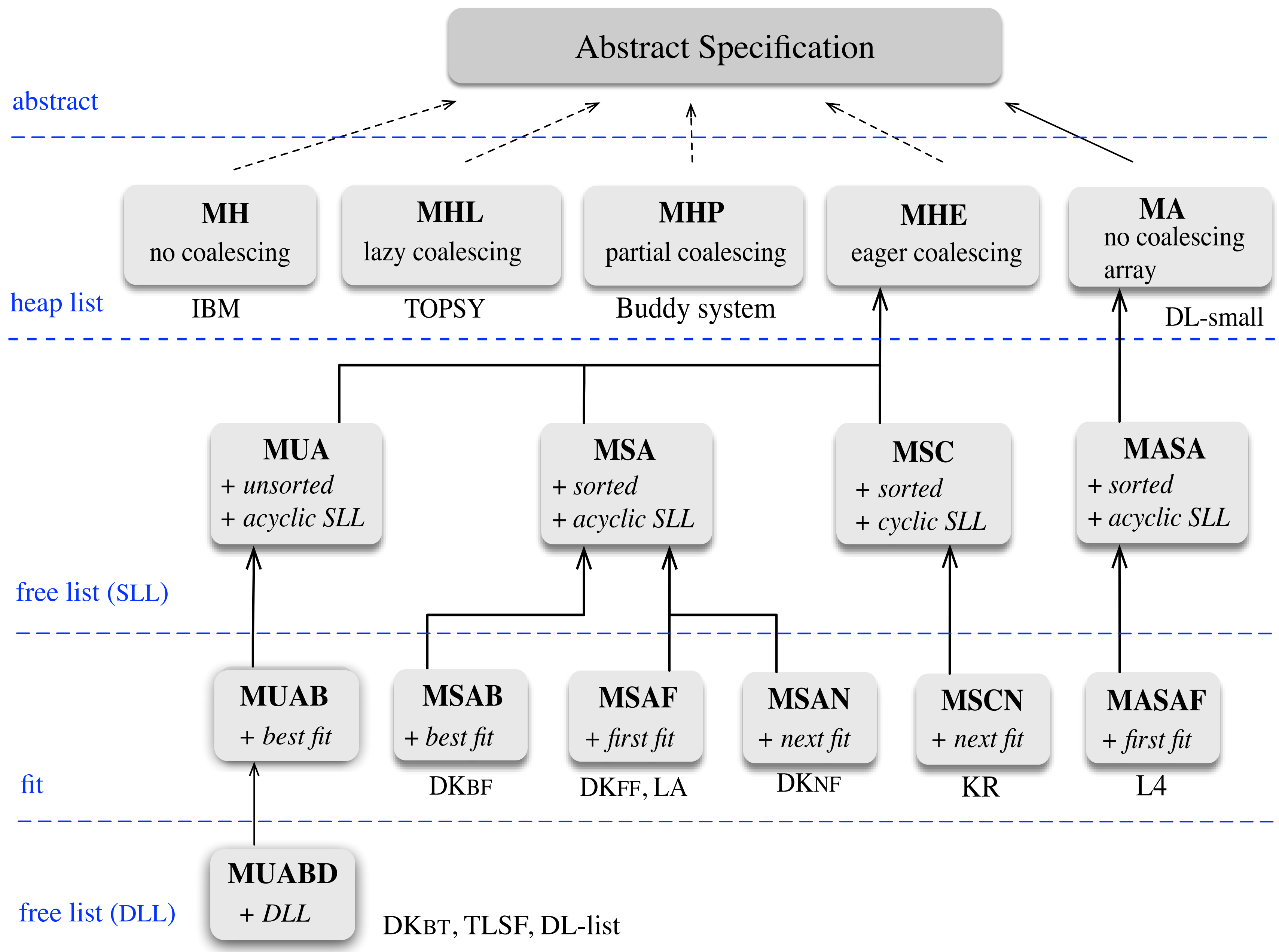


fig. A partial view of the hierarchy of models and the case studies it covers

Hierarchy of models

Theorem: Models consistency

Each model is proved.

Theorem: Refinement correctness

The refinement relations between models are valid.

Models	LOC	Proof obligations	Automatically discharged	Interactive proofs
MH	114	39	27(69%)	12(31%)
MHL	176	8	8(100%)	0(0%)
MHE	183	82	58(70%)	24(30%)
MHP	383	143	140(98%)	3(2%)
MA	168	20	20(100 %)	0 (0%)

Models	LOC	Proof obligations	Automatically discharged	Interactive proofs
MUA	219	36	30(83%)	6(17%)
MSA	197	41	27(66%)	14(34%)
MSC	205	37	30(82%)	7(18%)
MSAB	202	2	2(100%)	0(0%)
MSAF	202	2	2(100%)	0(0%)
MSAN	200	2	2(100%)	0(0%)
MSCN	221	40	36(88%)	4(12%)
MUABD	241	9	9(100%)	0(0%)
MASA	182	21	18(85.6%)	3(14.4%)
MASF	186	2	2(100%)	0(0%)

fig. Statistics on proofs

ISMM'17, SCIS'17 (journal)

PART II:
Algorithmic verification by static analysis

Dynamic Memory Allocators implementation

- Small but critical piece of code
- Variety of policies and techniques [*Wilson et al 95*]
- Combines low-level (**pointer arithmetics**, system calls) and high level (dynamic data structures) code
- Complex properties (invariants) on both levels

Properties

- Spatial properties for structure of disjoint memory
- Intricate numerical properties for data, e.g., memory's size and content
- Different levels of abstractions (heap and free lists)

Aim: automatically infer DMA properties

- Logical abstract domain on Separation Logic *[O'Hearn,Reynolds,Yang'01]*
- Combination of domains
- Hierarchical abstraction

Dynamic Memory Allocators implementation

- Small but critical piece of code
- Variety of policies and techniques [*Wilson et al 95*]
- Combines low-level (**pointer arithmetics**, system calls) and high level (dynamic data structures) code
- Complex properties (invariants) on both levels

Properties

- Spatial properties for structure of disjoint memory
- Intricate numerical properties for data, e.g., memory's size and content
- Different levels of abstractions (heap and free lists)

Aim: automatically infer DMA properties

- Logical abstract domain on Separation Logic *[O'Hearn,Reynolds,Yang'01]*
- Combination of domains
- Hierarchical abstraction

Dynamic Memory Allocators implementation

- Small but critical piece of code
- Variety of policies and techniques [*Wilson et al 95*]
- Combines low-level (**pointer arithmetics**, system calls) and high level (dynamic data structures) code
- Complex properties (invariants) on both levels

Properties

- Spatial properties for structure of disjoint memory
- Intricate numerical properties for data, e.g., memory's size and content
- Different levels of abstractions (heap and free lists)

Aim: automatically infer DMA properties

- Logical abstract domain on Separation Logic *[O'Hearn,Reynolds,Yang'01]*
- Combination of domains
- Hierarchical abstraction

Static analysis based on abstract interpretation [Cousot 77,79]

- Design **abstract domains** to capture properties
- Lattice operators $(S, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- Termination or acceleration of iteration (widening operation)
- Abstract transformers (assignments, condition tests, …)

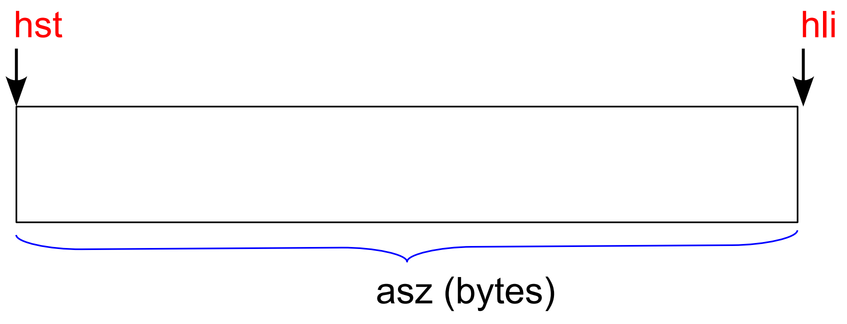
Logic-based shape analysis

- Abstract elements uses formulae from logic [Distefano et al 06]
- Entailment represents partial order $(\phi \Rightarrow \psi \Leftrightarrow \phi \sqsubseteq \psi)$

Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

Raw memory region



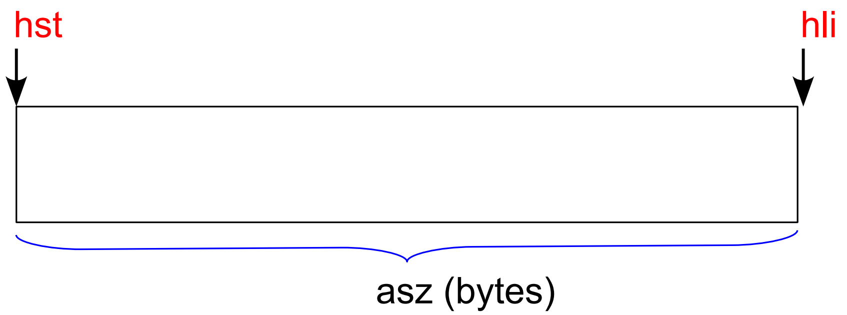

```
void minit(int asz)
{ … hst=sbrk(asz); hli=sbrk(0); … }
```

Memory Abstraction with Inductive Segments

Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

Raw memory region



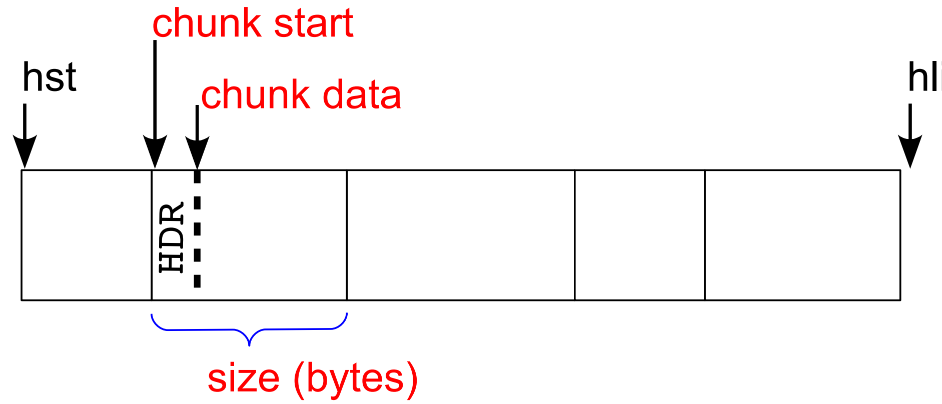

```
void minit(int asz)
{ … hst=sbrk(asz); hli=sbrk(0); … }
```

$$\mathsf{blk}(hst; hli) \;\wedge\; hli - hst = \mathsf{asz}$$

[Calcagno et at' 06]

Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$




```
typedef struct hdr_s {
  size_t size;
  bool isfree;
  struct hdr_s *fnx; } HDR;
```

Chunk region

$$\mathsf{chk}(hst; a_1) \star \mathsf{chk}(a_1; a_2) \star \cdots \star \mathsf{chk}(a_n; hli)$$

$$\mathsf{chk}(X; Y) \triangleq \exists Z \,.\, \mathsf{chd}(X; Z) \star \mathsf{blk}(Z; Y) \wedge Y - X = X \,.\, size$$

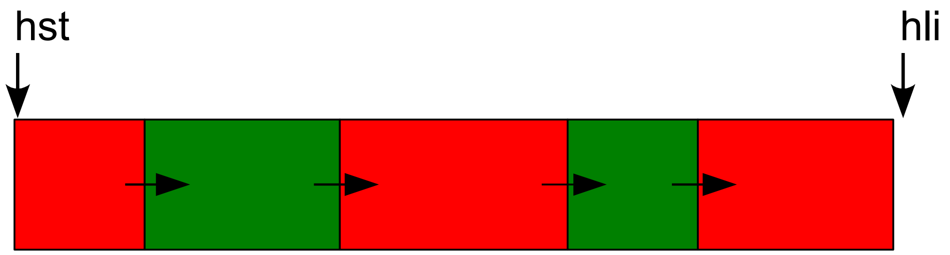
$$\mathsf{chd}(X; Y) \triangleq \mathsf{blk}(X; Y) \wedge Y - X = |\mathsf{HDR}| \wedge X \equiv_{|\mathsf{HDR}|} 0$$

[O'Hearn et al'01,Calcagno et al'06]

Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

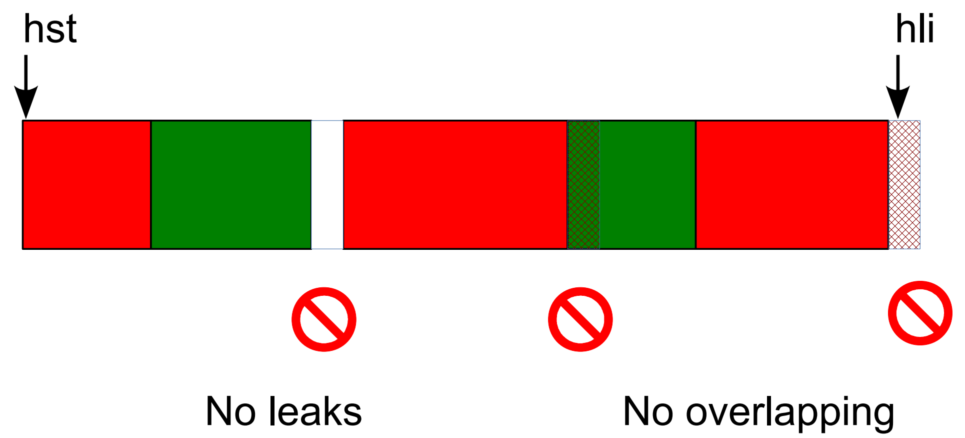
Heap list



Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

Heap list

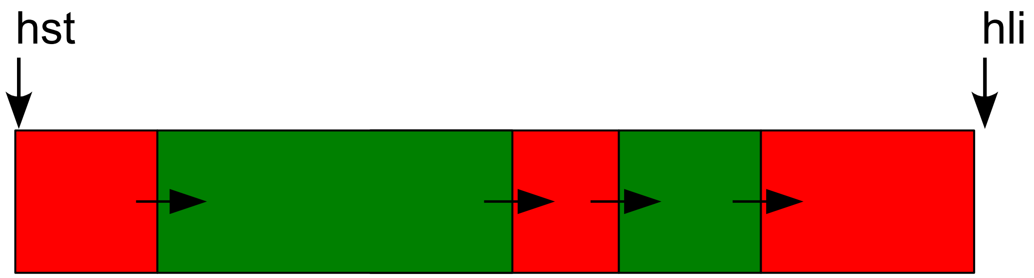


$$\mathsf{hls}(X;Y)[W] \triangleq \mathsf{emp} \wedge X = Y \wedge W = \epsilon$$
$$\vee\ \exists Z, W' \cdot \mathsf{chk}(X;Z) \star \mathsf{hls}(Z;Y)[W'] \wedge W = [X]\,.\,W'$$

Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

Heap list with coalescing policy

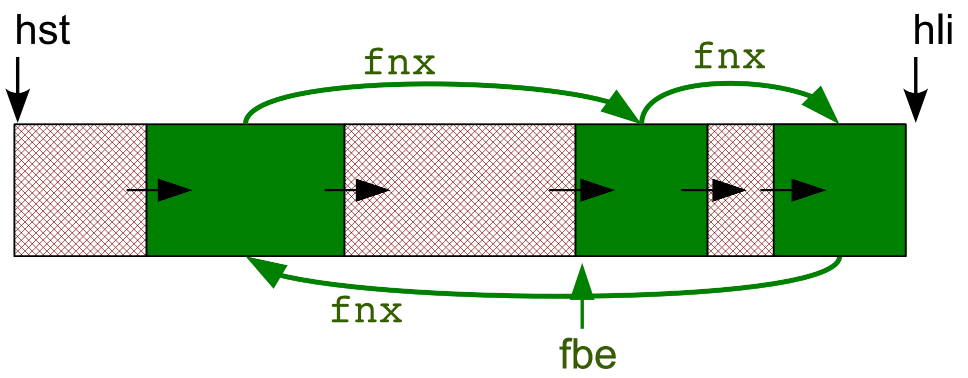


$$\mathsf{hlsc}(X, f_x; Y, f_y)[W] \triangleq \mathsf{emp} \wedge X = Y \wedge W = \epsilon \wedge 0 \le f_x + f_y \le 1$$
$$\vee\ (\exists Z, W', f \cdot \mathsf{chk}(X; Z) \star \mathsf{hlsc}(Z, f; Y, f_y)[W'] \wedge W = [X]\,.\,W'$$
$$\wedge\ f = X\,.\,\mathbf{isfree} \wedge 0 \le X\,.\,\mathbf{isfree} + f_y \le 1)$$

Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

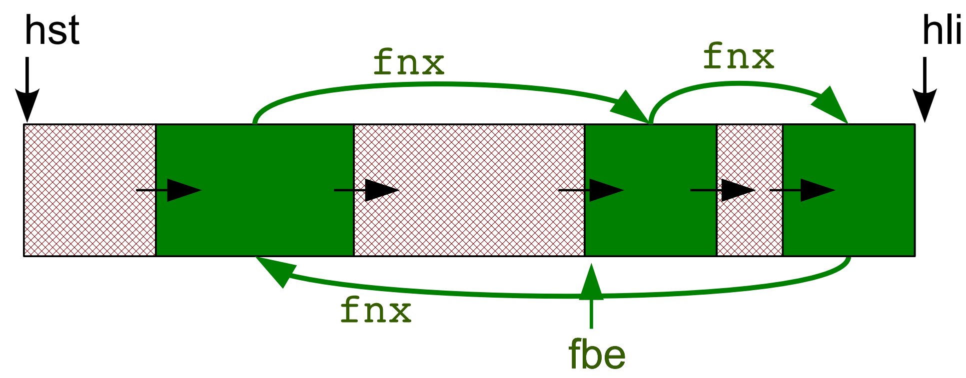
Free list



Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

Free list

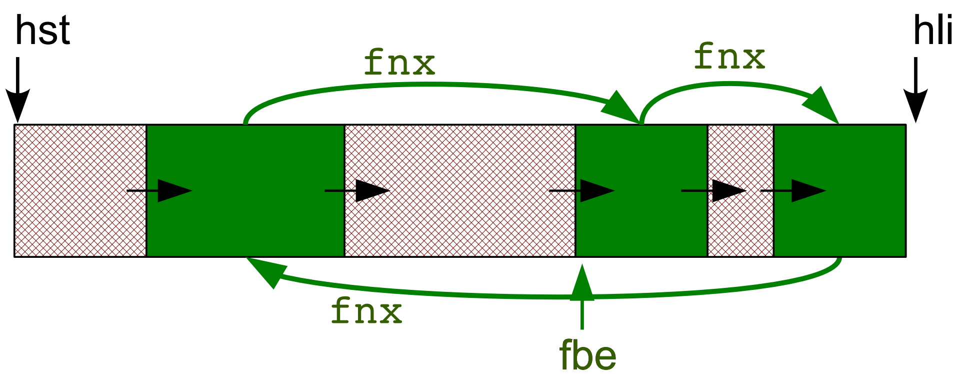


$$\boxed{\mathsf{fls}(X;Y)[W]} \triangleq \mathsf{emp} \wedge X = Y \wedge W = \epsilon$$
$$\vee\ \exists Z, W' \cdot \mathsf{fck}(X;Z) \star \boxed{\mathsf{fls}(Z;Y)[W']} \wedge W = [X]\,.\,W' \wedge X \neq Y$$

Separation logic with inductive predicates

- Atomic predicates (raw memory region)
- Inductively-defined predicates (disjoint memory blocks)
- Separating conjunction $\phi \star \psi$

Free list



$$\mathsf{fls}(X;Y)[W] \triangleq \mathsf{emp} \wedge X = Y \wedge W = \epsilon$$
$$\vee\ \exists Z, W' \cdot \mathsf{fck}(X;Z) \star \mathsf{fls}(Z;Y)[W'] \wedge W = [X] \,.\, W' \wedge X \neq Y$$

$$\mathsf{hls}(hst;hli)[W_H] \Supset \exists Z, W' \cdot \mathsf{fck}(\mathsf{fbe};Z) \star \mathsf{fls}(Z;\mathsf{fbe})[W'] \wedge W_F = [\mathsf{fbe}] \,.\, W'$$

combination symbol

Spatial part of SLMA

$$\Sigma_H ::= \mathsf{emp} \mid X \mapsto x \mid \mathsf{blk}(X;Y) \mid \mathsf{chd}(X;Y) \mid \mathsf{chk}(X;Y) \mid \Sigma_H \star \Sigma_H \mid \mathsf{hls}(X;Y)[W] \mid \mathsf{hlsc}(X,i;Y,j)[W]$$

$$\Sigma_F ::= \mathsf{emp} \mid \mathsf{fck}(X;Y) \mid \mathsf{fls}(X;Y)[W] \mid \mathsf{flso}(X,i;Y,j) \mid \Sigma_F \star \Sigma_F$$

Hierarchical conjunction of spatial formulas

$$\Sigma ::= \Sigma_H \Supset \Sigma_F$$

Spatial part of SLMA

$$\Sigma_H ::= \mathsf{emp} \mid X \mapsto x \mid \mathsf{blk}(X;Y) \mid \mathsf{chd}(X;Y) \mid \mathsf{chk}(X;Y) \mid \Sigma_H \star \Sigma_H \mid \mathsf{hls}(X;Y)[W] \mid \mathsf{hlsc}(X,i;Y,j)[W]$$

$$\Sigma_F ::= \mathsf{emp} \mid \mathsf{fck}(X;Y) \mid \mathsf{fls}(X;Y)[W] \mid \mathsf{flso}(X,i;Y,j) \mid \Sigma_F \star \Sigma_F$$

Hierarchical conjunction of spatial formulas

$$\Sigma ::= \Sigma_H \Supset \Sigma_F$$

By semantics:

Σ_H: sequence of addresses in the heap list

Σ_F: sequence of addresses in the free list

To specify overlapping of memory region, then $\Supset$ requires

$$\forall X \in W_F \Rightarrow X \in W_H$$

Separation Logic fragment: SLMA

Spatial part of SLMA

$$\Sigma_H ::= \mathsf{emp} \mid X \mapsto x \mid \mathsf{blk}(X;Y) \mid \mathsf{chd}(X;Y) \mid \mathsf{chk}(X;Y) \mid \Sigma_H \star \Sigma_H \mid \mathsf{hls}(X;Y)[W] \mid \mathsf{hlsc}(X,i;Y,j)[W]$$

$$\Sigma_F ::= \mathsf{emp} \mid \mathsf{fck}(X;Y) \mid \mathsf{fls}(X;Y)[W] \mid \mathsf{flso}(X,i;Y,j) \mid \Sigma_F \star \Sigma_F$$

Hierarchical conjunction of spatial formulas

$$\Sigma ::= \Sigma_H \Supset \Sigma_F$$

Pure formulas as location (sequence) and numerical constrains

$$\Pi ::= A \mid \Pi_\forall \mid \Pi_W \qquad \Pi_\forall ::= \forall X \in W \cdot A_G \Rightarrow A_U \mid \Pi_\forall \land \Pi_\forall$$

$$L ::= X \mid X\,.\,\mathbf{fnx} \qquad \Pi_W ::= W_H = w \land W_F = w$$

$$A ::= L - L \# t \mid \Delta \mid A \land A \qquad w ::= \epsilon \mid [x] \mid W \mid w\,.\,w$$

SLMA captures the complex invariants of DMA

First-fit: (choice of a free chunk of *req* size)

$$\mathsf{hls}(hst;\mathsf{hli})[W_H] \Supset \mathsf{fls}(\mathsf{fbe};Y_2)[W_1] \star \mathsf{fck}(Y_2;Y_3) \star \mathsf{fls}(Y_3;\mathsf{nil})[W_2]$$
$$\wedge\, Y_2\,.\,\mathbf{size} \geq req \wedge \forall X \in W_1 \cdot X\,.\,\mathbf{size} < req$$
$$\wedge\, W_F = W_1\,.\,[Y_2]\,.\,W_2$$

Best-fit:

$$\mathsf{hls}(hst;\mathsf{hli})[W_H] \Supset \mathsf{fls}(\mathsf{fbe};Y_2)[W_1] \star \mathsf{fck}(Y_2;Y_3) \star \mathsf{fls}(Y_3;\mathsf{nil})[W_2]$$
$$\wedge\, Y_2\,.\,\mathbf{size} \geq req \wedge \forall X \in W_1, W_2 \cdot X\,.\,\mathbf{size} \geq req \Rightarrow X\,.\,\mathbf{size} > Y_2\,.\,\mathbf{size}$$
$$\wedge\, W_F = W_1\,.\,[Y_2]\,.\,W_2$$

Satisfiability problem for SLMA is undecidable

- Decidable pure part of SLMA for integer constraints Π_N
- Undecidable array logic fragment Π_W

Entailment checking fro SLMA is undecidable

- Undecidable entire pure part of SLMA (sequence constrains)
- Undecidable spatial part (fragment of SL with inductive predicates and data constraints)

Abstract domain

1. Numerical domain *[Apron]* (polyhedra) - **arithmetic constraints** $\Pi_N \in \mathbb{N}^\sharp$

$$\mathcal{N}^\sharp = (\mathbb{N}^\sharp, \sqsubseteq^{\mathbb{N}}, \sqcup^{\mathbb{N}}, \sqcap^{\mathbb{N}}, \bot^{\mathbb{N}}, \top^{\mathbb{N}}), \quad \nabla^{\mathbb{N}}$$

Abstract domain

1. Numerical domain *[Apron]* (polyhedra) - **arithmetic constraints** $\Pi_N \in \mathbb{N}^\sharp$

$$\mathcal{N}^\sharp = (\mathbb{N}^\sharp, \sqsubseteq^{\mathbb{N}}, \sqcup^{\mathbb{N}}, \sqcap^{\mathbb{N}}, \bot^{\mathbb{N}}, \top^{\mathbb{N}}), \quad \bigtriangledown^{\mathbb{N}}$$

2. Data words domain *[Bouajjani et al'11]* - **sequence constrains** $\Pi_W \in \mathbb{W}^\sharp$

$$\mathcal{D}^\sharp = (\mathbb{W}^\sharp, \sqsubseteq^{\mathbb{W}}, \sqcup^{\mathbb{W}}, \sqcap^{\mathbb{W}}, \bot^{\mathbb{W}}, \top^{\mathbb{W}}), \quad \bigtriangledown^{\mathbb{W}}$$

Abstract domain

3. Shape abstract domain - **spatial part** Σ

$$\mathscr{G}^\sharp = (\mathbb{G}^\sharp, \sqsubseteq^{\mathbb{G}}, \sqcup^{\mathbb{G}}, \sqcap^{\mathbb{G}}, \bot^{\mathbb{G}}, \top^{\mathbb{G}})$$

$G \in \mathbb{G}^\sharp$: Representative of the Gaifman graph of Σ

Abstract domain

3. Shape abstract domain - **spatial part** Σ

$$\mathscr{G}^\sharp = (\mathbb{G}^\sharp, \sqsubseteq^{\mathbb{G}}, \sqcup^{\mathbb{G}}, \sqcap^{\mathbb{G}}, \bot^{\mathbb{G}}, \top^{\mathbb{G}})$$

4. Shape-value domain (cofibered product of $\mathscr{G}^\sharp, \mathscr{N}^\sharp, \mathscr{D}^\sharp$)

$$\mathbb{M}^\sharp \triangleq \mathbb{G}^\sharp \rightrightarrows (\mathbb{N}^\sharp \times \mathbb{W}^\sharp)$$

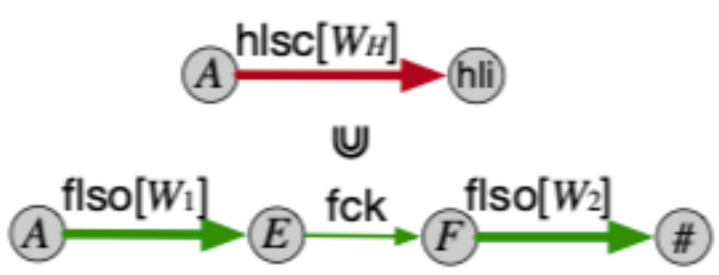
Abstract domain

3. Shape abstract domain - **spatial part** Σ

$$\mathscr{G}^\sharp = (\mathbb{G}^\sharp, \sqsubseteq^{\mathbb{G}}, \sqcup^{\mathbb{G}}, \sqcap^{\mathbb{G}}, \bot^{\mathbb{G}}, \top^{\mathbb{G}})$$

4. Shape-value domain (cofibered product of $\mathscr{G}^\sharp, \mathscr{N}^\sharp, \mathscr{D}^\sharp$)

$$\mathbb{M}^\sharp \triangleq \mathbb{G}^\sharp \rightrightarrows (\mathbb{N}^\sharp \times \mathbb{W}^\sharp)$$



$$\bigwedge \begin{array}{l} (\mathcal{F}_{\mathtt{size}}(E) \geq \mathbf{nunits}) \wedge \\ (\forall X \in W_1 \cdot \mathcal{F}_{\mathtt{size}}(X) < \mathbf{nunits}) \wedge \\ (W_F = W_1.[E].W_2) \end{array}$$

Abstract domain

3. Shape abstract domain - **spatial part** Σ

$$\mathscr{G}^\sharp = (\mathbb{G}^\sharp, \sqsubseteq^{\mathbb{G}}, \sqcup^{\mathbb{G}}, \sqcap^{\mathbb{G}}, \bot^{\mathbb{G}}, \top^{\mathbb{G}})$$

4. Shape-value domain (cofibered product of $\mathscr{G}^\sharp, \mathscr{N}^\sharp, \mathscr{D}^\sharp$)

$$\mathbb{M}^\sharp \triangleq \mathbb{G}^\sharp \rightrightarrows (\mathbb{N}^\sharp \times \mathbb{W}^\sharp)$$

5. Disjunctive abstraction $\mathscr{A}^\sharp$

$$\mathbb{A}^\sharp \triangleq \mathscr{P}(\mathbb{M}^\sharp), \quad \gamma_{\mathbb{A}}(A^\sharp) \triangleq \bigcup \{\gamma_{\mathbb{M}}(m^\sharp) \mid m^\sharp \in A^\sharp\}$$

Lattice operations: ordering and join

$$A^\sharp = (G, \Pi_N, \Pi_W) \in \mathbb{M}^\sharp, \quad B^\sharp = (G', \Pi'_N, \Pi'_W) \in \mathbb{M}^\sharp$$

$A^\sharp \sqsubseteq^{\mathbb{M}} B^\sharp$ **i.e.** $G \sim_\sigma G' \quad \wedge \quad (\Pi_N \sqsubseteq^{\mathbb{N}} \Pi'_N \wedge \Pi_W \sqsubseteq^{\mathbb{W}} \Pi'_W)$

$A^\sharp \sqcup^{\mathbb{M}} B^\sharp$ **i.e.** $G_\sigma \wedge \ (\Pi_N \sqcup^{\mathbb{N}} \Pi'_N) \wedge (\Pi_W \sqcup^{\mathbb{W}} \Pi'_W)$

Lattice operations: ordering and join

$$A^\sharp = (G, \Pi_N, \Pi_W) \in \mathbb{M}^\sharp, \quad B^\sharp = (G', \Pi'_N, \Pi'_W) \in \mathbb{M}^\sharp$$

$A^\sharp \sqsubseteq^{\mathbb{M}} B^\sharp$ **i.e.** $G \sim_\sigma G' \quad \wedge \quad (\Pi_N \sqsubseteq^{\mathbb{N}} \Pi'_N \wedge \Pi_W \sqsubseteq^{\mathbb{W}} \Pi'_W)$

$A^\sharp \sqcup^{\mathbb{M}} B^\sharp$ **i.e.** $G_\sigma \wedge (\Pi_N \sqcup^{\mathbb{N}} \Pi'_N) \wedge (\Pi_W \sqcup^{\mathbb{W}} \Pi'_W)$

Theorem: soundness of $\sqsubseteq^{\mathbb{M}}$, $\sqcup^{\mathbb{M}}$

If $A^\sharp \sqsubseteq^{\mathbb{M}} B^\sharp$, *then* $\gamma_{\mathbb{M}}(A^\sharp) \subseteq \gamma_{\mathbb{M}}(B^\sharp)$

For any $A^\sharp, B^\sharp \in \mathbb{M}^\sharp$, $\gamma_{\mathbb{M}}(A^\sharp) \cup \gamma_{\mathbb{M}}(B^\sharp) \subseteq \gamma_{\mathbb{M}}(A^\sharp \sqcup^{\mathbb{M}} B^\sharp)$

Logic-based abstract domain

Lattice operations **folding:** eliminate nodes not labeled by program variables by applying lemmas:

- Predicate definition $P(\ldots) \triangleq \vee_i \phi_i$ gives

$$\phi_i \Rightarrow P(\ldots)$$

- List segment composition $P \in \{\textbf{hls, hlsc, fls, flso}\}$:

$$P(X;Y)[W_1] \star P(Y;Z)[W_2] \wedge W = W_1 \,.\, W_2 \Rightarrow P(X;Z)[W]$$

- blk lemmas, e.g. :

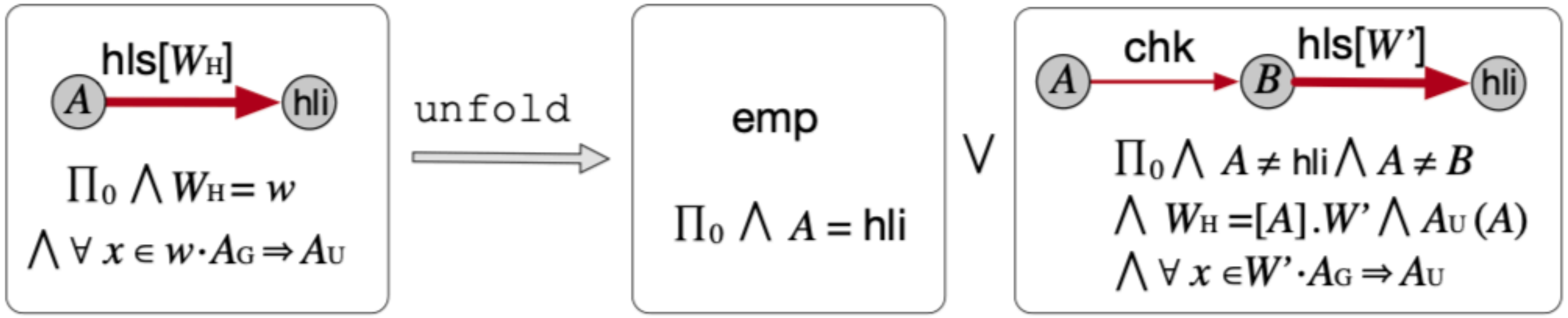
$$\textbf{blk}(X;Y) \star \textbf{blk}(Y;Z) \wedge X \leq Y \leq Z \Rightarrow \textbf{blk}(X;Z)$$

Lattice operation materialisation: unfolding summary

$$\mathbf{Unfold}^\sharp : A^\sharp \rightarrow \mathscr{P}_{fin}(A^\sharp) \qquad (A^\sharp \in \mathbb{M}^\sharp)$$

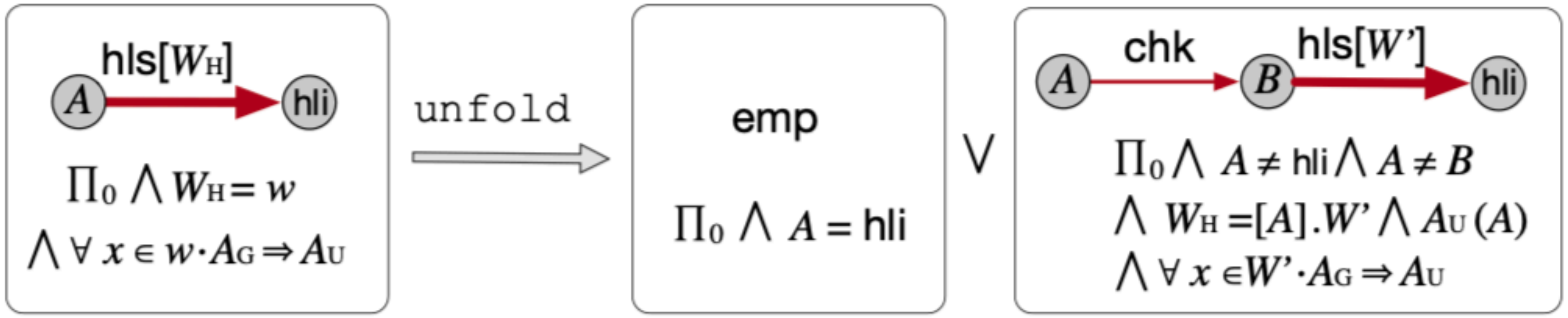
Lattice operation materialisation: unfolding summary

$$\textbf{Unfold}^\sharp : A^\sharp \rightarrow \mathscr{P}_{fin}(A^\sharp) \qquad (A^\sharp \in \mathbb{M}^\sharp)$$



Lattice operation materialisation: unfolding summary

$$\textbf{Unfold}^\sharp : A^\sharp \rightarrow \mathscr{P}_{fin}(A^\sharp) \qquad (A^\sharp \in \mathbb{M}^\sharp)$$



Theorem: soundness of $\textbf{Unfold}^\sharp$

If $\textbf{Unfold}^\sharp$ *transforms* $A^\sharp$ *into a finite number of fisjuncts* $A_1^\sharp \vee A_2^\sharp \vee \ldots A_n^\sharp$, *then* $\gamma_{\mathbb{M}} \subseteq \bigcup_{0 \leq i \leq n} \gamma_{\mathbb{M}}(A_i^\sharp)$

Fields and Hierarchical Unfolding

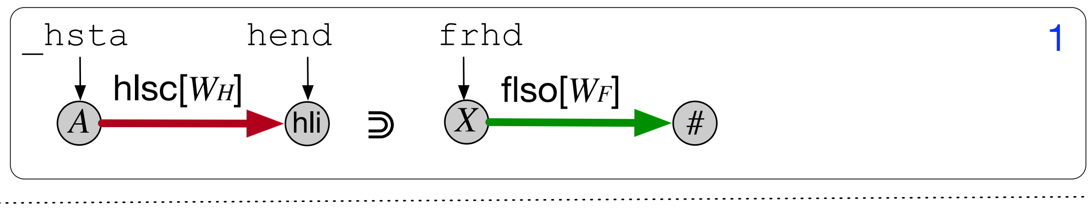
Let fix **blk** $\prec_P$ **chd** $\prec_P$ **chk** $\prec_P$ **fck** $\prec_P$ **hls, hlsc, fls, flso** $(Q \preceq_P P \triangleq (Q \prec_P P) \vee (Q = P))$

Given an atom $P(X; \ldots)$ and a statement ***s*** accessing X,

then **apply rules of (unfold)** P to obtain atom $Q(X; \ldots)$ s.t. $Q \preceq_P P$ and:

- if ***s*** reads $X.f$, then $Q \preceq_P$ **fck** ,
- if ***s*** assigns $X.isfree$ or $X.fnx$, then $Q \preceq_P$ **chk** ,
- if ***s*** mutates X using pointer arithmetic or assigns $X.size$, then $Q \preceq_P$ **chd** .

Hierarchical folding and unfolding

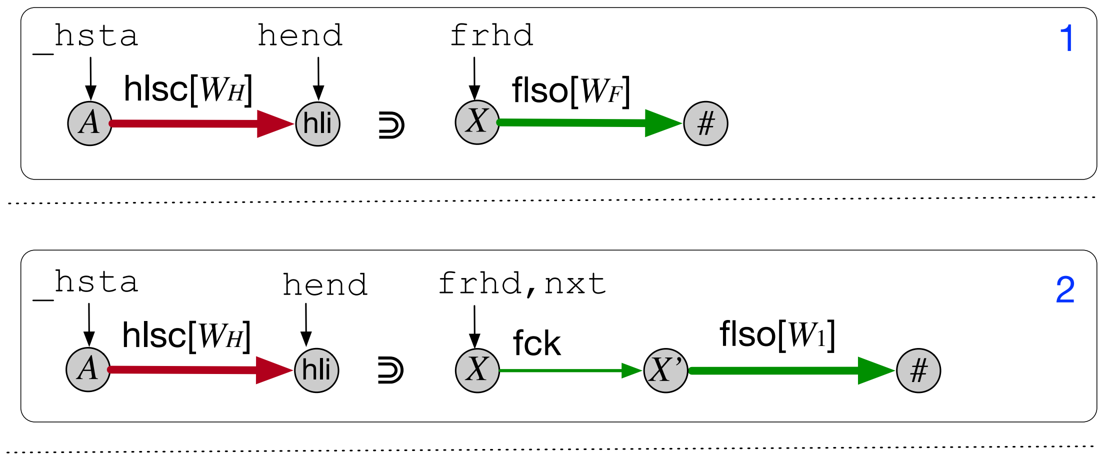



```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
       prv = nxt, nxt = nxt->fnx) {
    if (nxt->size >= nunits) {
      if (nxt->size > nunits) {
        nxt->size -= nunits;
        nxt += nxt->size;
        nxt->size = nunits;
      } else {
        if (prv == NULL)
          frhd = nxt->fnx;
        else
          ...
```

before loop

Logic-based abstract domain

Hierarchical folding and unfolding



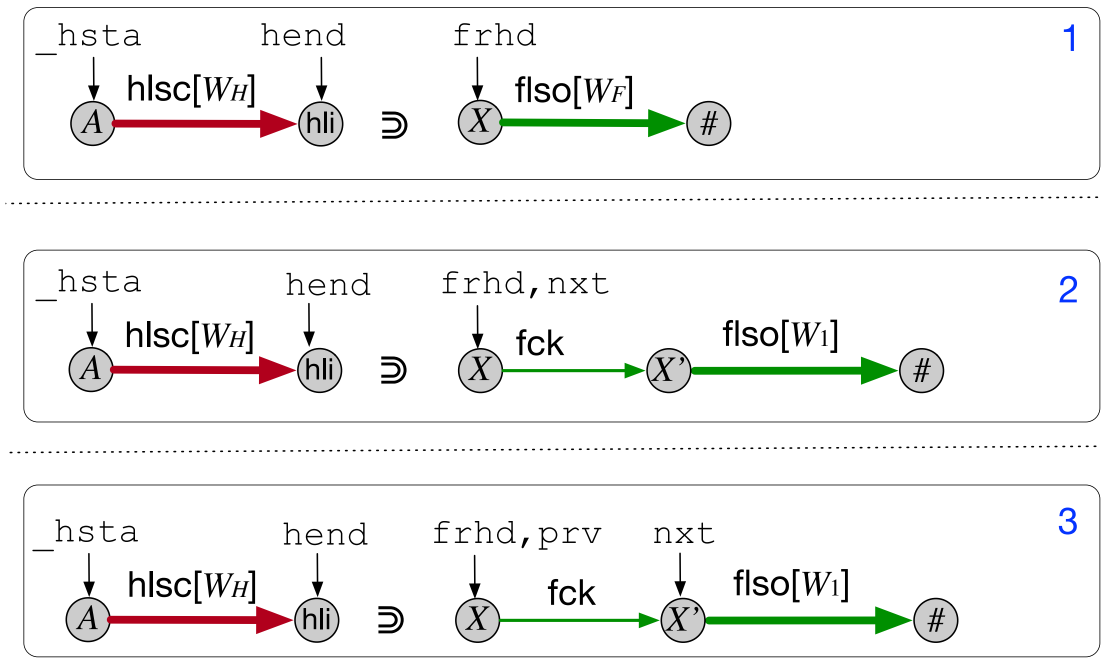

```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
      prv = nxt, nxt = nxt->fnx) {
    if (nxt->size >= nunits) {
      if (nxt->size > nunits) {
        nxt->size -= nunits;
        nxt += nxt->size;
        nxt->size = nunits;
      } else {
        if (prv == NULL)
          frhd = nxt->fnx;
        else

          ...
```

Unfold free list summary

Logic-based abstract domain

Hierarchical folding and unfolding



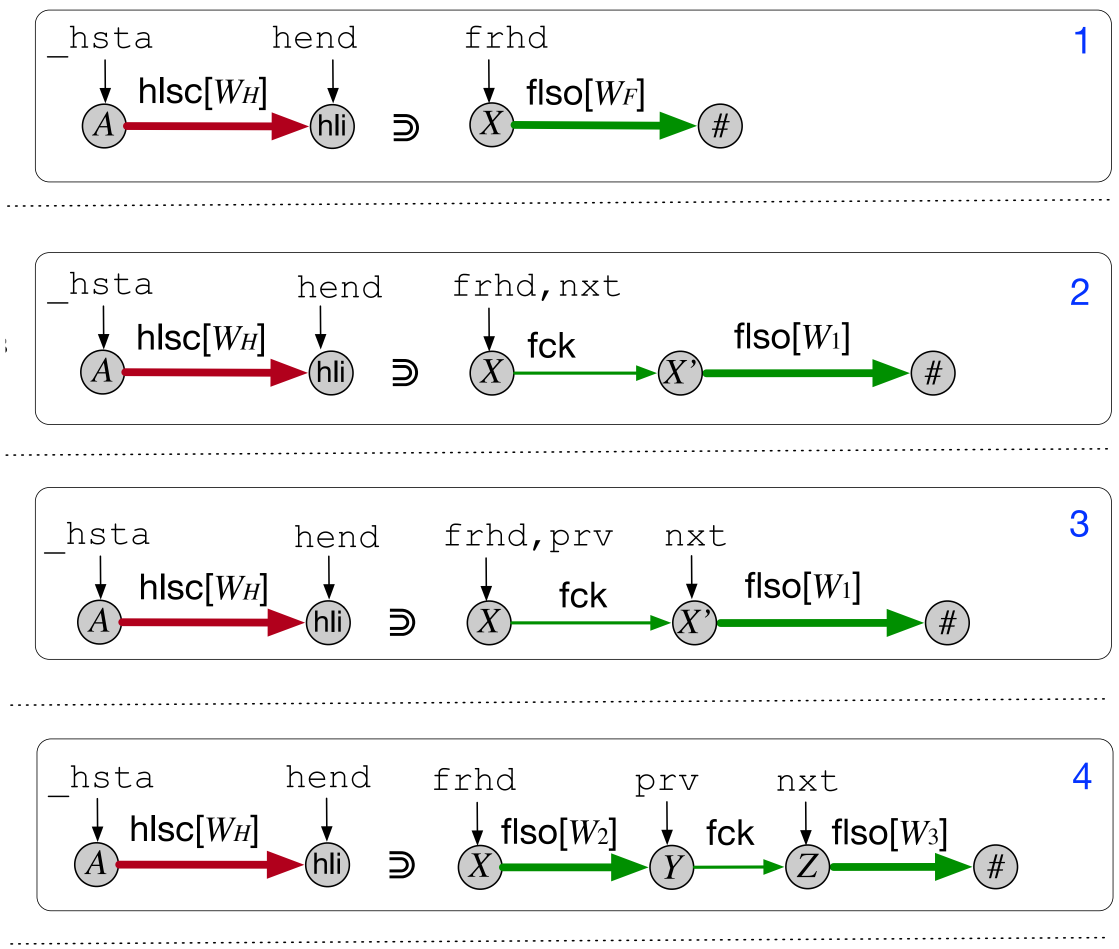

```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
       prv = nxt, nxt = nxt->fnx) {
   if (nxt->size >= nunits) {
    if (nxt->size > nunits) {
     nxt->size -= nunits;
     nxt += nxt->size;
     nxt->size = nunits;
    } else {
     if (prv == NULL)
      frhd = nxt->fnx;
     else

      ...
```

Unfold free list summary

Logic-based abstract domain

Hierarchical folding and unfolding



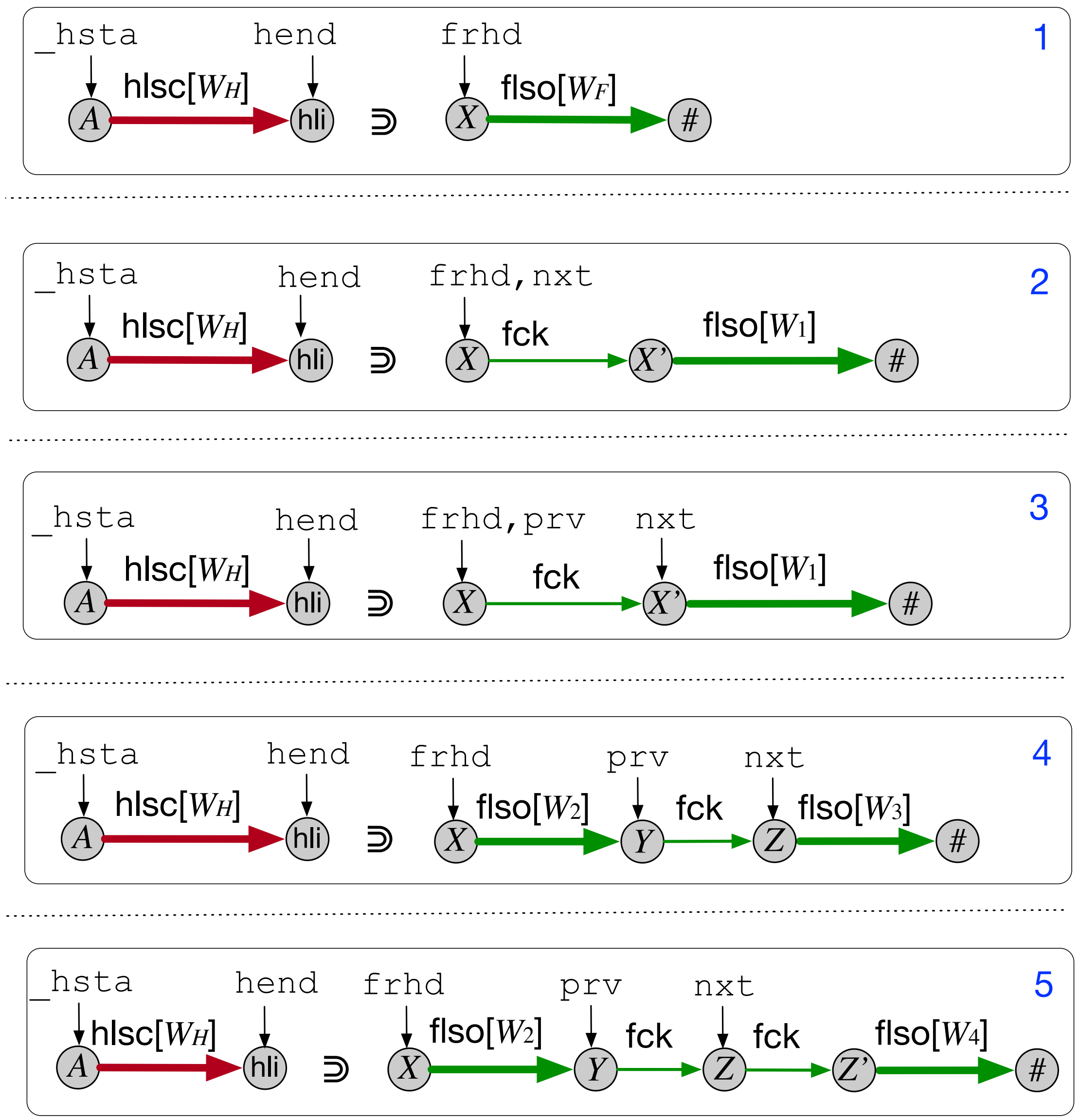

```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
        prv = nxt, nxt = nxt->fnx) {
    if (nxt->size >= nunits) {
      if (nxt->size > nunits) {
        nxt->size -= nunits;
        nxt += nxt->size;
        nxt->size = nunits;
      } else {
        if (prv == NULL)
          frhd = nxt->fnx;
        else

          ...
```

i-th iteration

Logic-based abstract domain

Hierarchical folding and unfolding

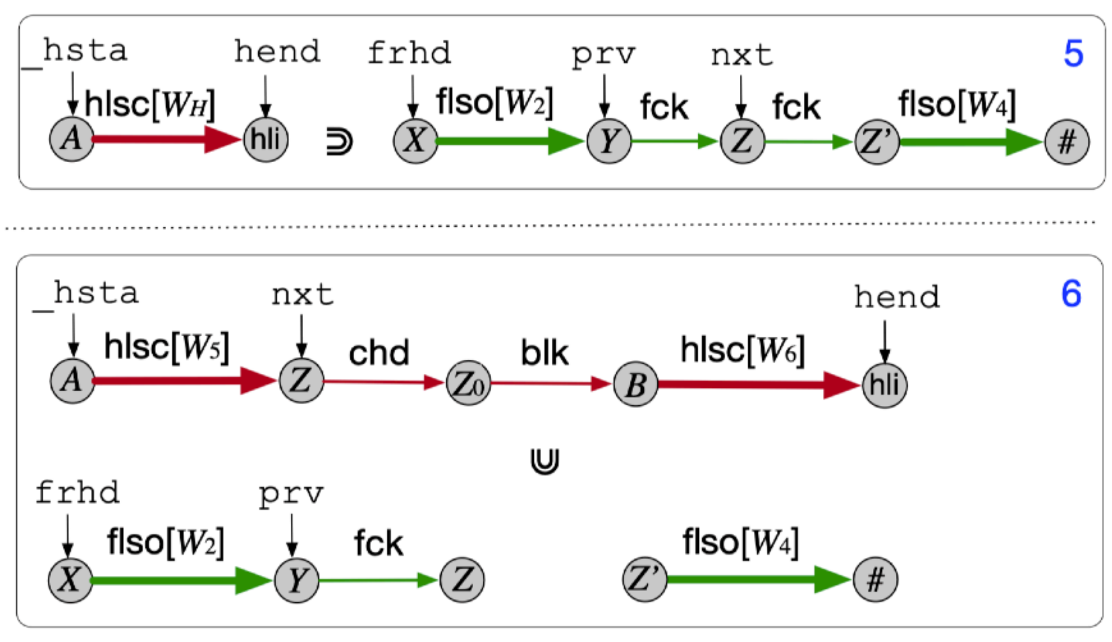



```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
      prv = nxt, nxt = nxt->fnx) {
    if (nxt->size >= nunits) {
      if (nxt->size > nunits) {
        nxt->size -= nunits;
        nxt += nxt->size;
        nxt->size = nunits;
      } else {
        if (prv == NULL)
          frhd = nxt->fnx;
        else
          ...
```

read size field

Logic-based abstract domain

Hierarchical folding and unfolding



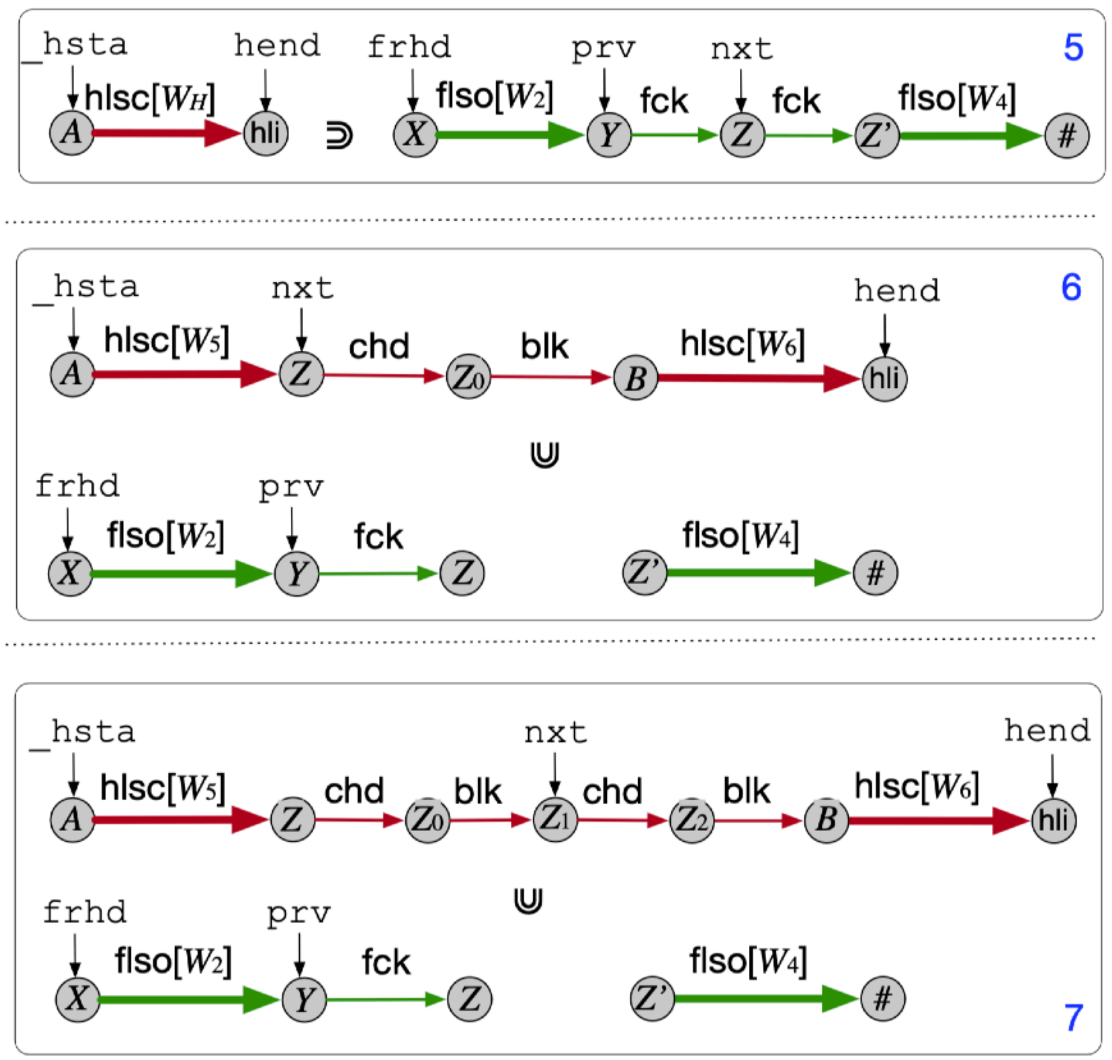

```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
      prv = nxt, nxt = nxt->fnx) {
    if (nxt->size >= nunits) {
      if (nxt->size > nunits) {
        nxt->size -= nunits;
        nxt += nxt->size;
        nxt->size = nunits;
      } else {
        if (prv == NULL)
          frhd = nxt->fnx;
        else

          ...
```

write size field

Logic-based abstract domain

Hierarchical folding and unfolding

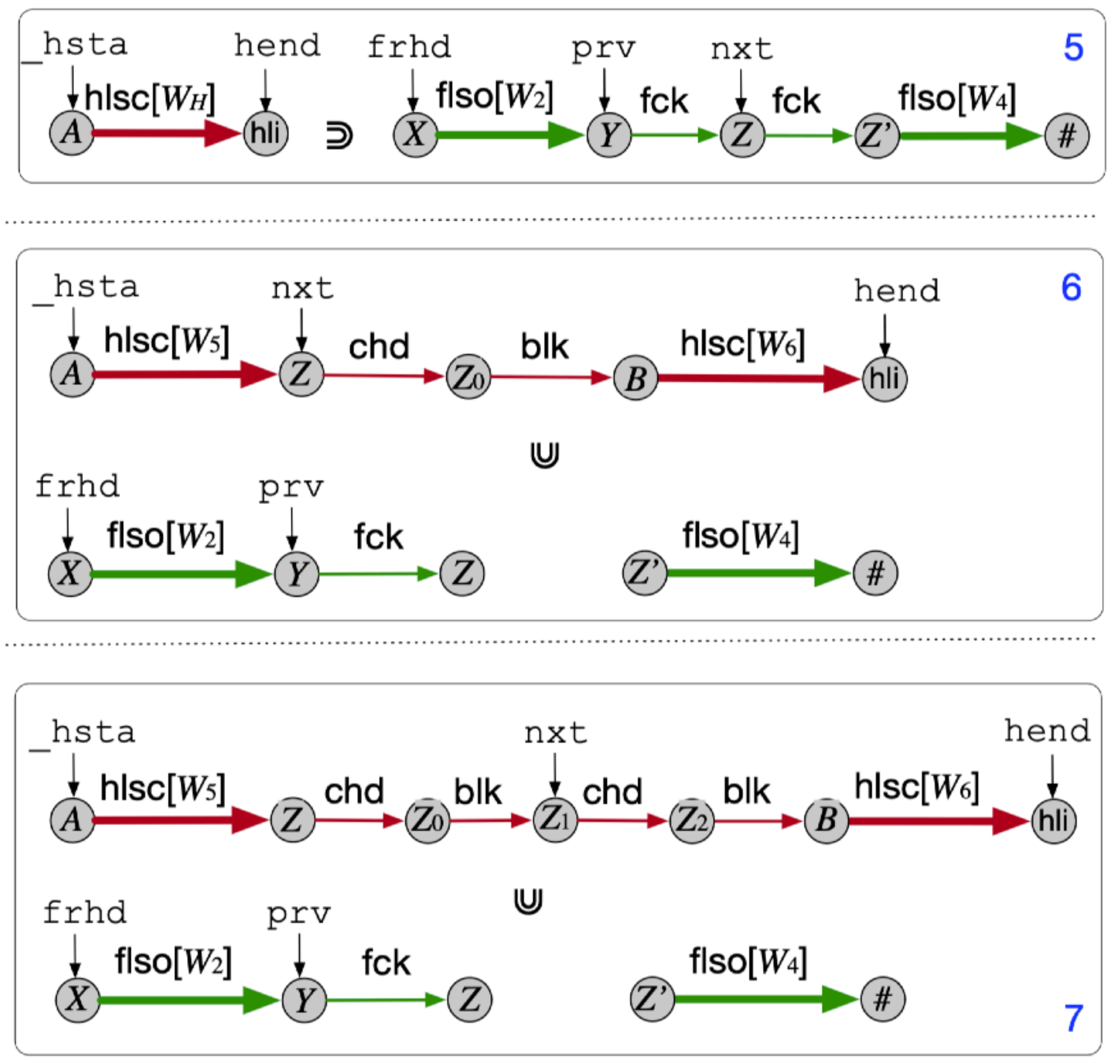



```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
      prv = nxt, nxt = nxt->fnx) {
   if (nxt->size >= nunits) {
    if (nxt->size > nunits) {
     nxt->size -= nunits;
     nxt += nxt->size;
     nxt->size = nunits;
    } else {
     if (prv == NULL)
      frhd = nxt->fnx;
     else

     ...
```

write size field

Hierarchical folding and unfolding




```
void* malloc(size_t nbytes) {
  HDR *nxt, *prv;
  size_t nunits =
    (nbytes+sizeof(HDR)-1)/sizeof(HDR) + 1;
  for (prv = NULL, nxt = frhd; nxt;
       prv = nxt, nxt = nxt->fnx) {
    if (nxt->size >= nunits) {
      if (nxt->size > nunits) {
        nxt->size -= nunits;
        nxt += nxt->size;
        nxt->size = nunits;
      } else {
        if (prv == NULL)
          frhd = nxt->fnx;
        else

         ...
```

write size field

Experimental

Static analyser MMEN

- Frama-c plugin (Ocaml 38k LOC)
- Pointer arithmetics
- Low level system calls, e.g., sbrk
- Verifies a set of DMAs

LOPSTR'16

Future work

- Modelling and verification for **concurrent** memory algorithms (B, CIVL, etc)
- Other components of OS kernel
- Extension of the logic
- Scalable analysis tool

Publications

2018	**Tool paper: Static Analyser for Dynamic Memory Allocators** (submit soon) **Journal paper: Hierarchical Shape Abstraction for Analysis of Dynamic Memory Allocators**
2017	**Formal Modelling of List Based Dynamic Memory Allocators**. **Bin Fang**, Mihaela Sighireanu, Geguang Pu. Journal of SCIENCE CHINA Information Sciences, 2017.
2017	**A Refinement Hierarchy for Free List Memory Allocators**. **Bin Fang**, Mihaela Sighireanu. ACM SIGPLAN International Symposium on Memory Management (ISMM) 2017.
2016	**Hierarchical Shape Abstraction of Free-List Memory Allocators**. **Bin Fang**, Mihaela Sighireanu. 26th International Symposium on Logic-Based Program Synthesis and Transformation LOPSTR 2016.
2015	**Formal Development of a Real-Time Operating System Memory Manager**. Wen Su, Jean-Raymond Abrial, Geguang Pu, **Bin Fang**. 20th International Conference on Engineering of Complex Computer Systems ICECCS 2015.
2014	**Automated Coverage-Driven Test Data Generation Using Dynamic Symbolic Execution**. Ting Su, Siyuan Jiang, Geguang Pu, **Bin Fang**, Jifeng He, Jun Yan, Jianjun Zhao. Eighth International Conference on Software Security and Reliability, SERE 2014.
2014	**Runtime Verification by Convergent Formula Progression**. Yan Shen, Jianwen Li, Zheng Wang, **Bin Fang**, Geguang Pu and Wangwei Liu. 21st Asia-Pacific Software Engineering Conference APSEC 2014.

Thank you! Questions ?