Modelling, Specification and Formal Analysis of Complex Software Systems

Precise Static Analysis of Programs with Dynamic Memory

Mihaela Sighireanu

IRIF, University Paris Diderot & CNRS

VTSA 2015

Outline

Stack	$\mathtt{Stacks} \triangleq [(\mathbf{CP} \times \mathbf{P} \times (\mathbf{DV} \nrightarrow \mathbb{D} \ \cup \ \mathbf{RV} \nrightarrow \mathbb{L}))^*] \ni \mathtt{S}$
Memory	$\mathtt{Mem} \triangleq \mathtt{Stacks} \times \mathtt{Heaps} \ni \mathtt{m}$
Configurations	$\mathtt{Config} \triangleq \mathbf{CP} \times (\mathtt{Mem} \cup \{\mathtt{merr}\}) \ni \mathtt{C}$

$$\frac{\forall v_i \in \overrightarrow{vin} \,.(\mathtt{S},\mathtt{H}) \vdash v_i \rightsquigarrow c_i \neq \mathtt{merr}}{(\ell,(\mathtt{S},\mathtt{H})) \vdash v{=}P(\overrightarrow{vin},\overrightarrow{vout}) \rightsquigarrow (\mathit{start}_P,(\mathit{push}(\mathtt{S},\ell+1,P,v,\overrightarrow{vout},\overrightarrow{c_i},\mathtt{lv}_P),\mathtt{H}))}$$

$$\frac{\mathit{top}(\mathtt{S}) = (\ell, P, v, \ldots) \quad (\mathtt{S},\mathtt{H})(v') = c}{(\ell',(\mathtt{S},\mathtt{H})) \vdash \mathtt{return}\ v' \rightsquigarrow (\ell,(\mathit{pop}(\mathtt{S}),\mathtt{H})[v \leftarrow c])}$$

Another source of infinity is the unbounded stack that usually stores locations in the heap.

Aim

Compute an abstraction of the relation between the input and output configurations of a procedure, *i.e.* the procedure summary or contract.

Context sensitive: the summary depends on an abstraction of the calling stack

> *"If* p *is called before* q*, it returns 0, otherwise 1."*
>
> ⟶ insight on the full program behaviour, expressive
>
> ⟶ analysis done for each call point

Context insensitive: the summary is independent of the calling stack

> *"If* p *is called it returns 0 or 1."*
>
> ⟶ insight on the procedure behaviour, but less precise
>
> ⟶ analysis done independently of callers

Main steps:

Case 1: Compute summary information for each procedure
... at each calling point with “equivalent stack” runs

Case 2: Use summary information at procedure calls...
... if the abstraction of reaching stack fits the already computed ones

Main steps:

Case 1: Compute summary information for each procedure
… at each calling point with "equivalent stack" runs

Case 2: Use summary information at procedure calls…
… if the abstraction of reaching stack fits the already computed ones

Classic approaches for summary computation:

- Functional approach: [Sharir&Pnueli,81],[Knoop&Steffen,92]
  Summary is a function mapping abstract input to abstract output
- Relational approach: [Cousot&Cousot,77]
  Summary is a relation between input and output
- Call string approach: [Sharir&Pnueli,81], [Khedker&Karkare,08]
  Maps string abstractions of the call stack to abstract configs.

Functional Context-Sensitive

Aim

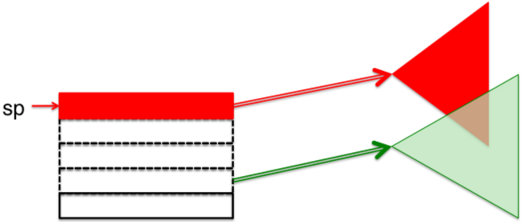
Compute a function $\mathit{summary}_P : \mathbf{CP} \not\to (\mathcal{A}_{\mathbb{H}} \to \mathcal{A}_{mH})$ mapping each control point of the procedure $q \in \mathbf{CP}$ to a function which associates every (G_0, W_0) abstract heap reachable at start_P to the abstract heap (G_q, W_q) reachable at q.

```
   int length(list* l) {
1:   int len = 0;
2:   if (l == NULL)
3:     len=0;
4:   else {
5:     len=1+length(l->next);
6:   }
7:   return len;
8: }
```

q	(G_0, W_0)	(G_q, W_q)
2	$l = \boxtimes$	$l = \boxtimes$
	$\mathtt{ls}^+(l, \boxtimes)$	$\mathtt{ls}^+(l, \boxtimes)$
...	...	...
8	$l = \boxtimes$	$\$ret = 0 \wedge l = \boxtimes$
	$\mathtt{ls}^+(l, \boxtimes)$	$\$ret \geqslant 1 \wedge \mathtt{ls}^+(l, \boxtimes)$

Problem

The local heap of a procedure may be accessed from the stack bypassing the actual parameters.

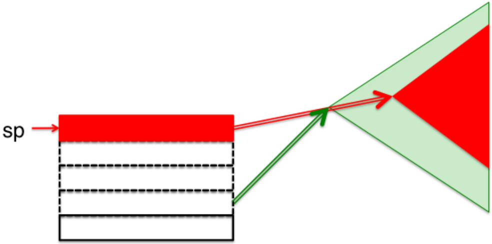


Bad Consequence

Context sensitive analyses shall track also these interferences!

Observation

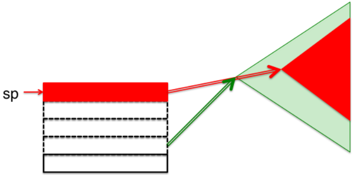
In a large class of programs with procedure calls, the local heap is reachable from the stack by passing through the actual parameters.

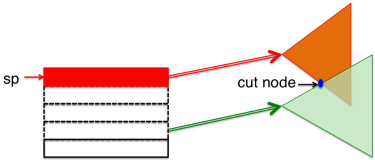


Consequence

For this class, the computation of summaries is compositional.

[Rinetzky *et al*,05]





Definition

A call is cut point free if all local heap cut nodes are reachable from the stack through the procedure parameters. A cut point free program has only cut point free procedure calls.

Let V be the set of formal parameters and local variables.

Definition

A concrete inter-procedural configurations is a pair of heap configurations (H^0, H_q) where:

- H^0 is the local heap at $\mathtt{start}_P$ over a new vocabulary V^0
  $\longrightarrow$ similar to `old` notation in JML
- H is the heap at the control point q of the procedure over $V \cup \{\$ret\}$

Definition

A concrete procedure summary is the set $\{(H^0, H_{\mathtt{end}_P})\}$.

Outline

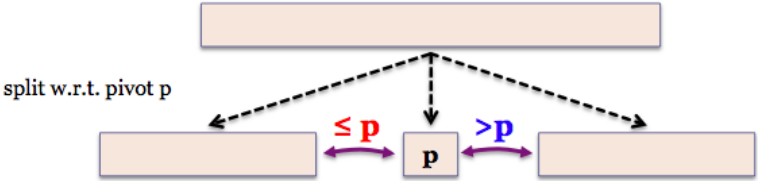
Programs with Lists and Data

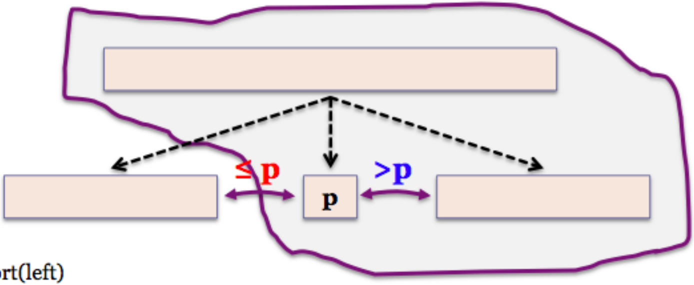
— Inter-procedural Analysis —

joint work with A. Bouajjani, C. Drăgoi, C. Enea

PLDI'11

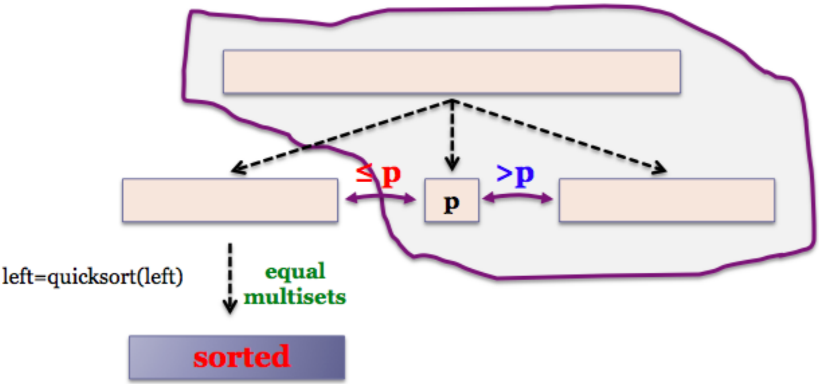
Running Example: Quicksort on Lists (with Copy)



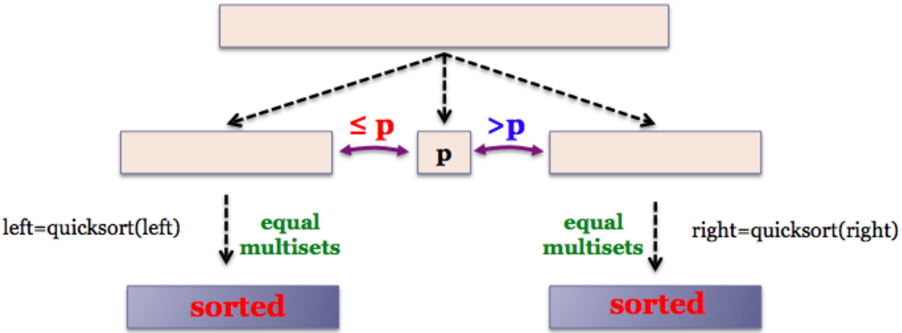


left=quicksort(left)

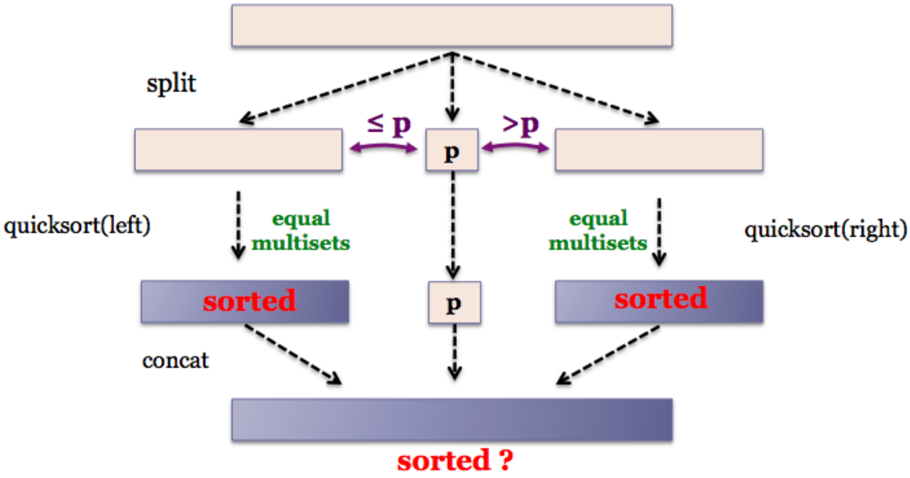
Running Example: Quicksort on Lists (with Copy)

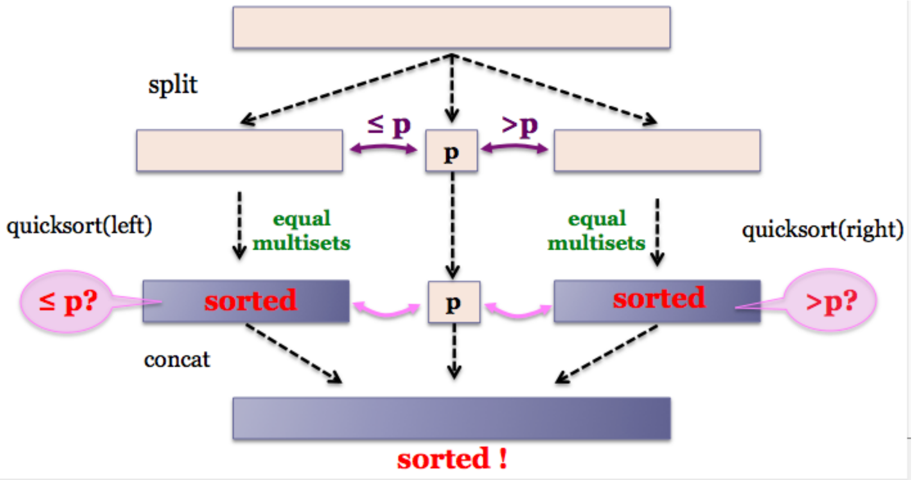


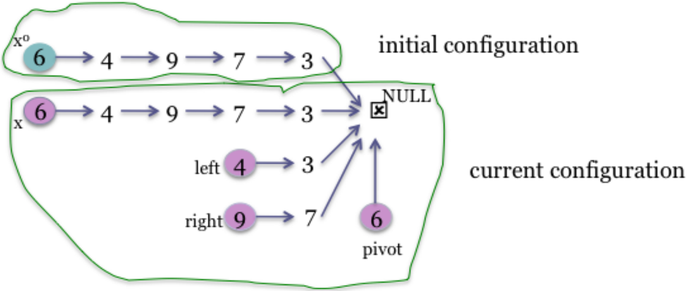
Running Example: Quicksort on Lists (with Copy)



Running Example: Quicksort on Lists (with Copy)

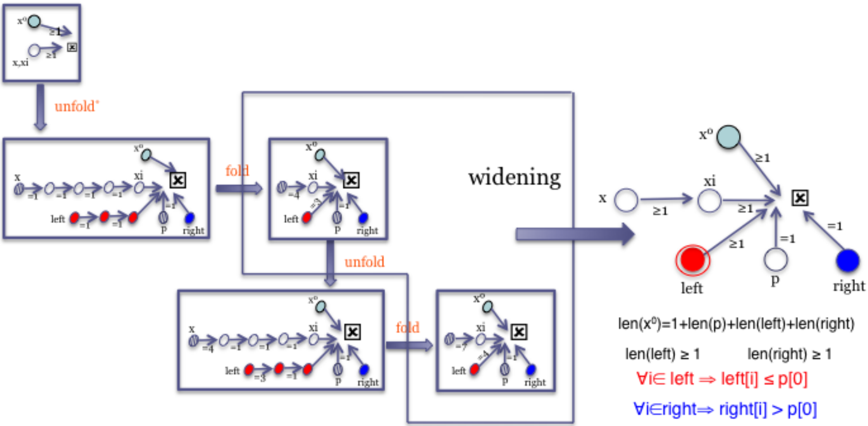



split
≤ p
p
>p
quicksort(left)
equal
multisets
equal
multisets
quicksort(right)
≤ p?
sorted
p
sorted
>p?
concat
sorted !



$$\alpha\big((H^0, H)\big) \triangleq (G^0 * G,\ W^0 \wedge W)$$

Analysing Quicksort: $\mathcal{A}^{\mathbb{U}}$ Domain



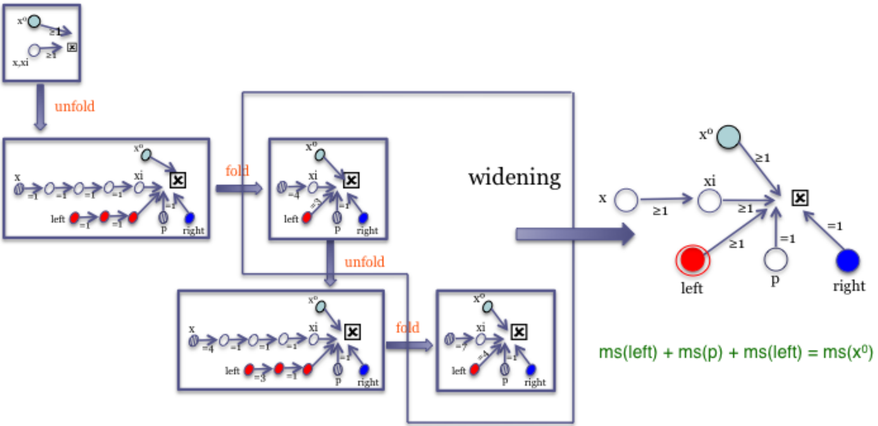
len(x^0)=1+len(p)+len(left)+len(right)

len(left) ≥ 1 len(right) ≥ 1

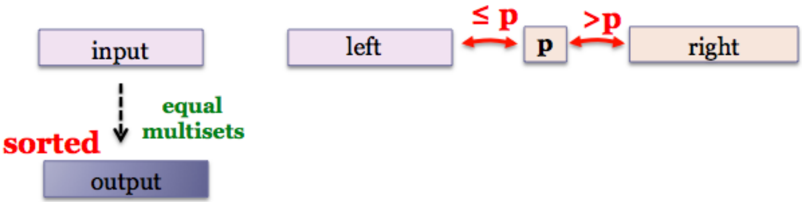
$\forall i \in$ left $\Rightarrow$ left[i] $\leq$ p[0]

$\forall i \in$right$\Rightarrow$ right[i] > p[0]

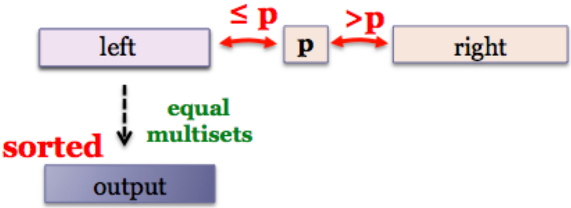
Analysing Quicksort: $\mathcal{A}^{\mathrm{M}}$ Domain



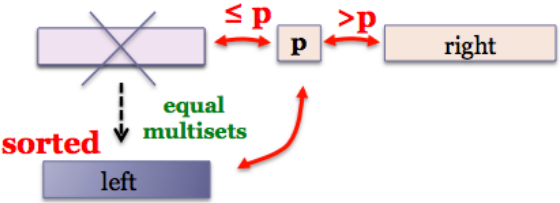
left = quicksort(left)

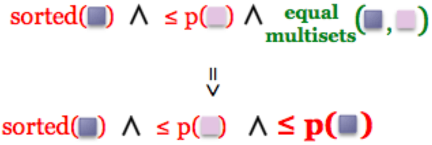


left = quicksort(left)

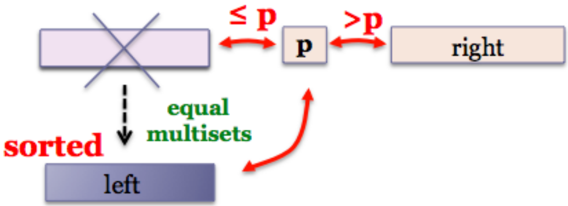


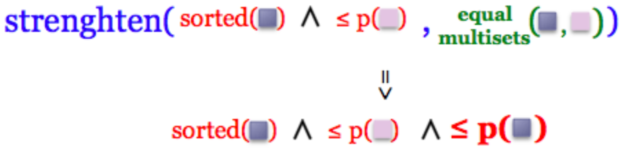
left = quicksort(left)

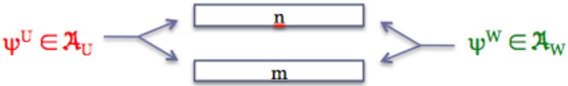


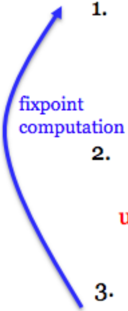


left = quicksort(left)

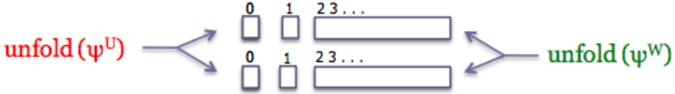






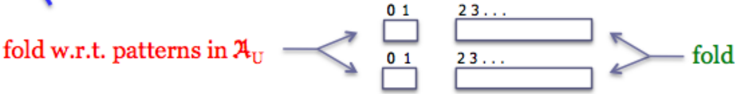


1. unfold a bounded length prefix of n and m

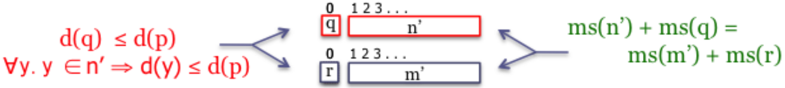


2. intersect the constraints on the prefixes

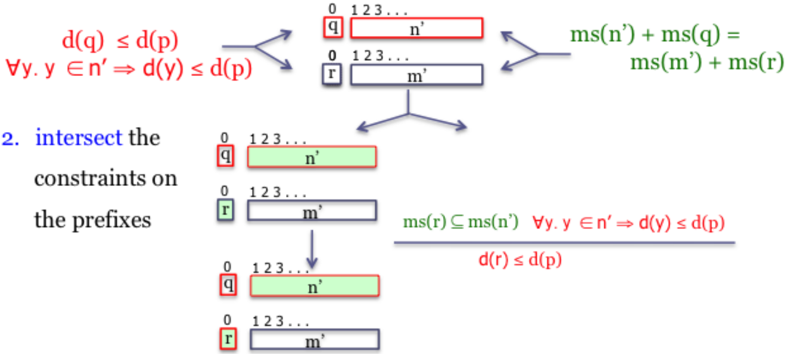
3. fold the prefixes and collect the information using universal formulas



1. unfold a bounded length prefix of n and m

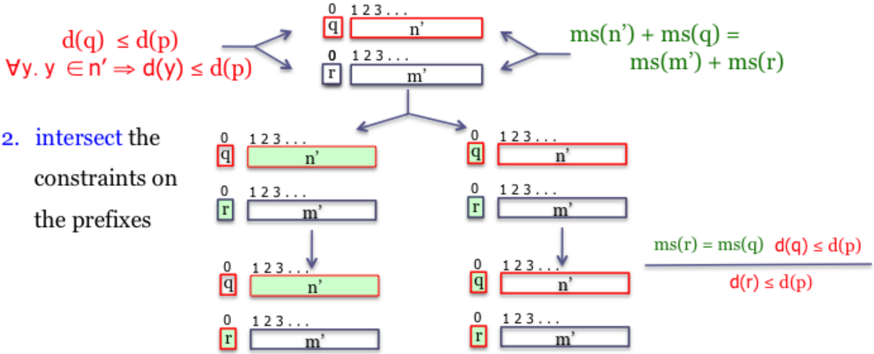


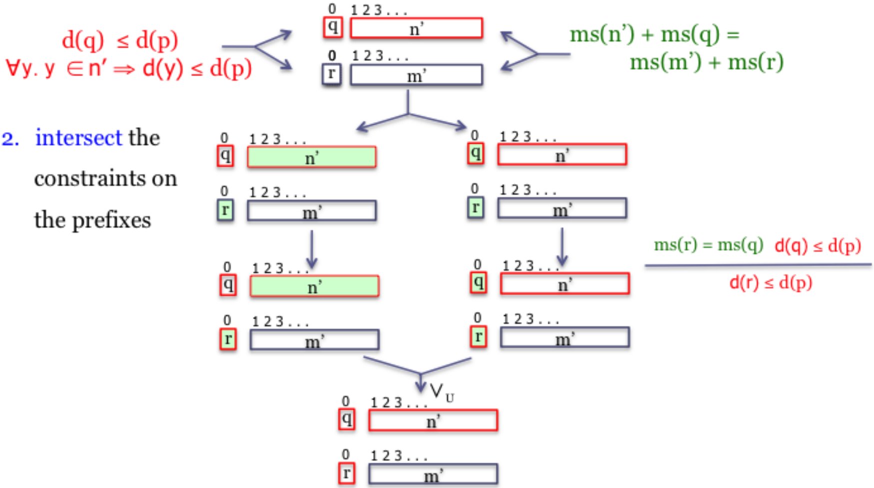
2. intersect the constraints on the prefixes



Strengthen Procedure: Example

2. intersect the constraints on the prefixes





2. intersect the constraints on the prefixes

Experimental Results

class	fun	nesting (loop,rec)	$\mathcal{A}_{\mathbb{M}}$ t (s)	$\mathcal{A}_{\mathbb{U}}$ $\mathcal{P}$	$\mathcal{A}_{\mathbb{U}}$ t (s)	Examples of summaries synthesized
sll	*create*	$(0,-)$	<1	$P_=, P_1$	<1	
	addfst	–	<1	$P_=$	<1	
	addlst	$(0,1)$	<1	$P_=$	<1	$\rho^\#_{\mathbb{U}}(create(\&x,\ell)):\ \mathtt{hd}(x)=0 \land \mathtt{len}(x)=\ell \land \forall y \in \mathtt{tl}(x) \Rightarrow x[y]=0$
	delfst	–	<1	$P_=$	<1	
	dellst	$(0,1)$	<1	$P_=$	<1	
map	*init*(v)	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^\#_{\mathbb{U}}(init(v,x)):\ \mathtt{len}(x^0)=\mathtt{len}(x) \land \mathtt{hd}(x)=v \land \forall y \in \mathtt{tl}(x).\ x[y]=v$
	initSeq	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^\#_{\mathbb{U}}(add(v,x)):\ \mathtt{len}(x^0)=\mathtt{len}(x) \land \mathtt{hd}(x)=\mathtt{hd}(x^0)+v \land$
	add(v)	$(0,1)$	<1	$P_=$	<1	$\forall y_1 \in \mathtt{tl}(x), y_2 \in \mathtt{tl}(x^0).\ y_1=y_2 \Rightarrow x[y_1]=x^0[y_2]+v$
map2	*add*(v)	$(0,1)$	<1	$P_=$	<1	$\rho^\#_{\mathbb{U}}(add(v,x,z)):\ \mathtt{len}(x^0)=\mathtt{len}(x) \land \mathtt{len}(z^0)=\mathtt{len}(z) \land eq(x,x^0) \land$
	copy	$(0,1)$	<1	$P_=$	<1	$\forall y_1 \in \mathtt{tl}(x), y_2 \in \mathtt{tl}(z).\ y_1=y_2 \Rightarrow x[y_1]+v=z[y_2]$
fold	*del*Pred	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^\#_{\mathbb{M}}(split(v,x,\&l,\&u)):\ \mathtt{ms}(x)=\mathtt{ms}(x^0)=\mathtt{ms}(l)\cup\mathtt{ms}(u)$
	max	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^\#_{\mathbb{U}}(split(v,x,\&l,\&u)):\ equal(x,x^0) \land \mathtt{len}(x)=\mathtt{len}(l)+\mathtt{len}(u) \land$
	clone	$(0,1)$	<1	$P_=$	<1	$l[0] \le v \land \forall y \in \mathtt{tl}(l) \Rightarrow l[y] \le v \land$
	split	$(0,1)$	<1	$P_=, P_1$	<1	$u[0] > v \land \forall y \in \mathtt{tl}(u) \Rightarrow u[y] > v$
fold2	*equal*	$(0,1)$	<1	$P_=$	<1	$\rho^\#_{\mathbb{M}}(merge(x,z,\&r)):\ \mathtt{ms}(x)\cup\mathtt{ms}(z)=\mathtt{ms}(r) \land \mathtt{ms}(x^0)=\mathtt{ms}(x) \land \ldots$
	concat	$(0,1)$	<1	$P_=, P_1, P_2$	<3	$\rho^\#_{\mathbb{U}}(merge(x,z,\&r)):\ equal(x,x^0) \land equal(z,z^0) \land sorted(x^0) \land sorted(z^0) \land$
	merge	$(0,1)$	<1	$P_=, P_1, P_2$	<3	$sorted(r) \land \mathtt{len}(x)+\mathtt{len}(z)=\mathtt{len}(r)$
sort	*bubble*	$(1,-)$	<1	$P_=, P_1, P_2$	<3	
	insert	$(1,-)$	<1	$P_=, P_1, P_2$	<3	$\rho^\#_{\mathbb{M}}(quicksort(x)):\ \mathtt{ms}(x)=\mathtt{ms}(x^0)=\mathtt{ms}(res)$
	quick	$(-,2)$	<2	$P_=, P_1, P_2$	<4	$\rho^\#_{\mathbb{U}}(quicksort(x)):\ equal(x,x^0) \land sorted(res)$
	merge	$(-,2)$	<2	$P_=, P_1, P_2$	<4	

Outline

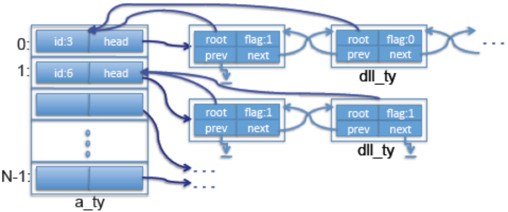
Reasoning about Composite Data Structures

— using a FO Logic Framework —

joint work with A. Bouajjani, C. Drăgoi, C. Enea

CONCUR'09

```
struct a_ty {
    int id;
    dll_ty* head;
}
struct dll_ty {
    bool flag;
    a_ty* root;
    dll_ty* next, *prev;
}
a_ty arr[N];
```



Structure: “the array contains in each cell a reference to an acyclic doubly linked list”

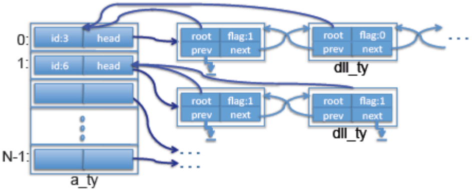
Sizes: “the array is sorted in decreasing order w.r.t. the lengths of lists stored”

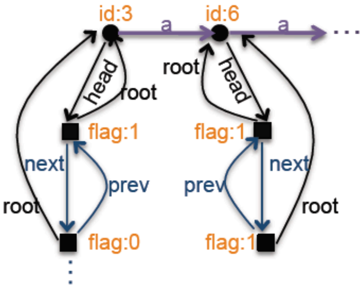
Data: “the array is sorted w.r.t. the values of the field `id`”

Recall: Heap Graph Model

Heaps are represented as labeled directed graphs called heap graphs

```
struct a_ty {
    int id;
    dll_ty* head;
}
struct dll_ty {
    bool flag;
    a_ty* root;
    dll_ty* next, *prev;
}
a_ty arr[N];
```





The graph is deterministic

The array fields create acyclic distinct paths

A Very Expressive Logic

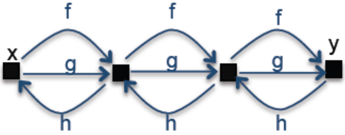
- assume $\mathbb{D}$ the data domain where data fields take values
- assume $\mathrm{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$ a first order logic on $\mathbb{D}$, with operations in $\mathbb{O}$ and predicates in $\mathbb{P}$

gCSL is a multi-sorted first order logic on graphs parametrized by $\mathrm{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$

$$\mathrm{gCSL} = \mathrm{FO} \ + \ \text{reachability} \ + \ \begin{array}{c}\text{arithmetical}\\ \text{constraints}\end{array} \ + \ \mathrm{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$$

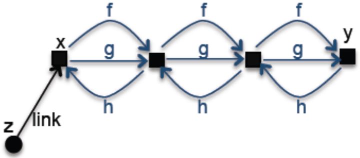
Reachability Predicates

$$x \xrightarrow{\{f, g, \overline{h}\}} y$$



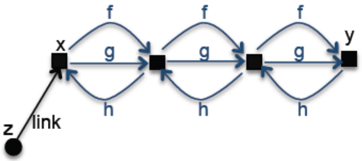
Reachability Predicates

$$x \xrightarrow{\{f,g,\overline{h}\}} y \qquad link(z) = x$$



$$x \xrightarrow{\{f,g,\overline{h}\},l} y$$

$$x \xrightarrow{\{f,g,\overline{h}\}} y \qquad link(z) = x$$



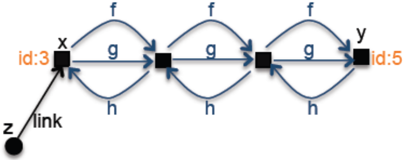
$$id(x) = 3$$

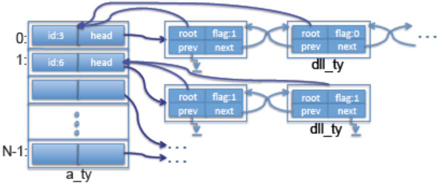
$$\exists c.\ id(x) + id(y) + c \geq 9$$

$$x \xrightarrow{\{f,g,\overline{h}\},l} y \wedge l = 3 \wedge v \xrightarrow{\{g\},l'} w$$

$$l' < l \wedge l + l' \geq 4$$

$$x \xrightarrow{\{f,g,\overline{h}\}} y \qquad link(z) = x$$





Structure: "the array contains in each cell a reference to an acyclic doubly linked list"

$$\forall i \; \exists x, y.\; x = \mathtt{head}(a[i]) \wedge x \xrightarrow{\{\mathtt{next}, \overline{\mathtt{prev}}\}} y$$

Sizes: "the array is sorted w.r.t. the lengths of lists stored"

$$\forall j, j'.\; j < j' \implies \exists x, x', l, l'.\; (x = \mathtt{head}(a[j]) \wedge x' = \mathtt{head}(a[j'])\wedge$$

$$x \xrightarrow{\{\mathtt{next}\}, l} \mathtt{null} \wedge x' \xrightarrow{\{\mathtt{next}\}, l'} \mathtt{null} \wedge l' \leq l)$$

Data: "the array is sorted w.r.t. the values of the field id"

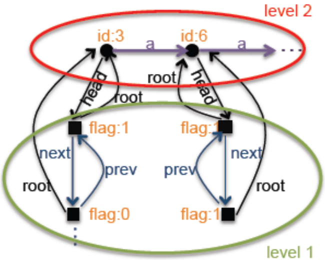
$$\forall i, j.\; i < j \implies \mathtt{id}(a[i]) < \mathtt{id}(a[j])$$

The satisfiability problem of gCSL is undecidable

- when data are restricted to finite domains (such as booleans), the logic subsumes the first-order logic on graphs with reachability
- when the models are restricted to simple structures, like sequences or arrays, for very simple data logics such as $(\mathbb{N}, =)$, the fragment $\forall^*\exists^*$ is undecidable

An ordered partition over $\mathcal{RT}$ is a mapping $\sigma : \mathcal{RT} \to \{1, \ldots, N\}$

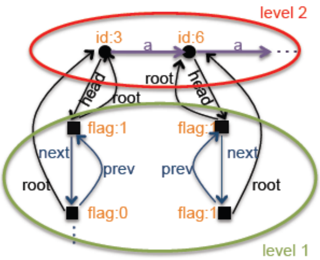
- a type $R \in \mathcal{RT}$ is of *level* k iff $\sigma(R) = k$



CSL Fragment

An ordered partition over $\mathcal{RT}$ is a mapping $\sigma : \mathcal{RT} \to \{1, \ldots, N\}$

- a type $R \in \mathcal{RT}$ is of *level* k iff $\sigma(R) = k$



For $1 \leq k \leq |\sigma|$,
CSL is the smallest set of formulas closed under disjunction and conjunction, which contains all the closed formulas of the form:

$$\exists^*_{\leq k}\ \forall^*_k\ \ \exists^*_{\leq k-1}\ \forall^*_{k-1}\ \ \ldots\ \ \exists^*_1\ \forall^*_1\ \ \{\exists_d, \forall_d\}^*.\ \phi$$

ϕ is a quantifier-free formula in gCSL

For $1 \leq k \leq |\sigma|$,
CSL is the smallest set of formulas closed under disjunction and conjunction, which contains all the closed formulas of the form:

$$\exists^*_{\leq k}\ \forall^*_k\ \ \exists^*_{\leq k-1}\ \forall^*_{k-1}\ \ldots\ \exists^*_1\ \forall^*_1\ \ \{\exists_d, \forall_d\}^*.\ \phi$$

ϕ is a quantifier-free formula in gCSL such that:

- REACH: for any $x \xrightarrow{A,ind} x'$, x and x' are free or existential variables
- UNIVIDX: two universal index variables can be used only in $j < j'$ or $j = j'$

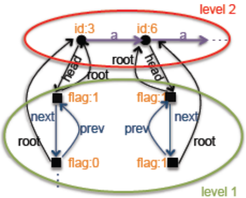
$$\text{YES} \quad \forall j, j'.\ j < j' \Rightarrow data(a[j]) < data(a[j'])$$

$$\text{NO} \quad \forall j, j'.\ j + 1 = j' \Rightarrow data(a[j]) < data(a[j'])$$

- LEV: atomic constraints on lengths of lists and array indexes involve only one level

$$\text{YES } \exists x, x', l_1 \exists z, z', l_2.\ x \xrightarrow{\{f\}, l_1} x' \wedge z \xrightarrow{\{f\}, l_2} z' \wedge l_1 \geq 4 \wedge l_2 \geq 0$$

$$\text{NO } \exists x, x', l_1 \exists z, z', l_2.\ x \xrightarrow{\{f\}, l_1} x' \wedge z \xrightarrow{\{f\}, l_2} z' \wedge l_1 + l_2 \geq 0$$



Structure: "the array contains in each cell a reference to an acyclic doubly linked list"

$$\forall i\ \exists x, y.\ x = \mathtt{head}(a[i]) \wedge x \xrightarrow{\{\mathtt{next},\mathtt{prev}\}} y$$

Sizes: "the array is sorted w.r.t. the lengths of lists stored"

$$\forall j, j'.\ j < j' \Longrightarrow \exists x, x', l, l'.\ (x = \mathtt{head}(a[j]) \wedge x' = \mathtt{head}(a[j']) \wedge$$

$$x \xrightarrow{\{\mathtt{next}\},l} \mathtt{null} \wedge x' \xrightarrow{\{\mathtt{next}\},l'} \mathtt{null} \wedge l' \leq l)$$

Data: "the array is sorted w.r.t. the values of the field id"

$$\forall i, j.\ i < j \Longrightarrow \mathtt{id}(a[i]) < \mathtt{id}(a[j])$$

Theorem

The satisfiability of CSL is decidable if the satisfiability of the underlying first order logic $\mathsf{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$ *is decidable*

Let

$$\varphi_k = \exists^*_{\leq k}\mathbf{r}\ \forall^*_k\mathbf{p}\ \ \exists^*_{\leq k-1}\mathbf{r}'\ \forall^*_{k-1}\mathbf{p}'\ \ldots\ \exists^*_1\mathbf{r}''\ \forall^*_1\mathbf{p}''\ \ \{\exists_d, \forall_d\}^*.\ \phi$$

1. compute φ_{k-1} equi-satisfiable to φ_k such that

$$\varphi_{k-1} = \exists^*_{\leq k-1}\mathbf{z}\ \forall^*_{k-1}\mathbf{w}\ \ldots\ \exists^*_1\mathbf{z}'\ \forall^*_1\mathbf{w}'\ \ \{\exists_d, \forall_d\}^*.\ \phi'$$

   until it ends up with a formula over variables of level 1

$$\varphi = \exists^*_1\mathbf{x}\ \forall^*_1\mathbf{y}\ \ \{\exists_d, \forall_d\}^*.\ \phi''$$

2. reduce the satisfiability of φ to the satisfiability of a formula in $\mathsf{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$

Theorem

The satisfiability of CSL is decidable if the satisfiability of the underlying first order logic FO$(\mathbb{D}, \mathbb{O}, \mathbb{P})$ is decidable

Let

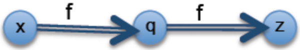
$$\varphi = \exists_1^* \mathbf{x}\ \forall_1^* \mathbf{y}\ \ \{\exists_d, \forall_d\}^* .\ \phi''$$

1. compute the set of small models for the reachability and size constraints
2. for each small model, build a FO$(\mathbb{D}, \mathbb{O}, \mathbb{P})$ formula ψ

   If one of ψ is satisfiable then φ is satisfiable.

Computing Small Models

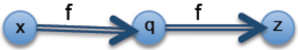
$$\varphi = \exists x, q, z \;.\; x \xrightarrow{\{f\}} q \wedge x \xrightarrow{\{f\}} z \;\wedge\; q \neq z$$

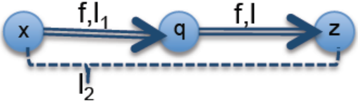


- φ has two small models of size three

Computing Small Models

$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\},l_1} q \wedge x \xrightarrow{\{f\},l_2} z\ \wedge\ q \neq z$$
$$\wedge\ l_1 + l_2 \geq 8$$





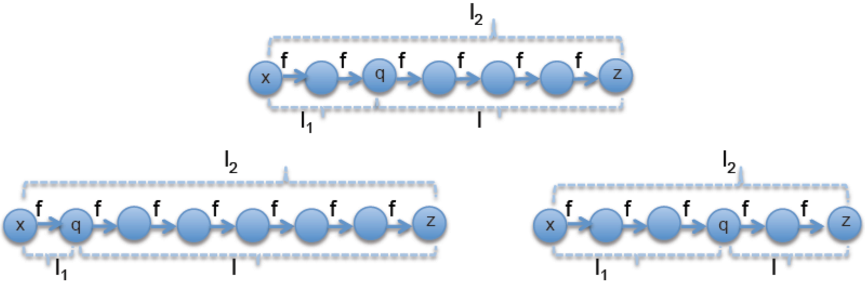
$$l_1 + l = l_2 \wedge l_1 + l_2 \geq 8$$

$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z\ \wedge\ q \neq z$$
$$\wedge\ l_1 + l_2 \geq 8$$

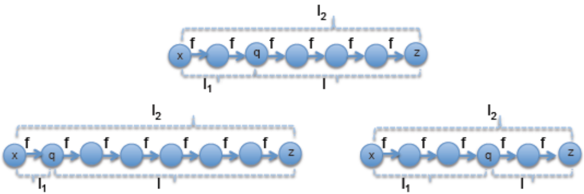
- minimal solutions (l_1, l_2, l) for $l_1 + l = l_2 \wedge l_1 + l_2 \geq 8$

$$\mathcal{M} = \{(1,7,6), (2,6,4), (3,5,2)\}$$

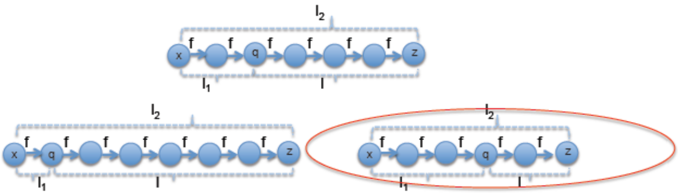
- small models for φ



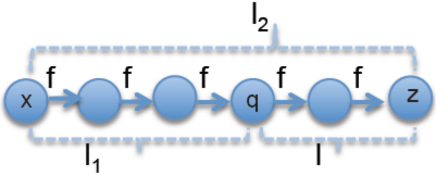
$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z\ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\ g(x) = 0 \wedge g(q) = 2$$
$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



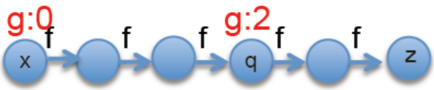
$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z\ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\ g(x) = 0 \wedge g(q) = 2$$
$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \land x \xrightarrow{\{f\}, l_2} z\ \land\ q \neq z$$
$$\land l_1 + l_2 \geq 8$$
$$\land\, g(x) = 0 \land g(q) = 2$$
$$\land\, \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z\ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\ g(x) = 0 \wedge g(q) = 2$$
$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$

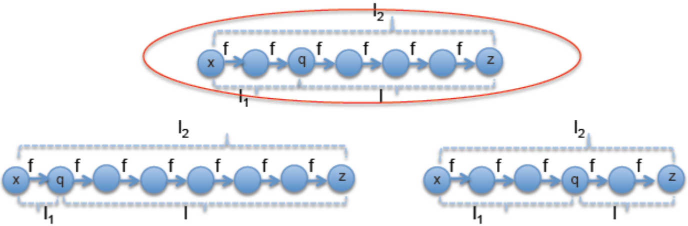


$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\},l_1} q \wedge x \xrightarrow{\{f\},l_2} z\ \wedge\ q \neq z$$

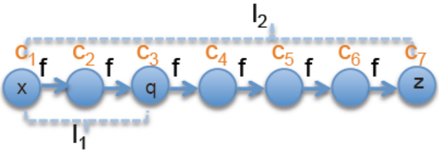
$$\wedge l_1 + l_2 \geq 8$$

$$\wedge\ g(x) = 0 \wedge g(q) = 2$$

$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z\ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\ g(x) = 0 \wedge g(q) = 2$$
$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



$$\psi_1 = \exists c_1, c_2, c_3, c_4, c_5, c_5, c_6, c_7.\ \bigwedge_{i \neq j} c_i \neq c_j$$
$$\mathit{true} \wedge \mathit{true}\ \wedge\ \mathit{true}\ \wedge \mathit{true}$$
$$\wedge\ c_1 = 0 \wedge c_3 = 2$$
$$\wedge\ \bigwedge_{1 \leq i < j \leq 7} c_i < c_j$$

$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z \ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\ g(x) = 0 \wedge g(q) = 2$$
$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$

$$\psi_1 = \exists c_1, c_2, c_3, c_4, c_5, c_5, c_6, c_7.\ \bigwedge_{i \neq j} c_i \neq c_j$$
$$\mathit{true} \wedge \mathit{true}\ \wedge\ \mathit{true}\ \wedge \mathit{true}$$
$$\wedge\ c_1 = 0 \wedge c_3 = 2$$
$$\wedge\ \bigwedge_{1 \leq i < j \leq 7} c_i < c_j$$

Decision Procedure: Summary

1. choose a small model for the reachability and size constraints; if there are no models then φ is unsatisfiable
2. build a FO($\mathbb{D}, \mathbb{O}, \mathbb{P}$) formula ψ for the selected small model
3. check the satisfiability of ψ

Remark

The complexity of the reduction procedure is NP^{MOILP} when the number of universally quantified variables is fixed.

CSL and Program Verification

Theorem

If the satisfiability of the underlying first order logic $FO(\mathbb{D}, \mathbb{O}, \mathbb{P})$ *is decidable, then the satisfiability of CSL is decidable*

Theorem

For any basic statement S *and any CSL formula* φ*, we can compute in polynomial time a formula* $\mathtt{post}(S, \varphi)$ *describing the strongest post-condition of* φ *by* S*.*

Outline

Observation

The limits of specifying complex heap shapes in SL are given by the class of inductive predicates allowed.

However, the classical data structures may be specified.

Exercise: Specify the shape of the following data structures:

- Binary trees
- Doubly linked lists segments
- Tree with linked leaves

Observation

The limits of specifying complex heap shapes in SL are given by the class of inductive predicates allowed.

However, the classical data structures may be specified.

Exercise: Specify the shape of the following data structures:

$$\begin{aligned}
\mathtt{dll}(E, L, P, F) &\triangleq (E = F \wedge L = P \wedge \mathit{emp}) \vee \big(E \neq F \wedge L \neq P \wedge \\
&\quad \exists X.\ E \mapsto \{(\mathtt{nxt}, X), (\mathtt{prv}, P)\} * \mathtt{dll}(X, L, E, F)\ \big) && (1)\\
\mathtt{btree}(E) &\triangleq (E = \boxtimes \wedge \mathit{emp}) \vee \big(E \neq \boxtimes \wedge \\
&\quad \exists X, Y.\ E \mapsto \{(\mathtt{lson}, X), (\mathtt{rson}, Y)\} * \mathtt{btree}(X) * \mathtt{btree}(Y)\big) && (2)\\
\mathtt{tll}(R, P, E, F) &\triangleq (R = E \wedge R \mapsto \{(\mathtt{lson}, \boxtimes), (\mathtt{rson}, \boxtimes), (\mathtt{parent}, P), (\mathtt{nxt}, F)\})\ \vee \\
&\quad \big(R \neq E \wedge \exists X, Y, Z.\ \ R \mapsto \{(\mathtt{lson}, X), (\mathtt{rson}, Y), (\mathtt{parent}, P), (\mathtt{nxt}, Z)\} * \\
&\qquad \mathtt{tll}(X, R, E, Z) * \mathtt{tll}(Y, R, Z, F)\big) && (3)
\end{aligned}$$

The fragment allowing these specifications has good theoretical properties:

- decidability of satisfiability [Brotherston *et al*, 14]
  $\longrightarrow$ by reduction boolean equations
- decidability of the entailment [Iosif *et al*, 13]
  $\longrightarrow$ by reduction to MSO on graphs with bounded width

Recently, efficient dedicated solvers have been released, *e.g.*:

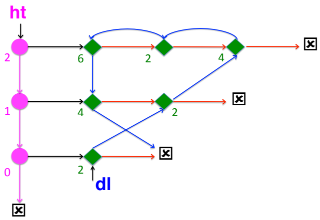
- Asterix [Perez&Rybalchenko,11]
- Cyclist-SL and SAT-SL [Gorogiannis *et al*,12]
- SLEEK [Chin *et al*, 10]
- SLIDE [Iosif *et al*, 14]
- SPEN [Enea,Lengal,S.,Vojnar, 14]

Follow them on SL-COMP competition:

- 6 solvers involved (freely available on StarExec)
- more than 600 benchmarks

`www.liafa.univ-paris-diderot.fr/slcomp`

- Introducing content and size constraints [Chin *et al*, 10],[S. *et al*, 15]
- Adding pre-field separation to express overlaid data structures [Yang *et al*,11],[Enea *et al*, 13]



$$\mathtt{nll}_\beta(\mathrm{h}, \boxtimes, \boxtimes) \circledast \mathtt{ls}_\delta(\mathrm{dl}, \boxtimes) \wedge \beta(\Diamond) = \delta(\Diamond)$$

- Shape analysis benefits from Separation Logic compositional reasoning.
- Shape analysis may be extended to content and size analysis.
- Efficiency is obtained using sound syntax-oriented procedures.
- Sound procedures for undecidable logic fragments may be obtained by applying static analysis.