

From Pointer Systems to Counter Systems using Shape Analysis

Arnaud Sangnier

EDF R&D, LSV, ENS Cachan & CNRS

joint work with: Sébastien Bardin, Alain Finkel, Etienne Lozes
LSV, ENS Cachan & CNRS

ACI Sécurité Informatique - 13th March 2006

Motivation (I)

Checking safety properties on systems manipulating dynamic linked lists

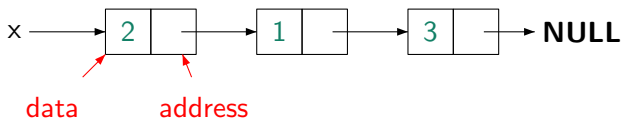
```
void deleteAll(List x) {  
    List elem;  
    while (x!=NULL) {  
        elem=x;  
        x=x->n;  
        elem->n=NULL;  
        free(elem);  
    }  
}
```

```
List reverse(List x) {  
    List y,t;  
    y =NULL;  
    while (x!=NULL) {  
        t=y;  
        y=x;  
        x=x->n;  
        y->n=t;  
        t=NULL;  
    }  
    return y;  
}
```

Motivation (II)

We manipulate pointer variables + memory heap

- Memory heap = collection of memory cells
- A memory cell contains : data or address



Properties

- Absence of memory violation (Segmentation fault)
- Absence of memory leak
- Qualitative properties on the memory heap (shape analysis)
 - Ex: The *reverse* function returns an acyclic list.
- Quantitative properties on the memory heap
 - Ex: The *reverse* function returns a list of the same size as the input list.

TVLA [Sagiv, Reps, Wilhelm 2002]

- Abstract interpretation
- The user has to provide control predicates

PALE [Moller, Schwartzbach 2001]

- Models the heap with a decidable monadic second order logic
- The user has to annotate the program

Smallfoot [Berdine, Calcagno, O'Hearn 2005]

- Checks separation logic specifications

[Bardin, Finkel, Nowak 2004]

- Symbolic representation of the memory heap using graphs
- Were looking for an acceleration technique

[Bouajjani, Habermehl, Moro, Vojnar 2005]

- Abstract regular model checking
- Regular expressions represent the memory heap

[Bozga, Iosif 2005]

- Alias logic with counters

- Modeling the memory heap using counters
- Translate the actions of the program into actions on the counters
- Propose a two-step analysis procedure

- 1 Models
- 2 Symbolic Computation
- 3 Translation
- 4 Analysis
- 5 Conclusion

A system model

$\mathcal{M} = (\mathcal{D}, \mathcal{G}, \mathcal{A})$ where

- \mathcal{D} is an infinite set of data
- $\mathcal{G} \subseteq 2^{\mathcal{D}}$ is a set of guards
- $\mathcal{A} \subseteq \mathcal{D}^{\mathcal{D}}$ is a set of actions

A system

A system $S = (Q, \delta)$ in $\mathcal{M} = (\mathcal{D}, \mathcal{G}, \mathcal{A})$ with

- Q is a finite set of control states
- $\delta \subset Q \times \mathcal{G} \times \mathcal{A} \times Q$ is a transition relation.

Semantics of a system

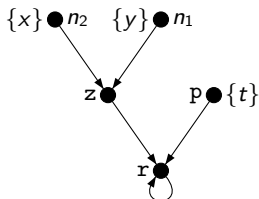
A transition system $TS(S) = (Q \times \mathcal{D}, \rightarrow)$ where $\rightarrow \subseteq (Q \times \mathcal{D})^2$ is defined as

- $(q, d) \rightarrow (q', d')$ iff $\exists g, a$ such that $(q, g, a, q') \in \delta$, $d \in g$ and $a(d) = d'$

Modeling the memory heap

Memory graph $(N, next, var)$ [BFN 04]

- N is a finite set of nodes with $\{z, p, r\} \subset N$
- $next : N \rightarrow N$ is a function such that :
 - $next^{-1}(\{r\}) = \{z, p, r\}$
- $var : N \rightarrow 2^{\mathbb{V}}$ is such that $\{var(n)\}_{n \in N}$ forms a partition of the set of pointer variables \mathbb{V}



$$N = \{n_1, n_2, z, p, r\}$$

$$next(n_1) = next(n_2) = z$$

$$var(n_1) = \{y\}, var(n_2) = \{x\}, \\ var(p) = \{t\}$$

The pointer model

- **Data** are memory graphs
- Syntax of **guards** :

$$g ::= \text{True} \mid \text{IsNull}(x) \mid \neg \text{IsNull}(x) \mid \dots$$

- Syntax of **actions** :

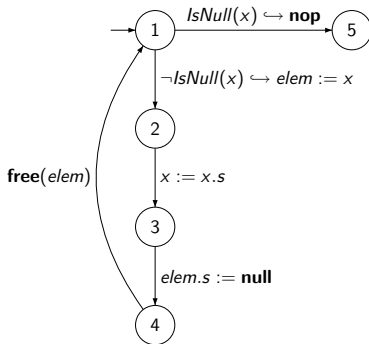
$$a ::= x := E \mid x.s := E \mid x := \mathbf{new} \mid \mathbf{free}(x) \mid \mathbf{nop}$$

where E is either **null**, or x or $x.s$.

A pointer system

The program *deleteAll* that deletes all the elements of a list

```
void deleteAll(List x) {  
  List elem;  
  while (x!=NULL) {  
    elem=x;  
    x=x->n;  
    elem->n=NULL;  
    free(elem);  
  }  
}
```



Definition [Memory violation]

A memory graph has a memory violation if $\text{var}(\mathbf{x}) \neq \emptyset$

Definition [Memory leakage]

A memory graph has a memory leakage if N contains at least one node not reachable from a node labeled by a variable.

Theorem [BFN 04]

The reachability of memory leakage and of memory violation are undecidable problems for pointer systems with at least 3 pointer variables.

The counter model ;

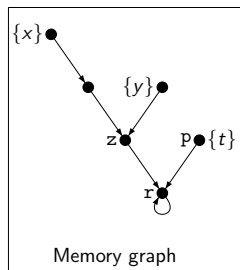
- **Data** are vectors of $\mathbb{N}^{\mathbb{K}}$ (\mathbb{K} is a finite set of counters)
- **Guards** are Presburger formulae interpreted in $\mathbb{N}^{\mathbb{K}}$
- **Actions** are linear functions in $\mathbb{N}^{\mathbb{K}}$

We can use the tool FAST ([Bardin, Finkel, Leroux, Petrucci 2003]) to analyze counter systems:

- Symbolic verification of counter system terminates often in practice

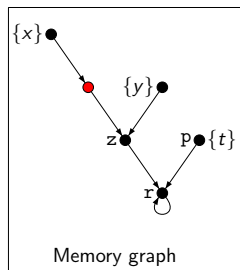
- 1 Models
- 2 Symbolic Computation**
- 3 Translation
- 4 Analysis
- 5 Conclusion

Memory shapes



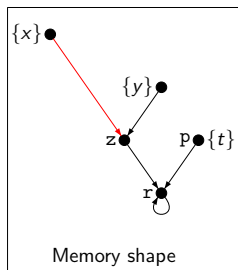
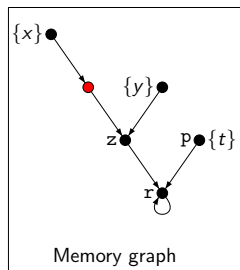
Memory shapes

- Compression of irrelevant nodes



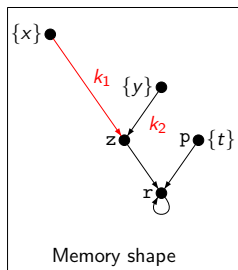
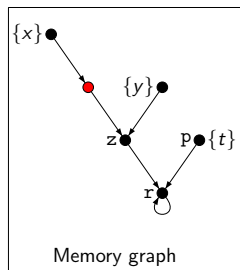
Memory shapes

- Compression of irrelevant nodes



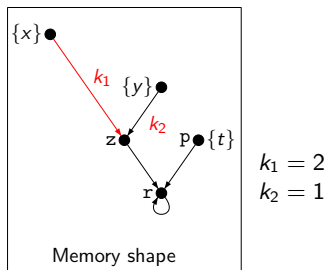
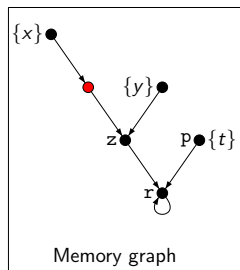
Memory shapes

- Compression of irrelevant nodes
- Labeling with strictly positive counters



Memory shapes

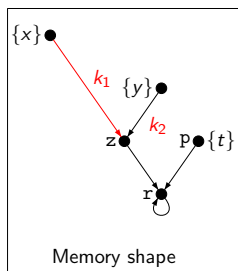
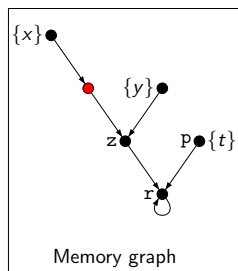
- Compression of irrelevant nodes
- Labeling with strictly positive counters



- A memory shape + a counter valuation = a unique memory graph (without memory leakage)

Memory shapes

- Compression of irrelevant nodes
- Labeling with strictly positive counters
- Two special memory shapes: *MemLeak* and *SegF*



$$k_1 = 2$$
$$k_2 = 1$$

- A memory shape + a counter valuation = a unique memory graph (without memory leakage)

Properties of memory shapes

We consider a finite set of pointer variables \mathbb{V} .

Number of counters

The number of counters in a memory shape is bounded by $2 \cdot |\mathbb{V}|$.

Number of memory shapes

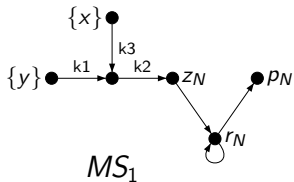
The number of memory shapes built over \mathbb{V} is bounded by $2 \cdot |\mathbb{V}|^{(3 \cdot |\mathbb{V}|)}$.

Equality test

Testing equality of two memory shapes can be done in linear time.

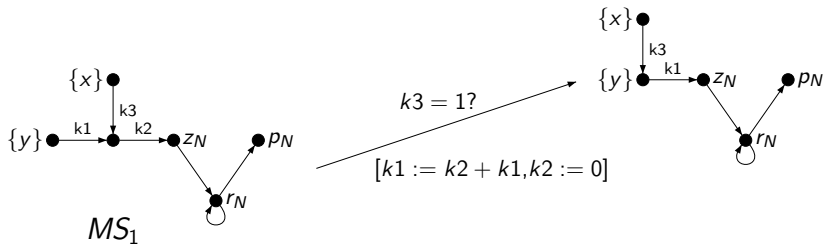
Symbolic computation on memory shapes (I)

- Effect of the action $x.s := y$ on the memory shape MS_1



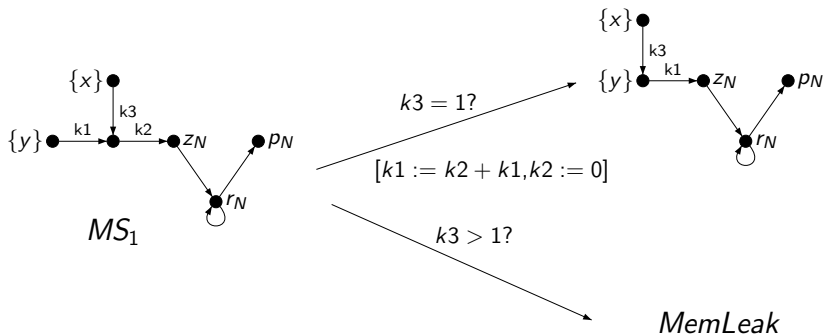
Symbolic computation on memory shapes (I)

- Effect of the action $x.s := y$ on the memory shape MS_1



Symbolic computation on memory shapes (I)

- Effect of the action $x.s := y$ on the memory shape MS_1



The symbolic function TEST

- Decides whether a memory shape satisfies a pointer guard (ie all the underlying memory graphs for any valuation satisfy it).

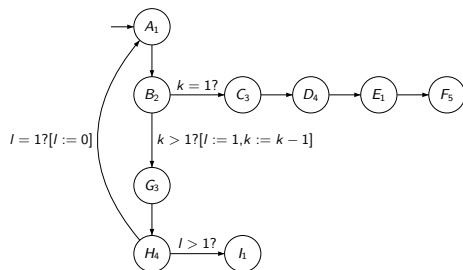
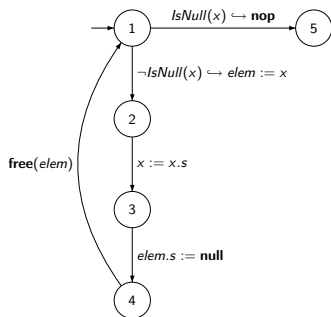
The symbolic function POST

- Produces, from a given memory shape and a pointer action, the set $\{(\phi_i, f_i, MS_i)\}$ of all possible issues (the ϕ_i 's are Presburger formulae and the f_i 's are linear functions)

- 1 Models
- 2 Symbolic Computation
- 3 Translation**
- 4 Analysis
- 5 Conclusion

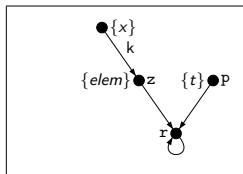
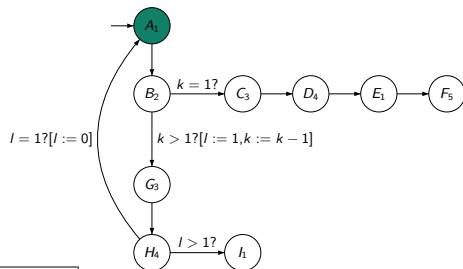
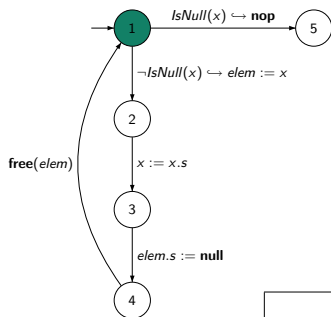
Building of the counter system

- To each control state of the counter system we associate a memory shape and a control state of the pointer system



Building of the counter system

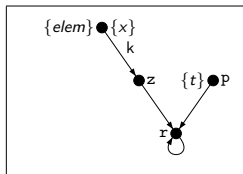
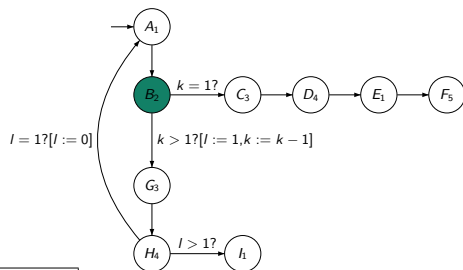
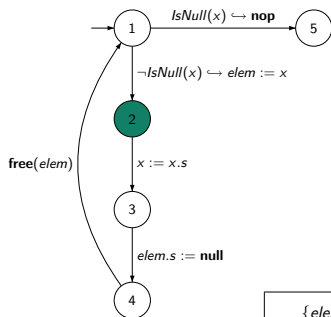
- To each control state of the counter system we associate a memory shape and a control state of the pointer system



Translation

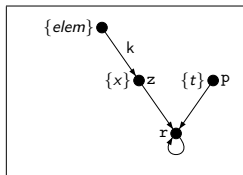
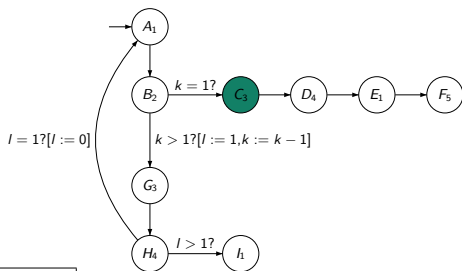
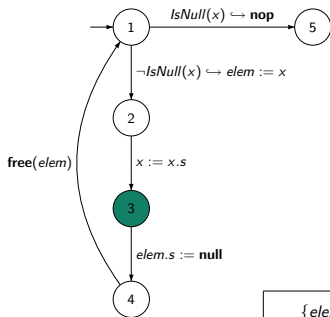
Building of the counter system

- To each control state of the counter system we associate a memory shape and a control state of the pointer system



Building of the counter system

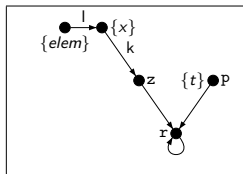
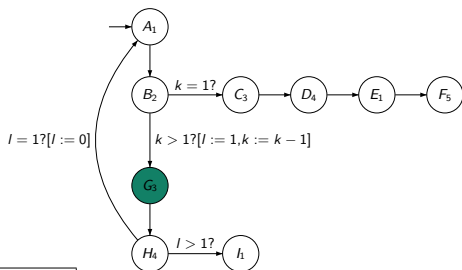
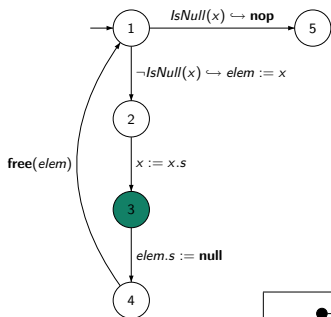
- To each control state of the counter system we associate a memory shape and a control state of the pointer system



Translation

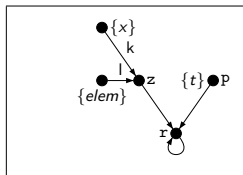
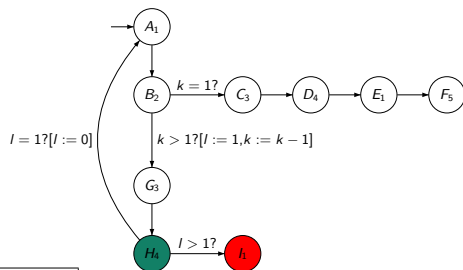
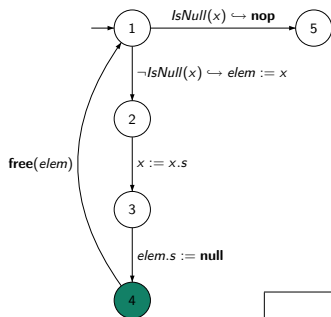
Building of the counter system

- To each control state of the counter system we associate a memory shape and a control state of the pointer system



Building of the counter system

- To each control state of the counter system we associate a memory shape and a control state of the pointer system



Translation

The translation is effective.

Bisimulation [BFLS 2006]

The following relation :

$$\mathcal{R} = \left\{ ((q, \langle MS, val \rangle), ((q, MS), val)) \mid q \in Q_p, MS \in \mathcal{MS}, val \in \mathbb{N}^{\mathbb{K}} \right\}$$

is a bisimulation between the transitions systems of the pointer system and the counter system.

- 1 Models
- 2 Symbolic Computation
- 3 Translation
- 4 Analysis**
- 5 Conclusion

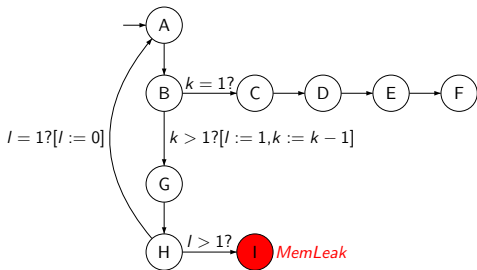
Over-approximation

The set of memory shapes appearing in the control states of the counter system is an over-approximation of the memory shapes reachable in the pointer system.

This leads to a 2 step analysis :

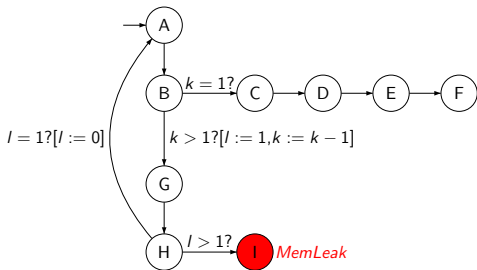
- 1 Checking if the unsafe states are present in the control states of the counter system
- 2 In the positive case, checking if they are effectively reachable using FAST

2 steps analysis (example)



- The memory shape *SegF* does not appear in the counter system
- The memory shape *MemLeak* does in state I

2 steps analysis (example)



- The memory shape *SegF* does not appear in the counter system
- The memory shape *MemLeak* does in state I
- BUT the control state I is not reachable

- 1 Models
- 2 Symbolic Computation
- 3 Translation
- 4 Analysis
- 5 Conclusion**

Concluding remarks

- Our translation is automatic
- It allows the verification of quantitative properties on classical examples
- A prototype has been implemented in O'CAML

On going work

- Improve the translation to reduce the produced counter systems
- Add counter variables in the pointer systems