

Towards Model Checking Pointer Systems

Arnaud Sangnier

EDF R&D, LSV, ENS Cachan & CNRS

joint work with Alain Finkel and Etienne Lozes (LSV)

ILC - 5th November 2007

Cape Town - South Africa

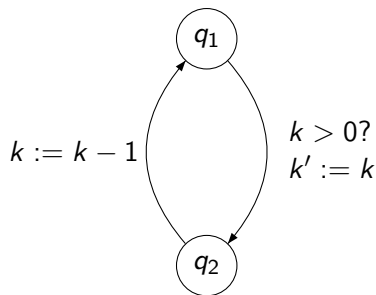
Checking safety and temporal properties on systems manipulating dynamic linked lists

```
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;  
}
```

- Providing a symbolic framework for programs manipulating pointers
- Defining a temporal logic using this framework
- Proposing a semi-algorithm to decide model-checking based on a translation towards counter systems
- Studying decidability

- 1 Preliminaries
- 2 Model Checking
- 3 From Pointer Systems to Counter Systems
- 4 Decidability
- 5 Conclusion

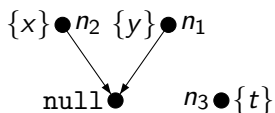


- Manipulates integer variables
- Each transition labeled with :
 - **A guard** :
Presburger formula
 - **An action** :
linear function
- Configuration (q, v)
 - q : control state
 - v : counter valuation

Modeling the memory heap

A memory graph $MG = (N, succ, loc)$

- N is a finite set of nodes such that $\text{null} \notin N$
- $succ$ is a partial function from N to $N \cup \{\text{null}\}$
- $loc : V \rightarrow N \cup \{\text{null}\}$
- $\forall n \in N, \exists v \in V$ such that $n \in succ^*(loc(v))$

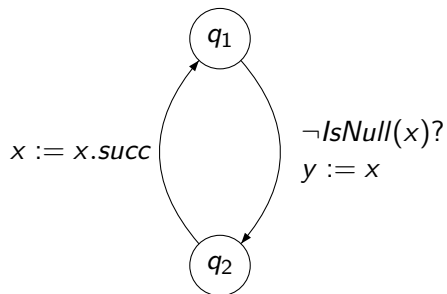


$$N = \{n_1, n_2, n_3\}$$

$$succ(n_1) = succ(n_2) = \text{null}$$

$$loc(y) = n_1, loc(x) = n_2,$$

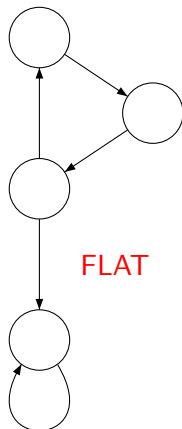
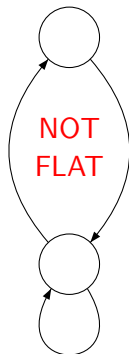
$$loc(t) = n_3$$



- Manipulates pointer variables
- Each transition labeled with :
 - **A guard** :
Test to *null*
 - **An action** :
 $x := y, x := x.succ$
 $x.succ = y, x := malloc$
 $free(x)$
- Configuration (q, MG) :
 - q : control state
 - MG : memory graph

Flat systems

- A system is flat if every control state belongs to at most one cycle with no repeated vertex



- 1 Preliminaries
- 2 Model Checking**
- 3 From Pointer Systems to Counter Systems
- 4 Decidability
- 5 Conclusion

FOCTL* [DFGvD06]

$$\phi ::= q \mid \psi_{Pres} \mid \exists y. \phi \mid \neg \phi \mid \phi \wedge \phi \mid \mathbf{X}\phi \mid \phi \mathbf{U}\phi \mid \mathbf{A}\phi$$

Hypothesis : Let $CS = \langle Q, E \rangle$ be a **flat** counter system with the **finite monoid** property.

Finkel-Leroux 2002

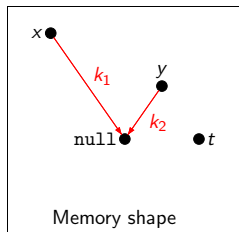
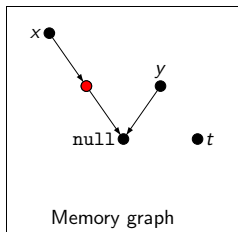
The relation \rightarrow^* is effectively Presburger definable

The tool FAST ([Bardin et al. 2003, 2006]) can be used to solve reachability problems for counter systems.

Demri-Finkel-Goranko-Van Drimmelen 2006

The symbolic model checking of FOCTL* is decidable for CS

A temporal logic for pointer systems



$$a(k_1) = 2$$

$$a(k_2) = 1$$

- Memory shape + Presburger formula = Infinite set of memory graphs

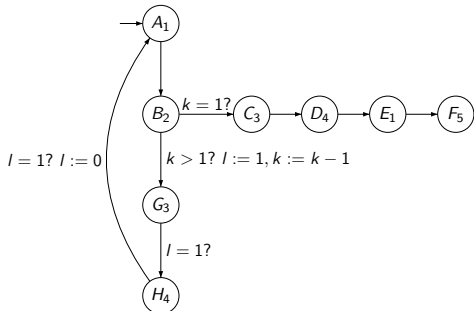
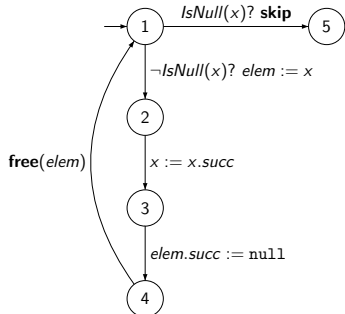
CTL_{mem}^*

$$\phi ::= q \mid \langle MS, \psi_{Pres} \rangle \mid \neg\phi \mid \phi \wedge \phi \mid X\phi \mid \phi U \phi \mid A\phi$$

- 1 Preliminaries
- 2 Model Checking
- 3 From Pointer Systems to Counter Systems**
- 4 Decidability
- 5 Conclusion

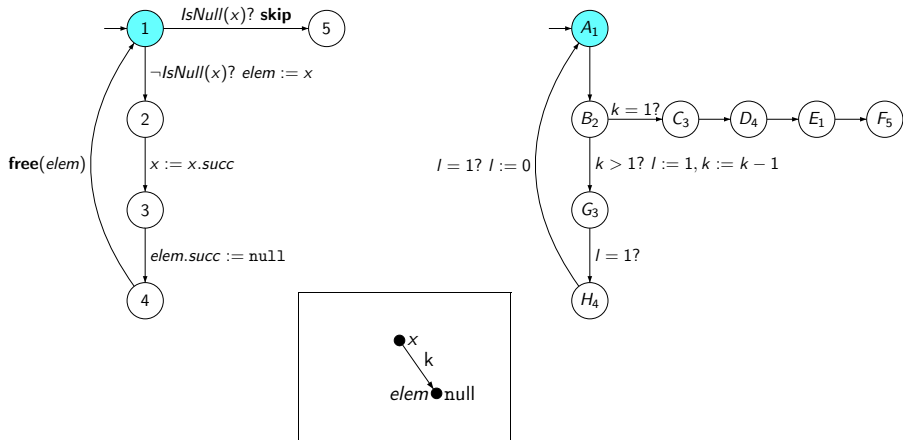
A first translation [BFLS06]

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



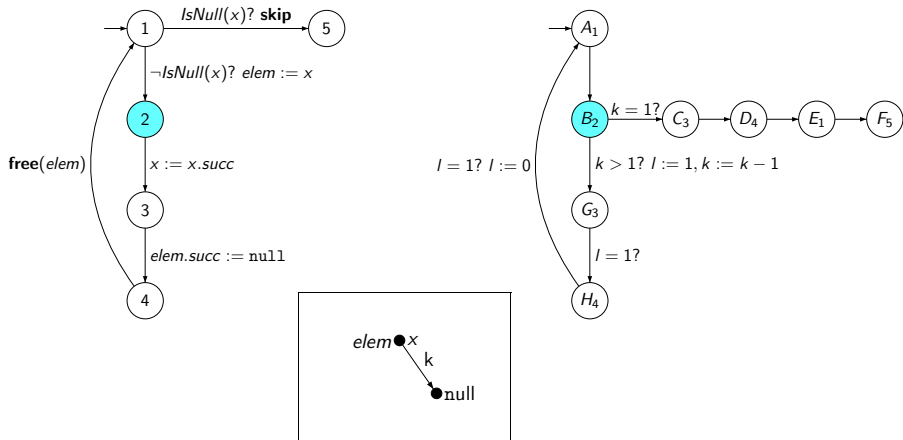
A first translation [BFLS06]

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



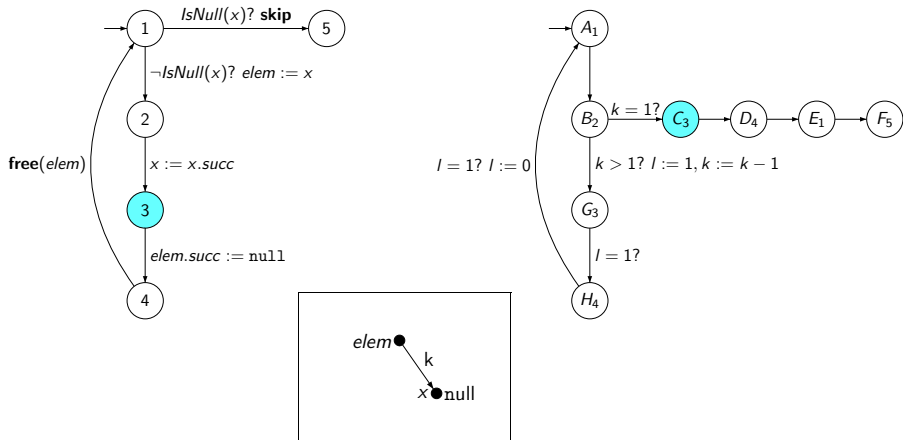
A first translation [BFLS06]

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



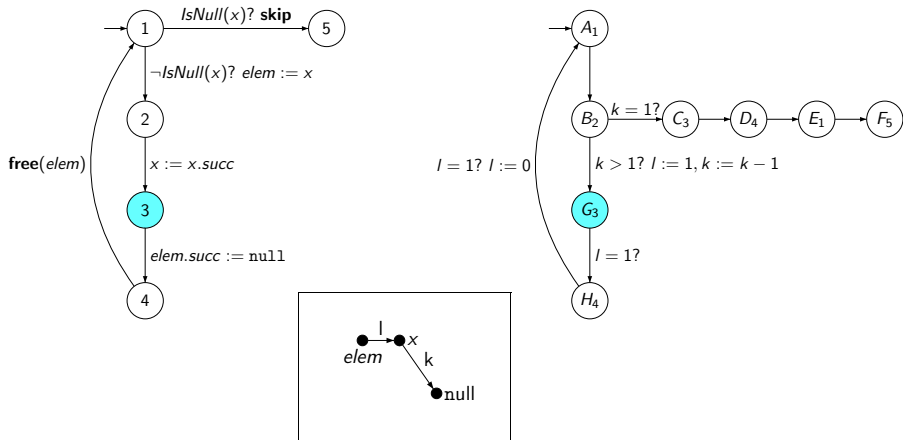
A first translation [BFLS06]

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



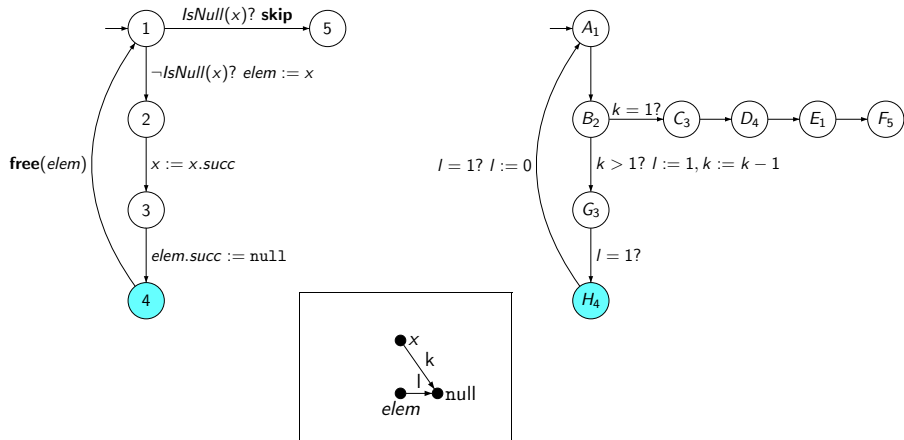
A first translation [BFLS06]

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



A first translation [BFLS06]

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



The obtained counter system :

- has the **finite monoid** property
- is **bisimilar** to the original pointer system

HENCE

- When the obtained counter system is flat, the symbolic model-checking is decidable

BUT

- Even for flat pointer systems, the obtained counter system is not flat

- 1 Preliminaries
- 2 Model Checking
- 3 From Pointer Systems to Counter Systems
- 4 Decidability**
- 5 Conclusion

Hypothesis : We consider:

- PS a **flat** pointer system without destructive update (ie action of the form $x.succ = e$)
- $INIT = (q_0, (MS_0, True))$
- $BAD = \bigcup_{i \in \{1, \dots, r\}} (q_i, (MS_i, True))$

Then:

Bozga-Iosif 2007

The safety problem is undecidable

Bozga-Iosif 2007

The safety problem is decidable if MS_0 has at most one cyclic list

With a new translation which preserves **flatness** for pointer systems **without destructive update**, we deduce :

Decidability of safety

For flat pointer systems without destructive update and without alias test, the safety problem is decidable.

→ This result does not extend to model checking!!!

With a new translation which preserves **flatness** for pointer systems **without destructive update**, we deduce :

Decidability of safety

For flat pointer systems without destructive update and without alias test, the safety problem is decidable.

→ This result does not extend to model checking!!!

Decidability of symbolic model checking

For flat pointer systems without destructive update and with an initial acyclic configuration, the symbolic model checking is decidable.

- 1 Preliminaries
- 2 Model Checking
- 3 From Pointer Systems to Counter Systems
- 4 Decidability
- 5 Conclusion**

Conclusion

- A framework to express temporal properties on pointer systems
- A semi-algorithm to verify symbolic model-checking problems
- Some decidability results

Future work

- Extend the used methods to more complex data structures
- Implementation