Toward Symbolic Verification of Programs Handling Pointers

Sébastien Bardin
Alain Finkel
David Nowak

LSV, CNRS & ENS Cachan, France

{bardin,finkel,nowak}@lsv.ens-cachan.fr

Programs Handling Pointers

- Pointers are useful but dangerous.
- They cannot always be avoided in critical systems.
- Verification is difficult because of aliasing.
- We aim at verifying that segmentation fault or memory leak will not happen under the following hypotheses:
  - we do not consider data;
  - we do not consider pointer arithmetic;
  - at first, we only consider linked lists.

Verification of programs handling pointers

Shape analysis

Sagiv, Reps and Wilhelm, POPL'99

Pointer assertion logic

Jensen, Joergensen, Klarlund and Schwartzbach, PLDI '97

Separation logic

Reynolds, Millennial Perspectives in Computer Science, 2000

O'Hearn, Reynolds and Yang, CSL'01

pointer alias analysis, points-to analysis

Many many papers

Model checking

No paper

Model checking

- Model checking attemps to reply automatically to questions of the form:

  Does my model $\mathcal{M}$ satisfy my property φ?

- In case φ is a safety property, this amounts to a reachability question in the model $\mathcal{M}$:

  $$\text{Reachable states} \cap \text{Bad states} = \emptyset$$

Infinite state systems

- A method to help termination of the model checking algorithm is needed.
  - widening (overapproximation, terminates)
  - acceleration (exact, semi-algorithm)

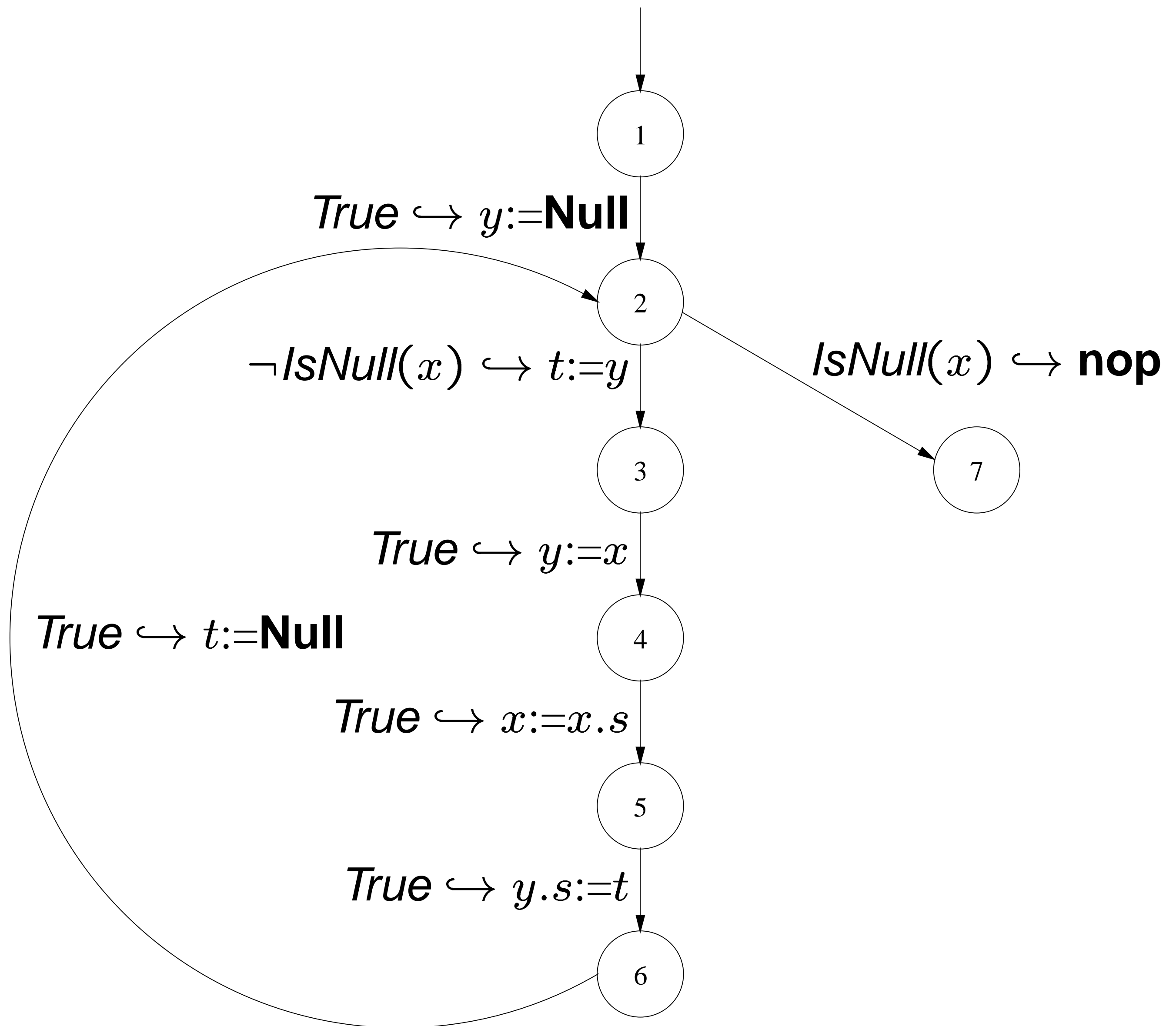
    It consists in computing in one step the effect of an arbitrary number of executions of a transition.

- Acceleration techniques strongly depend on symbolic representation
  - computable canonical form
  - closure under union, union computable
  - inclusion decidable
  - efficiency

An example of C program handling pointers

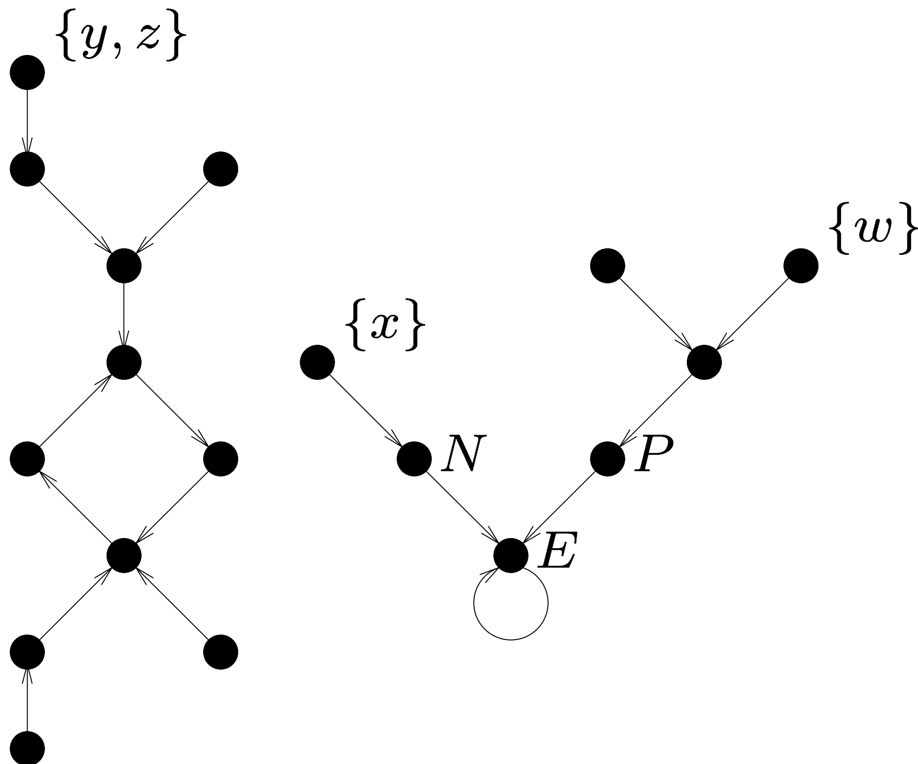
```
/* list.h */
typedef struct node
{
  struct node* n;
  int data;
}* List;

/* reverse.c */
#include ``list.h''
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;
}
```

Pointer automaton



The semantics of a pointer automaton is given in terms of a transition system whose states are memory graphs.

A memory graph is either *SegF* or a graph such as

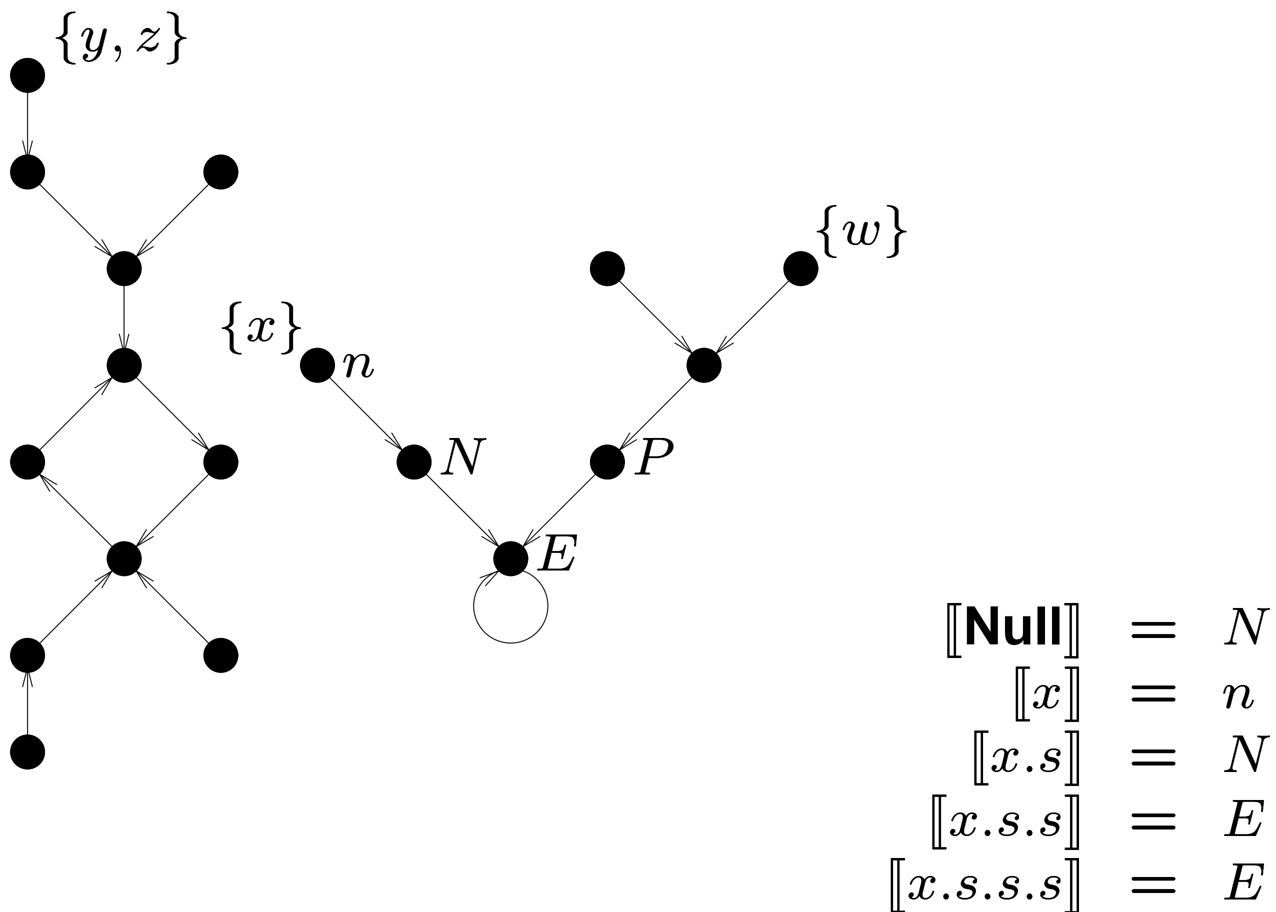


- Black points represent memory cells. Edges represent pointers.

- Each memory cell is labelled by the set of pointer variables which point to this cell.

- The output degree of each node is exactly 1: we only consider linked lists.

- Transition systems with memory graphs as states will be in general infinite.



- Here x is a well-formed linked list of length 1.

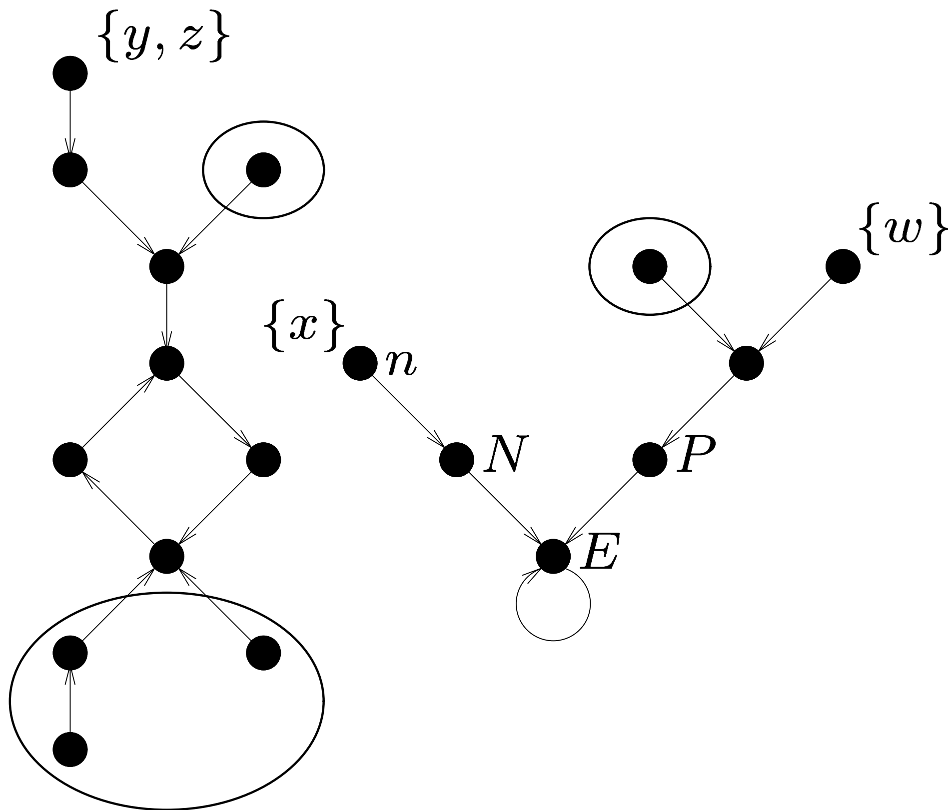
- N, P and E are special nodes which do not denote real memory cells:

  – N (Null) denotes null pointer expressions such as **Null** or $x.s$.

  – P (Pit) denotes deallocated pointer expressions such as $w.s.s$.

  – E (Error) denotes meaningless pointer expressions such as **Null**$.s$, $x.s.s$, $x.s.s.s$ or $w.s.s.s$.

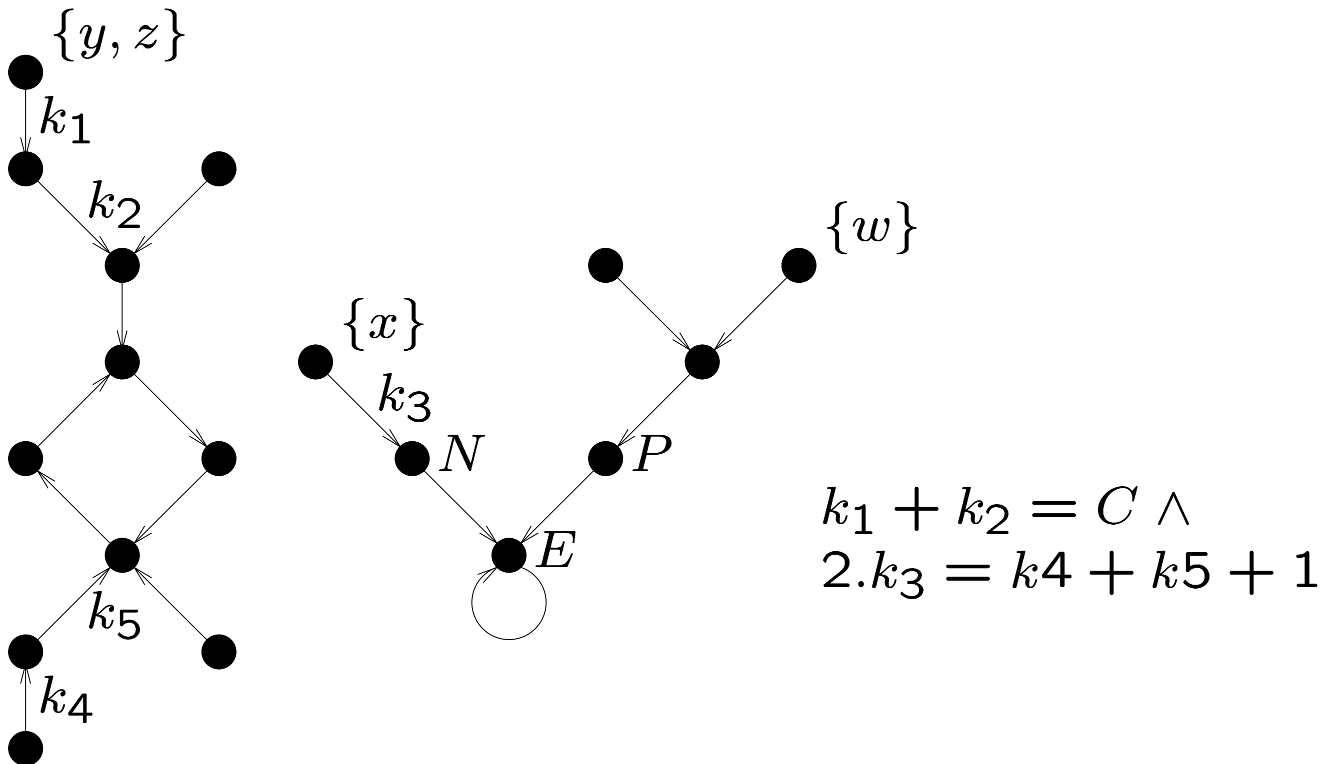
Dead nodes



A dead node is a node which cannot be reached from a node labelled by a non-empty set of variables.

Presence of dead nodes means there is a memory leak.

An SMS is a finite set of atomic SMSs such as



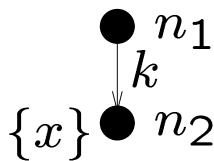
- Edges are labelled by counter variables which represent the number of pointers between the two nodes.

- Counters are constrained by a Presburger formula. Unconstrained counters are omitted.

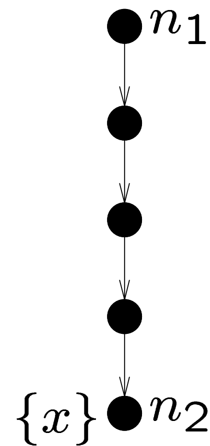
- Counters labelling edges (N, E), (P, E) and (E, E) are implicitly equal to 1 (in all SMSs).

- An SMS M represents an infinite set $\gamma(M)$ of memory graphs.

Assignment



$v(k) = 4$



- An assignment satisfying a formula ϕ is a valuation function v for counters such that $\phi[k_i{:=}v(k_i)]$ is true.

- The assignment of an SMS (G, ϕ) w.r.t. v is the memory graph G' built from G where each edge labelled by a counter variable k is replaced by a chain of $v(k)$ edges.

Concretization

Atomic SMSs

- $\gamma(G, \phi)$, where G is not *SegF*, is the set of all possible assignments of (G, ϕ).

- $\gamma(\textit{SegF}, \phi) = \begin{cases} \{\textit{SegF}\} \text{ if } \phi \text{ is satisfiable} \\ \emptyset \text{ otherwise} \end{cases}$

SMSs

- $\gamma(\{M_1, \ldots, M_n\}) = \bigcup_{i=1}^{n} \gamma(M_i)$

  where M_1, ..., M_n are atomic SMSs.

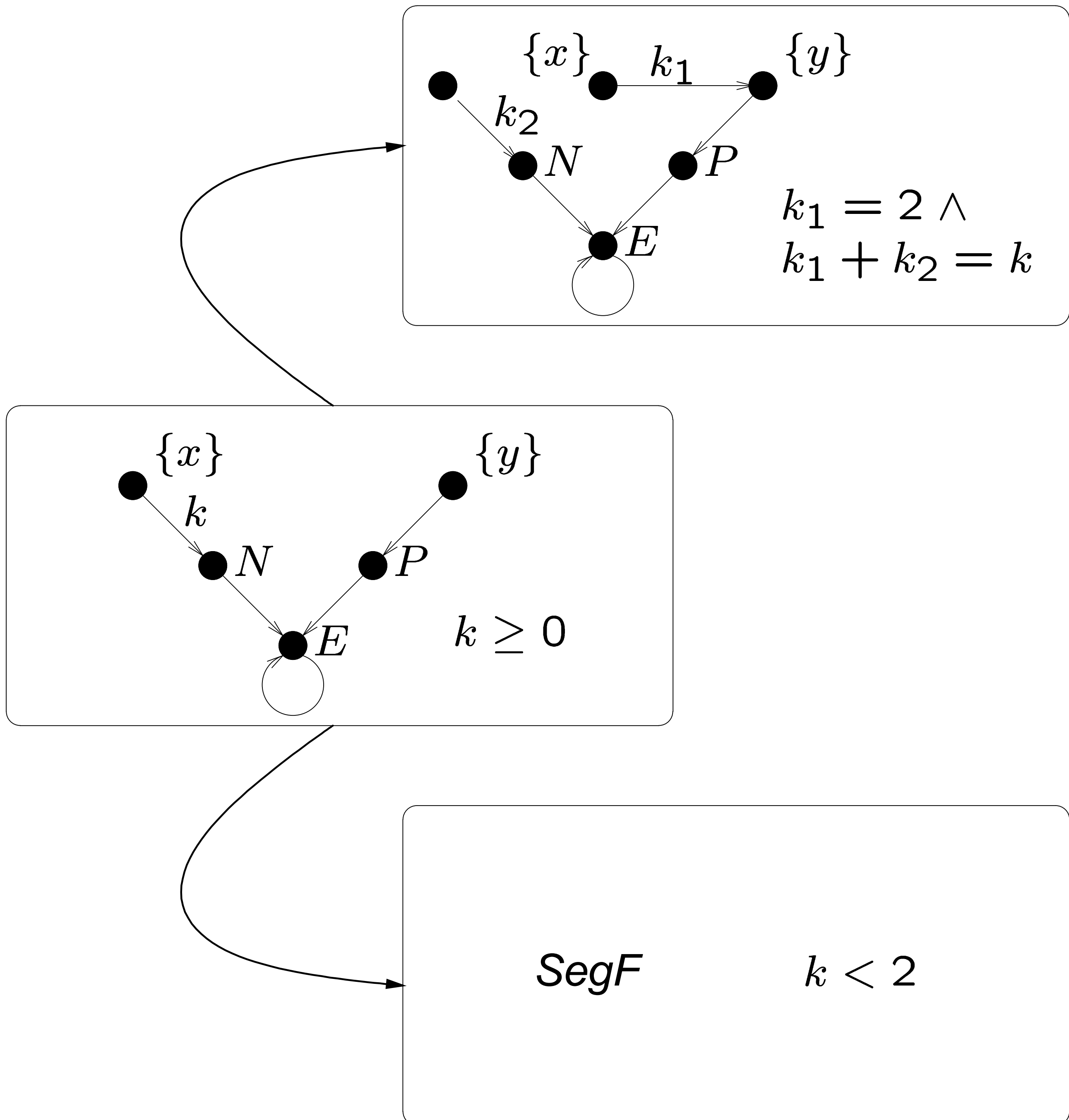
Union $\sqcup$ of SMSs such that

$$\gamma(M_1 \sqcup M2) = \gamma(M_1) \cup \gamma(M_2)$$

is defined the evident way.

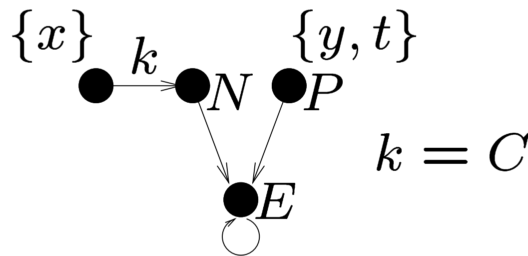
Execution of pointer automata

Example: an execution of the instruction $x.s.s$**:=**y

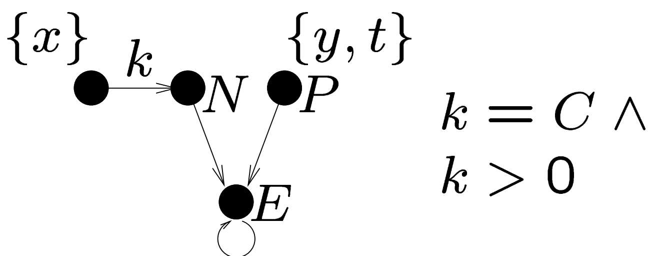


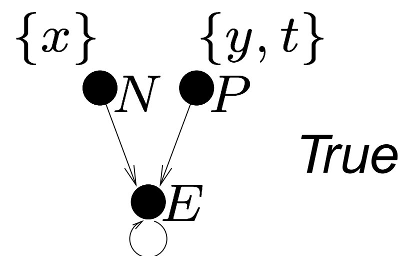
Each SMS M has a computable canonical form

An SMS



and its canonical form





An atomic SMS in canonical form is such that

- each counter is different from 0;
- only core nodes appear, i.e.
  - nodes N, S and E,
  - nodes labelled by a non empty set of variables,
  - nodes of input degree greater than or equal to 2.

An SMS in canonical form is a finite set of atomic SMSs in canonical form such that it does not contain isomorphic graphs.

Inclusion

We define a decidable relation $\sqsubseteq$ on SMSs such that

$$M_1 \sqsubseteq M_2 \;\Leftrightarrow\; \gamma(M_1) \subseteq \gamma(M_2)$$

It is defined on SMSs by:

- $(G, \phi) \sqsubseteq (G', \phi')$ iff G and G' are isomorphic and $\phi \Rightarrow \phi'$ is valid.

- $(\mathit{SegF}, \phi) \sqsubseteq (\mathit{SegF}, \phi')$ iff ϕ is not satisfiable or ϕ' is satisfiable.

- $M_1 \sqcup M_2 \;\sqsubseteq\; M$ iff $M_1 \sqsubseteq M$ and $M_2 \sqsubseteq M$.

- $M \;\sqsubseteq\; M_1 \sqcup M_2$ iff $M \sqsubseteq M_1$ and $M \sqsubseteq M_2$.

Undecidability results

NoSegF A state of the form $(q, (\textit{SegF}, \phi))$ where q is a control state and ϕ is satisfiable is not reachable.

NoMemLeak A state (q, M) where q is a control state and M is a memory leak is not reachable.

Theorem. **NoSegF** and **NoMemLeak** are undecidable for pointer automata with at least 3 pointer variables.

Future work

- We can check invariants (represented as SMSs) on pointer automata by computing one iteration. But it is difficult on realistic examples.

- We can try to isolate decidable subclasses. But we did not find realistic enough subclasses.

- We focus on acceleration techniques:
  - They can compute the set of reachable states.
  - They can compute invariants.
  - They have proved to be efficient on other models such as counter, clock or lossy channel systems.

- We will extend our symbolic representation to other structures than linked lists.

Acceleration