# Where could SPIN go next?
# A unifying approach to
# exploring
# infinite state spaces

Pierre Wolper

University of Liège

in order to verify infinite state systems ?

- Reachability,

- Reachability,

- Reachability,

- and repeated reachability.

Why ?

- It is necessary : if you cannot determine what states are reachable, there is not much you can do.

- It is sufficient : most verification problems are reducible to reachability or repeated reachability.

## Reducing verification to reachability
## Linear-time model checking

Verify that all the computations of a program $P$ satisfy a temporal logic specification $f$

Solution [VW86].

- Build the automaton for $\neg f$: $A_{\neg f}$

- Check whether there is a reachable accepting state in the product $P \times A_{\neg f}$ (safety properties), or

- Check whether there is a repeatedly reachable accepting state in the product $P \times A_{\neg f}$ (general properties).

## Reducing verification to reachability
## Branching-time model checking

Verify that the computation tree of a program $P$ satisfies a branching-time temporal logic specification $f$

Solution [KVW94].

- Build the alternating automaton for $\neg f$: $A_{\neg f}$.

- Check whether the product $P \times A_{\neg f}$ is nonempty. This product is a Boolean graph and nonemptiness can be checked by a form of Boolean reachability.

## Reducing verification to reachability
## Bisimulation, simulation, ...

Check whether there is a simulation relation between a transition system $P_1$ and a transition system $P_2$.

Solution [Lar92,Ver95]

- Build a Boolean graph from $P_1$ and $P_2$.

- Explore the Boolean graph in order to detect bisimulation or simulation failures.

## Deciding reachability

Two basic options.

- Forwards reachability:

  - Start with the initial state,

  - add states that are immediately reachable from the current set

  - until a fixpoint is reached.

- Backwards reachability:

  - Starting with the state(s) for which reachability is to be decided,

  - add the immediate predecessors of the current set of states, until the initial state or a fixpoint is reached.

**Problem:** For infinite state spaces, these procedures usually never stop.

## Solution

Compute with (infinite) *sets* of states at each step rather than with individual states. But,

- How does one move from individual states to sets of states ?

- How does one represent and compute with (infinite) sets of states ?

- How does one guarantee that the search terminates ?

## From states to Infinite sets of states

## 1) Searching forwards

- Usually, unique initial state.

- So, applying the transition relation of the program will never move beyond finite set of states.

Solution : compute directly the result of iterating the transition relation.

## Iterating transition relations

**Problem :** Given a state space $U$, a set of states $S \subseteq U$ and a transition relation $R \subseteq U \times U$, compute $R^*(S)$.

- In general this is impossible even if $R$ is very simple, e.g. the transition relation of a Turing machine.

- But, it can be done for some restricted $R$'s. Furthermore, even computing an infinite subset $r^*(S) \subseteq R^*(S)$ is useful. Indeed, the procedure can by repeated from $r^*(S)$, which is a potentially infinite set

## The loop first search: a pragmatic approach

Ideas

- Distinguish the control part and the data part of states, the infinite nature of the state space coming from the data part.

- Move from control point to control point, but attempt *first* to compute in one step the result of repeatedly applying sequences of transitions that lead from one control point to the same control point (*loops*)

- When this computation succeeds, carry on the state space exploration, handling simultaneously all data values that are possible for each control point.

## An example of a loop-first search



$$x := 0 \qquad x <= 1000000 \qquad x > 1000000$$
$$x := x + 2$$
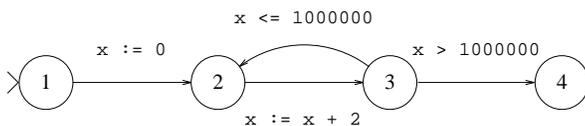
$(\textcircled{1}, \bot)$
$(\textcircled{2}, 0)$
$(\textcircled{2}, 2k)$ with $0 \leq k \in \mathbf{N} \leq 500000$
$(\textcircled{3}, 2k + 2)$ with $0 \leq k \in \mathbf{N} \leq 500000$
$(\textcircled{4}, 1000002)$

Note that a loop first search allows one to go arbitrarily deep into a computation in one step.

## From states to Infinite sets of states

### 2) Searching backwards

- Sometimes possible to start with an infinite set of states.

- From a given state, the initial state is always reachable in a finite number of steps.

- However, if it is not, the backwards computation might need to go arbitrarily deep before terminating.

So, no general advantage on a forwards search and does not allow the computation of a representation of the set of all reachable states.

## Representing and computing with sets of values

Requirements on the representation :

1. For a represented set $S$, and all transitions $r$, $r(S)$ must be computable and representable.

2. For a represented set $S$, and some transitions $r$, $r^*(S)$ must be computable and representable.

3. To decide termination, inclusion $S \subseteq S'$ must be decidable.

4. Sufficient conditions for 1 and 3: Representable sets are algorithmically closed under intersection, projection (existential quantification) and complementation; emptiness is decidable.

## Representing sets of integers

First idea : linear constrained sets.

- Yes, but iterating a simple operation like $x := x + 3$ yields sets which are periodic unions of linear constrained sets
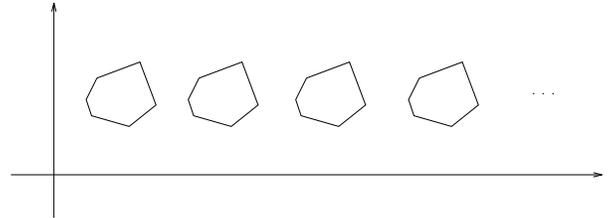


- One needs means to represent periodicity !

## Representing sets of integers II

- Use a logical formalism, e.g. Presburger Arithmetic (first-order arithmetic without multiplication).

$$\exists k\ x_0 (x = x_0 + 5k \wedge 1 \le x_0 \le 3 \\ \wedge 2 \le y \le 4)$$

  - Expressiveness is sufficient,

  - The problem is computing with such a logical representation.

- Alternative : use automata to represent sets of integers.

## Representing set of integers
## Encoding numbers

- Binary representation,

- Unbounded numbers,

- Most significant bit first.

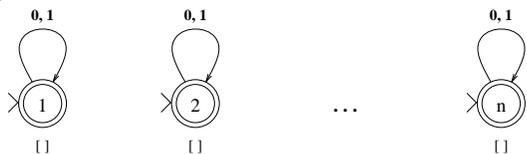- 2's complement for negative numbers (at least $p$ pits for a number $x$ such that $-2^{p-1} \le x < 2^{p-1}$).

Examples :

$$4 \ : \ 0100, 00100, 000100, \ldots$$
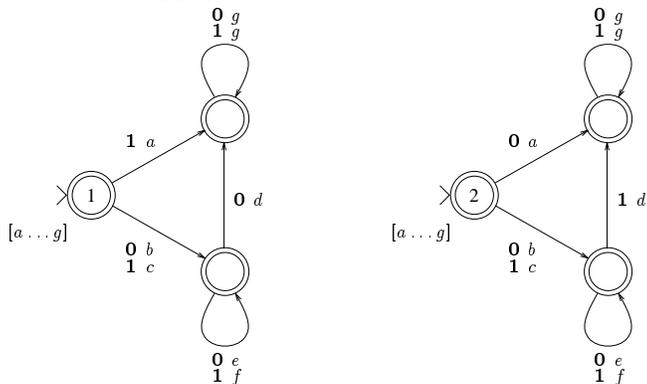$$-4 \ : \ 100, 1100, 11100, \ldots$$

The encodings of any Presburger definable set of integer vectors are accepted by finite automata and computing with sets of integers is reduced to computing with finite automata.
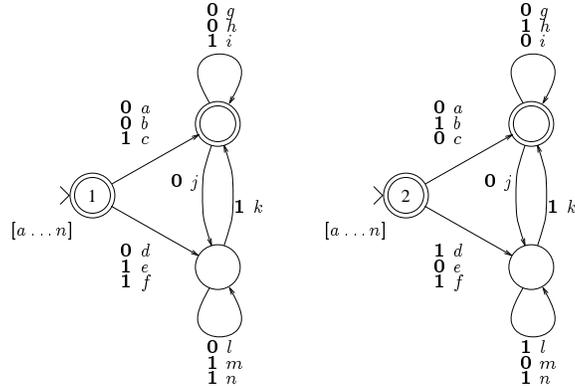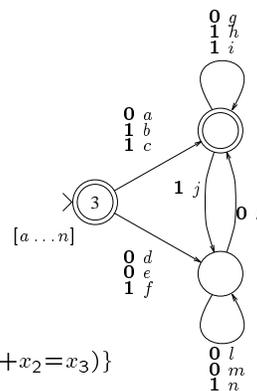
# Automata for Elementary predicates

$A_{\mathbf{Z}^n}$

**0, 1**     **0, 1**     **0, 1**

(1)    (2)    ...    (n)

[]     []     []

$A_{\{(x_1,x_2)\,|\,x_1\le x_2\}}$

$\begin{matrix}0 & g\\1 & g\end{matrix}$

**1** $a$    **0** $d$

$[a \ldots g]$   (1)   $\begin{matrix}0 & b\\1 & c\end{matrix}$   $\begin{matrix}0 & e\\1 & f\end{matrix}$

$\begin{matrix}0 & g\\1 & g\end{matrix}$

**0** $a$    **1** $d$

$[a \ldots g]$   (2)   $\begin{matrix}0 & b\\1 & c\end{matrix}$   $\begin{matrix}0 & e\\1 & f\end{matrix}$

17

---

$\begin{matrix}0 & g\\0 & h\\1 & i\end{matrix}$

$\begin{matrix}0 & a\\0 & b\\1 & c\end{matrix}$   **0** $j$   **1** $k$

$[a \ldots n]$   (1)   $\begin{matrix}0 & d\\1 & e\\1 & f\end{matrix}$   $\begin{matrix}0 & l\\1 & m\\1 & n\end{matrix}$

$\begin{matrix}0 & g\\1 & h\\0 & i\end{matrix}$

$\begin{matrix}0 & a\\1 & b\\0 & c\end{matrix}$   **0** $j$   **1** $k$

$[a \ldots n]$   (2)   $\begin{matrix}1 & d\\0 & e\\1 & f\end{matrix}$   $\begin{matrix}1 & l\\0 & m\\1 & n\end{matrix}$

$\begin{matrix}0 & g\\1 & h\\1 & i\end{matrix}$

$\begin{matrix}0 & a\\1 & b\\1 & c\end{matrix}$   **1** $j$   **0** $k$

$[a \ldots n]$   (3)   $\begin{matrix}0 & d\\0 & e\\1 & f\end{matrix}$   $\begin{matrix}0 & l\\0 & m\\1 & n\end{matrix}$

$A_{\{(x_1,x_2,x_3)\,|\,x_1+x_2=x_3\}}$

18

---

# Iterating operations on integers

A simple case.

- An instruction $I \equiv T\vec{x} \le \vec{u} \to \vec{x} := A\vec{x} + \vec{b}$,
  with $A$ idempotent ($A^2 = A$)

- An initial value $\vec{x}_0$

Compute the values obtained by the
repeated execution of $I$ on $\vec{x}_0$ :

$$
\begin{aligned}
A\vec{x}_0 &+ & &+ \vec{b}\\
A\vec{x}_0 &+ &A\vec{b} &+ \vec{b}\\
A\vec{x}_0 &+ &2A\vec{b} &+ \vec{b}\\
&\vdots & \vdots &\\
A\vec{x}_0 &+ &kA\vec{b} &+ \vec{b}\\
&\vdots & \vdots &
\end{aligned}
$$

Cycle precondition : $T(A\vec{x}_0 + kA\vec{b} + \vec{b}) \le \vec{u}$

More general results by B. Boigelot.

19

---

# Communicating with message queues
## Representing queue contents

- Given a message alphabet $\Sigma$, the
  content of a queue is a word over $\Sigma$.

- Sets of queue contents can be
  represented by languages (sets of words)
  accepted by finite automata.

- If there are several queues, one uses the
  concatenation of their content.

This representation is the QDD of Boigelot
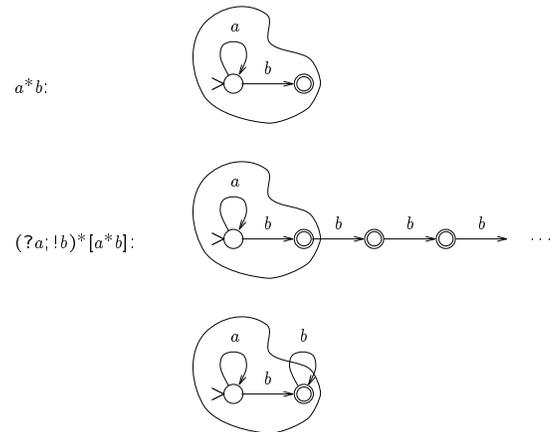and Godefroid.

20

## Iterating sequences of operations on queues

- Consider send $!a$ and receive $?a$ operations, and finite sequences $\sigma$ of such operations..

- For a language of queue contents $L$, let $\sigma(L) = \{\sigma(w) \mid w \in L\}$.

- The problem is to compute $\sigma^*(L) = \bigcup_{k \geq 0} \sigma^k(L)$ when $L$ is regular.

- For a single queue, this is always possible and the result is always regular.

## Computing $\sigma^*(L)$
## An example

$a^*b$:



$(?a;!b)^*[a^*b]$:

## Sequences of operations on multiple queues

- Iterating operations can move one beyond sets of content recognizable by finite automata $(q_1!a, q_2!b)^*$

- It is nevertheless still possible to compute the result of iterating operations when the result is recognizable by a finite automaton.
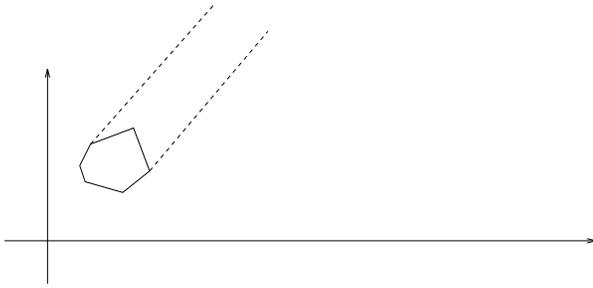
## Unifying
## Real-Time and Hybrid Systems

A similar analysis technique can be used.

- Hybrid systems have states that are composed of a control part and real variables.

- The passing of time is the type of transition that makes the jump from finite to infinite sets of states. They are similar to the iterated transitions of a loop-first search.

- For the analyzable cases, the possible values of the real variables are linearly constrained sets of reals. These are representable by automata operating on infinite words.

- One can combine iterated transitions and time transitions [BBR97].

## Time passing transitions
## Example

## Unifying
## Pushdown systems

- Finite-state systems with one pushdown stack have a regular set of reachable states.

- This set is easily computable.

- From this computation model-checking algorithms can be obtained.

## The reachable states of
## pushdown systems

A *pushdown automaton* $A$ is a quadruple

$$A = (Q, \Gamma, \Delta, q^0)$$

where

- $Q$ is a finite set of control states,

- $\Gamma$ is a stack alphabet,

- $\Delta \subset$
  $\{(Q \times \Gamma) \times (Q \times \{\epsilon\})\} \cup \{(Q \times \{\epsilon\}) \times (Q \times \Gamma)\}$
  is a transition relation ($\epsilon$ denotes the empty word),

- $q^0$ is the initial control state.

Given a pushdown automaton
$A = (Q, \Gamma, \Delta, q_0)$ the reachable subset of
$Q \times \Gamma^*$ is characterized by
$A_r = (Q_r, \Sigma_r, \Delta_r, q_r^0, F_r)$ where

- the set of states $Q_r$ is $Q$,

- $\Sigma_r = \Gamma$,

- $\Delta_r \subseteq Q_r \times (\Sigma_r \cup \{\varepsilon\}) \times Q_r$ is the smallest relation such that

  - if $(q, a_+, q') \in \Delta$, then $(q, a, q') \in \Delta_r$ and,

  - if $(q, a_-, q') \in \Delta$ and $(q'', a, q) \in \Delta_r^*$, then $(q'', \varepsilon, q') \in \Delta_r$,

- $q_r^0 = q^0$,

- $F_r = Q_r$.

## The unifying ideas

- Do a search of the state space using special transitions that can generate an infinite set of reachable states in one step.

- Use a suitable representation of sets of values. In many cases this can be taken to be a representation of regular sets.

- Different representations can be used for different types of data, but they can be combined.

- Enumerative exploration is still present as a fall-back.

## Termination

- The search terminates when no new states can be generated.

- This is determined by checking that all new states that can be generated are in the set already obtained.

- A guarantee of termination implies decidability, *but is practically quite irrelevant*.

## Conclusions

- Traditional finite-state analysis can be naturally extended to handle infinite state-spaces.

- Infinite state spaces that can be easily analyzed are those that have finite-state sets of encodings.

- The challenge is to make all this work well in practice.