Abstract Regular Model Checking and application to programs over lists

with Ahmed BOUAJJANI, Pierre MORO, Tomas VOJNAR (TU Brno)

Peter Habermehl
LIAFA, University Paris 7

May 23rd, 2005

Introduction

- Regular model checking
- Modeling programs with 1-selector-linked structures
  - lists, circular lists, lists with sharing
- Applying abstract regular model checking
- Experimental results
- Conclusion and perspectives

Regular model checking

[Pnueli & al. 97], [Fribourg & al. 97], [Wolper, Boigelot, 98], [Bouajjani & al. 00]

- Configurations of systems are modeled as words over a finite alphabet Σ.
- Finite automata A over Σ represent (infinite) regular sets of words (configurations)
  - $Init$: set of initial configurations
  - Bad: set of bad configurations
- Transitions are modeled by a transducer τ (automata over $\Sigma \times \Sigma$).

Regular model checking

- A lot of infinite-state systems can be encoded in this way:
  - (Lossy) FIFO-channel systems, Push-down automata
  - Counter machines
  - Parameterised systems (parameterised number of identical components)
  - Programs over lists (with sharing)
  - Using trees: More general systems
- Given $Init$ and τ
- Reachable configurations in n steps: $\tau^n(Init)$
- $\tau^*(Init) := \bigcup_{k=0}^{\infty} \tau^k(Init)$ (not necessarily regular)

Regular model checking

- Basic model-checking problem : $\tau^*(Init) \cap Bad = \emptyset$?
- Several approaches exist
  [Abdulla, Boigelot, Bouajjani, Jonsson, Legay, Nilsson, d'Orso, Pnueli, Touili, Wolper,...]
- Calculating exact reachability sets or relations
  - Special classes of systems or transitions where τ^* can be calculated
  - General methods:
    * quotienting of iterated transducers
    * extrapolation of automata
- Calculating overapproximations of $\tau^*(Init)$ (invariants of τ)
  - Inference of regular languages [H., Vojnar 04]
  - Abstract regular model-checking [Bouajjani, H., Vojnar 04]
    * Applying the abstract-check-refine paradigm
    * Abstraction of automata representing configurations

Applying the abstract regular model-checking framework to programs with 1-selector-linked structures

- Encoding of configurations (stores) as words
  - similar to encoding of PALE [Jensen, Jorgensen, Klarlund, Schwartzbach 97]
- Encoding of program statements as transducers
- Adapting the existing abstraction schemata
- Advantages
  - Automatic verification
  - Automatic loop invariant generation

From programs to transducers

Example C-like program – reversing a list

- reverses a list pointed to by l
- Data is abstracted (finite domains can be handled)

$List\ x, y, l;$
1: $x = null;$
2: while $(l\ != null)\ \{$
3: $y = l \rightarrow next;$
4: $l \rightarrow next = x;$
5: $x = l;$
6: $l = y;\}$
7: $l = x;$
8: // end of program

Example

1: $x = null;$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

$1 \quad | \quad xy \quad | \quad \quad | \quad l \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot \quad |$

Example

1: $x = null;$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

1	xy		$l \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$	
2	y	x	$l \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$	

Example

1: $x = null$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

1	xy		$l \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$
2	y	x	$l \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$
3	y	x	$l \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$
4		x	$l \rightarrow y \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$
5		x	$l \rightarrow \bot \mid y \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$
6			$xl \rightarrow \bot \mid y \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$
2			$x \rightarrow \bot \mid ly \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$
			$etc.$

Example

1: $x = null$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

3			$x \rightarrow\rightarrow\rightarrow \bot$	$ly \rightarrow\rightarrow\rightarrow \bot$

Example

1: $x = null$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

3				$x \rightarrow\rightarrow\rightarrow \bot$	$ly \rightarrow\rightarrow\rightarrow \bot$
4				$x \rightarrow\rightarrow\rightarrow \bot$	$l \rightarrow y \rightarrow\rightarrow \bot$

Example

1: $x = null$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

3			$x \rightarrow\rightarrow\rightarrow \bot$	$ly \rightarrow\rightarrow\rightarrow \bot$	
4			$x \rightarrow\rightarrow\rightarrow \bot$	$l\rightarrow y \rightarrow\rightarrow \bot$	

Example

1: $x = null$
2: while $(l \mathrel{!}= null)$ $\{$
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y; \}$
7: $l = x;$

3				$x \rightarrow\rightarrow\rightarrow \bot \mid ly \rightarrow\rightarrow\rightarrow \bot$
4				$x \rightarrow\rightarrow\rightarrow \bot \mid l\rightarrow y \rightarrow\rightarrow \bot$
5				$xm_t \rightarrow\rightarrow\rightarrow \bot \mid l\rightarrow m_f \mid y \rightarrow\rightarrow \bot$

Example

1: $x = null$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

3			$x \rightarrow\rightarrow\rightarrow \bot$	$ly \rightarrow\rightarrow\rightarrow \bot$
4			$x \rightarrow\rightarrow\rightarrow \bot$	$l\rightarrow y \rightarrow\rightarrow \bot$
5			$xm_t \rightarrow\rightarrow\rightarrow \bot$	$l\rightarrow m_f \mid y \rightarrow\rightarrow \bot$
5			$l \rightarrow x \rightarrow\rightarrow\rightarrow \bot$	$y \rightarrow\rightarrow \bot$

Example

1: $x = null$
2: while $(l \mathrel{!}= null)$ {
3: $\quad y = l \rightarrow next;$
4: $\quad l \rightarrow next = x;$
5: $\quad x = l;$
6: $\quad l = y;\}$
7: $l = x;$

3				$x \rightarrow\rightarrow\rightarrow \bot \mid ly \rightarrow\rightarrow\rightarrow \bot$
4				$x \rightarrow\rightarrow\rightarrow \bot \mid l \rightarrow y \rightarrow\rightarrow \bot$
5				$xm_t \rightarrow\rightarrow\rightarrow \bot \mid l \rightarrow m_f \mid y \rightarrow\rightarrow \bot$
5				$l \rightarrow x \rightarrow\rightarrow\rightarrow \bot \mid y \rightarrow\rightarrow \bot$
				$etc.$
8		y		$xl \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow \bot$

Sets of configurations

- The set of initial configurations

$$Init = (1 \mid xy \mid \mid l \rightarrow\rightarrow^* \bot \mid)$$

- The set of reachable configurations at line 8 is given by

$$\tau^*(Init) = (8 \mid \mid y \mid xl \rightarrow\rightarrow^* \bot \mid)$$

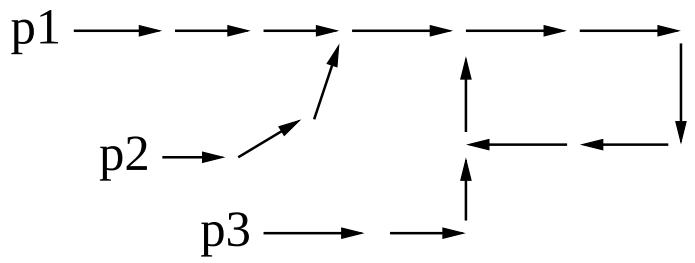
- A loop invariant

$$(2 \mid y \mid x \mid l \rightarrow\rightarrow^* \bot \mid) + (2 \mid \mid yl \mid x \rightarrow\rightarrow^* \bot \mid) + (2 \mid \mid \mid x \rightarrow\rightarrow^* \bot \mid ly \rightarrow\rightarrow^* \bot \mid)$$

Properties checked

- Basic properties
  - No garbage is created
  - No null pointer dereference
  - The result is a list
- more complex properties
  - The list is really reversed
  - $l \rightarrow^* fst \rightarrow snd \rightarrow^* \rightarrow \bot$ leads to $l \rightarrow^* snd \rightarrow fst \rightarrow^* \rightarrow \bot$

Using markers for shared (or circular) lists



- The store can contain shared parts and cycles
- Property: a store with k pointer variables without garbage can be encoded with at most k pairs of markers.
  here : $p1 \rightarrow\rightarrow\rightarrow n_t \rightarrow m_t \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow h_t \rightarrow m_f \mid p2 \rightarrow\rightarrow\rightarrow n_f \mid p3 \rightarrow\rightarrow\rightarrow h_f$
- We fix a number of markers : A configuration contains the set of unused markers
- Available pairs of markers are used "on demand" by the transducer
- If no pair of markers is available, a pair is eliminated if possible

Elimination of a pair of markers

Example
$| \; y \; m_t \longrightarrow \cdots \longrightarrow \bot \; | \; x \longrightarrow \cdots \longrightarrow m_f \; |$ is changed to
$| \; x \longrightarrow \cdots \longrightarrow y \longrightarrow \cdots \longrightarrow \bot$

- Involves shifting of parts words of arbitrary size
- Non-regular relation: can not be described as the application of one transducer
- But shifting of one symbol can be done with a transducer
- Transitive closure of this transducer on an input set realizes the shifting.

Encoding program statements as transducers

Automatic translation from program statements to transducers

- $free$, new
- Tests $x == y$ and $x == null$
- Assignments $x = null$ and $x = y$
- Assignment $y = x \rightarrow next$
- Assignment $x \rightarrow next = y$
  - realized using a new pair of markers if available
  - If no pair of markers is available, a pair is eliminated (if possible)

Abstract regular model-checking [04]

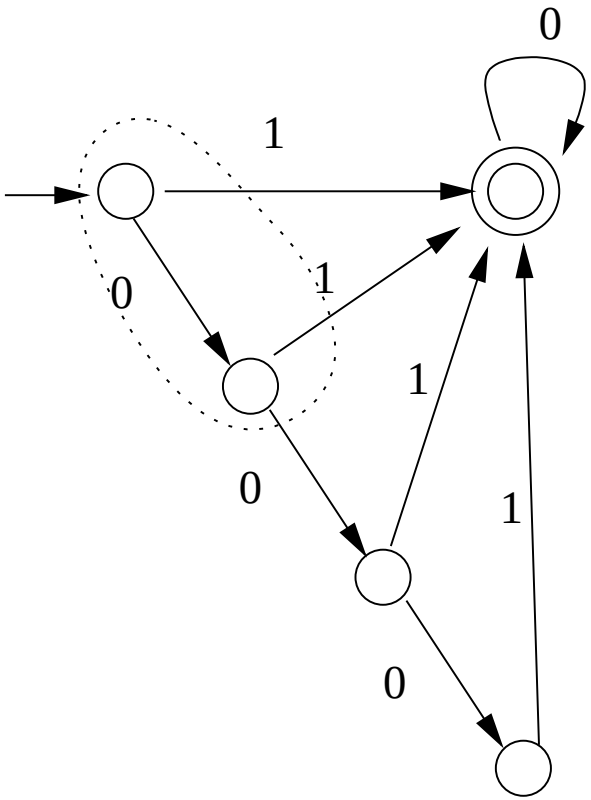
- $\tau^*(Init) \cap Bad = \emptyset$?
- Define a finite-range abstraction function α on automata
- Compute iteratively $(\alpha \circ \tau)^*(Init)$,
- If $(\alpha \circ \tau)^*(Init) \cap Bad = \emptyset$ then answer YES
- Otherwise, let θ be the computed symbolic path from $Init$ to Bad,
- Check if θ includes a concrete counterexample,
  - If yes, then answer NO,
  - Otherwise, define a refinement of α which excludes θ, and redo computation

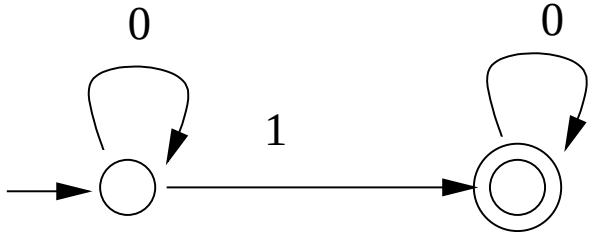
Abstractions

- In earlier work [CAV 04] we have defined representation-oriented abstractions
  - define a (finite index) equivalence relation on states of automata representing configurations
    - Equivalence based on languages of words up to finite length
    - Equivalence based on same status wrt "predicate" automata
  - collapse equivalent states

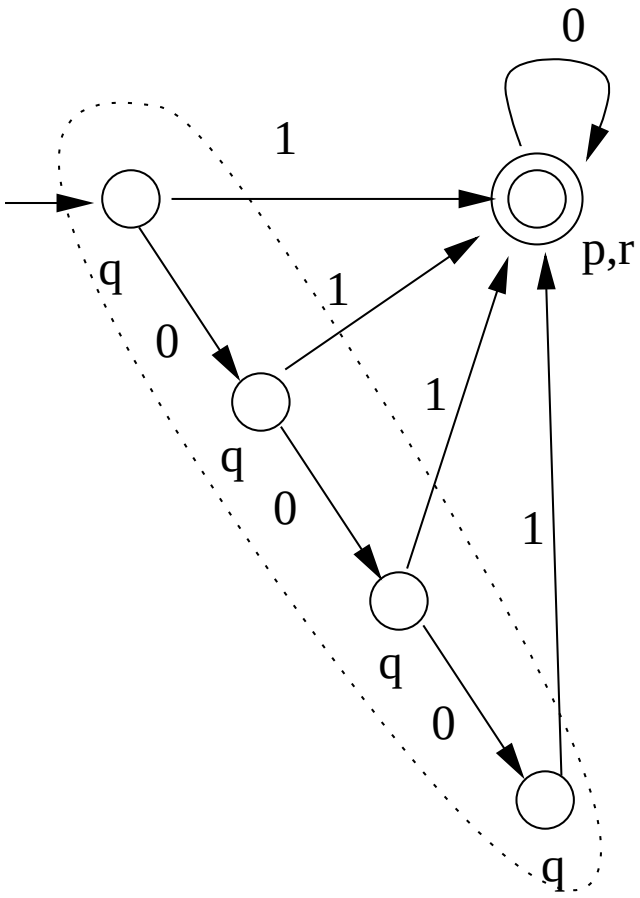
Example finite-length abstraction

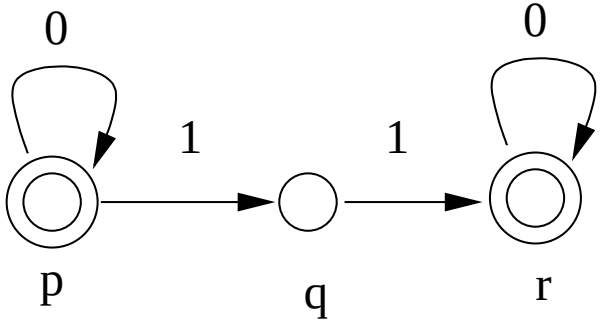
with words up to length 3

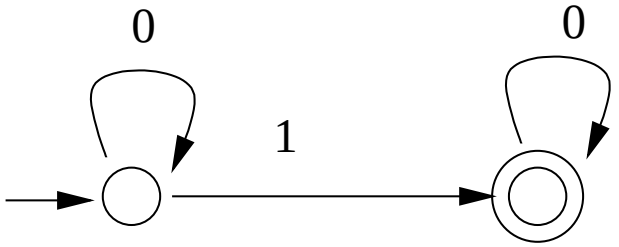




Example predicate abstraction







New abstractions

- Here, we propose configuration-oriented abstractions
  - defined on configurations
  - piecewise 0-k counter abstractions
  - Closure abstractions

Piecewise 0-k counter abstractions

- Let S be the set of *strong* symbols: they have a bounded number of occurrences in every word (e.g., separators, markers, pointer variables).

- Given a word w, consider its decomposition according to strong symbols:

$$w = w_0 s_1 w_1 s_2 w_2 \cdots s_n w_n$$

  where $s_i \in S$ and $w_i \in (\Sigma \setminus S)^*$.

- For $k \in \mathbb{N}^{>0}$, define the relation α_k such that, for every word w:

$$\alpha_k(w) = L_0 s_1 L_1 s_2 L_2 \cdots s_n L_n$$

  where $L_i = \{u \in (\Sigma \setminus S)^* \ : \ \forall a \in \Sigma \setminus S. |w_i|_a < k$ and $|u|_a = |w_i|_a$, or $|w_i|_a \geq k$ and $|u|_a \geq k\}$

- The relation α_k is regular (definable by a finite transducer) and finite range.

Closure abstractions

- Given $u \in \Sigma^+$, and $k \in \mathbb{N}^{>0}$, define an extrapolation relation:

$$R_{(u,k)} = \{(w, w') \ : \ w = u_1 u^k u_2 \text{ and } w' = u_1 u^k u^* u_2\}$$

- R an extrapolation relation and L a regular set $\Rightarrow R^*(L)$ is regular.

- An extrapolation system is a finite union of extrapolation relations.
  - Let $\prec \subseteq \ \Sigma^* \times \Sigma^*$ such that:
    - u is not a factor of v (i.e., u does not appear as a subword of v),
    - u cannot be written as $w_1 v^p w_2$ for any $p \in \mathbb{N}$, and two words w_1, w_2 such that w_1 is a suffix of v and w_2 is a prefix of v.
  - $\forall u, v \in \Sigma^*$, if $u \prec v$ then $\forall p \geq 0. \forall w_1, w_2 \in \Sigma^*. v^p \neq w_1 u w_2$.
  - Let $R = R_{(u_1,k_1)} \cup \cdots \cup R_{(u_n,k_n)}$. R is serialisable if the reflexive-transitive closure of $\prec$ is a partial ordering on the set $\{u_1^{k_1}, \ldots, u_n^{k_n}\}$.
  - $\rightsquigarrow$ Let $i_1, i_2, \ldots, i_n$ be a total ordering compatible with $\prec$. Then $R^* = R^*_{(u_{i_n},k_{i_n})} \circ \cdots \circ R^*_{(u_{i_1},k_{i_1})}$.

Experimental results

using new abstractions (up to 100 times faster)

Program	Markers	$\|M\|^{max}_{st.+tr.}$	T_{sec}
Reverse, basic consistency	0	51+105	0.3
Reverse, full	0	281+369	4.2
Faulty reverse	1	61+138	0.2
Insert, bas. cons.	0	81+102	0.5
Insert, bas. cons.	2	165+577	0.15
Insert, full	0	755+936	10.8
Delete, bas. cons.	0	55+113	0.3
Merge, bas. cons.	0	209+279	2.7
Merge, corr.mix.	0	1080+1415	40.4
Bubblesort, bas. cons.	2	2095+2872	305
Bubblesort, full	2	2339+2887	279
Circular list reverse, bas. cons.	3	655+764	5.4
Circular list reverse, full	3	2349+2822	50.6
Circular list rem.seg., bas. cons.	2	116+291	1.0

Conclusion and perspectives

- We have applied and adapted the abstract regular model-checking framework to programs over lists
  - Encoding
  - New abstractions based on configurations
- Current and future work:
  - Extension to trees done recently (Adam Rogalewicz)
  - Specialised treatment for counter automata with integers (using NDDs) and reals (using RVA)
  - Handling more general classes of dynamic data structures (using trees)
  - Combining with other sources of infinity (recursion, multithreading, unbounded data domains)