Software Verification Using Counter Automata

DCS Days, March 27th, 2009

Filip Konečný

konecny@imag.fr, ikonecny@fit.vutbr.cz

Cotutelle Ph.D.

Université Joseph Fourier

- supervisor – Radu IOSIF

Brno University of Technology

- Faculty of Information Technology
- VeriFIT – Formal Verification Research Group
- supervisor – Tomáš VOJNAR

Subject of the Thesis

title

- Symbolic Verification of Programs Using Counter Automata and Related Formalisms

aims

- verification of infinite-state systems – arrays, pointers, etc.

means

- formalisms to model systems – counter automata, etc.

Counter Automata

- verification of programs with lists

  Programs with Lists are Counter Automata, CAV'06
  A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar

- verification of generic hardware designs specified in VHDL

  Verifying Parametrised Hardware Designs Via Counter Automata
  A. Smrčka, T. Vojnar

- verification of multithreaded programs

  Towards the Automated Verification of Multithreaded Java Programs
  G. Delzanno, J.F. Raskin, L. Begin

Verification of Integer Array Programs

Automatic Verification of Integer Array Programs
M. Bozga, P. Habermehl, R. Iosif, F. Konečný, T. Vojnar
CAV'09

programs under consideration

- integer arrays, integer variables
- assignments, conditional statements, non-nested while loops

specification of pre-/post-conditions

- SIL (Single Index Logic) – to expresses array properties

verification task

- is the program's behaviour in accordance with the specified post-condition?
- approach:
  - infer post-condition w.r.t. the program and the user pre-condition
  - does the inferred post-condition entails the user-specified post-condition?

SIL – Single Index Logic

examples

- $\forall i.(0 \leq i \leq b) \Rightarrow (a[i] = i + 1)$
- $\forall i.(0 \leq i \leq b - 1) \Rightarrow (a[i] \leq a[i+1])$

SIL

- interpreted over integer arrays and integer variables
- comparisons of array ellements within a constant sized window
- comparisons use a single index variable (thus single index logic)
- decidable logic (modular translation to a CA where reachability is decidable)

Example: Array Rotation

Input: array a_1, parameter b_1 (number of used cells in a_1)
Data: array a_2 keeps a copy of a_1, scalar variable b_2 (keeps first element of a_1
Output: array a_1 is rotated by one to the left

$/*$ **pre-condition:** $(b_1 > 0) \;\wedge$ $*/$
$/*$ $\forall i.(0 \leq i \leq b_1 - 1) \Rightarrow (a_1[i] = a_2[i])$ $*/$

$b_2 := a_1[0];$
while $_{a:i=0}(i \leq b_1 - 2)$ **do**
  $a_1[i] := a_1[i+1];$
  $i++;$
end
$a_1[b_1 - 1] := b_2;$

$/*$ **post-condition:** $\forall i.(0 \leq i \leq b_1 - 2) \Rightarrow (a_1[i] = a_2[i+1]) \;\wedge$
$\forall i.(b_1 - 1 \leq i \leq b_1 - 1) \Rightarrow (a_1[i] = b_2)$ $*/$

Verification Framework

```
pre-condition          φ
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition         ψ
```

Verification Framework

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition
```

φ

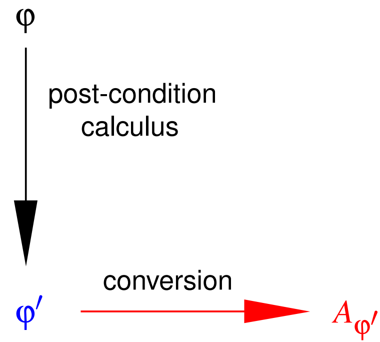
post-condition calculus

φ'

ψ

Verification Framework

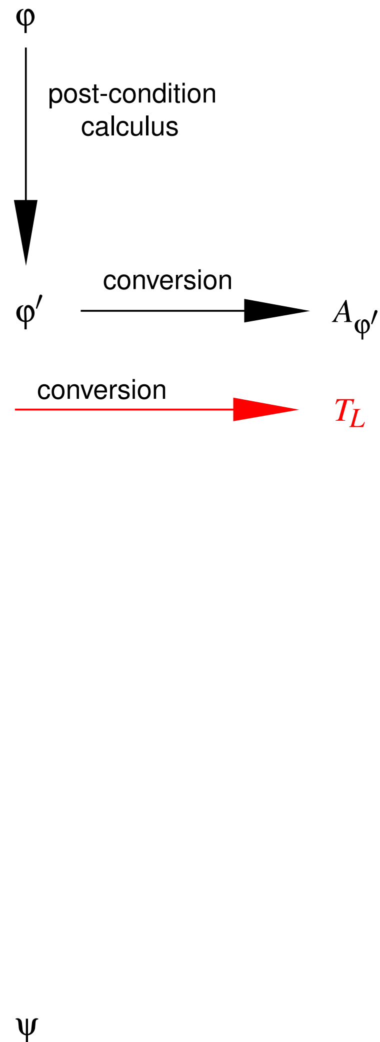
```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition
```



ψ

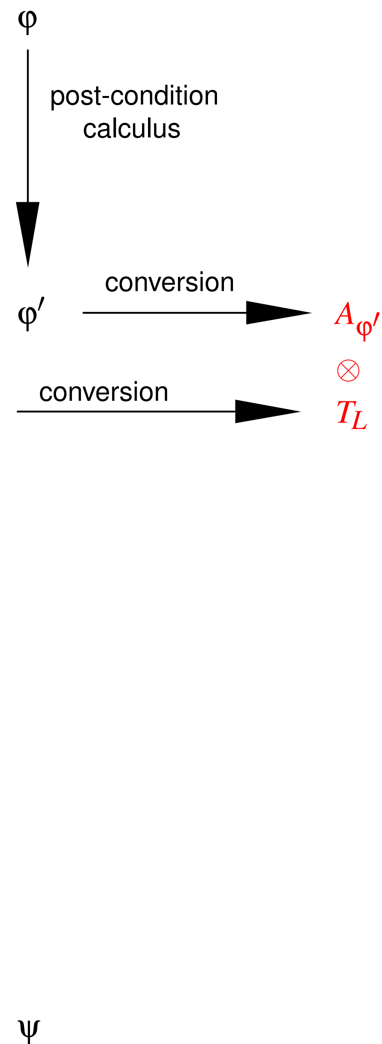
Verification Framework

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition
```



Verification Framework

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition
```



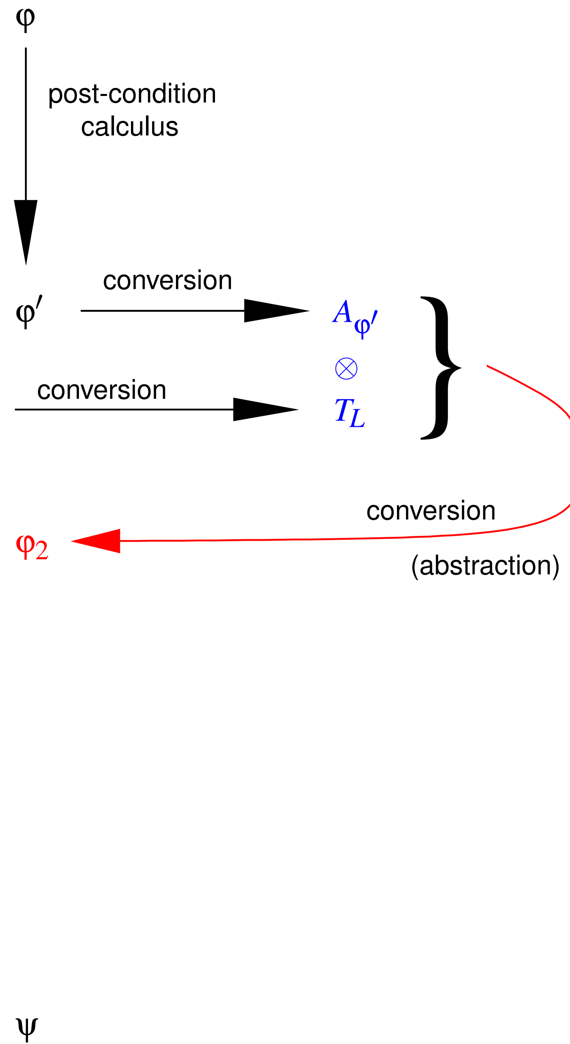
$A_{\varphi'}$

$\otimes$

T_L

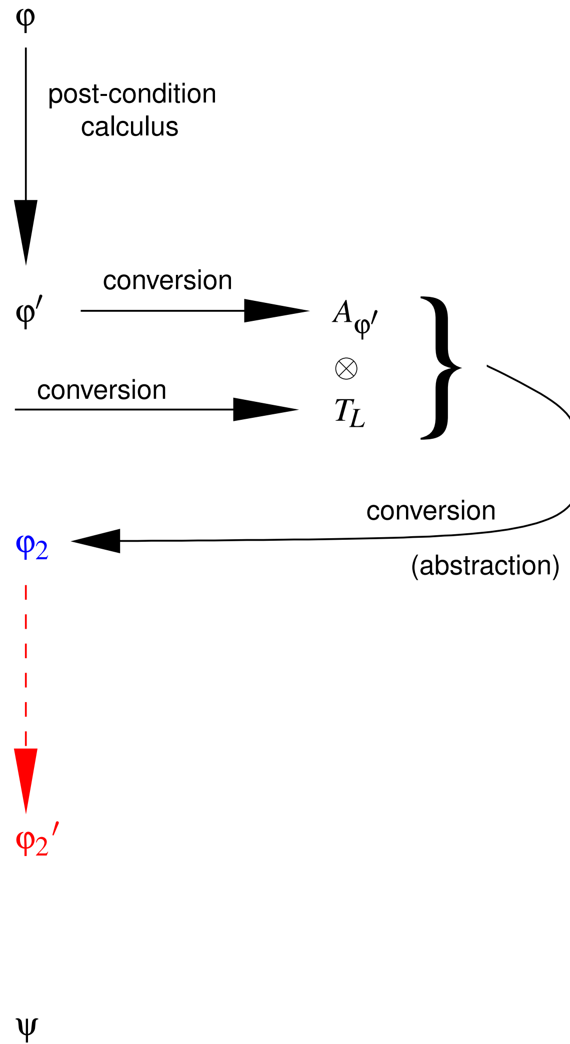
Verification Framework

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition
```



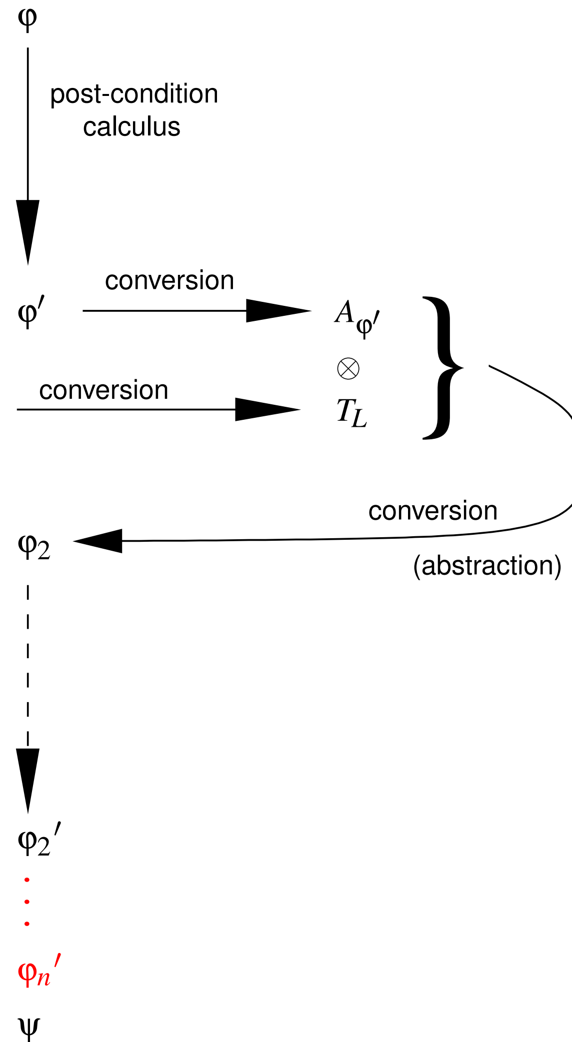
Verification Framework

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition
```



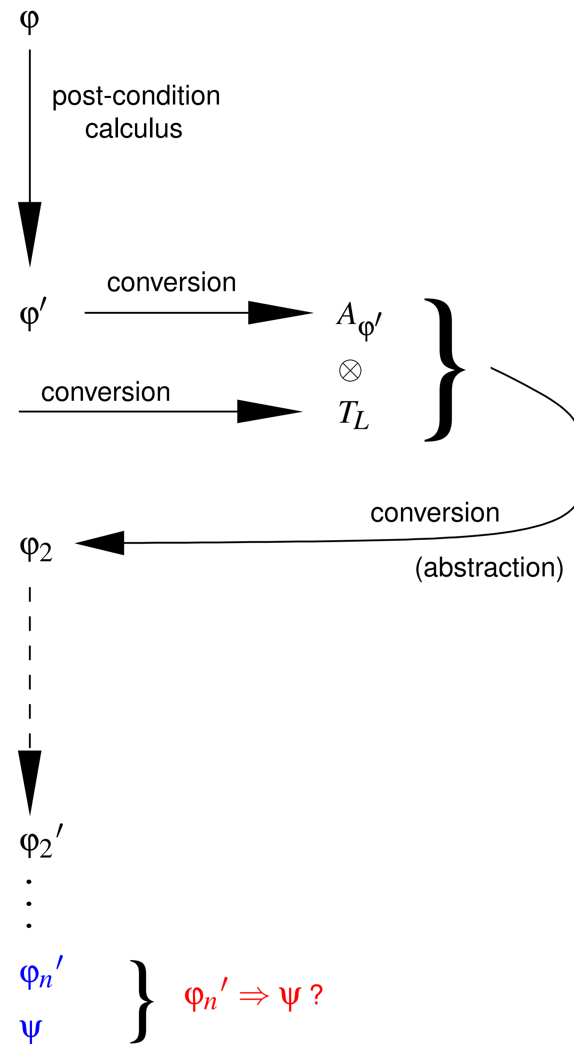
Verification Framework

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
 .
 .
 .

post-condition
```



Verification Framework

```
pre-condition
non-loop code {
   ...
   if() ...
   else if() ...
   else ...
   ...
}
loop {
   if() ...
   else if() ...
      ...
   else ...
}
non-loop code {
   ...
   if() ...
   else if() ...
   else ...
   ...
}
 .
 .
 .

post-condition
```



Encoding of Arrays in Counter Automata

- accepting runs of a CA correspond to models of a SIL formula expressing a certain array property
- array variable $\rightarrow$ 2 counters:
  - value counter
  - index counter
- e.g. array variable $a \rightarrow$ value counter x and index counter i
  - the connection of x and i – if value of i is v, then value of x is $a[v]$
  - force i to range between 0 and $|a|$ (i is 0 initially, then it either keeps its value or increases by 1 until the value reaches $|a|$)
  - an example of a run corresponding to array $a = [10, 12, 14]$:

i	0	0	0	1	2	2
x	10	10	10	12	14	14

- *state-consistent* automaton – runs respect the above restrictions
- one array can be encoded by many runs

Encoding of Arrays in Counter Transducers

- array variable $\rightarrow$ 3 counters:
  - input-value counter
  - output-value counter
  - index counter
- example of a run ($a = [10, 12, 14] \rightarrow a' = [11, 13, 15]$):

i	0	0	0	1	2	2	3	3
x^i	10	10	10	12	14	14	?	?
x^o	?	?	?	11	13	13	15	15

- *transition-consistent* transducer

States Recognized by Couter Automata

Notation

- $\vec{a} = \{a_1, a_2, \ldots, a_k\}$ – *array variables*
- $\vec{b} = \{b_1, b_2, \ldots, b_m\}$ – *scalar variables*
- $\langle \alpha, \iota \rangle$ – a state
  - $\alpha : \vec{a} \rightarrow \mathbb{Z}^*$
  - $\iota : \vec{b} \rightarrow \mathbb{Z}$
- $\Sigma(A)$ – the set of program states accepted by automaton A
- $\Theta(T)$ – the set of pairs program states accepted by transducer T

Post-image of $\Theta(T)$ over $\Sigma(A)$

- agreement of T and A on representation of states
- assurance of composability – *state-completeness*
- SIL $\rightarrow$ state-complete CA

Programs with a Single Loop

- φ – loop pre-condition
- ψ – loop post-condition
- $\Sigma(A_\varphi)$ – program states at the entry point of a loop
- $\Theta(T_L)$ – input-output pairs of states accepted by loop transducer
- $\Theta(T_L)(\Sigma(A_\varphi))$ – program states at the exit point of a loop
- entailment checking
  - $\Theta(T_L)(\Sigma(A_\varphi)) \subseteq \Sigma(A_\psi)$
  - $\Theta(T_L) \cap (\Sigma(A_\varphi) \times \Sigma(A_{\neg\psi}) = \emptyset$
- entailment checking using automata
  - $\Sigma(T_L \otimes A_\varphi \otimes A_{\neg\psi}) = \emptyset$?
  - $T_L \otimes A_\varphi \otimes A_{\neg\psi}$ – is a final state reachable?

Verification Framework – One Loop Case

```
pre-condition          φ
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
post-condition         ψ
```

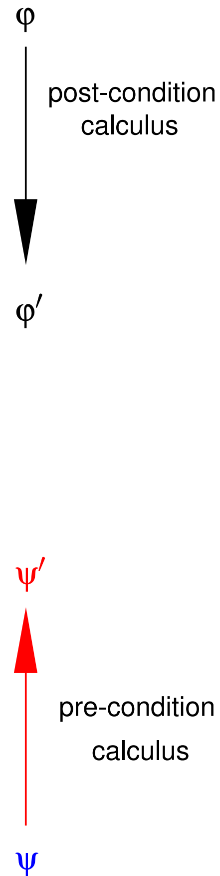
Verification Framework – One Loop Case

```
pre-condition                 φ
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}                             φ′
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
post-condition                ψ
```

post-condition calculus

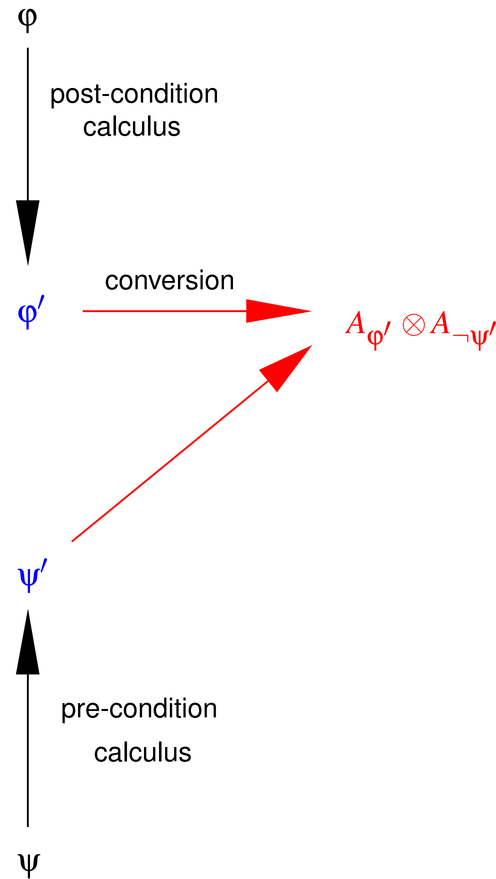
Verification Framework – One Loop Case

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
post-condition
```



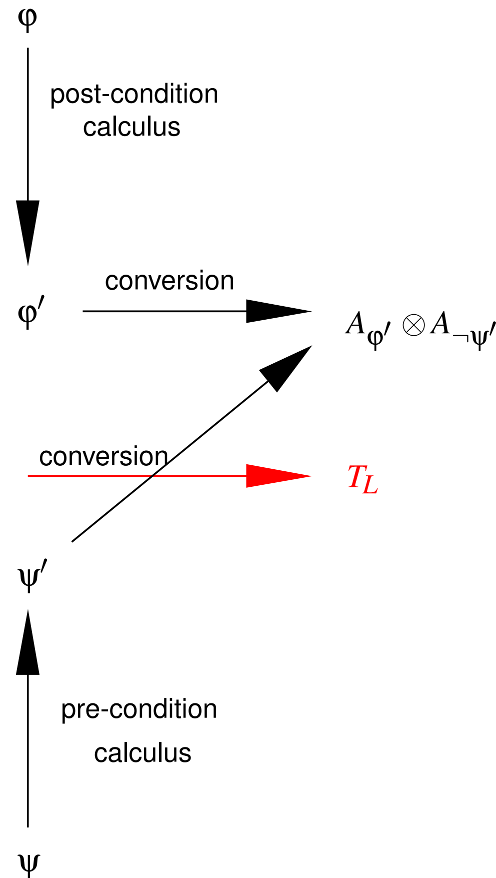
Verification Framework – One Loop Case

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
post-condition
```



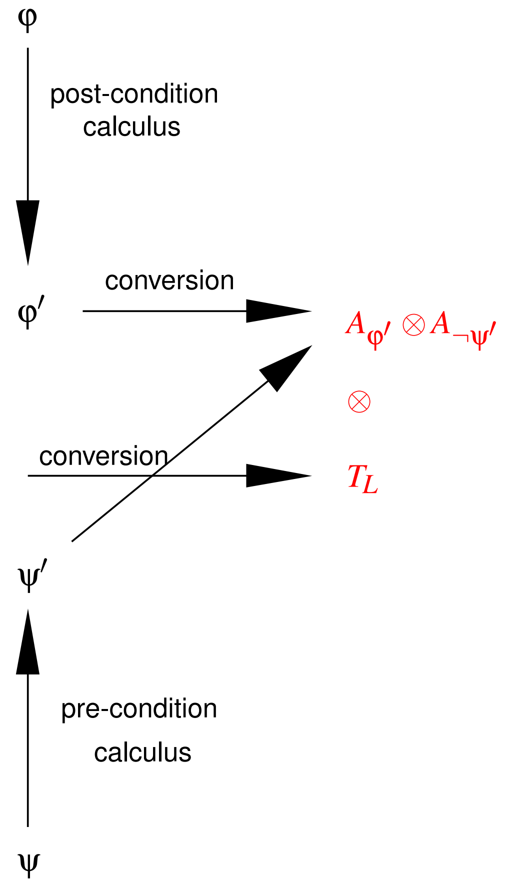
Verification Framework – One Loop Case

```
pre-condition
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
loop {
  if() ...
  else if() ...
    ...
  else ...
}
non-loop code {
  ...
  if() ...
  else if() ...
  else ...
  ...
}
post-condition
```



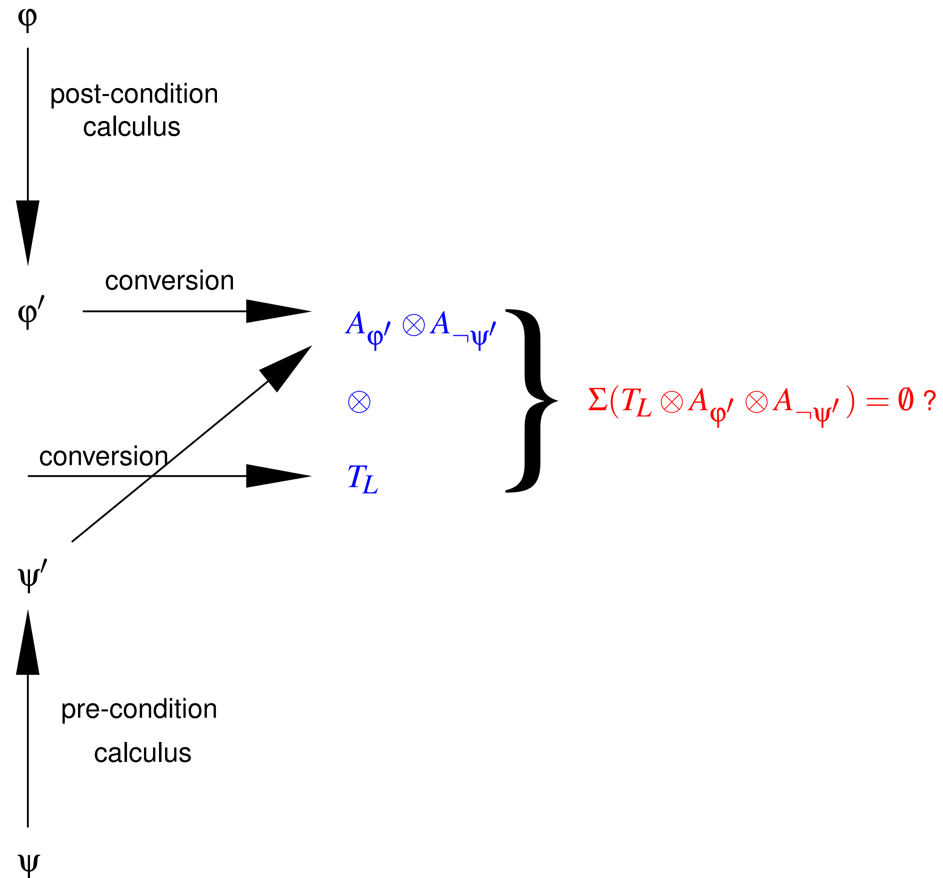
Verification Framework – One Loop Case

```
pre-condition
non-loop code {
   ...
   if() ...
   else if() ...
   else ...
   ...
}
loop {
   if() ...
   else if() ...
     ...
   else ...
}
non-loop code {
   ...
   if() ...
   else if() ...
   else ...
   ...
}
post-condition
```

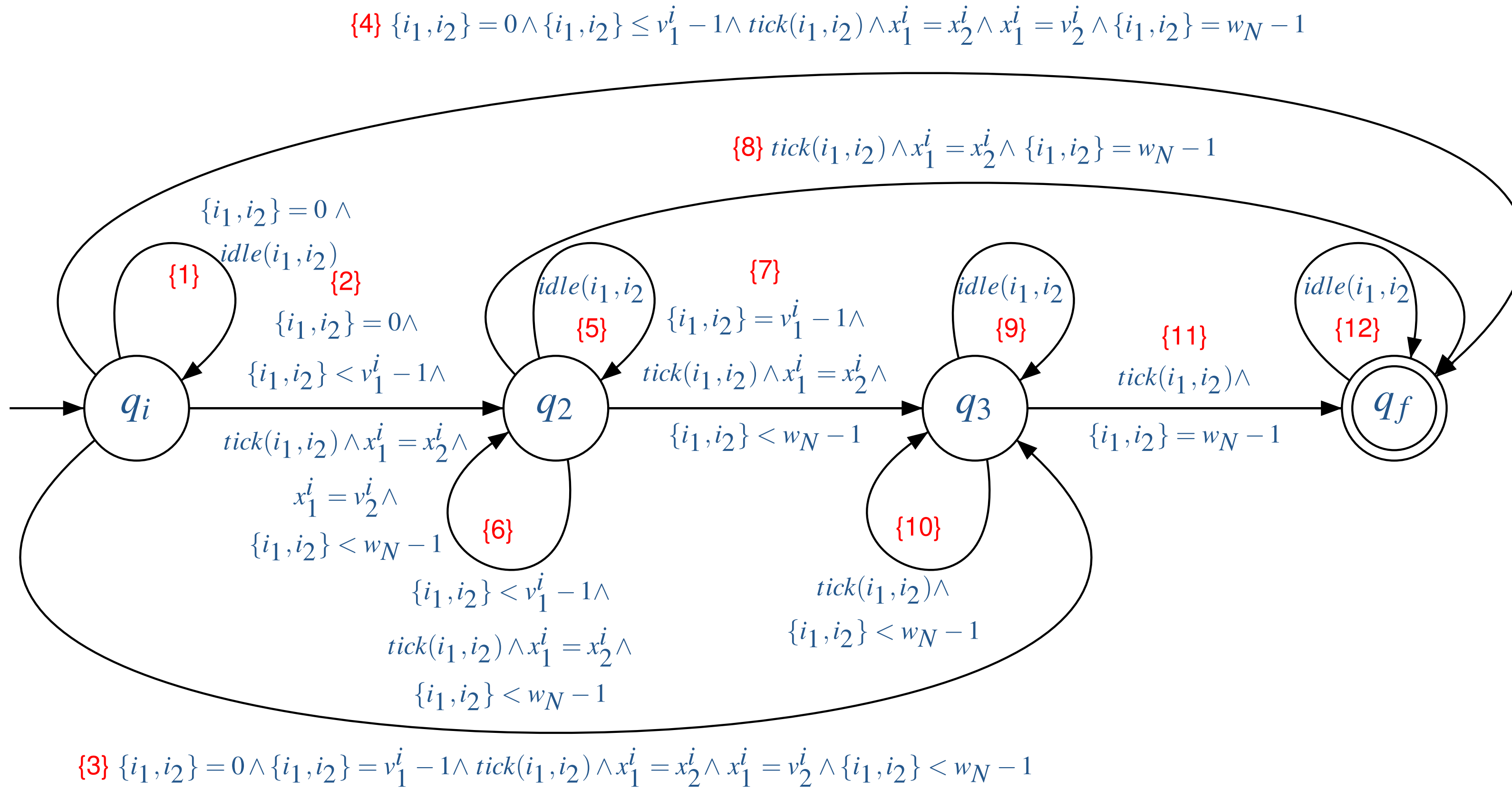


Verification Framework – One Loop Case

```
pre-condition
non-loop code {
   ...
   if() ...
   else if() ...
   else ...
   ...
}
loop {
   if() ...
   else if() ...
      ...
   else ...
}
non-loop code {
   ...
   if() ...
   else if() ...
   else ...
   ...
}
post-condition
```



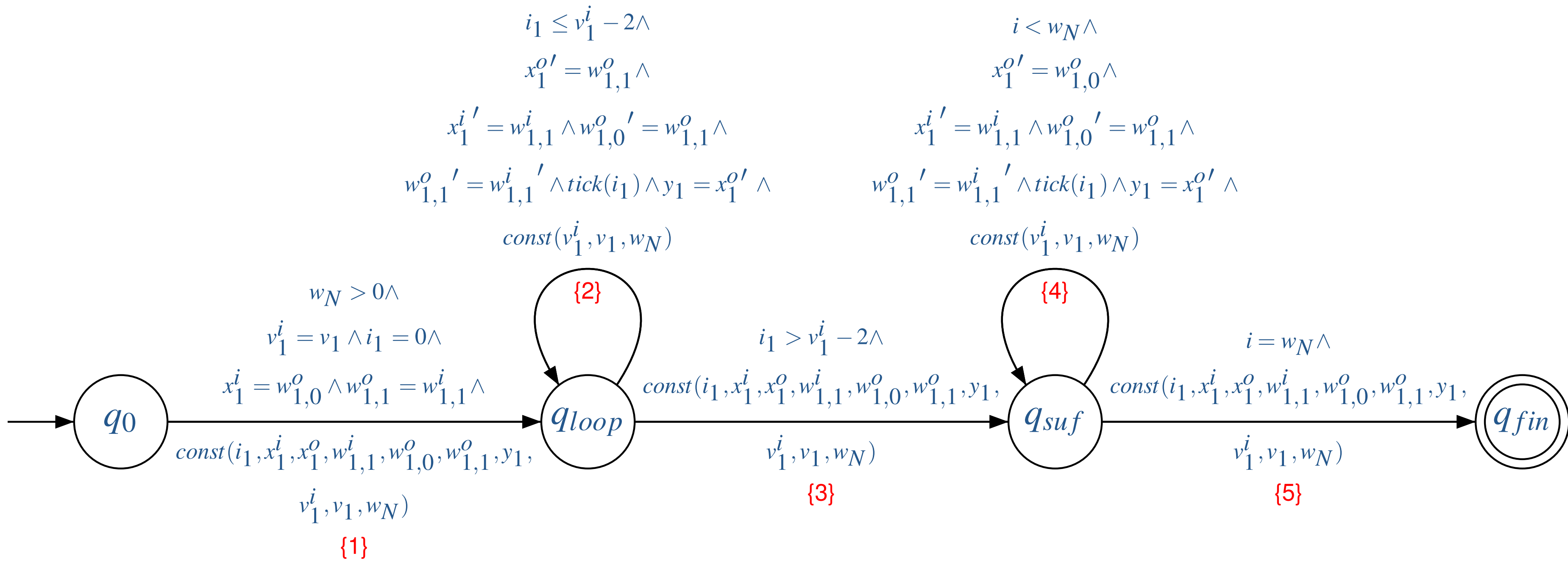
Rotation Example – Precondition Automaton



$$A_{\varphi'}$$

$$\forall i.(0 \leq i \leq b_1 - 1) \Rightarrow (a_1[i] = a_2[i])$$

Rotation Example – Loop Transducer



T_L

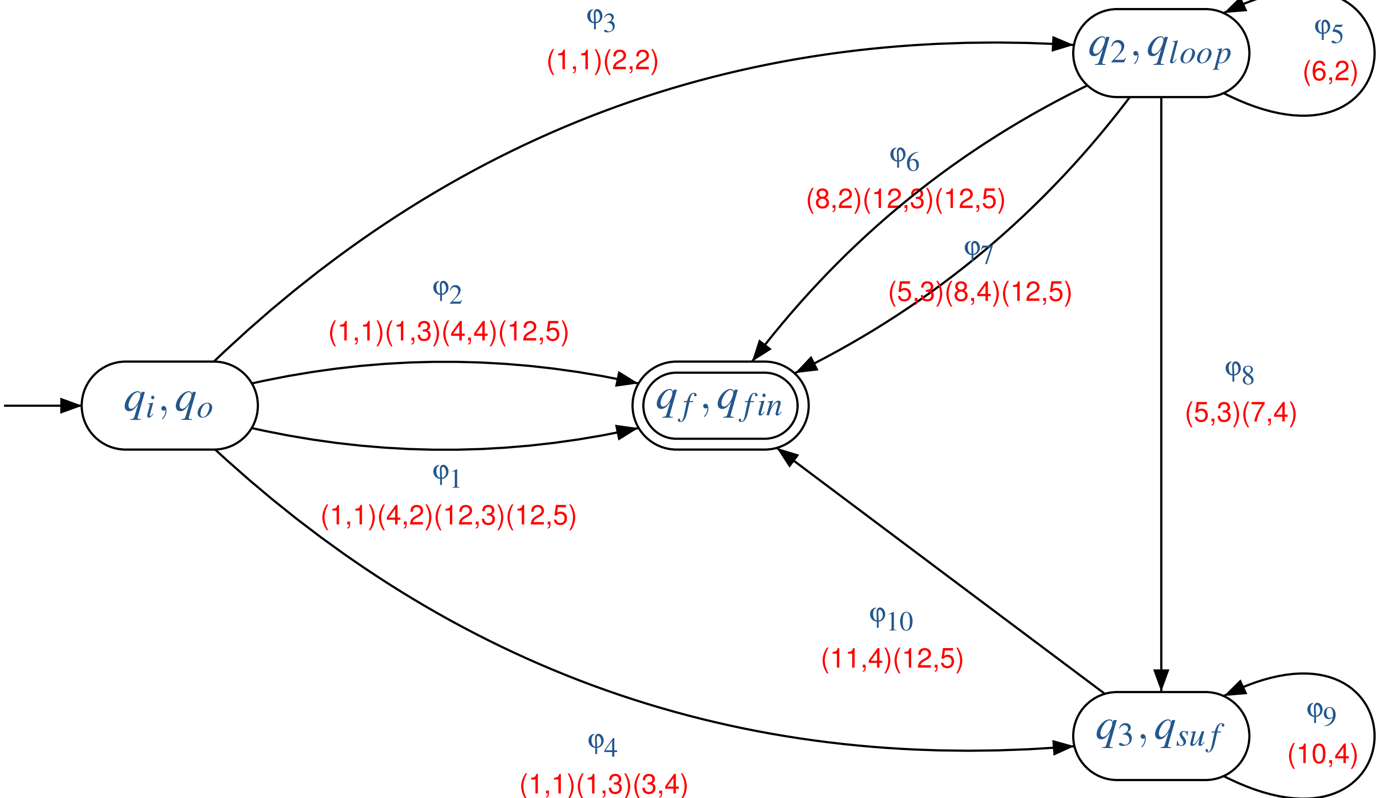
while $_{a:i=0}(i \leq b_1 - 2)$ **do**

$a_1[i] := a_1[i+1];$

$i++;$

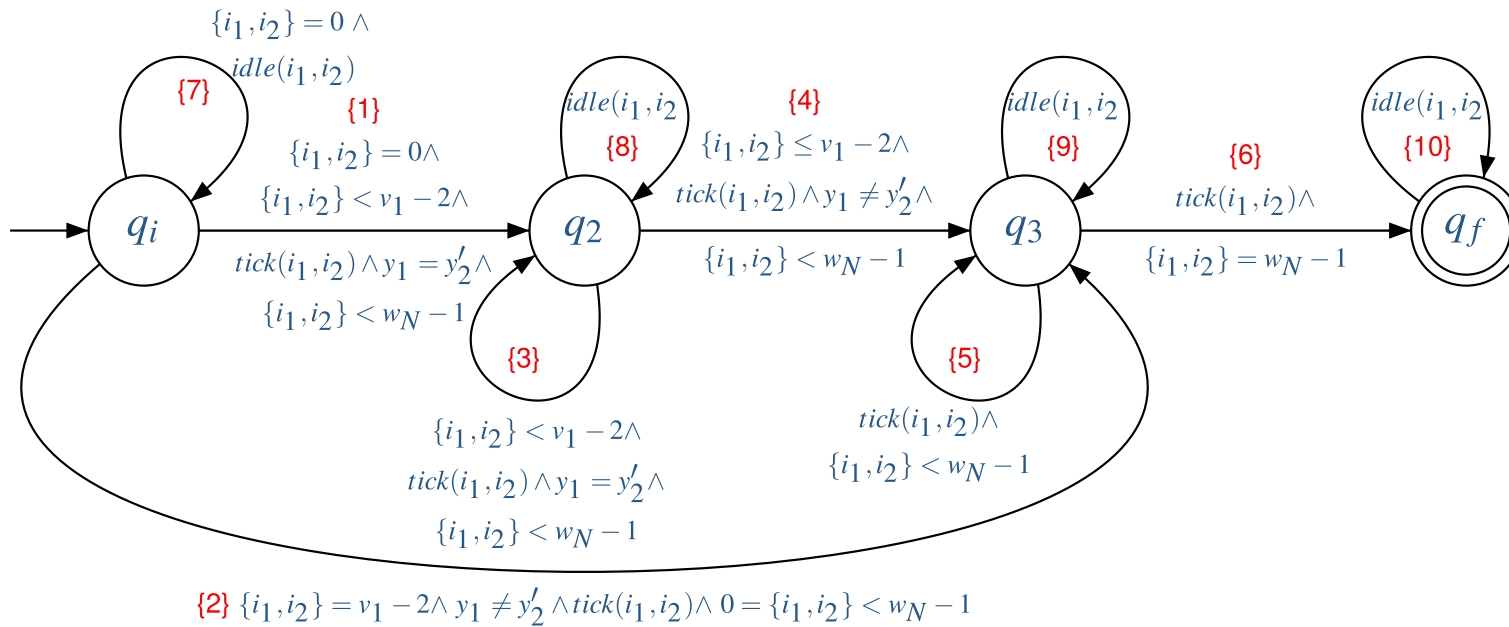
end

Rotation Example – Post-image



$T_L \otimes A_{\varphi'}$

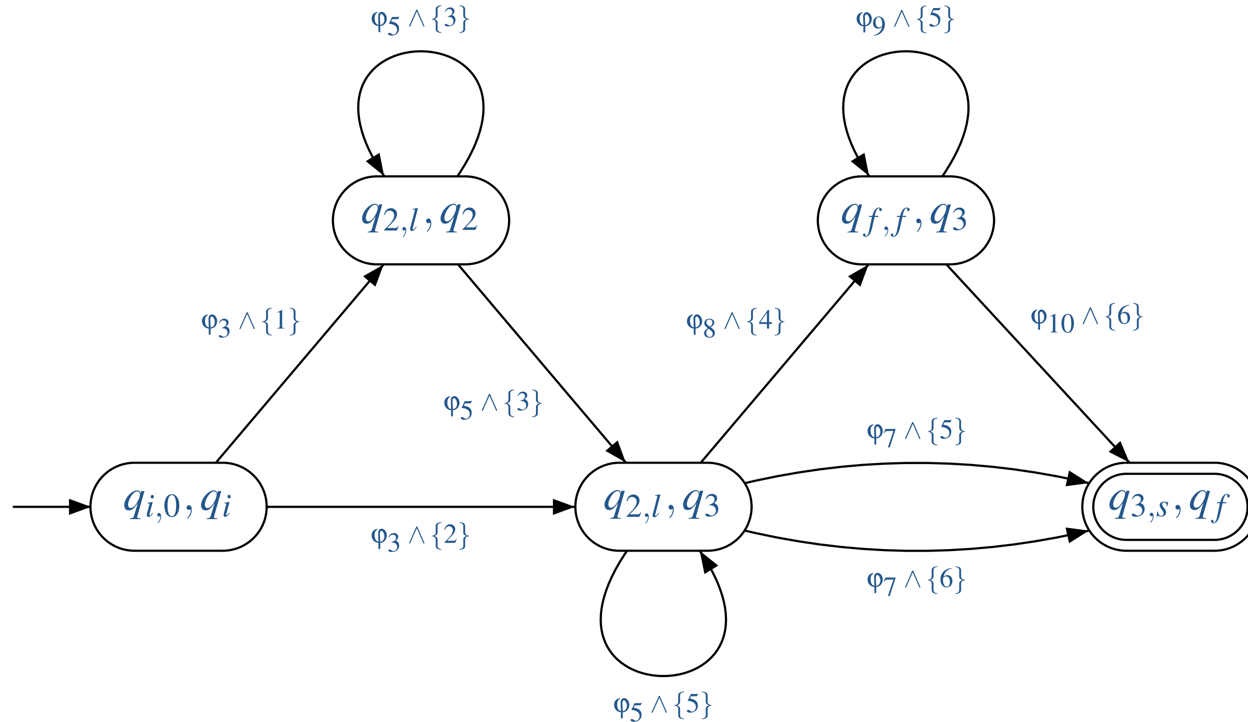
Rotation Example – Post-condition



$$A_{\psi'}$$

$$\neg \forall i.(0 \leq i \leq b_1 - 2) \Rightarrow (a_1[i] = a_2[i+1])$$

Rotation Example – Final Automaton



$$T_L \otimes A_{\varphi'} \otimes A_{\neg\psi'}$$

Alternative Array Verification Approaches

Finding Loop Invariants for Programs over Arrays Using a Theorem Prover
L. Kovács and A. Voronkov.
In *Proc. of FASE'09*, LNCS. Springer, 2009.

Array Abstractions from Proofs
R. Jhala and K. McMillan
In *CAV'07*, *LNCS* 4590. Springer, 2007

Discovering Properties about Arrays in Simple Programs
N. Halbwachs and M. Péron
In *Proc. of PLDI'08*. ACM, 2008

Future Work

- implementation of the techniques introduced
- FLATA – a counter automata tool
- current features:
  - transitive closure computation for loops (for DBM and octagonal contraints)
  - several simplification techniques – an attempt to flatten CAs
  - reachability relation for flat CAs with octagonal loops
  - front-end for .armc, .fast formats
- proposed features:
  - translations from SIL and array programs
  - optimizations of operations on CA inspired by their applications to array verification
  - abstraction techniques
  - translation back to SIL

Future Work

explore further current framework

- don’t abstract, stay at the automata level

array programs

- interprocedural verification
- multithreading
- with pointers

other applications of counter automata

- parameterized systems