What *else* is decidable about integer arrays ?

Peter Habermehl[1,2] Radu Iosif[3] Tomas Vojnar[4]

[1]LSV, ENS Cachan, CNRS, INRIA

[2]LIAFA, CNRS, Université Paris Diderot

[3]VERIMAG, CNRS, Université Joseph Fourier, INPG

[4]FIT, Brno University of Technology

April 4th, 2008

Motivating example

Verification of programs handling integer arrays

```
for (k=0, l=0; k < n; k++, l+=2)

{ c[l] = a[k];
  c[l + 1] = b[k]; }
```

Motivating example

Verification of programs handling integer arrays

$\{\{\ \mathbf{n > 0}\ \wedge\ \forall \mathbf{i}, \mathbf{j}\ .\ \mathbf{0} \leq \mathbf{i}, \mathbf{j} < \mathbf{n} \rightarrow \mathbf{a[i]} \leq \mathbf{b[j]}\ \}\}$

for (k=0, l=0; k < n; k++, l+=2)

{ c[l] = a[k];
  c[l + 1] = b[k]; }

Motivating example

Verification of programs handling integer arrays

$\{\{\ \mathbf{n > 0\ \wedge\ \forall i, j\ .\ 0 \leq i, j < n \rightarrow a[i] \leq b[j]}\ \}\}$

for (k=0, l=0; k < n; k++, l+=2)

{ c[l] = a[k];
  c[l + 1] = b[k]; }

$\{\{\ \mathbf{n > 0\ \wedge\ \forall i, j\ .\ 0 \leq i, j < 2n\ \wedge\ i \equiv_2 0\ \wedge\ j \equiv_2 1 \rightarrow c[i] \leq c[j]}\ \}\}$

Motivating example

Verification of programs handling integer arrays

$$\{\{\ \mathbf{n > 0\ \wedge\ \forall i, j\ .\ 0 \leq i, j < n \rightarrow a[i] \leq b[j]}\ \}\}$$

for (k=0, l=0; k < n; k++, l+=2)

$$\{\{\ \mathbf{n > 0\ \wedge\ k \leq n\ \wedge\ l = 2k\ \wedge}$$
$$\mathbf{\forall i, j\ .\ 0 \leq i, j < 2k\ \wedge\ i \equiv_2 0\ \wedge\ j \equiv_2 1 \rightarrow c[i] \leq c[j]\ \wedge}$$
$$\mathbf{\forall i, j\ .\ 0 \leq i, j < n \rightarrow a[i] \leq b[j]}\ \}\}$$

```
{ c[l] = a[k];
  c[l + 1] = b[k]; }
```

$$\{\{\ \mathbf{n > 0\ \wedge\ \forall i, j\ .\ 0 \leq i, j < 2n\ \wedge\ i \equiv_2 0\ \wedge\ j \equiv_2 1 \rightarrow c[i] \leq c[j]}\ \}\}$$

Example verification condition

to prove the invariant check validity of :

$$
\begin{array}{l}
\forall\ \mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}', \mathbf{c}, \mathbf{c}', \mathbf{n}, \mathbf{n}', \mathbf{k}, \mathbf{k}', \mathbf{l}, \mathbf{l}'. \\
\quad n > 0 \ \wedge\ k \leq n \ \wedge\ l = 2k \ \wedge \\
\quad (\forall \mathbf{i}, \mathbf{j}\ .\ 0 \leq i, j < 2k \ \wedge\ i \equiv_2 0 \ \wedge\ j \equiv_2 1 \rightarrow c[i] \leq c[j]) \ \wedge \\
\quad (\forall \mathbf{i}, \mathbf{j}\ .\ 0 \leq i, j < n \rightarrow a[i] \leq b[j]) \ \wedge \\
\quad k < n \ \wedge\ k' = k + 1 \ \wedge\ l' = l + 2 \ \wedge\ n' = n \ \wedge \\
\quad (\forall \mathbf{i}\ .\ a'[i] = a[i]) \ \wedge\ (\forall \mathbf{i}\ .\ b'[i] = b[i]) \ \wedge \\
\quad (\forall \mathbf{i}\ .\ i < l \rightarrow c'[i] = c[i]) \ \wedge\ (\forall \mathbf{i}\ .\ i > l + 1 \rightarrow c'[i] = c[i]) \ \wedge \\
\quad c'[l] = a[k] \ \wedge\ c'[l+1] = b[k] \\
\quad \longrightarrow \\
\quad n' > 0 \ \wedge\ k' \leq n' \ \wedge\ l' = 2k' \ \wedge \\
\quad (\forall \mathbf{i}, \mathbf{j}\ .\ 0 \leq i, j < 2k' \ \wedge\ i \equiv_2 0 \ \wedge\ j \equiv_2 1 \rightarrow c'[i] \leq c'[j]) \ \wedge \\
\quad (\forall \mathbf{i}, \mathbf{j}\ .\ 0 \leq i, j < n \rightarrow a'[i] \leq b'[j])
\end{array}
$$

Example verification condition

check satisfiability of

$$
\begin{aligned}
&\exists\ \mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}', \mathbf{c}, \mathbf{c}', \mathbf{n}, \mathbf{n}', \mathbf{k}, \mathbf{k}', \mathbf{l}, \mathbf{l}'.\\
&\quad n > 0 \ \wedge\ k \leq n \ \wedge\ l = 2k \ \wedge\\
&\quad (\forall \mathbf{i}, \mathbf{j} \ .\ 0 \leq i, j < 2k \ \wedge\ i \equiv_2 0 \ \wedge\ j \equiv_2 1 \rightarrow c[i] \leq c[j]) \ \wedge\\
&\quad (\forall \mathbf{i}, \mathbf{j} \ .\ 0 \leq i, j < n \rightarrow a[i] \leq b[j]) \ \wedge\\
&\quad k < n \ \wedge\ k' = k + 1 \ \wedge\ l' = l + 2 \ \wedge\ n' = n \ \wedge\\
&\quad (\forall \mathbf{i} \ .\ a'[i] = a[i]) \ \wedge\ (\forall \mathbf{i} \ .\ b'[i] = b[i]) \ \wedge\\
&\quad (\forall \mathbf{i} \ .\ i < l \rightarrow c'[i] = c[i]) \ \wedge\ (\forall \mathbf{i} \ .\ i > l + 1 \rightarrow c'[i] = c[i]) \ \wedge\\
&\quad c'[l] = a[k] \ \wedge\ c'[l+1] = b[k]\\
&\quad \wedge\\
&\quad (n' \leq 0 \ \vee\ k' > n' \ \vee\ l' < 2k' \ \vee\ l' > 2k' \ \vee\\
&\quad (\exists \mathbf{i}, \mathbf{j} \ .\ 0 \leq i, j < 2k' \ \wedge\ i \equiv_2 0 \ \wedge\ j \equiv_2 1 \ \wedge\ c'[i] > c'[j]) \ \vee\\
&\quad (\exists \mathbf{i}, \mathbf{j} \ .\ 0 \leq i, j < n \ \wedge\ a'[i] > b'[j]))
\end{aligned}
$$

Example verification condition

check satisfiability of

$$
\begin{array}{l}
\exists\ \mathbf{a}, \mathbf{a'}, \mathbf{b}, \mathbf{b'}, \mathbf{c}, \mathbf{c'}, \mathbf{n}, \mathbf{n'}, \mathbf{k}, \mathbf{k'}, \mathbf{l}, \mathbf{l'}, \mathbf{i_1}, \mathbf{i_2}, \mathbf{j_1}, \mathbf{j_2}. \\
n > 0 \ \wedge \ k \leq n \ \wedge \ l = 2k \ \wedge \\
\quad (\forall \mathbf{i}, \mathbf{j} \ . \ 0 \leq i, j < 2k \ \wedge \ i \equiv_2 0 \ \wedge \ j \equiv_2 1 \rightarrow c[i] \leq c[j]) \ \wedge \\
\quad (\forall \mathbf{i}, \mathbf{j} \ . \ 0 \leq i, j < n \rightarrow a[i] \leq b[j]) \ \wedge \\
\quad k < n \ \wedge \ k' = k + 1 \ \wedge \ l' = l + 2 \ \wedge \ n' = n \ \wedge \\
\quad (\forall \mathbf{i} \ . \ a'[i] = a[i]) \ \wedge \ (\forall \mathbf{i} \ . \ b'[i] = b[i]) \ \wedge \\
\quad (\forall \mathbf{i} \ . \ i < l \rightarrow c'[i] = c[i]) \ \wedge \ (\forall \mathbf{i} \ . \ i > l + 1 \rightarrow c'[i] = c[i]) \ \wedge \\
\quad c'[l] = a[k] \ \wedge \ c'[l+1] = b[k] \\
\quad \wedge \\
\quad (n' \leq 0 \ \vee \ k' > n' \ \vee \ l' < 2k' \ \vee \ l' > 2k' \ \vee \\
\quad (0 \leq i_1, j_1 < 2k' \ \wedge \ i_1 \equiv_2 0 \ \wedge \ j_1 \equiv_2 1 \ \wedge \ c'[i_1] > c'[j_1]) \ \vee \\
\quad (0 \leq i_2, j_2 < n \ \wedge \ a'[i_2] > b'[j_2]))
\end{array}
$$

Introduction

- Goal: A decidable logic over integer arrays powerful enough to decide verification conditions
- one needs to express properties of the form $\exists^*\forall^*(G \rightarrow V)$
  - leads to undecidability
  - restriction is needed
- [Bradley et al. 2006] and [Bouajjani et al. 2007] do not allow simultaneous references to $a[i]$ and $a[i+1]$
- here, we do not allow disjunctions in V

Easier example and sketch of decision procedure

$$\exists k.\forall i.(0 \leq i < k) \rightarrow a[i] + 1 \leq b[i+1]$$

- we consider bothways infinite arrays

Easier example and sketch of decision procedure

$$\exists k.\forall i.(0 \leq i < k) \rightarrow a[i] + 1 \leq b[i+1]$$

- we consider bothways infinite arrays
- decision procedure :
  - transform a formula into a normal form (composed of simple formulae)
    - models of simple formulae correspond to (infinite) constraint graphs
  - given a simple formula φ construct a counter automaton A_φ
    - one counter for the position in the arrays
    - a counter for each array
    - recognizes bi-infinite sequences
  - check emptiness of A_φ
    - The language of A_φ corresponds to models of the constraint graph and models of φ
    - Emptiness of this type of counter automaton is decidable

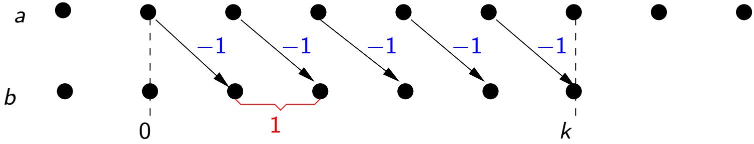
Example translation

$$\exists k.\forall i.(0 \leq i < k) \rightarrow a[i]+1 \leq b[i+1]$$

Example translation

$$\exists k . \forall i . (0 \leq i < k) \rightarrow a[i]+1 \leq b[i+1]$$

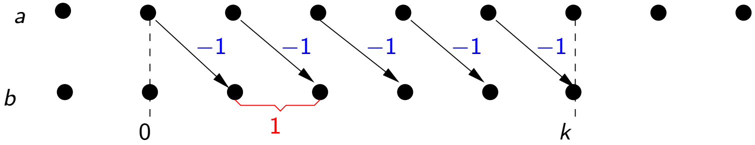
corresponds to



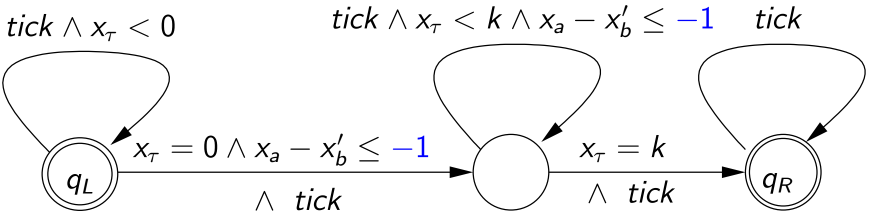
Example translation

$$\exists k.\forall i.(0 \le i < k) \rightarrow a[i]+1 \le b[i+1]$$

corresponds to



and is translated into



where $tick : x'_\tau = x_\tau + 1$

Flat bi-infinite Büchi counter automata (FBCA)

- A FBCA A is a tuple $\langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where
  - $\mathbf{x}$ is a set of counters
    - including a set $\mathbf{k}$ of parameters whose values never change
  - Q is the set of states,
  - $L \subset Q$ and $R \subset Q$ are the left and right accepting states,
  - $\rightarrow$ is the transition relation given by rules $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$
    - $\varphi(\mathbf{x}, \mathbf{x}')$ is a parametric difference constraint formula (conjunction of terms like $x - y' \leq c$ but also $x \leq k$ where k is a parameter)
  - The control structure is flat (no nested loops)

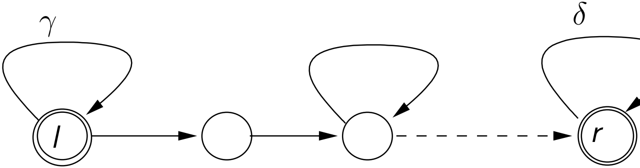
Flat bi-infinite Büchi counter automata (FBCA)

- A FBCA A is a tuple $\langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where
  - $\mathbf{x}$ is a set of counters
    - including a set $\mathbf{k}$ of parameters whose values never change
  - Q is the set of states,
  - $L \subset Q$ and $R \subset Q$ are the left and right accepting states,
  - $\rightarrow$ is the transition relation given by rules $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$
    - $\varphi(\mathbf{x}, \mathbf{x}')$ is a parametric difference constraint formula (conjunction of terms like $x - y' \leq c$ but also $x \leq k$ where k is a parameter)
  - The control structure is flat (no nested loops)
- A configuration c is tuple (q, ν) with $q \in Q$ and ν a valuation of the counters

Flat bi-infinite Büchi counter automata (FBCA)

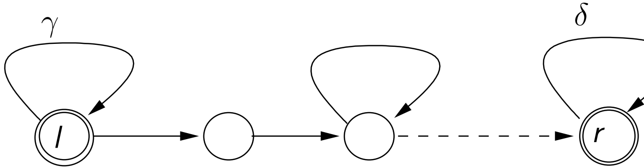
- A FBCA A is a tuple $\langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where
  - $\mathbf{x}$ is a set of counters
    - including a set $\mathbf{k}$ of parameters whose values never change
  - Q is the set of states,
  - $L \subset Q$ and $R \subset Q$ are the left and right accepting states,
  - $\rightarrow$ is the transition relation given by rules $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$
    - $\varphi(\mathbf{x}, \mathbf{x}')$ is a parametric difference constraint formula (conjunction of terms like $x - y' \leq c$ but also $x \leq k$ where k is a parameter)
  - The control structure is flat (no nested loops)
- A configuration c is tuple (q, ν) with $q \in Q$ and ν a valuation of the counters
- FBCA accept bi-infinite sequences of configurations of the form $\ldots c_{-2} c_{-1} c_0 c_1 c_2 \ldots$ such that a state of L (resp. R) is visited infinitely often on the left (resp. right)

Emptiness of FBCA is decidable



- Fix a left-accepting state l and a right accepting state r with loops γ and δ

Emptiness of FBCA is decidable



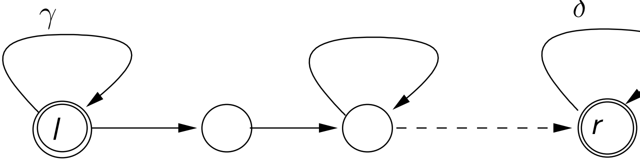
- Fix a left-accepting state l and a right accepting state r with loops γ and δ
- There is an accepting run through l and r iff

$$\Phi_{l,r} : \exists \mathbf{x} \exists \mathbf{x}' \ . \ I_{l,\overleftarrow{\gamma}}(\mathbf{x}) \ \wedge \ R_{l,r}(\mathbf{x}, \mathbf{x}') \ \wedge \ I_{r,\delta}(\mathbf{x}')$$

is satisfiable

  - $I_{l,\overleftarrow{\gamma}}(\mathbf{x})$: There is an infinite computation along $\overleftarrow{\gamma}$ starting from $\mathbf{x}$
  - $R_{l,r}(\mathbf{x}, \mathbf{x}')$: There is a computation from l to r
  - $I_{r,\delta}(\mathbf{x}')$: There is an infinite computation along δ starting from $\mathbf{x}'$

Emptiness of FBCA is decidable



- Fix a left-accepting state l and a right accepting state r with loops γ and δ
- There is an accepting run through l and r iff

$$\Phi_{l,r} : \exists \mathbf{x} \exists \mathbf{x}' \,.\, I_{l,\overleftarrow{\gamma}}(\mathbf{x}) \;\wedge\; R_{l,r}(\mathbf{x},\mathbf{x}') \;\wedge\; I_{r,\delta}(\mathbf{x}')$$

  is satisfiable
  - $I_{l,\overleftarrow{\gamma}}(\mathbf{x})$: There is an infinite computation along $\overleftarrow{\gamma}$ starting from $\mathbf{x}$
  - $R_{l,r}(\mathbf{x},\mathbf{x}')$: There is a computation from l to r
  - $I_{r,\delta}(\mathbf{x}')$: There is an infinite computation along δ starting from $\mathbf{x}'$
- The formulae can be given using the theory of (flat) counter automata with difference constraints [Comon, Jurski 1998] [Bozga, Iosif 2006]

Array logic (LIA) and its decision procedure

- Syntax
  - $\exists\, \mathbf{k}\, \exists\, \mathbf{a}\, \mathcal{BC}(\forall\, \mathbf{i}.\, \mathbf{G} \rightarrow \mathbf{V})$
  - G is of the form $\bigvee \bigwedge \varphi(\mathbf{k}, \mathbf{i})$
    - ★ φ is either a difference constraint (like $i - j \leq c$) or a modulo constraint (like $i \equiv_s c$)
  - V is of the form $\bigwedge \psi(\mathbf{a}, \mathbf{k}, \mathbf{i})$
    - ★ ψ is a difference constraint on array value expressions (like $a[i] - b[i + n] \leq c$)

Array logic (LIA) and its decision procedure

- Syntax
  - $\exists\, \mathbf{k}\, \exists\, \mathbf{a}\, \mathcal{BC}(\forall\, \mathbf{i}.\ \mathbf{G} \rightarrow \mathbf{V})$
  - G is of the form $\bigvee \bigwedge \varphi(\mathbf{k}, \mathbf{i})$
    - ★ φ is either a difference constraint (like $i - j \leq c$) or a modulo constraint (like $i \equiv_s c$)
  - V is of the form $\bigwedge \psi(\mathbf{a}, \mathbf{k}, \mathbf{i})$
    - ★ ψ is a difference constraint on array value expressions (like $a[i] - b[i + n] \leq c$)
- Decision procedure :
  - transform a formula into a normal form (composed of simple formulae)
    - ★ models of simple formulae correspond to (infinite) constraint graphs
  - given a simple formula φ construct a counter automaton A_φ

Normal form

Every LIA formula can be written equivalently in the following normal form

$$\exists \mathbf{k} \exists \mathbf{a} \,.\, \bigvee_p \Big(\bigwedge_q \phi_{pq}(\mathbf{a}, \mathbf{k}) \Big) \wedge \theta_p(\mathbf{k}) \qquad \text{(NF)}$$

where **a** is a set of array variables, **k** is a set of integer variables, and

- θ_p is a conjunction of terms of the forms:
  - $g(\mathbf{k}) \geq 0$
  - $g(\mathbf{k}) \equiv_s t$

  where g is a linear combination of the variables in **k**, and $0 \leq t < s$

Normal form

- ϕ_{pq} is a formula of the following forms, for $\sim \in \{\leq, \geq\}$ and some $m \in \mathbb{N}$, $0 \leq t < s$, $0 \leq v < u$, $p, q \in \mathbb{Z}$ and the $f_k, g_k, f_k^1, g_k^1, f_k^2, g_k^2, h(\mathbf{k})$ are linear combinations of parameters

$$\forall i \ . \ \bigwedge_{k=1}^{K} f_k \leq i \ \wedge \ \bigwedge_{l=1}^{L} i \leq g_l \ \wedge \ i \equiv_s t \rightarrow a[i] \sim h(\mathbf{k}) \qquad \text{(F1)}$$

Normal form

- ϕ_{pq} is a formula of the following forms, for $\sim \in \{\leq, \geq\}$ and some $m \in \mathbb{N}$, $0 \leq t < s$, $0 \leq v < u$, $p, q \in \mathbb{Z}$ and the $f_k, g_k, f_k^1, g_k^1, f_k^2, g_k^2, h(\mathbf{k})$ are linear combinations of parameters

$$\forall i \ . \ \bigwedge_{k=1}^{K} f_k \leq i \ \wedge \ \bigwedge_{l=1}^{L} i \leq g_l \ \wedge \ i \equiv_s t \rightarrow a[i] \sim h(\mathbf{k}) \qquad \text{(F1)}$$

$$\forall i \ . \ \bigwedge_{k=1}^{K} f_k \leq i \ \wedge \ \bigwedge_{l=1}^{L} i \leq g_l \ \wedge \ i \equiv_s t \rightarrow a[i] - b[i+p] \sim q \qquad \text{(F2)}$$

Normal form

- ϕ_{pq} is a formula of the following forms, for $\sim \in \{\leq, \geq\}$ and some $m \in \mathbb{N}$, $0 \leq t < s$, $0 \leq v < u$, $p, q \in \mathbb{Z}$ and the $f_k, g_k, f_k^1, g_k^1, f_k^2, g_k^2, h(\mathbf{k})$ are linear combinations of parameters

$$\forall i \ . \ \bigwedge_{k=1}^{K} f_k \leq i \ \wedge \ \bigwedge_{l=1}^{L} i \leq g_l \ \wedge \ i \equiv_s t \rightarrow a[i] \sim h(\mathbf{k}) \qquad \text{(F1)}$$

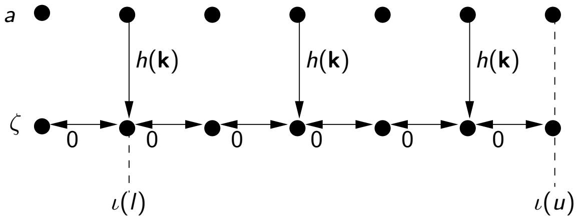
$$\forall i \ . \ \bigwedge_{k=1}^{K} f_k \leq i \ \wedge \ \bigwedge_{l=1}^{L} i \leq g_l \ \wedge \ i \equiv_s t \rightarrow a[i] - b[i+p] \sim q \qquad \text{(F2)}$$

$$\begin{gathered}\forall i, j \ . \ \textstyle\bigwedge_{k=1}^{K_1} f_k^1 \leq i \ \wedge \ \bigwedge_{l=1}^{L_1} i \leq g_l^1 \ \wedge \ \bigwedge_{k=1}^{K_2} f_k^2 \leq j \ \wedge \ \bigwedge_{l=1}^{L_2} j \leq g_l^2 \ \wedge \\ i - j \leq p \ \wedge \ i \equiv_s t \ \wedge \ j \equiv_u v \rightarrow a[i] - b[j] \sim q\end{gathered} \qquad \text{(F3)}$$

Decision procedure

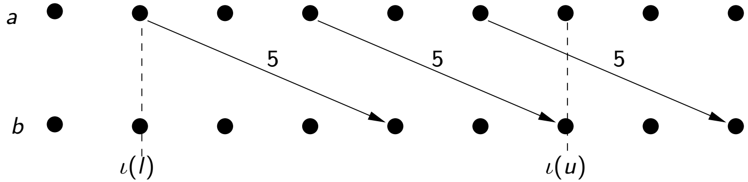
- transform a formula into a normal form (composed of simple formulae)
    - models of simple formulae correspond to (infinite) constraint graphs
- given a simple formula φ construct a counter automaton A_φ

Formulae $\rightarrow$ Constraint graphs



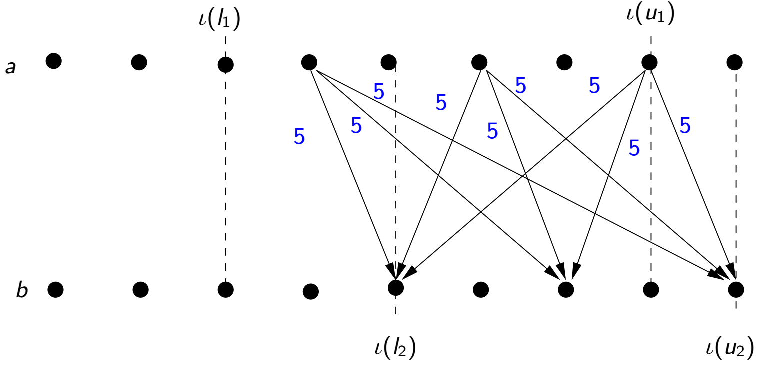
Constraint graph for $\forall i \ . \ l \leq i \leq u \ \wedge \ i \equiv_2 0 \rightarrow a[i] \leq h(\mathbf{k})$

Formulae $\rightarrow$ Constraint graphs



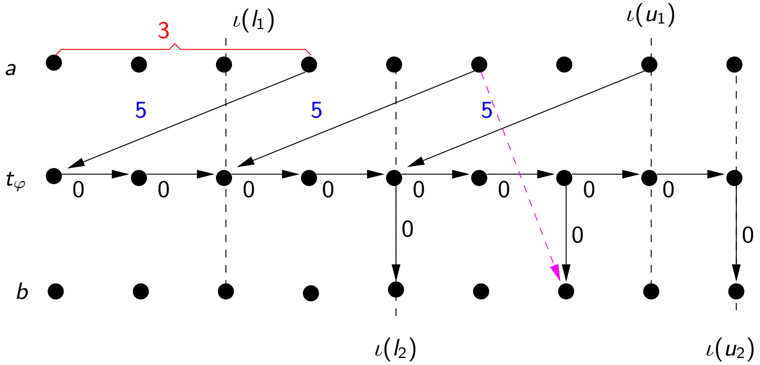
Constraint graph for $\forall i. l \leq i \leq u \wedge i \equiv_2 0 \rightarrow a[i] - b[i+3] \leq 5$

Formulae → Constraint graphs



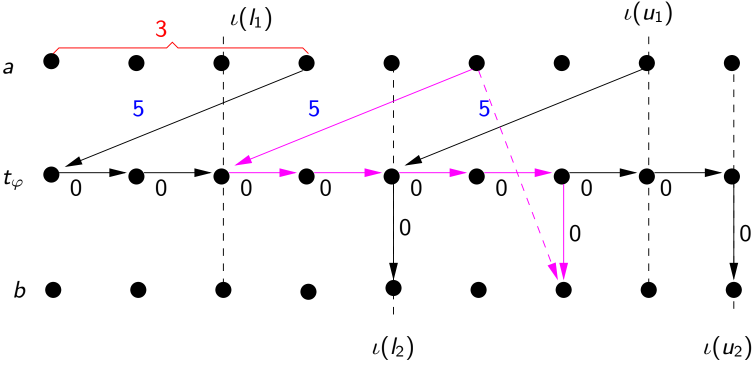
Constraint graph for
$\forall i, j. l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$

Formulae $\rightarrow$ Constraint graphs



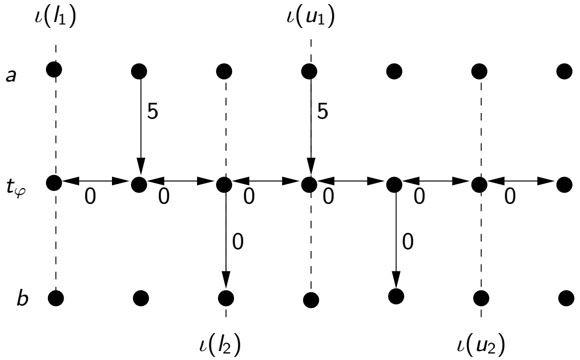
Constraint graph for
$\forall i,j. l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$

Formulae $\rightarrow$ Constraint graphs



Constraint graph for
$\forall i,j. l_1 \le i \le u_1 \wedge l_2 \le j \le u_2 \wedge i - j \le 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \le 5$

Formulae $\rightarrow$ Constraint graphs



Constraint graph for $\forall i, j . l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i \equiv_2 0 \wedge j \equiv_2 1$
$\rightarrow a[i] - b[j] \leq 5$

Decision procedure

- transform a formula into a normal form (composed of simple formulae)
  - models of simple formulae correspond to (infinite) constraint graphs
- given a simple formula φ construct a counter automaton A_φ

From constraint graphs to counter automata

- For each type of edge (horizontal, vertical and diagonal), we have a different automaton template (A_{hor}, A_{ver} and A_{diag}).

From constraint graphs to counter automata

- For each type of edge (horizontal, vertical and diagonal), we have a different automaton template (A_{hor}, A_{ver} and A_{diag}).
- Instances of these automata templates can then be composed by a product to obtain the automaton corresponding to a simple formula

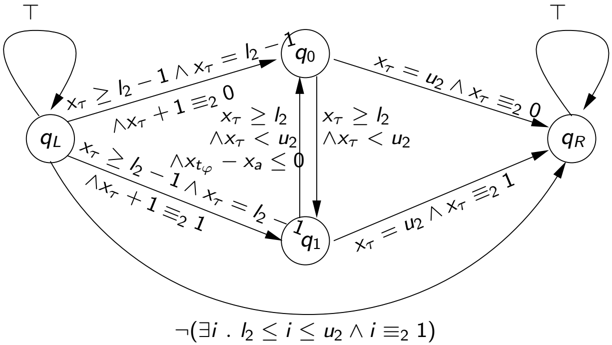
From constraint graphs to counter automata

- For each type of edge (horizontal, vertical and diagonal), we have a different automaton template (A_{hor}, A_{ver} and A_{diag}).
- Instances of these automata templates can then be composed by a product to obtain the automaton corresponding to a simple formula
- to obtain the automaton A_φ corresponding to φ we build the automata for each simple formula of the normal form of φ and compose them

From constraint graphs to counter automata

- For each type of edge (horizontal, vertical and diagonal), we have a different automaton template (A_{hor}, A_{ver} and A_{diag}).
- Instances of these automata templates can then be composed by a product to obtain the automaton corresponding to a simple formula
- to obtain the automaton A_φ corresponding to φ we build the automata for each simple formula of the normal form of φ and compose them
- The automata synchronize on common variables

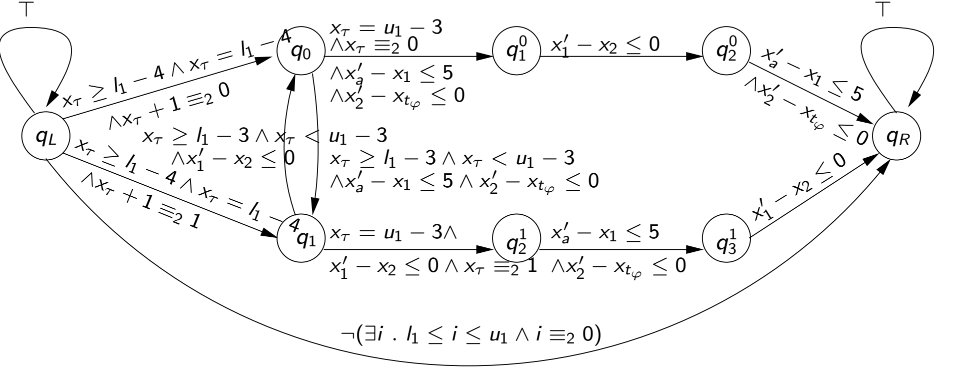
Counter automata for formulae



The FBCA for the vertical edges in the formula

$\varphi : \forall i, j. l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$

Counter automata for formulae



The FBCA for the diagonal edges in the formula
$\varphi : \forall i, j. l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$

Future work

- Studying complexity
- Implementation (of a restricted logic)
- Invariant generation
- Extensions, restrictions

Logic of Integer Arrays (LIA) syntax

$n, m, s, t..$	$\in$	$\mathbb{Z}$	constants $(0 \leq t < s)$
$k, l, \ldots$	$\in$	$BVar$	array-bound variables
$i, j, \ldots$	$\in$	$IVar$	index variables
$a, b, \ldots$	$\in$	$AVar$	array variables
B	$:=$	$n \mid k \mid B + B \mid B - B$	array-bound terms
I	$:=$	$i \mid I + n$	index terms
A	$:=$	$a[I] \mid a[B]$	array terms
G	$:=$	$B \leq I \mid I \leq B \mid I - I \leq n \mid I \equiv_s t \mid G \vee G \mid G \wedge G$	guard expressions
V	$:=$	$A \leq B \mid B \leq A \mid A - A \leq n \mid V \wedge V$	value expressions
C	$:=$	$B \leq n \mid B \equiv_s t$	array-bound constraints
P	$:=$	$\top \rightarrow V \mid G \rightarrow V \mid \forall i \,.\, P$	array properties
U	$:=$	$P \mid C \mid \neg U \mid U \vee U \mid U \wedge U$	universal formulae
F	$:=$	$U \mid \exists k \,.\, F \mid \exists a \,.\, F$	**LIA** formulae