Using Abstract Acceleration in the Verification of Logico-Numerical Data-Flow Programs

Peter Schrammel and Bertrand Jeannet
{peter.schrammel,bertrand.jeannet}@inria.fr

INRIA Rhône-Alpes

Synchron'10

Analysis of Numerical Programs

Reachability analysis of numerical programs

- Abstract interpretation (Cousot and Cousot 1977)
  - Termination, but over-approximation
- Acceleration (Finkel and Leroux 2002)
  - Exact result, but no guarantee for termination

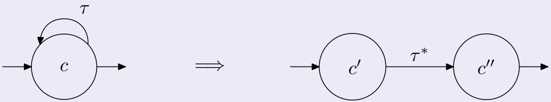
Analysis of Numerical Programs

Reachability analysis of numerical programs

- Abstract interpretation (Cousot and Cousot 1977)
  - Termination, but over-approximation
- Acceleration (Finkel and Leroux 2002)
  - Exact result, but no guarantee for termination

Acceleration

- Method for treating *loops* in numerical automata
- Replace a loop transition τ by its *transitive closure* τ^*



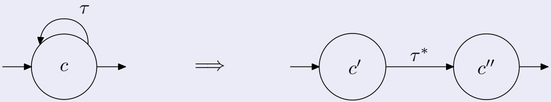
Analysis of Numerical Programs

Reachability analysis of numerical programs

- Abstract interpretation (Cousot and Cousot 1977)
  - Termination, but over-approximation
- Acceleration (Finkel and Leroux 2002)
  - Exact result, but no guarantee for termination

Acceleration

- Method for treating *loops* in numerical automata
- Replace a loop transition τ by its *transitive closure* τ^*



Abstract Acceleration (Gonnord and Halbwachs 2006)

- Computation of the convex hull $\tau^{\otimes}$ of the exact result τ^*

Application to Logico-Numerical Data-Flow Programs

Application to e.g. Lustre programs?

Issues

1. Input variables:
   - Boolean variables:
     - Encoded as non-determinism in the control flow graph (CFG)
   - Numerical variables
2. *Implicit* control flow → Discover a CFG w.r.t. Boolean variables
   1. **Conventional approach:**
      1. Reduction to numerical automaton by *enumeration* of Boolean states
      2. → Combinatorial explosion
   2. **Our approach:**
      1. Symbolic handling of Boolean variables
      2. Approximation method: Decoupling
      3. Controlled partitioning using heuristics

Outline

1. Introduction

2. Abstract Acceleration with Numerical Inputs
   - Abstract Acceleration
   - Abstract Acceleration with Numerical Inputs
   - Translations with Simple Guards
   - Translations with Resets and Simple Guards
   - General Guards
   - Comparison with Widening

3. Application to Logico-Numerical Data-Flow Programs
   - Conventional Approach
   - Decoupling
   - Partitioning
   - Experimental Results

4. Conclusion

Outline

1. Introduction

2. Abstract Acceleration with Numerical Inputs
   - Abstract Acceleration
   - Abstract Acceleration with Numerical Inputs
   - Translations with Simple Guards
   - Translations with Resets and Simple Guards
   - General Guards
   - Comparison with Widening

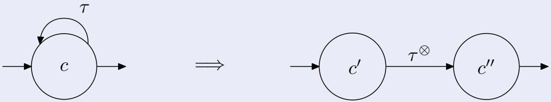
3. Application to Logico-Numerical Data-Flow Programs
   - Conventional Approach
   - Decoupling
   - Partitioning
   - Experimental Results

4. Conclusion

Abstract Acceleration

Abstract Acceleration (Gonnord and Halbwachs 2006)

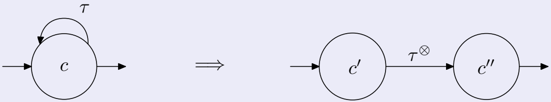
- Computation of a convex polyhedron $\tau^{\otimes}$ close to the exact result τ^*
- Acceleration of self-loops: $\tau : \underbrace{\mathbf{Ax} \le \mathbf{v}}_{\text{guard } G} \rightarrow \underbrace{\mathbf{x}' = \mathbf{Cx} + \mathbf{d}}_{\text{action}}$



Abstract Acceleration

Abstract Acceleration (Gonnord and Halbwachs 2006)

- Computation of a convex polyhedron $\tau^{\otimes}$ close to the exact result τ^*
- Acceleration of self-loops: $\tau : \underbrace{\mathbf{Ax} \le \mathbf{v}}_{\text{guard } G} \rightarrow \underbrace{\mathbf{x}' = \mathbf{Cx} + \mathbf{d}}_{\text{action}}$



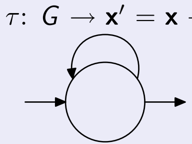
Accelerable Transitions

- Resets: $G \rightarrow \mathbf{x} := \mathbf{d}$
- Translations: $G \rightarrow \mathbf{x} := \mathbf{x} + \mathbf{d}$
- Translations with resets: $G \rightarrow \mathbf{x} := \mathbf{Cx} + \mathbf{d}$ where $\mathbf{C} = diag(\ldots, c_i, \ldots), c_i \in \{0, 1\}$
- Periodic affine transformations: $G \rightarrow \mathbf{x} := \mathbf{Cx} + \mathbf{d}$ where $\exists p > 0 : \mathbf{C}^p = \mathbf{C}^{2p}$

Example

Translations

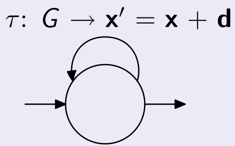
- $\tau : G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$
- Accelerated transition: $\tau^{\otimes}(X) = X \sqcup \big(((X \sqcap G) \nearrow \mathbf{d}) \sqcap G(\mathbf{x} - \mathbf{d})\big)$

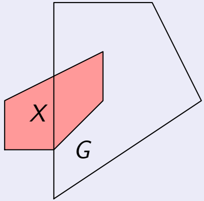


Example

Translations

- $\tau : G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$
- Accelerated transition: $\tau^{\otimes}(X) = X \sqcup (((X \sqcap G) \nearrow \mathbf{d}) \sqcap G(\mathbf{x} - \mathbf{d}))$



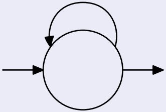


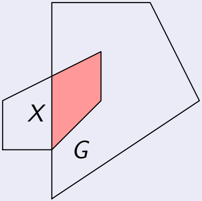
Example

Translations

- $\tau : G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$
- Accelerated transition: $\tau^{\otimes}(X) = X \sqcup (((X \sqcap G) \nearrow \mathbf{d}) \sqcap G(\mathbf{x} - \mathbf{d}))$

$\tau\colon\ G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$



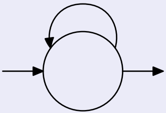


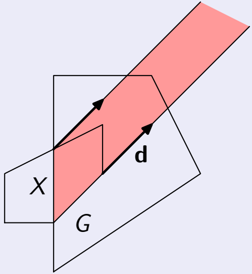
Example

Translations

- $\tau : G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$
- Accelerated transition: $\tau^{\otimes}(X) = X \sqcup ((\boxed{(X \sqcap G) \nearrow \mathbf{d}}) \sqcap G(\mathbf{x} - \mathbf{d}))$

τ: $G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$

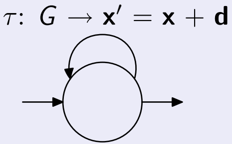


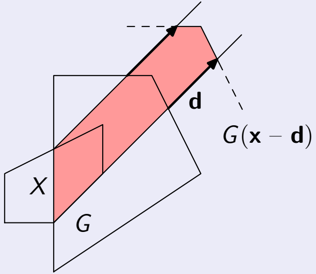


Example

Translations

- $\tau : G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$
- Accelerated transition: $\tau^{\otimes}(X) = X \sqcup \left(\boxed{((X \sqcap G) \nearrow \mathbf{d}) \sqcap G(\mathbf{x} - \mathbf{d})} \right)$

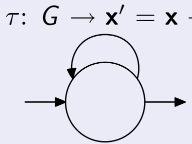


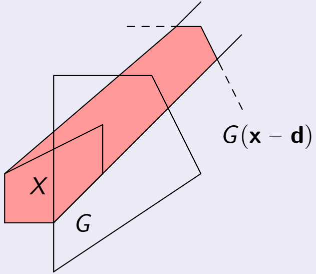


Example

Translations

- $\tau : G \rightarrow \mathbf{x}' = \mathbf{x} + \mathbf{d}$
- Accelerated transition: $\tau^{\otimes}(X) = \boxed{X \sqcup (((X \sqcap G) \nearrow \mathbf{d}) \sqcap G(\mathbf{x} - \mathbf{d}))}$





Outline

Abstract Acceleration with Numerical Inputs

- Extension to numerical inputs ξ:

$$\tau : \underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{L} \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \xi \end{pmatrix} \leq \begin{pmatrix} \mathbf{v} \\ \mathbf{k} \end{pmatrix}}_{\mathbf{Ax}+\mathbf{L}\boldsymbol{\xi}\leq\mathbf{v} \,\wedge\, \mathbf{J}\boldsymbol{\xi}\leq\mathbf{k}} \quad \rightarrow \quad \underbrace{\mathbf{x}' = \begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \xi \end{pmatrix} + \mathbf{u}}_{\mathbf{x}'=\mathbf{Cx}+\mathbf{T}\boldsymbol{\xi}+\mathbf{u}}$$

- $L = 0$ (*"simple guards"*):
  - $\rightarrow$ No interaction between inputs and state variables in the guard
  - Translations
  - Translations with resets
- $L \neq 0$ (*"general guards"*):
  - No more accelerable $\rightarrow$ approximated solution

Abstract Acceleration with Numerical Inputs

Translations with Simple Guards

- $\tau : \underbrace{\mathbf{A}\mathbf{x} \leq \mathbf{v}}_{G} \;\wedge\; \mathbf{J}\boldsymbol{\xi} \leq \mathbf{k} \quad \rightarrow \quad \mathbf{x}' = \mathbf{x} + \underbrace{\mathbf{T}\boldsymbol{\xi} + \mathbf{u}}_{D}$

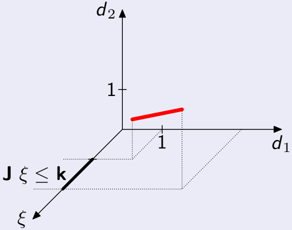
$$\text{Example: } \tau(X) : \left| \begin{array}{r} x_1 + x_2 \leq 4 \\ 1 \leq \xi \leq 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$$

Abstract Acceleration with Numerical Inputs

Translations with Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \leq \mathbf{v}}_{G} \ \wedge \ \mathbf{J}\boldsymbol{\xi} \leq \mathbf{k} \quad \rightarrow \quad \mathbf{x}' = \mathbf{x} + \underbrace{\mathbf{T}\boldsymbol{\xi} + \mathbf{u}}_{D}$
- $\tau(X) = (X \sqcap G) + D$ $\qquad D = \{\mathbf{d} \mid \exists \boldsymbol{\xi} : \mathbf{J}\boldsymbol{\xi} \leq \mathbf{k} \wedge \mathbf{d} = \mathbf{T}\boldsymbol{\xi} + \mathbf{u}\}$

Example: $\tau(X) : \left| \begin{array}{c} x_1 + x_2 \leq 4 \\ 1 \leq \xi \leq 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$

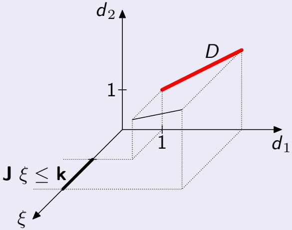


Abstract Acceleration with Numerical Inputs

Translations with Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \leq \mathbf{v}}_{G} \;\wedge\; \mathbf{J}\boldsymbol{\xi} \leq \mathbf{k} \quad \rightarrow \quad \mathbf{x}' = \mathbf{x} + \underbrace{\mathbf{T}\boldsymbol{\xi} + \mathbf{u}}_{D}$
- $\tau(X) = (X \sqcap G) + D$ $\qquad D = \{\mathbf{d} \mid \exists \boldsymbol{\xi} : \mathbf{J}\boldsymbol{\xi} \leq \mathbf{k} \wedge \mathbf{d} = \mathbf{T}\boldsymbol{\xi} + \mathbf{u}\}$

$$\text{Example: } \tau(X) : \left| \begin{array}{c} x_1 + x_2 \leq 4 \\ 1 \leq \xi \leq 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$$

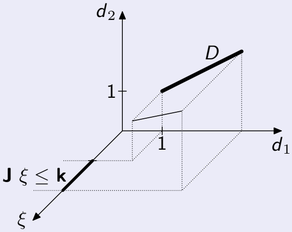


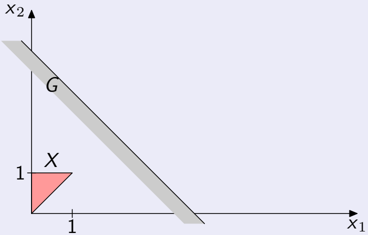
Abstract Acceleration with Numerical Inputs

Translations with Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \leq \mathbf{v}}_{G} \;\wedge\; \mathbf{J}\xi \leq \mathbf{k} \quad\rightarrow\quad \mathbf{x}' = \mathbf{x} + \underbrace{\mathbf{T}\xi + \mathbf{u}}_{D}$
- $\tau(X) = (X \sqcap G) + D \qquad D = \{\mathbf{d} \mid \exists\xi : \mathbf{J}\xi \leq \mathbf{k} \wedge \mathbf{d} = \mathbf{T}\xi + \mathbf{u}\}$
- $\tau^{\otimes}(X) = X \sqcup \tau((\boxed{X} \sqcap G) \nearrow D)$

Example: $\tau(X) : \left| \begin{array}{c} x_1 + x_2 \leq 4 \\ 1 \leq \xi \leq 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$



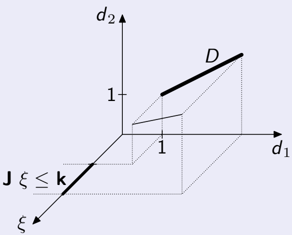


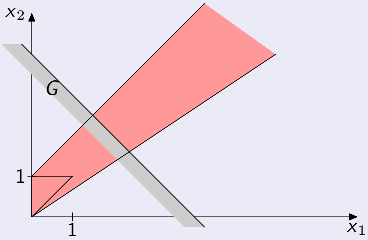
Abstract Acceleration with Numerical Inputs

Translations with Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \leq \mathbf{v}}_{G} \;\wedge\; \mathbf{J}\xi \leq \mathbf{k} \quad \to \quad \mathbf{x'} = \mathbf{x} + \underbrace{\mathbf{T}\xi + \mathbf{u}}_{D}$
- $\tau(X) = (X \sqcap G) + D \qquad\qquad D = \{\mathbf{d} \mid \exists \xi : \mathbf{J}\xi \leq \mathbf{k} \wedge \mathbf{d} = \mathbf{T}\xi + \mathbf{u}\}$
- $\tau^{\otimes}(X) = X \sqcup \tau\big(\boxed{(X \sqcap G) \nearrow D}\big)$

Example: $\tau(X) : \left| \begin{array}{c} x_1 + x_2 \leq 4 \\ 1 \leq \xi \leq 2 \end{array} \right. \to \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$



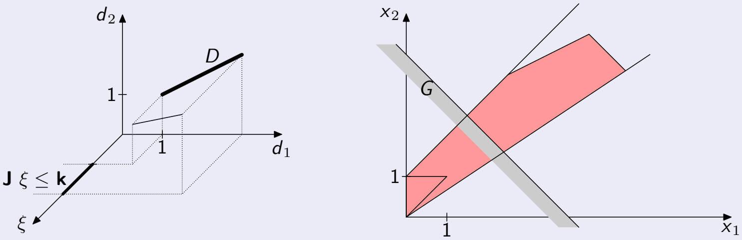


Abstract Acceleration with Numerical Inputs

Translations with Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \le \mathbf{v}}_{G} \;\wedge\; \mathbf{J}\boldsymbol{\xi} \le \mathbf{k} \quad \rightarrow \quad \mathbf{x}' = \mathbf{x} + \underbrace{\mathbf{T}\boldsymbol{\xi} + \mathbf{u}}_{D}$
- $\tau(X) = (X \sqcap G) + D \qquad D = \{\mathbf{d} \mid \exists \boldsymbol{\xi} : \mathbf{J}\boldsymbol{\xi} \le \mathbf{k} \wedge \mathbf{d} = \mathbf{T}\boldsymbol{\xi} + \mathbf{u}\}$
- $\tau^{\otimes}(X) = \boxed{X \sqcup \tau((X \sqcap G) \nearrow D)}$

Example: $\tau(X) : \left| \begin{array}{c} x_1 + x_2 \le 4 \\ 1 \le \xi \le 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$



Abstract Acceleration with Numerical Inputs

Translations with Resets and Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \leq \mathbf{v}}_{G} \;\wedge\; \mathbf{J}\xi \leq \mathbf{k} \quad \rightarrow \quad \mathbf{x}' = \mathbf{Cx} + \underbrace{\mathbf{T}\xi + \mathbf{u}}_{D}$

$$\text{Example: } \tau(X) : \left| \begin{array}{r} x_1 + 2x_2 \leq 3 \\ 0 \leq \xi \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi \end{array} \right.$$

Abstract Acceleration with Numerical Inputs

Translations with Resets and Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \leq \mathbf{v}}_{G} \;\wedge\; \mathbf{J}\boldsymbol{\xi} \leq \mathbf{k} \quad\rightarrow\quad \mathbf{x}' = \mathbf{Cx} + \underbrace{\mathbf{T}\boldsymbol{\xi} + \mathbf{u}}_{D}$
- $\tau(X) = (X \sqcap G)^t + D \qquad D = \{\mathbf{d} \mid \exists \boldsymbol{\xi} : \mathbf{J}\boldsymbol{\xi} \leq \mathbf{k} \wedge \mathbf{d} = \mathbf{T}\boldsymbol{\xi} + \mathbf{u}\}$

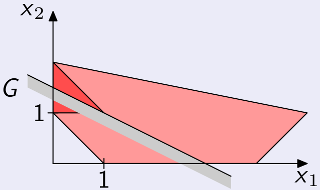
$$\text{Example: } \tau(X) : \left| \begin{array}{c} x_1 + 2x_2 \leq 3 \\ 0 \leq \xi \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi \end{array} \right.$$

Abstract Acceleration with Numerical Inputs

Translations with Resets and Simple Guards

- $\tau : \underbrace{\mathbf{Ax} \leq \mathbf{v}}_{G} \ \wedge\ \mathbf{J}\xi \leq \mathbf{k} \quad \rightarrow \quad \mathbf{x}' = \mathbf{Cx} + \underbrace{\mathbf{T}\xi + \mathbf{u}}_{D}$
- $\tau(X) = (X \sqcap G)^t + D \qquad D = \{\mathbf{d} \mid \exists \xi : \mathbf{J}\xi \leq \mathbf{k} \wedge \mathbf{d} = \mathbf{T}\xi + \mathbf{u}\}$
- $\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \tau\Big(\big((\tau(X) \sqcap G)^t \nearrow D^t\big) + D^r\Big)$

Example: $\tau(X) : \left| \begin{array}{c} x_1 + 2x_2 \leq 3 \\ 0 \leq \xi \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi \end{array} \right.$



Abstract Acceleration with Numerical Inputs

General Guards

- $\tau : \begin{pmatrix} \mathbf{A} & \mathbf{L} \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \xi \end{pmatrix} \leq \begin{pmatrix} \mathbf{v} \\ \mathbf{k} \end{pmatrix} \quad \rightarrow \quad \mathbf{x}' = \begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \xi \end{pmatrix} + \mathbf{u}$
- $L \neq 0 \Longrightarrow$ can code a *general affine transformation* by a reset to input:
  - $(\mathbf{Ax} \leq \mathbf{v}) \rightarrow \mathbf{x}' = \mathbf{Cx} + \mathbf{d} \quad \Longleftrightarrow \quad (\mathbf{Ax} \leq \mathbf{v} \wedge \xi = \mathbf{Cx} + \mathbf{d}) \rightarrow \mathbf{x}' = \xi$
- $\rightarrow$ Not accelerable

Abstract Acceleration with Numerical Inputs

General Guards

- $\tau : \begin{pmatrix} \mathbf{A} & \mathbf{L} \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \xi \end{pmatrix} \leq \begin{pmatrix} \mathbf{v} \\ \mathbf{k} \end{pmatrix} \quad \rightarrow \quad \mathbf{x}' = \begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \xi \end{pmatrix} + \mathbf{u}$
- $L \neq 0 \implies$ can code a *general affine transformation* by a reset to input:
  - $(\mathbf{Ax} \leq \mathbf{v}) \rightarrow \mathbf{x}' = \mathbf{Cx} + \mathbf{d} \quad \Longleftrightarrow \quad (\mathbf{Ax} \leq \mathbf{v} \wedge \xi = \mathbf{Cx} + \mathbf{d}) \rightarrow \mathbf{x}' = \xi$
- $\rightarrow$ Not accelerable

Weakened Guards

- $\overline{G} = \underbrace{(\exists \xi : G)}_{\mathbf{A}'\mathbf{x} \leq \mathbf{v}'} \wedge \underbrace{(\exists \mathbf{x} : G)}_{\mathbf{J}'\xi \leq \mathbf{k}'}$
- Resort to methods for translation and translation/reset

Outline

Comparison with Widening I

Widening

1. Delayed by N iterations
2. Widening until convergence
3. Descending iterations

Comparison with Widening I

Widening

1. Delayed by N iterations
2. Widening until convergence
3. Descending iterations

Comparison: Translation with general guard

$$\tau(X) : \left| \begin{array}{rcl} 2x_1 + x_2 + \xi & \leq & 6 \\ x_2 - \xi & \leq & 2 \\ 0 \leq \xi & \leq & 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = x_2 + 1 \end{array} \right.$$

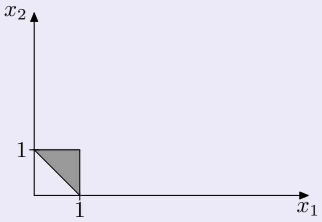
Comparison with Widening I

Widening

1. Delayed by N iterations
2. Widening until convergence
3. Descending iterations

Comparison: Translation with general guard

$$\tau(X) : \left| \begin{array}{rcl} 2x_1 + x_2 + \xi & \leq & 6 \\ x_2 - \xi & \leq & 2 \\ 0 \leq \xi & \leq & 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = x_2 + 1 \end{array} \right.$$



Comparison with Widening I

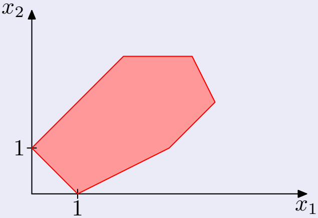
Widening

1. Delayed by N iterations
2. Widening until convergence
3. Descending iterations

Comparison: Translation with general guard

$$\tau(X) : \left| \begin{array}{rcl} 2x_1 + x_2 + \xi & \leq & 6 \\ x_2 - \xi & \leq & 2 \\ 0 \leq \xi & \leq & 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = x_2 + 1 \end{array} \right.$$

- Convex hull of the exact result



Comparison with Widening I

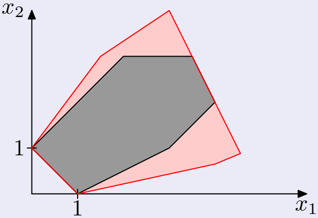
Widening

1. Delayed by N iterations
2. Widening until convergence
3. Descending iterations

Comparison: Translation with general guard

$$\tau(X) : \left| \begin{array}{rcl} 2x_1 + x_2 + \xi & \leq & 6 \\ x_2 - \xi & \leq & 2 \\ 0 \leq \xi & \leq & 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = x_2 + 1 \end{array} \right.$$

- Convex hull of the exact result
- Widening with delay $N = 0$ and 3 descending iterations



Comparison with Widening I

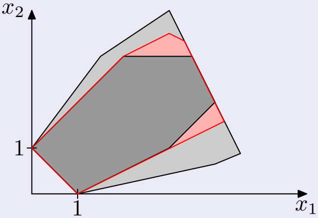
Widening

1. Delayed by N iterations
2. Widening until convergence
3. Descending iterations

Comparison: Translation with general guard

$$\tau(X) : \left| \begin{array}{rcl} 2x_1 + x_2 + \xi & \leq & 6 \\ x_2 - \xi & \leq & 2 \\ 0 \leq \xi & \leq & 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = x_2 + 1 \end{array} \right.$$

- Convex hull of the exact result
- Widening with delay $N = 0$ and 3 descending iterations
- Abstract acceleration

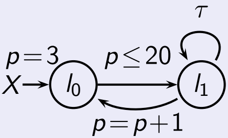


Comparison with Widening II

Comparison: Nested loops

$$\tau : \left| \begin{array}{rcl} 2x_1 + 2x_2 & \leq & p \\ 0 \leq \xi & \leq & 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi \\ p' = p \end{array} \right.$$

- Widening with delay $N = 2$ and one descending iteration: $\{0 \leq x_1 \wedge 1 \leq x_1 + x_2 \wedge 3 \leq p\}$

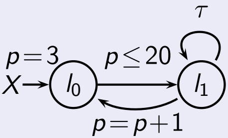


Comparison with Widening II

Comparison: Nested loops

$$\tau : \left| \begin{array}{rcl} 2x_1 + 2x_2 & \leq & p \\ 0 \leq \xi & \leq & 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi \\ p' = p \end{array} \right.$$

- Widening with delay $N = 2$ and one descending iteration: $\{0 \leq x_1 \wedge 1 \leq x_1 + x_2 \wedge 3 \leq p\}$
- *Inner loop* with abstract acceleration. *Outer loop:* widening with one descending iteration: $\{0 \leq x_1 \leq 12 \wedge 0 \leq x_2 \leq 3 \wedge 3 \leq p \leq 20\}$ (over-approximated)



Conclusion

Abstract acceleration with numerical inputs

Acceleration vs. Widening

- Different principles:
  - *Acceleration* is based on the program structure, whereas
  - *widening* is based on the structure of the abstract domain.
- Approximations are more predictible:
  - Widening is not monotonous – acceleration is.
- Acceleration of inner loops facilitates widening in nested loops situations.
  - Acceleration also applied in descending iterations.
- Widening needed for
  - handling non-accelerable transitions
  - ensuring convergence in the case of multiple self-loops and nested loops

Outline

Application of Acceleration to Data-Flow Programs

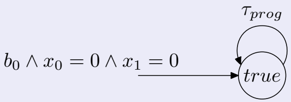
Application to e.g. Lustre programs?

Issues

1. Input variables:
   - Boolean variables:
     - Encoded as non-determinism in the control flow graph (CFG)
   - Numerical variables $\surd$
2. *Implicit* control flow → Discover a CFG w.r.t. Boolean variables
   1. **Conventional approach:**
      1. Reduction to numerical automaton by *enumeration* of Boolean states
      2. → Combinatorial explosion
   2. **Our approach:**
      1. Symbolic handling of Boolean variables
      2. Approximation method: Decoupling
      3. Controlled partitioning using heuristics

Conventional Approach

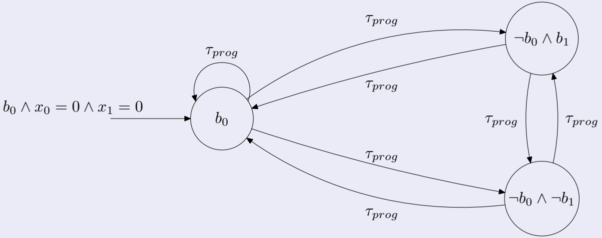
Transformations



Conventional Approach

Transformations

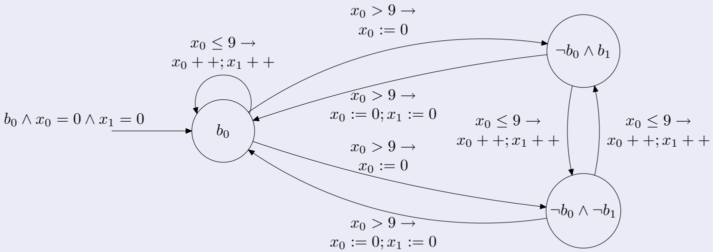
1. Boolean state space enumeration



Conventional Approach

Transformations

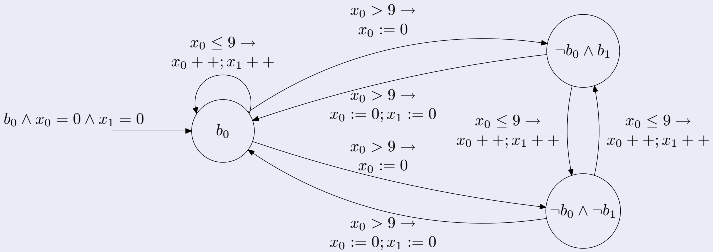
1. Boolean state space enumeration
2. Transition refinement by source and destination location



Conventional Approach

Transformations

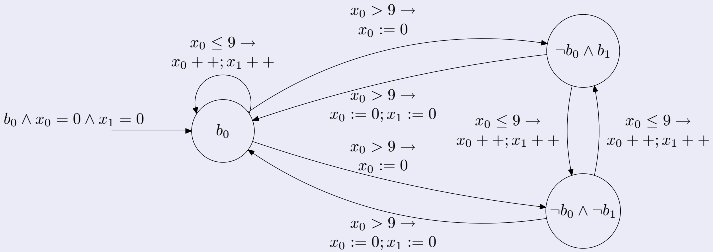
1. Boolean state space enumeration
2. Transition refinement by source and destination location
3. Elimination of Boolean input variables



Conventional Approach

Transformations

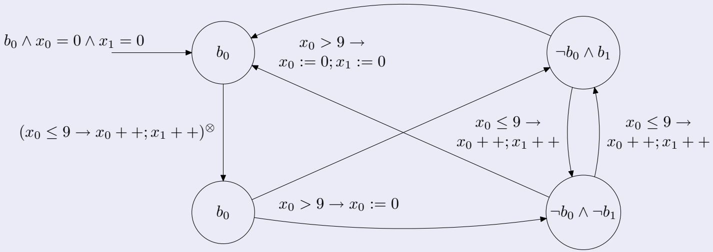
1. Boolean state space enumeration
2. Transition refinement by source and destination location
3. Elimination of Boolean input variables
4. Convexification of numerical guards



Conventional Approach

Transformations

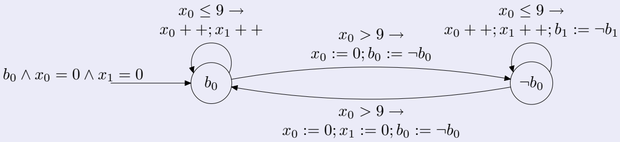
1. Boolean state space enumeration
2. Transition refinement by source and destination location
3. Elimination of Boolean input variables
4. Convexification of numerical guards
5. "Flattening" of accelerable self-loops



Partitioning and Acceleration

Intuition: Self-loops with Boolean Identity

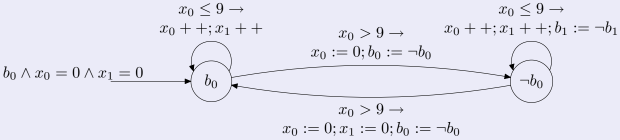
- Partition until we have a CFG where the Boolean part of the transition function is the identity.



Partitioning and Acceleration

Intuition: Self-loops with Boolean Identity

- Partition until we have a CFG where the Boolean part of the transition function is the identity.

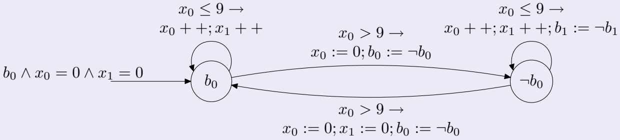


- Partitioning not necessary at all!!!

Partitioning and Acceleration

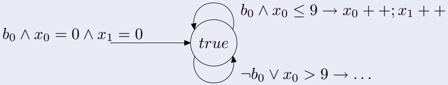
Intuition: Self-loops with Boolean Identity

- Partition until we have a CFG where the Boolean part of the transition function is the identity.



- Partitioning not necessary at all!!!

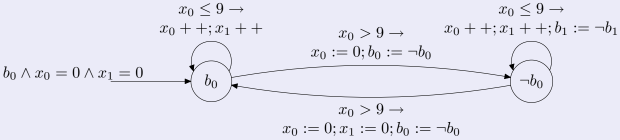
Acceleration without Partitioning



Partitioning and Acceleration

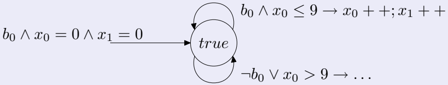
Intuition: Self-loops with Boolean Identity

- Partition until we have a CFG where the Boolean part of the transition function is the identity.



- Partitioning not necessary at all!!!

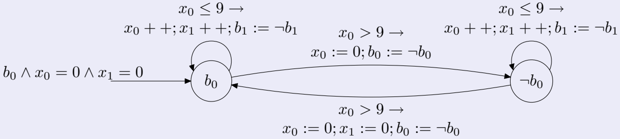
Acceleration without Partitioning



- Partitioning and acceleration are orthogonal!

Decoupling

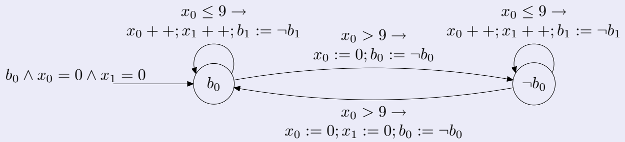
Problem



- Boolean identity too restrictive $\longrightarrow$ Rarely applicable

Decoupling

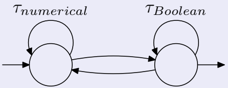
Problem



- Boolean identity too restrictive → Rarely applicable

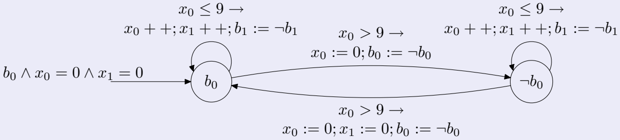
Idea: Decoupling

- Decoupling numerical and Boolean parts of the transition function → approximation



Decoupling

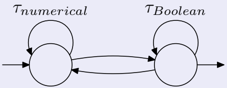
Problem

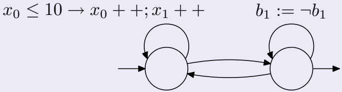


- Boolean identity too restrictive $\longrightarrow$ Rarely applicable

Idea: Decoupling

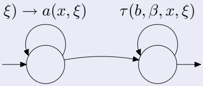
- Decoupling numerical and Boolean parts of the transition function $\longrightarrow$ approximation





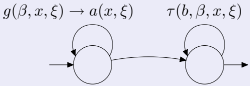
Decoupling Variants

- Numerical equations independent of Boolean equations

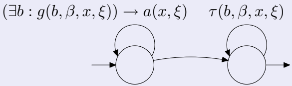


Decoupling Variants

- Numerical equations independent of Boolean equations

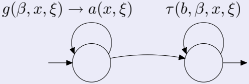


- Inputization of Boolean state variables in Numerical equations

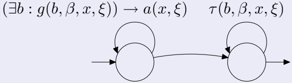


Decoupling Variants

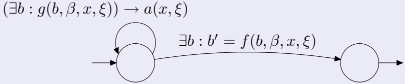
- Numerical equations independent of Boolean equations



- Inputization of Boolean state variables in Numerical equations

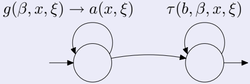


- Inputization of unstable Boolean state variables in Boolean equations

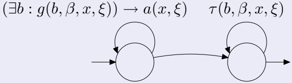


Decoupling Variants

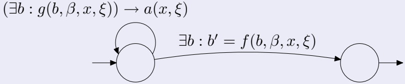
- Numerical equations independent of Boolean equations



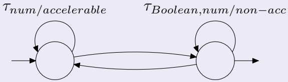
- Inputization of Boolean state variables in Numerical equations



- Inputization of unstable Boolean state variables in Boolean equations



- Decoupling accelerable and non-accelerable equations



Outline

Why partitioning?

Gain in precision!

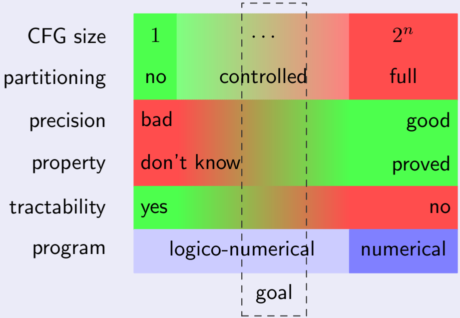
1. More targeted application of *widening* (at loop heads only)
2. Explicit *disjunctive* abstract domain → Less precision loss in unions

Why partitioning?

Gain in precision!

1. More targeted application of *widening* (at loop heads only)
2. Explicit *disjunctive* abstract domain → Less precision loss in unions

But...



Partitioning by Numerical Actions

Idea: Equivalence classes w.r.t. numerical actions

- Intuition: Same set of actions executed in the same Boolean states.

$$b_1 \sim b_2 \Leftrightarrow \begin{cases} \forall \beta, \mathcal{C}: & \mathcal{A}(b_1, \beta, \mathcal{C}) \Rightarrow \mathcal{A}(b_2, \beta, \mathcal{C}) \wedge f^x(b_1, \beta, \mathcal{C}) = f^x(b_2, \beta, \mathcal{C}) \\ \quad \textit{and} & \textit{vice versa} \end{cases}$$

Partitioning by Numerical Actions

Idea: Equivalence classes w.r.t. numerical actions

- Intuition: Same set of actions executed in the same Boolean states.

$$b_1 \sim b_2 \Leftrightarrow \begin{cases} \forall \beta, \mathcal{C}: & \mathcal{A}(b_1,\beta,\mathcal{C}) \Rightarrow \mathcal{A}(b_2,\beta,\mathcal{C}) \wedge f^x(b_1,\beta,\mathcal{C}) = f^x(b_2,\beta,\mathcal{C}) \\ \quad \textit{and} & \textit{vice versa} \end{cases}$$

- Example:

$$x_0' = \begin{cases} x_0+1 & \text{if } \neg b_0 \wedge \neg b_1 \wedge x_0 \leq 10 \wedge \beta \vee b_0 \wedge \neg b_1 \wedge x_0 \leq 20 \\ 0 & \text{if } \neg b_0 \wedge \neg b_1 \wedge x_0 > 10 \wedge x_1 > 10 \\ x_0 & \text{else} \end{cases}$$

$$x_1' = \begin{cases} x_1+1 & \text{if } \neg b_0 \wedge \neg b_1 \wedge x_1 \leq 10 \wedge \neg\beta \\ x_1 & \text{else} \end{cases}$$

$$x_2' = \begin{cases} x_2+1 & \text{if } \neg b_0 \wedge \neg b_1 \wedge (x_0 \leq 10 \wedge \beta \vee x_1 \leq 10 \wedge \neg\beta) \vee b_0 \wedge \neg b_1 \\ x_2 & \text{else} \end{cases}$$

Partitioning by Numerical Actions

Idea: Equivalence classes w.r.t. numerical actions

- Intuition: Same set of actions executed in the same Boolean states.

$$b_1 \sim b_2 \Leftrightarrow \left\{ \begin{array}{rl} \forall \beta, \mathcal{C}: & \mathcal{A}(b_1, \beta, \mathcal{C}) \Rightarrow \mathcal{A}(b_2, \beta, \mathcal{C}) \wedge f^x(b_1, \beta, \mathcal{C}) = f^x(b_2, \beta, \mathcal{C}) \\ \textit{and} & \textit{vice versa} \end{array} \right.$$

- Example:

$$\begin{array}{lll} [\neg b_0 \wedge \neg b_1] & (x_0', x_1', x_2') = & \left\{ \begin{array}{llll} (x_0+1, & x_1, & x_2+1) & \text{if } x_0 \leq 10 \wedge \beta \\ (x_0, & x_1+1, & x_2+1) & \text{if } x_1 \leq 10 \wedge \neg\beta \\ (0, & x_1, & x_2) & \text{if } x_0 > 10 \wedge x_1 > 10 \\ (x_0, & x_1, & x_2) & \text{else} \end{array} \right. \\ [b_0 \wedge \neg b_1] & (x_0', x_1', x_2') = & \left\{ \begin{array}{llll} (x_0+1, & x_1, & x_2+1) & \text{if } x_0 \leq 20 \\ (x_0, & x_1, & x_2+1) & \text{if } x_0 > 20 \\ (x_0, & x_1, & x_2) & \text{else} \end{array} \right. \\ [b_0 \wedge b_1] & (x_0', x_1', x_2') = & \left\{ \begin{array}{lll} (x_0, & x_1, & x_2) \end{array} \right. \end{array}$$

Partitioning by Numerical Actions

Idea: Equivalence classes w.r.t. numerical actions

- Intuition: Same set of actions executed in the same Boolean states.

$$b_1 \sim b_2 \Leftrightarrow \begin{cases} \forall \beta, \mathcal{C} : & \mathcal{A}(b_1, \beta, \mathcal{C}) \Rightarrow \mathcal{A}(b_2, \beta, \mathcal{C}) \wedge f^x(b_1, \beta, \mathcal{C}) = f^x(b_2, \beta, \mathcal{C}) \\ \quad \textit{and} & \textit{vice versa} \end{cases}$$

- Example:

$$\begin{array}{ll} [\neg b_0 \wedge \neg b_1] & (x_0', x_1', x_2') = \begin{cases} (x_0 + 1, & x_1, & x_2 + 1) & \text{if } x_0 \leq 10 \wedge \beta \\ (x_0, & x_1 + 1, & x_2 + 1) & \text{if } x_1 \leq 10 \wedge \neg\beta \\ (0, & x_1, & x_2) & \text{if } x_0 > 10 \wedge x_1 > 10 \\ (x_0, & x_1, & x_2) & \text{else} \end{cases} \\ [b_0 \wedge \neg b_1] & (x_0', x_1', x_2') = \begin{cases} (x_0 + 1, & x_1, & x_2 + 1) & \text{if } x_0 \leq 20 \\ (x_0, & x_1, & x_2 + 1) & \text{if } x_0 > 20 \\ (x_0, & x_1, & x_2) & \text{else} \end{cases} \\ [b_0 \wedge b_1] & (x_0', x_1', x_2') = \{ \; (x_0, \quad x_1, \quad x_2) \end{array}$$

- Nice property: Numerical equations independent of Boolean equations.

Partitioning by Numerical Actions

Idea: Equivalence classes w.r.t. numerical actions

- Intuition: Same set of actions executed in the same Boolean states.

$$b_1 \sim b_2 \Leftrightarrow \left\{ \begin{array}{ll} \forall \beta, \mathcal{C}: & \mathcal{A}(b_1,\beta,\mathcal{C}) \Rightarrow \mathcal{A}(b_2,\beta,\mathcal{C}) \wedge f^x(b_1,\beta,\mathcal{C}) = f^x(b_2,\beta,\mathcal{C}) \\ \textit{and} & \textit{vice versa} \end{array} \right.$$

- Example:

$$\begin{array}{lll}
[\neg b_0 \wedge \neg b_1] & (x_0',x_1',x_2') = & \left\{ \begin{array}{llll} (x_0+1, & x_1, & x_2+1) & \text{if } x_0 \leq 10 \wedge \beta \\ (x_0, & x_1+1, & x_2+1) & \text{if } x_1 \leq 10 \wedge \neg\beta \\ (0, & x_1, & x_2) & \text{if } x_0 > 10 \wedge x_1 > 10 \\ (x_0, & x_1, & x_2) & \text{else} \end{array} \right. \\
[b_0 \wedge \neg b_1] & (x_0',x_1',x_2') = & \left\{ \begin{array}{llll} (x_0+1, & x_1, & x_2+1) & \text{if } x_0 \leq 20 \\ (x_0, & x_1, & x_2+1) & \text{if } x_0 > 20 \\ (x_0, & x_1, & x_2) & \text{else} \end{array} \right. \\
[b_0 \wedge b_1] & (x_0',x_1',x_2') = & \left\{ \begin{array}{lll} (x_0, & x_1, & x_2) \end{array} \right.
\end{array}$$

- Nice property: Numerical equations independent of Boolean equations.
- Variants: Different quantifications

Partitioning by Numerical Actions

Idea: Equivalence classes w.r.t. numerical actions

- Intuition: Same set of actions executed in the same Boolean states.

$$b_1 \sim b_2 \Leftrightarrow \begin{cases} \forall \beta, \mathcal{C}: & \mathcal{A}(b_1,\beta,\mathcal{C}) \Rightarrow \mathcal{A}(b_2,\beta,\mathcal{C}) \wedge f^x(b_1,\beta,\mathcal{C}) = f^x(b_2,\beta,\mathcal{C}) \\ \quad \textit{and} & \textit{vice versa} \end{cases}$$

- Example:

$$[\neg b_0 \wedge \neg b_1] \quad (x_0', x_1', x_2') = \begin{cases} (x_0+1, & x_1, & x_2+1) & \text{if } x_0 \leq 10 \wedge \beta \\ (x_0, & x_1+1, & x_2+1) & \text{if } x_1 \leq 10 \wedge \neg\beta \\ (0, & x_1, & x_2) & \text{if } x_0 > 10 \wedge x_1 > 10 \\ (x_0, & x_1, & x_2) & \text{else} \end{cases}$$

$$[b_0 \wedge \neg b_1] \quad (x_0', x_1', x_2') = \begin{cases} (x_0+1, & x_1, & x_2+1) & \text{if } x_0 \leq 20 \\ (x_0, & x_1, & x_2+1) & \text{if } x_0 > 20 \\ (x_0, & x_1, & x_2) & \text{else} \end{cases}$$

$$[b_0 \wedge b_1] \quad (x_0', x_1', x_2') = \{ (x_0, \; x_1, \; x_2)$$

- Nice property: Numerical equations independent of Boolean equations.
- Variants: Different quantifications
- Refinement by Boolean backward bisimulation

Some Experimental Results

- Tool NBACCEL based on the abstract domain library BDDAPRON
- Small, but difficult benchmarks

	Boolean states	time NBACCEL	time NBAC
Escalator 1	9	0.54	–
Escalator 2	259	3.98	1.48
Gate 1	5	0.54	–
Traffic 1	13	0.45	3.52
Traffic 2	16	1.74	–

- Larger benchmarks

	Boolean states	time NBACCEL	time NBAC
LCM quest 0a	72	0.07	0.06
LCM quest 0b	541	0.20	0.31
LCM quest 0c	16432	0.32	0.49
LCM quest 1	32992	2.23	3.46
LCM quest 2	33013	5.22	15.14

Outline

Conclusion

Application of abstract acceleration to logico-numerical data-flow programs

Decoupling and Partitioning

- Acceleration can be applied independently of partitioning.
- Decoupling enlarges the applicability of acceleration.
- Partitioning heuristics w.r.t. numerical actions

Conclusion

Application of abstract acceleration to logico-numerical data-flow programs

Decoupling and Partitioning

- Acceleration can be applied independently of partitioning.
- Decoupling enlarges the applicability of acceleration.
- Partitioning heuristics w.r.t. numerical actions

Current and Future Work

- Combination with dynamic partitioning (also using numerical constraints)
- Backward acceleration
- Application to discretized hybrid systems
  - Non-standard semantics (Benveniste, Caillaud and Pouzet 2010)

Using Abstract Acceleration in the Verification of Logico-Numerical Data-Flow Programs

Peter Schrammel and Bertrand Jeannet
{peter.schrammel,bertrand.jeannet}@inria.fr

INRIA Rhône-Alpes

Synchron'10