Discrete Controller Synthesis for Logico-numerical Systems with ReaX

Nicolas BERTHIER Hervé MARCHAND

February 4, 2014

Ctrl-A Seminar

Outline

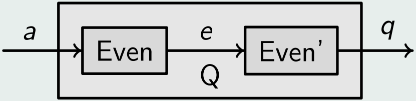
- Discrete Controller Synthesis for Logico-numerical Programs
- State of the Art
- Discrete Controller Synthesis Principles
- ReaX: Technical Choices, Implementation & Evaluations
- Conclusions

Outline

- Discrete Controller Synthesis for Logico-numerical Programs
  - Boolean Reactive/Synchronous Programs
  - Logico-numerical Reactive/Synchronous Programs
  - Discrete Controller Synthesis Problem Statement
- State of the Art
- Discrete Controller Synthesis Principles
- ReaX: Technical Choices, Implementation & Evaluations
- Conclusions

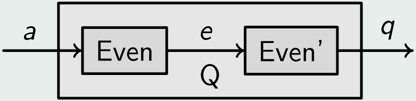
Example Finite Reactive/Synchronous Program

Synchronous Circuit

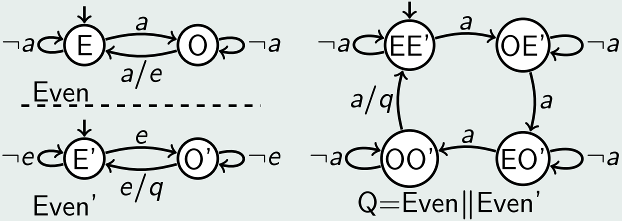


Example Finite Reactive/Synchronous Program

Synchronous Circuit

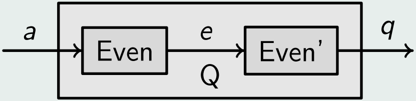


Automata

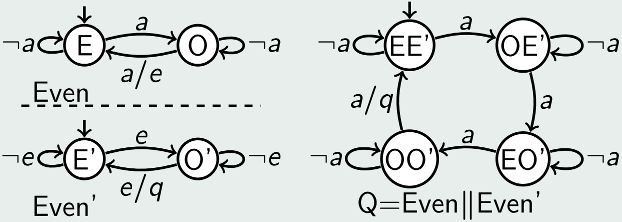


Example Finite Reactive/Synchronous Program

Synchronous Circuit



Automata



Equations

$$\rightsquigarrow \quad \begin{cases} x' = (\neg x \wedge a) \vee (x \wedge \neg a) & e = x \wedge a & \neg x_0 & \leftarrow \text{Even} \\ y' = (\neg y \wedge e) \vee (y \wedge \neg e) & q = y \wedge e & \neg y_0 & \leftarrow \text{Even'} \end{cases}$$

Example Finite Reactive/Synchronous Program (cont'd)

Equations

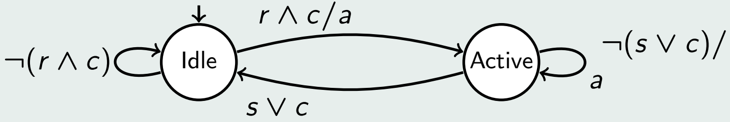
$$\begin{cases} x' = (\neg x \wedge a) \vee (x \wedge \neg a) & e = x \wedge a & \neg x_0 \\ y' = (\neg y \wedge e) \vee (y \wedge \neg e) & q = y \wedge e & \neg y_0 \end{cases}$$

Textual Form (nbac Format)

```
state x, y: bool;                              (* State Variables *)
      q: bool;                                        (* (Output) *)
input a: bool;
local e: bool;          (* Substituted in Expressions Below *)
definition
  e = x and a;
transition (* Transition Function, Assigning Every State *)
  x' = (not x and a) or (x and not a);       (* ... Variable *)
  y' = (not y and e) or (y and not e);
  q' = y and e;
initial not x and not y;      (* Initial State(s) Predicate *)
assertion true;                          (* Assertion Predicate *)
```

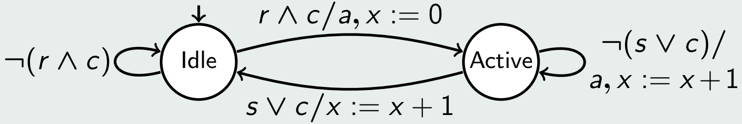
Example Logico-numerical Reactive/Synchronous Program

Example Logico-numerical Task Model



Example Logico-numerical Reactive/Synchronous Program

Example Logico-numerical Task Model



Expressing Reactive/Synchronous Programs

Definition (Symbolic Transition System — STS)

$S = \langle X, I, T, A, \Theta_0 \rangle$ where:

- $X = \langle x_1, \ldots, x_n \rangle$ ← Vector of State Variables
- $I = \langle i_1, \ldots, i_m \rangle$ ← Vector of Input Variables
- $T = \langle t_1, \ldots, t_n \rangle$, t_i: Expression on $X \cup I$ ← Transition Function
- A: Predicate on $X \cup I$ ← Assertion
- Θ_0: Predicate on X ← Initial State(s)

Definition (Arithmetic Symbolic Transition System — ASTS)

STS S $(= \langle X, I, T, A, \Theta_0 \rangle)$ where:

- $X = \mathbb{B}^s \cup \mathbb{R}^{s'} \cup \mathbb{Z}^{s''}$ $(s + s' + s'' = n)$
- $I = \mathbb{B}^t \cup \mathbb{R}^{t'} \cup \mathbb{Z}^{t''}$ $(t + t' + t'' = m)$
- T, A and Θ_0 Involve Arithmetic Expressions

Reactive/Synchronous Program Invariant

Definition (Invariant Property for an STS)

Given an ASTS $S = \langle X, I, T, A, \Theta_0 \rangle$, a Predicate Φ over X is an *Invariant* of S (Noted $S \models \Phi$) iff:

- "All Executions of S Remain in States Satisfying Φ" *i.e.,*

$\forall x_0 \in \mathcal{D}_X$ ← Initial State

$\forall (\iota_0, \ldots, \iota_p) \in \mathcal{D}_I^p$ ← Sequence of p Vectors of Inputs

$$\Theta_0(x_0) \wedge \forall i \in [0, p], A(T(\ldots T(x_0, \iota_0) \ldots, \iota_i))$$

$$\Rightarrow \forall i \in [0, p], \Phi(T(\ldots T(x_0, \iota_0) \ldots, \iota_i))$$

Discrete Controller Synthesis (DCS) Problem

- Initiated by Ramadge and Wonham 1989[1] (Formal Languages Setting)

Controller Synthesis Problem for Invariant Enforcement in ASTSs

Given an ASTS $S = \langle X, I_{uc} \uplus I_c, T, A, \Theta_0 \rangle$ and an Invariant Φ over X, Compute a Predicate A_Φ such that:

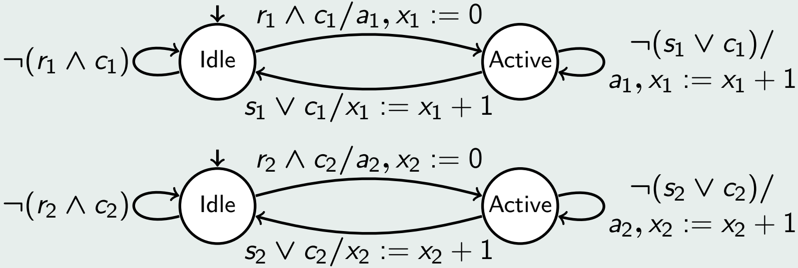
$$S' = \langle X, I_{uc} \uplus I_c, T, A_\Phi, \Theta_0 \rangle \models \Phi$$

- I_{uc} $\leftarrow$ *Non-controllable Input Variables*
- I_c $\leftarrow$ *Controllable Input Variables*

[1] Peter J. G. Ramadge and W. Murray Wonham. "The control of discrete event systems". In: *Proceedings of the IEEE* 77.1 (Jan. 1989), pp. 81–98.

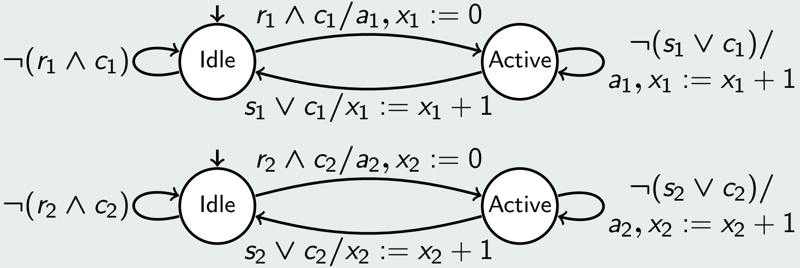
Discrete Controller Synthesis (DCS) Problem (cont'd)

Example DCS Problem for a Logico-numerical Program



Discrete Controller Synthesis (DCS) Problem (cont'd)

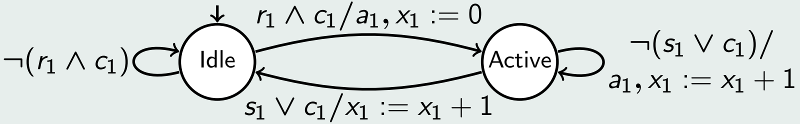
Example DCS Problem for a Logico-numerical Program

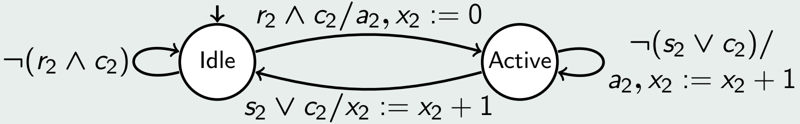


- $X = \langle t_1, t_2, x_1, x_2, a_1, a_2 \rangle$ $\qquad \mathcal{D}_X = \{\text{Idle}, \text{Active}\}^2 \times \mathbb{Z}^2 \times \mathbb{B}^2$
- $I_{uc} = \langle r_1, r_2, s_1, s_2 \rangle, I_c = \langle c_1, c_2 \rangle$ $\qquad \mathcal{D}_{I_{uc}} = \mathbb{B}^4,\ \mathcal{D}_{I_c} = \mathbb{B}^2$

Discrete Controller Synthesis (DCS) Problem (cont'd)

Example DCS Problem for a Logico-numerical Program





- $X = \langle t_1, t_2, x_1, x_2, a_1, a_2 \rangle$ $\mathcal{D}_X = \{\text{Idle}, \text{Active}\}^2 \times \mathbb{Z}^2 \times \mathbb{B}^2$
- $I_{uc} = \langle r_1, r_2, s_1, s_2 \rangle, I_c = \langle c_1, c_2 \rangle$ $\mathcal{D}_{I_{uc}} = \mathbb{B}^4, \mathcal{D}_{I_c} = \mathbb{B}^2$
- Enforcing Mutual Exclusion Between Active States:
$$\Phi(\langle t_1, t_2, x_1, x_2, a_1, a_2 \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle})$$
- Enforcing Bounds on x_i's:
$$\Phi(\langle t_1, t_2, x_1, x_2, a_1, a_2 \rangle) = (x_1 \leqslant 10 \wedge x_2 \leqslant 10)$$

Outline

- Discrete Controller Synthesis for Logico-numerical Programs
- State of the Art
  - For Other Kinds of Models
  - For Synchronous Languages
  - For Infinite-State Systems
- Discrete Controller Synthesis Principles
- ReaX: Technical Choices, Implementation & Evaluations
- Conclusions

State of the Art: Actual Tools

- Petrify[2]
- SynthKro[3]
- Supremica[4]
- Uppaal-tiga[5]
- ...

---

[2] Jordi Cortadella et al. "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers". In: *IEICE Transactions on Information and Systems* 80.3 (1997), pp. 315–325.

[3] Karine Altisen and Stavros Tripakis. "Tools for controller synthesis of timed systems". In: *Proceedings of the 2nd Workshop on Real-Time Tools (RT-TOOLS'02)*. 2002, pp. 2002–025.

[4] Knut Akesson et al. "Supremica: an integrated environment for verification, synthesis and simulation of discrete event systems". In: *8th International Workshop on Discrete Event Systems*. IEEE. 2006, pp. 384–385.

[5] Gerd Behrmann et al. "Uppaal-tiga: Time for playing games!" In: *Computer Aided Verification*. Springer. 2007, pp. 121–125.

State of the Art: Actual Tools for Synchronous Languages

- Sigali[6]
  - Polynomial Dynamical Systems ($\mathbb{Z}/3\mathbb{Z}$)
  - Symbolic Computations using *Ternary Decision Diagrams*
  - Finite Systems
- $\ldots\!\!\!/\,.$

[6] Hervé Marchand et al. "Synthesis of Discrete-Event Controllers based on the Signal Environment". In: *Discrete Event Dynamic System: Theory and Applications* 10.4 (Oct. 2000), pp. 325–346.

State of the Art: Actual Tools for Infinite-State Systems

- Smacs[7]
  - *Explicit* Control Flow Graph (*e.g.*, $X = \mathit{Locations} \cup \mathbb{Z}^n$)
    - $\rightsquigarrow$ Boolean State Space Explosion
  - *Partial Observability* of the Variables
  - Using *Abstract Interpretation Techniques*

[7] Gabriel Kalyon et al. "Symbolic Supervisory Control of Infinite Transition Systems under Partial Observation using Abstract Interpretation". In: *Discrete Event Dynamic Systems : Theory and Applications* 22.2 (2011), pp. 121–161.

Outline

- Discrete Controller Synthesis for Logico-numerical Programs
- State of the Art
- Discrete Controller Synthesis Principles
  - Infinite Transition Systems
  - Algorithmic Principle
  - Finite Case
  - Infinite Case
- ReaX: Technical Choices, Implementation & Evaluations
- Conclusions

Infinite Transition Systems

- Reasoning about State Spaces

Definition (Controllable Infinite Transition System of an ASTS)

One Associates to an ASTS $S = \langle X, I_{uc} \uplus I_c, T, A, \Theta_0 \rangle$ a *Controllable Infinite Transition System* $[S] = \langle \mathcal{X}, \mathcal{I}, \mathcal{T}_S, \mathcal{A}_S, \mathcal{X}_0 \rangle$ where:

- $\mathcal{X} = \mathcal{D}_X$ ← State Space
- $\mathcal{I} = \mathcal{U} \times \mathcal{C}$
  - $\mathcal{U} = \mathcal{D}_{I_{uc}}$ ← Non-controllable Input Space
  - $\mathcal{C} = \mathcal{D}_{I_c}$ ← Controllable Input Space
- $\mathcal{T}_S \subseteq \mathcal{X} \times \mathcal{I} \to \mathcal{X}$ ← Transition Function

  $= \lambda(x, \iota).\,(t_j(x, \iota))_{j \in [1,n]}$
- $\mathcal{A}_S \subseteq \mathcal{X} \times \mathcal{I}$ ← Assertion on Environment

  $= \{(x, \iota) \mid A(x, \iota) = true\}$
- $\mathcal{X}_0 \subseteq \mathcal{X}$ ← Initial States

  $= \{x \in \mathcal{X} \mid \Theta_0(x) = true\}$

Discrete Controller Synthesis: Algorithmic Principle

- Let $Bad \stackrel{\text{def}}{=} \{x \in \mathcal{X} \mid \Phi(x) = \mathit{false}\}$ $\leftarrow$ States to Avoid
- Computing States I_{Bad} "Uncontrollably" Reaching Bad $\leftarrow$ Fix-point
  - Using Pre-image Function $\mathcal{T}_S^{-1} : \wp(\mathcal{X}) \to \wp(\mathcal{X} \times \mathcal{U} \times \mathcal{C})$

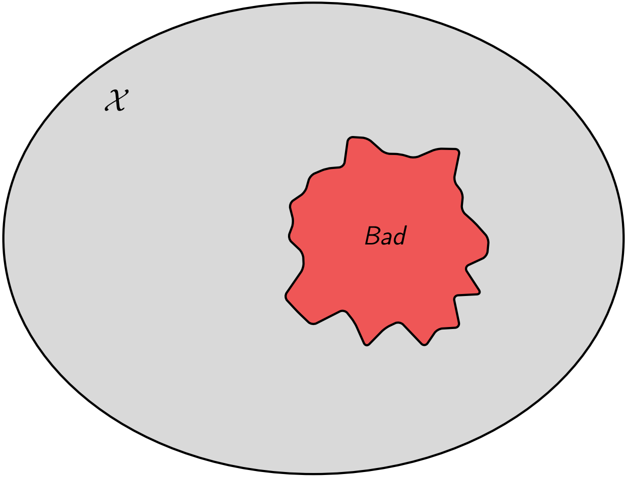
Discrete Controller Synthesis: Algorithmic Principle

- Let $Bad \stackrel{\text{def}}{=} \{x \in \mathcal{X} \mid \Phi(x) = \mathit{false}\}$ $\leftarrow$ States to Avoid
- Computing States I_{Bad} "Uncontrollably" Reaching Bad $\leftarrow$ Fix-point
  - Using Pre-image Function $\mathcal{T}_S^{-1} : \wp(\mathcal{X}) \rightarrow \wp(\mathcal{X} \times \mathcal{U} \times \mathcal{C})$

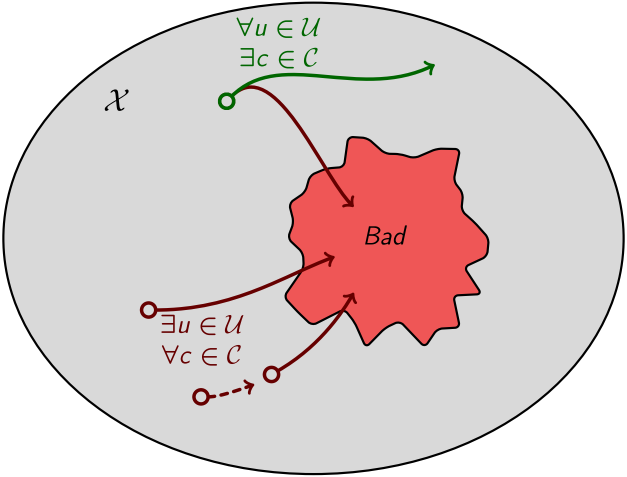
Finite Case

- State Variables on *Finite Domains* (*e.g.*, $\mathcal{X} = \mathbb{B}^n$)

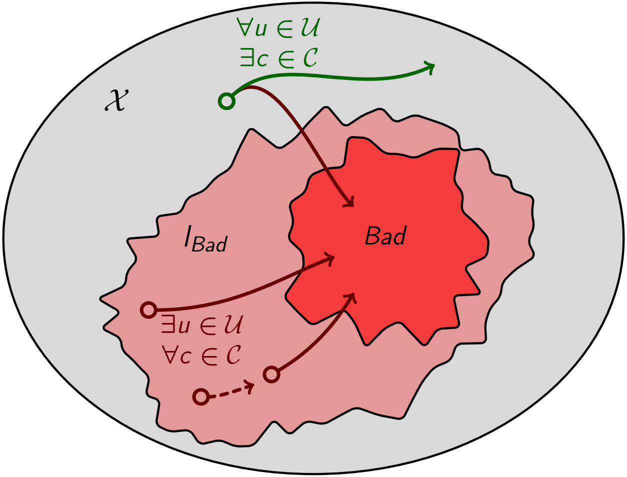
Discrete Controller Synthesis Principle in the Finite Case



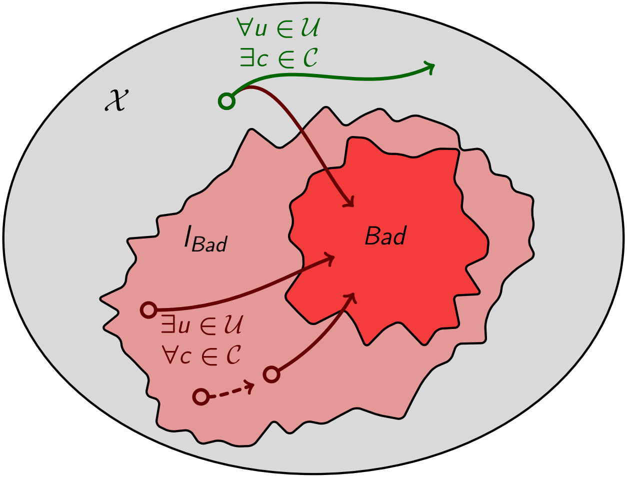
Discrete Controller Synthesis Principle in the Finite Case



Discrete Controller Synthesis Principle in the Finite Case



Discrete Controller Synthesis Principle in the Finite Case



$I_{Bad} = \mathrm{coreach}_u(Bad)$ $\qquad$ $\mathrm{coreach}_u(B) \stackrel{\text{def}}{=} \mathrm{lfp}(\lambda\beta.B \cup \mathrm{pre}_u(\beta))$

$$\mathrm{pre}_u(B) \stackrel{\text{def}}{=} \left\{x \in \mathcal{X} \,\middle|\, \exists u \in \mathcal{U}, \forall c \in \mathcal{C}, (x, u, c) \in \mathcal{T}_S^{-1}(B) \cap \mathcal{A}_S\right\}$$

Discrete Controller Synthesis: Algorithmic Principle

- Let $Bad \stackrel{\text{def}}{=} \{x \in \mathcal{X} \mid \Phi(x) = \mathit{false}\}$ ← States to Avoid
- Computing States I_{Bad} "Uncontrollably" Reaching *Bad* ← Fix-point
  - Using Pre-image Function $\mathcal{T}_S^{-1} \colon \wp(\mathcal{X}) \rightarrow \wp(\mathcal{X} \times \mathcal{U} \times \mathcal{C})$

Finite Case

- State Variables on *Finite Domains* $(e.g.,\ \mathcal{X} = \mathbb{B}^n)$

Discrete Controller Synthesis: Algorithmic Principle

- Let $Bad \stackrel{\text{def}}{=} \{x \in \mathcal{X} \mid \Phi(x) = \mathit{false}\}$ ← States to Avoid
- Computing States I_{Bad} "Uncontrollably" Reaching *Bad* ← Fix-point
  - Using Pre-image Function $\mathcal{T}_S^{-1} : \wp(\mathcal{X}) \to \wp(\mathcal{X} \times \mathcal{U} \times \mathcal{C})$
- Success iff $\mathcal{X}_0 \cap I_{Bad} = \varnothing$

  $\rightsquigarrow$ New Predicate ← Solution to the DCS Problem

Finite Case

- State Variables on *Finite Domains* (*e.g.*, $\mathcal{X} = \mathbb{B}^n$)

$\rightsquigarrow$ *Maximally Permissive Controller*

Discrete Controller Synthesis: Algorithmic Principle

- Let $Bad \stackrel{\text{def}}{=} \{x \in \mathcal{X} \mid \Phi(x) = \mathit{false}\}$ ← States to Avoid
- Computing States I_{Bad} "Uncontrollably" Reaching *Bad* ← Fix-point
  - Using Pre-image Function $\mathcal{T}_S^{-1} : \wp(\mathcal{X}) \rightarrow \wp(\mathcal{X} \times \mathcal{U} \times \mathcal{C})$
- Success iff $\mathcal{X}_0 \cap I_{Bad} = \varnothing$

  $\rightsquigarrow$ New Predicate ← Solution to the DCS Problem

Finite Case

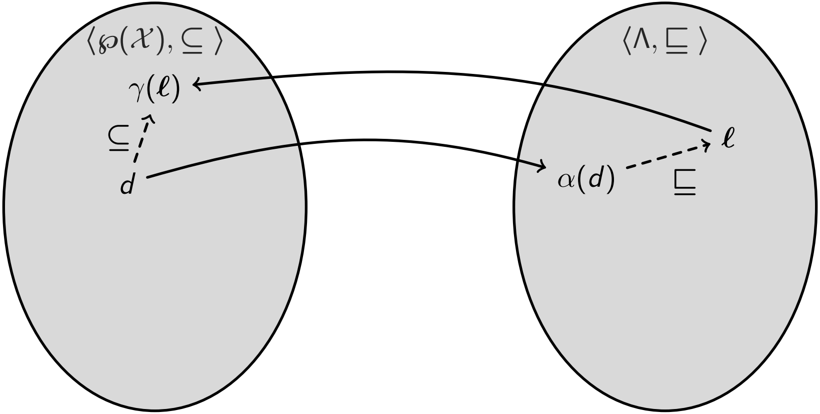
- State Variables on *Finite Domains* $(e.g., \mathcal{X} = \mathbb{B}^n)$

$\rightsquigarrow$ *Maximally Permissive Controller*

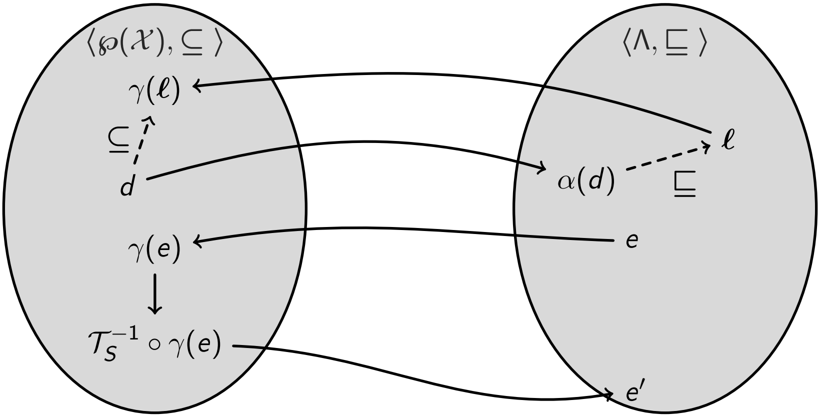
Infinite Case

- Allowing *Numerical* State Variables $(e.g., \mathcal{X} = \mathbb{B}^n \times \mathbb{Z}^m)$
- Undecidability Problem $\rightsquigarrow$ "Over-approximating Synthesis"

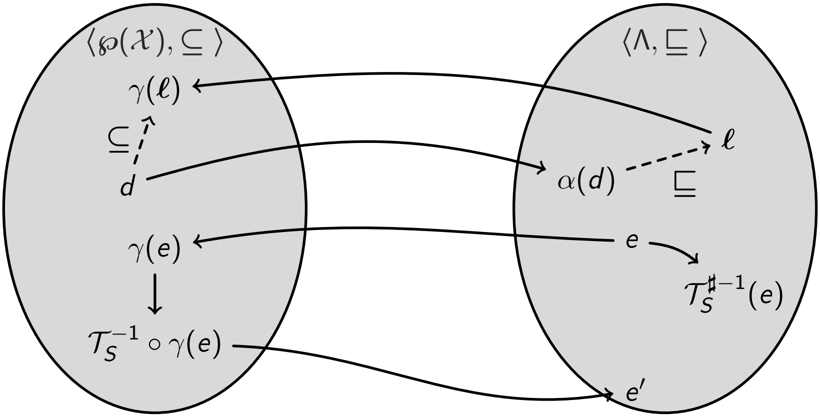
Abstract Interpretation Principles



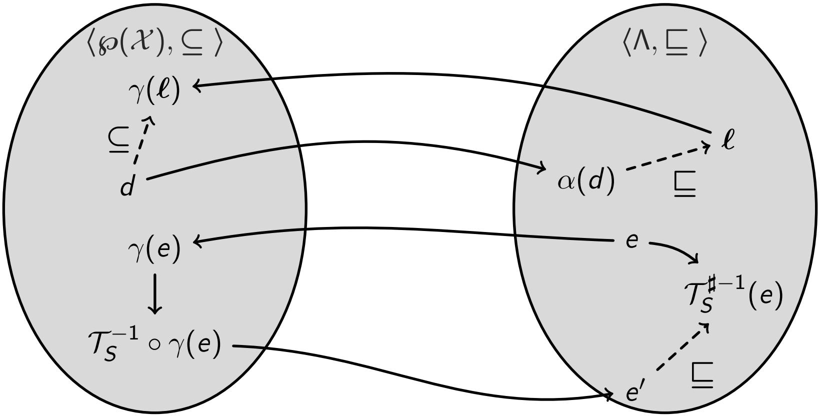
Abstract Interpretation Principles



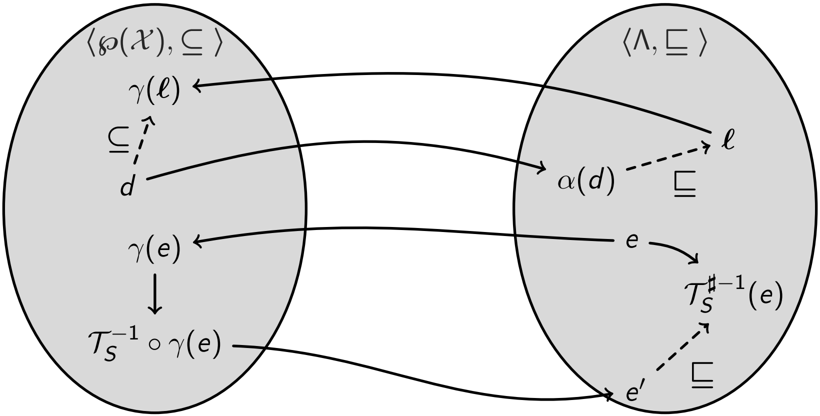
Abstract Interpretation Principles



Abstract Interpretation Principles



Abstract Interpretation Principles

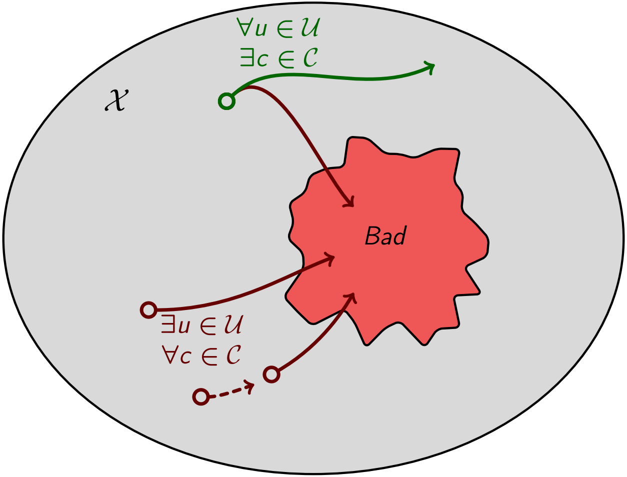


Requirements

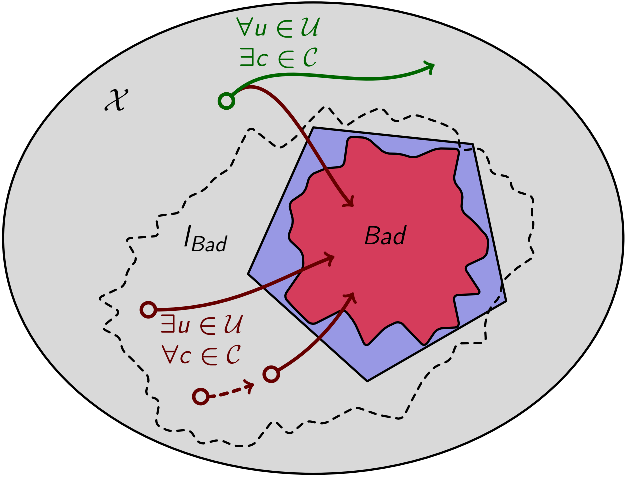
- $\langle \Lambda, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$, α and γ such that: $\wp(\mathcal{X}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$; $+\ \mathcal{T}_S^{\sharp-1}$

\+ Widening Operator $\nabla : \Lambda \times \Lambda \rightarrow \Lambda$ $\leftarrow$ Forcing Convergence

Over-approximating Discrete Controller Synthesis

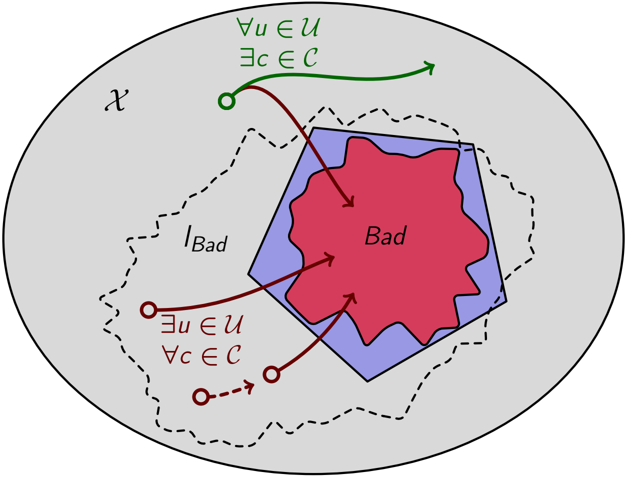


Over-approximating Discrete Controller Synthesis



- Computing $I'_{Bad} \supseteq I_{Bad}$

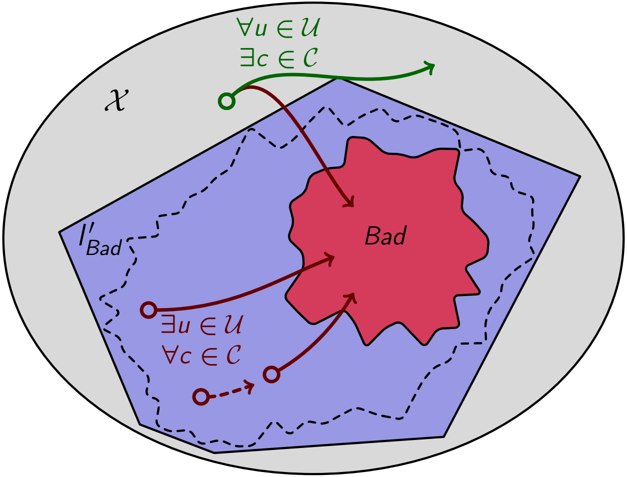
Over-approximating Discrete Controller Synthesis



$I_{Bad} = \mathrm{coreach}_u(Bad)$ $\qquad$ $\mathrm{coreach}_u(B) \stackrel{\mathrm{def}}{=} \mathrm{lfp}(\lambda\beta.B \cup \mathrm{pre}_u(\beta))$

$$\mathrm{pre}_u(B) \stackrel{\mathrm{def}}{=} \left\{x \in \mathcal{X} \;\middle|\; \exists u \in \mathcal{U}, \forall c \in \mathcal{C}, (x, u, c) \in \mathcal{T}_S^{-1}(B) \cap \mathcal{A}_S\right\}$$

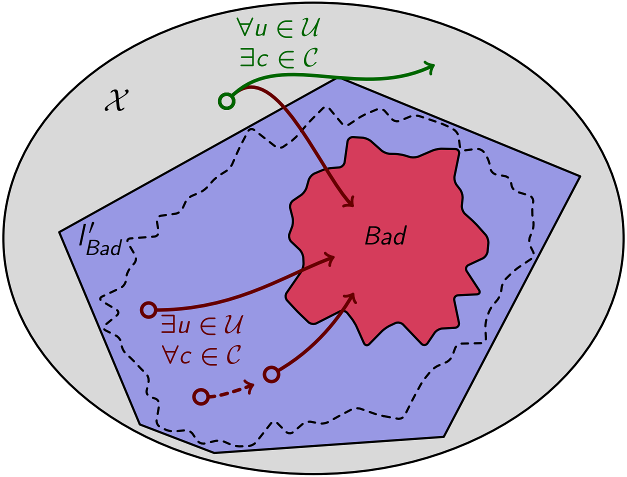
Over-approximating Discrete Controller Synthesis



$I'_{Bad} = \gamma \circ \mathrm{coreach}^\sharp_u \circ \alpha(Bad)$ $\qquad$ $\mathrm{coreach}^\sharp_u(B) \stackrel{\text{def}}{=} \mathrm{lfp}(\lambda\beta.B \sqcup \mathrm{pre}^\sharp_u(\beta))$

$$\mathrm{pre}^\sharp_u(B) \stackrel{\text{def}}{=} \exists^\sharp_{\mathcal{U}}\left(\forall^\sharp_{\mathcal{C}}\left(\mathcal{T}_S^{\sharp-1}(B) \sqcap \alpha(\mathcal{A}_S)\right)\right)$$

Over-approximating Discrete Controller Synthesis



$$I'_{Bad} = \gamma \circ \mathrm{coreach}^{\nabla}_{u} \circ \alpha(Bad) \qquad \mathrm{coreach}^{\nabla}_{u}(B) \stackrel{\mathrm{def}}{=} \mathrm{lfp}(\lambda\beta . B \nabla \mathrm{pre}^{\sharp}_{u}(\beta))$$

$$\mathrm{pre}^{\sharp}_{u}(B) \stackrel{\mathrm{def}}{=} \exists^{\sharp}_{\mathcal{U}}\left(\forall^{\sharp}_{\mathcal{C}}\left(\mathcal{T}^{\sharp -1}_{S}(B) \sqcap \alpha(\mathcal{A}_S)\right)\right)$$

Outline

- Discrete Controller Synthesis for Logico-numerical Programs
- State of the Art
- Discrete Controller Synthesis Principles
- ReaX: Technical Choices, Implementation & Evaluations
  - Over-approximating Logico-numerical State Spaces
  - Further Technical Choices
  - Implementation Details
  - Performance Comparison with Sigali
  - Evaluations of Over-approximating Synthesis
- Conclusions

Over-approximating Logico-numerical State Spaces

Over-approximating Numerical Spaces (*e.g.*, $\mathcal{X} = \mathbb{Z}^n \times \mathbb{R}^m$)

Numerical Domains Provide α, γ and $\mathcal{N}$ such that $\wp(\mathbb{Z}^n \times \mathbb{R}^m) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{N}$

[8] Peter Schrammel. "Logico-Numerical Verification Methods for Discrete and Hybrid Systems". PhD thesis. University of Grenoble, 2012.

Over-approximating Logico-numerical State Spaces

Over-approximating Numerical Spaces (*e.g.*, $\mathcal{X} = \mathbb{Z}^n \times \mathbb{R}^m$)

Numerical Domains Provide α, γ and $\mathcal{N}$ such that $\wp(\mathbb{Z}^n \times \mathbb{R}^m) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{N}$

- *Intervals*:

$$\bigwedge_{i \in [1, n+m]} (a_i \leqslant v_i \leqslant b_i)$$

- *Convex Polyhedra*: Conjunction of k Linear Constraints of the Form:

$$\left(\sum_{i \in [1, n+m]} a_i v_i\right) \leqslant b$$

[8] Peter Schrammel. "Logico-Numerical Verification Methods for Discrete and Hybrid Systems". PhD thesis. University of Grenoble, 2012.

Over-approximating Logico-numerical State Spaces

Over-approximating Numerical Spaces (*e.g.*, $\mathcal{X} = \mathbb{Z}^n \times \mathbb{R}^m$)

Numerical Domains Provide α, γ and $\mathcal{N}$ such that $\wp(\mathbb{Z}^n \times \mathbb{R}^m) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{N}$

- *Intervals*:

$$\bigwedge_{i \in [1,n+m]} (a_i \leqslant v_i \leqslant b_i)$$

- *Convex Polyhedra*: Conjunction of k Linear Constraints of the Form:

$$\left(\sum_{i \in [1,n+m]} a_i v_i\right) \leqslant b$$

Over-approximating Logico-numerical State Spaces (*e.g.*, $\mathcal{X} = \mathbb{B}^n \times \mathbb{Z}^m$)

Composing: One Numerical Domain $\mathcal{N}$ & One Boolean Domain[8]

[8] Peter Schrammel. "Logico-Numerical Verification Methods for Discrete and Hybrid Systems". PhD thesis. University of Grenoble, 2012.

Over-approximating Logico-numerical State Spaces

Over-approximating Numerical Spaces (*e.g.*, $\mathcal{X} = \mathbb{Z}^n \times \mathbb{R}^m$)

Numerical Domains Provide α, γ and $\mathcal{N}$ such that $\wp(\mathbb{Z}^n \times \mathbb{R}^m) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{N}$

- *Intervals*:

$$\bigwedge_{i\in[1,n+m]}(a_i \leqslant v_i \leqslant b_i)$$

- *Convex Polyhedra*: Conjunction of k Linear Constraints of the Form:

$$\left(\sum_{i\in[1,n+m]} a_i v_i\right) \leqslant b$$

Over-approximating Logico-numerical State Spaces (*e.g.*, $\mathcal{X} = \mathbb{B}^n \times \mathbb{Z}^m$)

Composing: One Numerical Domain $\mathcal{N}$ & One Boolean Domain[8]

- *Product*: $\wp(\mathbb{B}^n \times \mathbb{Z}^m \times \mathbb{R}^o) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \wp(\mathbb{B}^n) \times \mathcal{N}$

[8] Peter Schrammel. "Logico-Numerical Verification Methods for Discrete and Hybrid Systems". PhD thesis. University of Grenoble, 2012.

Over-approximating Logico-numerical State Spaces

Over-approximating Numerical Spaces (*e.g.*, $\mathcal{X} = \mathbb{Z}^n \times \mathbb{R}^m$)

Numerical Domains Provide α, γ and $\mathcal{N}$ such that $\wp(\mathbb{Z}^n \times \mathbb{R}^m) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{N}$

- *Intervals*:

$$\bigwedge_{i \in [1,n+m]} (a_i \leqslant v_i \leqslant b_i)$$

- *Convex Polyhedra*: Conjunction of k Linear Constraints of the Form:

$$\left(\sum_{i \in [1,n+m]} a_i v_i\right) \leqslant b$$

Over-approximating Logico-numerical State Spaces (*e.g.*, $\mathcal{X} = \mathbb{B}^n \times \mathbb{Z}^m$)

Composing: One Numerical Domain $\mathcal{N}$ & One Boolean Domain[8]

- *Product*: $\wp(\mathbb{B}^n \times \mathbb{Z}^m \times \mathbb{R}^o) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \wp(\mathbb{B}^n) \times \mathcal{N}$
- *Power*: $\wp(\mathbb{B}^n \times \mathbb{Z}^m \times \mathbb{R}^o) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathbb{B}^n \to \mathcal{N}$ $(= \mathcal{N}^{\mathbb{B}^n})$

[8] Peter Schrammel. "Logico-Numerical Verification Methods for Discrete and Hybrid Systems". PhD thesis. University of Grenoble, 2012.

Further Technical Choices

Restricting to $\mathcal{C} = \mathbb{B}^p$

- Computing the Universal Quantification in the Concrete Domain

$$\mathrm{pre}_u^{\sharp\prime}(B) \stackrel{\text{def}}{=} \exists^\sharp_{\mathcal{U}} \alpha \left\{ (x,u) \,\middle|\, \forall c \in \mathcal{C}, (x,u,c) \in \gamma\left(\mathcal{T}_S^{\sharp-1}(B) \sqcap \alpha(\mathcal{A}_S) \right) \right\}$$

- Permits *Triangularization* of the Controller[9]

[9] Gwenaël Delaval, Hervé Marchand, and Éric Rutten. "Contracts for modular discrete controller synthesis". In: *Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems*. Stockholm, Sweden, 2010, pp. 57–66.

ReaX: Implementation Details

Extension of ReaVer[10], Itself Exploiting BddApron[11]

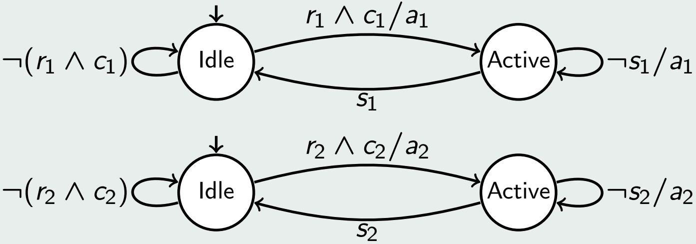
- Decision Diagrams: CUDD
- Numerical Abstract Domains of APRON; *e.g.*,
  - Intervals
  - Convex Polyhedra
- Combining *Binary Decision Diagrams* with *Numerical Abstract Domains*
  - $\rightsquigarrow$ Product $(\mathbb{B}^n \times \mathcal{N})$, Power $(\mathbb{B}^n \to \mathcal{N})$
- Pre-processing Features
  - Partial Control Flow Graph Generation
  - $\rightsquigarrow$ Improving Precision
- BZR Backend

[10] Peter Schrammel. "Logico-Numerical Verification Methods for Discrete and Hybrid Systems". PhD thesis. University of Grenoble, 2012.

[11] Bertrand Jeannet. *BddApron: A logico-numerical abstract domain library*. 2009. url: http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/.

Performance Comparison with SIGALI

Benchmark



- $X = \langle t_1, t_2, a_1, a_2 \rangle$ $\qquad \mathcal{D}_X = \{\text{Idle}, \text{Active}\}^2 \times \mathbb{B}^2$
- $I_{uc} = \langle r_1, r_2, s_1, s_2 \rangle$, $I_c = \langle c_1, c_2 \rangle$ $\qquad \mathcal{D}_{I_{uc}} = \mathbb{B}^4$, $\mathcal{D}_{I_c} = \mathbb{B}^2$
- Enforcing Mutual Exclusion Between Active States

$$\Phi(\langle t_1, t_2, \ldots \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle})$$

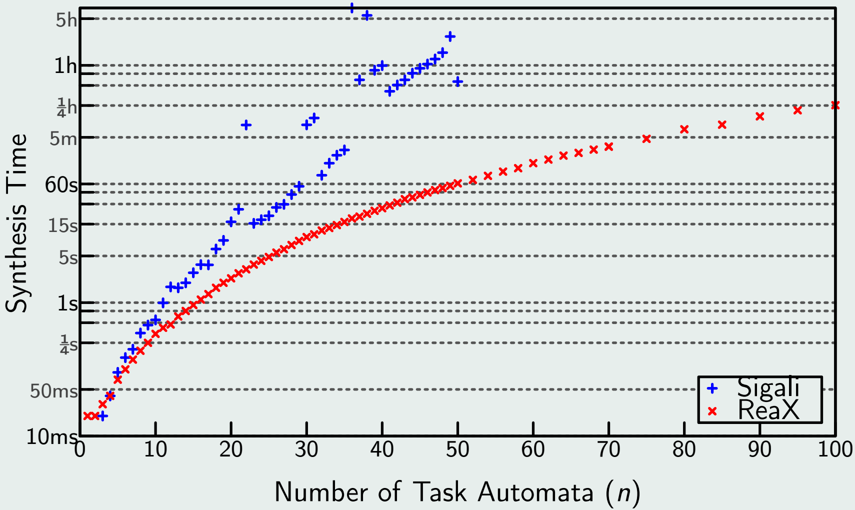
Performance Comparison with SIGALI

Benchmark

```
typedef State = enum {Idle, Active};
state t1, t2: State;                          (* State Variables *)
input r1, r2, s1, s2: bool;       (* Non-controllable Inputs *)
controllable c1, c2: bool;            (* Controllable Inputs *)
transition
  t1' = if t1 = Idle   and r1 and c1 then Active else
        if t1 = Active and s1        then Idle   else t1;
  t2' = if t2 = Idle   and r2 and c2 then Active else
        if t2 = Active and s2        then Idle   else t2;
initial t1 = Idle and t2 = Idle;
invariant t1 <> t2 or t1 = Idle;       (* To be Enforced (Φ) *)
```

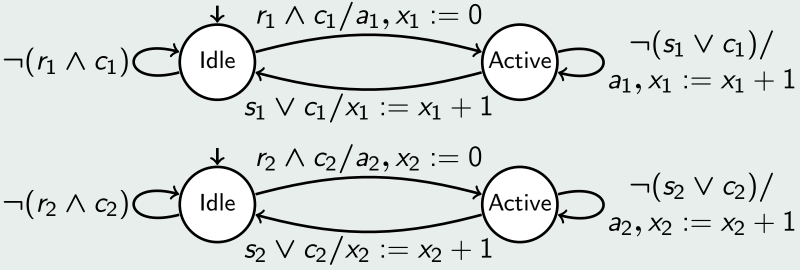
Performance Comparison with Sigali (cont'd)

Comparing Execution Times



Example Logico-numerical DCS Problem

Example Logico-numerical Program & Invariant Properties

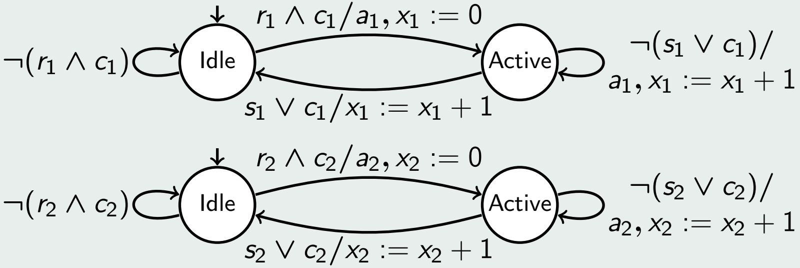


- $X = \langle t_1, t_2, x_1, x_2, a_1, a_2 \rangle$ $\quad\mathcal{D}_X = \{\text{Idle}, \text{Active}\}^2 \times \mathbb{Z}^2 \times \mathbb{B}^2$
- $I_{uc} = \langle r_1, r_2, s_1, s_2 \rangle, I_c = \langle c_1, c_2 \rangle$ $\quad\mathcal{D}_{I_{uc}} = \mathbb{B}^4, \mathcal{D}_{I_c} = \mathbb{B}^2$
- Enforcing Mutual Exclusion Between Active States

$$\Phi(\langle t_1, t_2, x_1, x_2 \dots \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle})$$

Example Logico-numerical DCS Problem

Example Logico-numerical Program & Invariant Properties

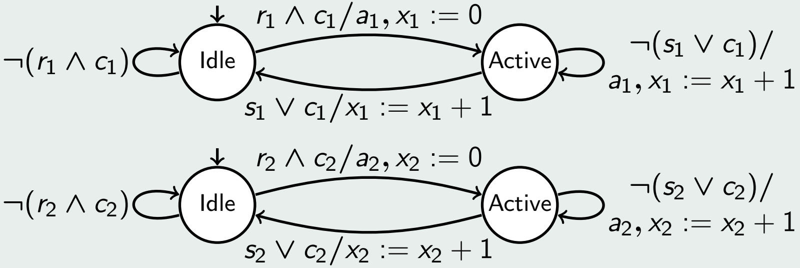


- $X = \langle t_1, t_2, x_1, x_2, a_1, a_2 \rangle$ $\qquad \mathcal{D}_X = \{\text{Idle}, \text{Active}\}^2 \times \mathbb{Z}^2 \times \mathbb{B}^2$
- $I_{uc} = \langle r_1, r_2, s_1, s_2 \rangle, I_c = \langle c_1, c_2 \rangle$ $\qquad \mathcal{D}_{I_{uc}} = \mathbb{B}^4, \mathcal{D}_{I_c} = \mathbb{B}^2$
- Enforcing Mutual Exclusion Between Active States & Bounds on x_i's

$$\Phi(\langle t_1, t_2, x_1, x_2 \ldots \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle}) \wedge (x_1 \leqslant 10 \wedge x_2 \leqslant 10)$$

Example Logico-numerical DCS Problem

Example Logico-numerical Program & Invariant Properties



- $X = \langle t_1, t_2, x_1, x_2, a_1, a_2 \rangle$ $\quad\mathcal{D}_X = \{\text{Idle}, \text{Active}\}^2 \times \mathbb{Z}^2 \times \mathbb{B}^2$
- $I_{uc} = \langle r_1, r_2, s_1, s_2 \rangle, I_c = \langle c_1, c_2 \rangle$ $\quad\mathcal{D}_{I_{uc}} = \mathbb{B}^4, \mathcal{D}_{I_c} = \mathbb{B}^2$
- Enforcing Mutual Exclusion Between Active States & Bounds on x_i's & Relational Constraints on x_i's

$$\Phi(\langle t_1, t_2, x_1, x_2 \ldots \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle}) \wedge (x_1 \leqslant 10 \wedge x_2 \leqslant 10) \wedge (x_1 \leqslant x_2)$$

Example Logico-numerical DCS Problem

Example Logico-numerical Program & Invariant Properties

```
typedef State = enum {Idle, Active};
state t1, t2: State;                               (* State Variables *)
      x1, x2: int;
input r1, r2, s1, s2: bool;              (* Non-controllable Inputs *)
controllable c1, c2: bool;                   (* Controllable Inputs *)
transition
  t1' = if t1 = Idle   and  r1 and c1  then Active else
        if t1 = Active and (s1 or  c1) then Idle   else t1;
  x1' = if t1 = Idle   and  r1 and c1  then 0      else
        if t1 = Active                 then x1 + 1 else x1;
  t2' = if t2 = Idle   and  r2 and c2  then Active else
        if t2 = Active and (s2 or  c2) then Idle   else t2;
  x2' = if t2 = Idle   and  r2 and c2  then 0      else
        if t2 = Active                 then x2 + 1 else x2;
initial x1 = 0 and t1 = Idle and x2 = 0 and t2 = Idle;
invariant t1 <> t2 or t1 = Idle;              (* To be Enforced (Φ) *)
```

Example Logico-numerical DCS Problem

Example Logico-numerical Program & Invariant Properties

```
typedef State = enum {Idle, Active};
state t1, t2: State;                          (* State Variables *)
      x1, x2: int;
input r1, r2, s1, s2: bool;       (* Non-controllable Inputs *)
controllable c1, c2: bool;            (* Controllable Inputs *)
transition
  t1' = if t1 = Idle   and  r1 and c1  then Active else
        if t1 = Active and (s1 or  c1) then Idle   else t1;
  x1' = if t1 = Idle   and  r1 and c1  then 0      else
        if t1 = Active                 then x1 + 1 else x1;
  t2' = if t2 = Idle   and  r2 and c2  then Active else
        if t2 = Active and (s2 or  c2) then Idle   else t2;
  x2' = if t2 = Idle   and  r2 and c2  then 0      else
        if t2 = Active                 then x2 + 1 else x2;
initial x1 = 0 and t1 = Idle and x2 = 0 and t2 = Idle;
invariant t1 <> t2 or t1 = Idle      (* To be Enforced (Φ) *)
      and x1 <= 10 and x2 <= 10;
```

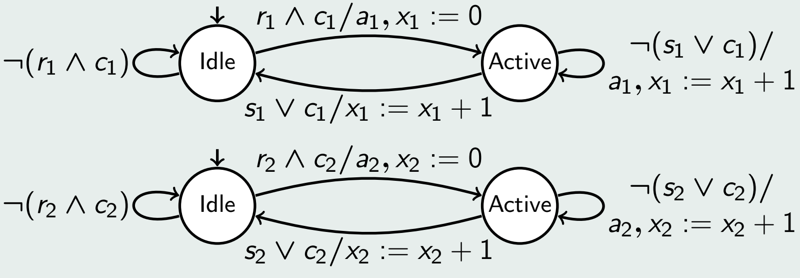
Example Logico-numerical DCS Problem

Example Logico-numerical Program & Invariant Properties

```
typedef State = enum {Idle, Active};
state t1, t2: State;                        (* State Variables *)
      x1, x2: int;
input r1, r2, s1, s2: bool;       (* Non-controllable Inputs *)
controllable c1, c2: bool;            (* Controllable Inputs *)
transition
  t1' = if t1 = Idle   and  r1 and c1  then Active else
        if t1 = Active and (s1 or  c1) then Idle   else t1;
  x1' = if t1 = Idle   and  r1 and c1  then 0      else
        if t1 = Active                 then x1 + 1 else x1;
  t2' = if t2 = Idle   and  r2 and c2  then Active else
        if t2 = Active and (s2 or  c2) then Idle   else t2;
  x2' = if t2 = Idle   and  r2 and c2  then 0      else
        if t2 = Active                 then x2 + 1 else x2;
initial x1 = 0 and t1 = Idle and x2 = 0 and t2 = Idle;
invariant t1 <> t2 or t1 = Idle        (* To be Enforced (Φ) *)
      and x1 <= 10 and x2 <= 10 and x1 <= x2;
```

Results of Over-approximating Synthesis

Program



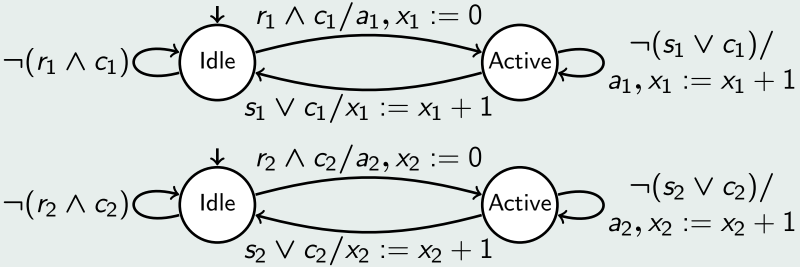
$\Phi(\langle t_1, t_2, x_1, x_2 \ldots \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle}) \wedge (x_1 \leqslant 10 \wedge x_2 \leqslant 10)$

- Power Domain, Intervals & Convex Polyhedra

$$I'^{c}_{Bad} = \left\{ \begin{array}{l} t_1 = \text{Idle} \wedge t_2 = \text{Idle} \wedge x_1 \leqslant 10 \wedge x_2 \leqslant 10 \vee \\ t_1 = \text{Idle} \wedge t_2 = \text{Active} \wedge x_1 \leqslant 10 \wedge x_2 \leqslant 9 \vee \\ t_1 = \text{Active} \wedge t_2 = \text{Idle} \wedge x_1 \leqslant 9 \wedge x_2 \leqslant 10 \end{array} \right\}$$

Results of Over-approximating Synthesis (cont'd)

Program



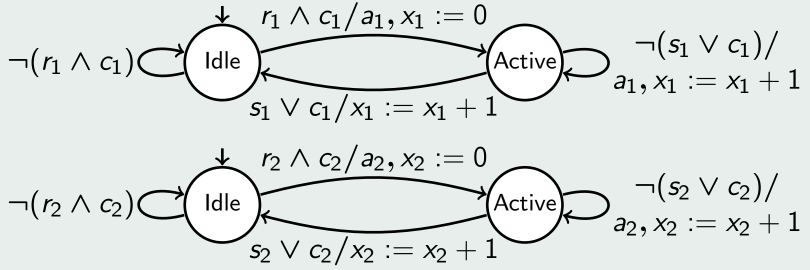
$\Phi(\langle t_1, t_2, x_1 \ldots \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle}) \wedge (x_1 \leqslant 10 \wedge x_2 \leqslant 10) \wedge (x_1 \leqslant x_2)$

- Power Domain, Intervals

$$
\begin{aligned}
I'_{Bad} &\supseteq \{t_1 = \text{Idle} \ \wedge \ t_2 = \text{Idle} \ \wedge \ x_1 < 11 \ \wedge \ x_2 < 10\} \\
&\supseteq \{t_1 = \text{Idle} \ \wedge \ t_2 = \text{Idle} \ \wedge \ x_1 = \ 0 \ \wedge \ x_2 = \ 0\} = \mathcal{X}_0
\end{aligned}
$$

Results of Over-approximating Synthesis (cont'd)

Program



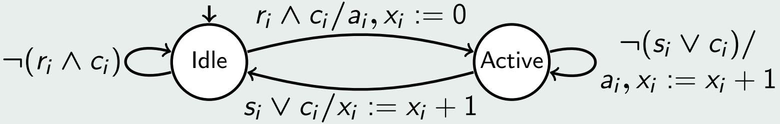
$\Phi(\langle t_1, t_2, x_1 \ldots \rangle) = (t_1 \neq t_2 \vee t_1 = \text{Idle}) \wedge (x_1 \leqslant 10 \wedge x_2 \leqslant 10) \wedge (x_1 \leqslant x_2)$

- Power Domain, Convex Polyhedra

$$I'^{c}_{Bad} = \left\{ \begin{array}{l} t_1 = \text{Idle} \;\wedge\; t_2 = \text{Idle} \quad \wedge x_1 \leqslant x_2 \leqslant 10 \vee \\ t_1 = \text{Idle} \;\wedge\; t_2 = \text{Active} \wedge x_1 \leqslant x_2 \leqslant \;\; 9 \vee \\ t_1 = \text{Active} \;\wedge\; t_2 = \text{Idle} \wedge x_1 < x_2 \leqslant 10 \end{array} \right\}$$

Performance Evaluation

Example Logico-numerical Programs & Invariant Properties



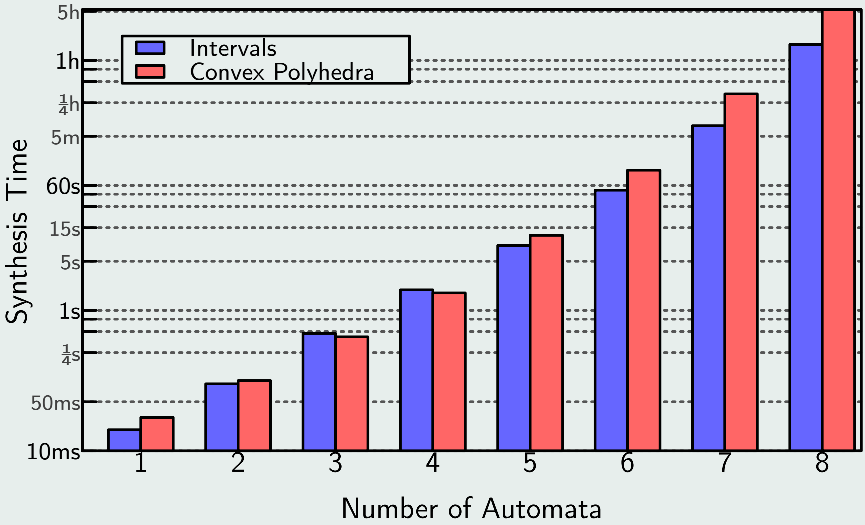
- $X = \langle t_1 \dots t_n, x_1 \dots x_n, a_1 \dots a_n \rangle$ $\mathcal{D}_X = \{\text{Idle}, \text{Active}\}^n \times \mathbb{Z}^n \times \mathbb{B}^n$
- $I_{uc} = \langle r_1 \dots r_n, s_1 \dots s_n \rangle, I_c = \langle c_1 \dots c_n \rangle$ $\mathcal{D}_{I_{uc}} = \mathbb{B}^{2n}, \mathcal{D}_{I_c} = \mathbb{B}^n$
- Enforcing Mutual Exclusion Between Active States & Bounds on x_i's

$$\Phi(\langle t_1 \dots t_n, x_1 \dots x_n, \dots \rangle) =$$

$$\bigoplus_{i \in [1,n]} (t_i = \text{Active}) \wedge \bigwedge_{i \in [1,n]} (x_i \leqslant 10)$$

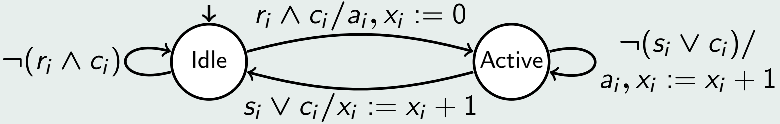
Performance of Over-approximating Synthesis

Performance Results: Mutual Exclusion & Bounds



Performance Evaluation

Example Logico-numerical Programs & Invariant Properties



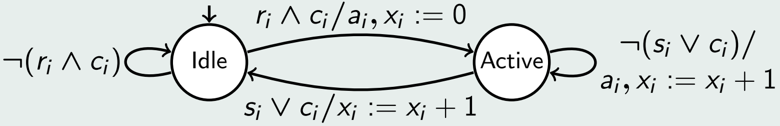
- $X = \langle t_1 \ldots t_n, x_1 \ldots x_n, a_1 \ldots a_n \rangle$ $\qquad \mathcal{D}_X = \{\text{Idle}, \text{Active}\}^n \times \mathbb{Z}^n \times \mathbb{B}^n$
- $I_{uc} = \langle r_1 \ldots r_n, s_1 \ldots s_n \rangle, I_c = \langle c_1 \ldots c_n \rangle$ $\qquad \mathcal{D}_{I_{uc}} = \mathbb{B}^{2n}, \mathcal{D}_{I_c} = \mathbb{B}^n$
- Enforcing Mutual Exclusion Between Active States & Bounds on x_i's

$$\Phi(\langle t_1 \ldots t_n, x_1 \ldots x_n, \ldots \rangle) =$$

$$\bigoplus_{i \in [1,n]} (t_i = \text{Active}) \wedge \bigwedge_{i \in [1,n]} (x_i \leqslant 10)$$

Performance Evaluation

Example Logico-numerical Programs & Invariant Properties



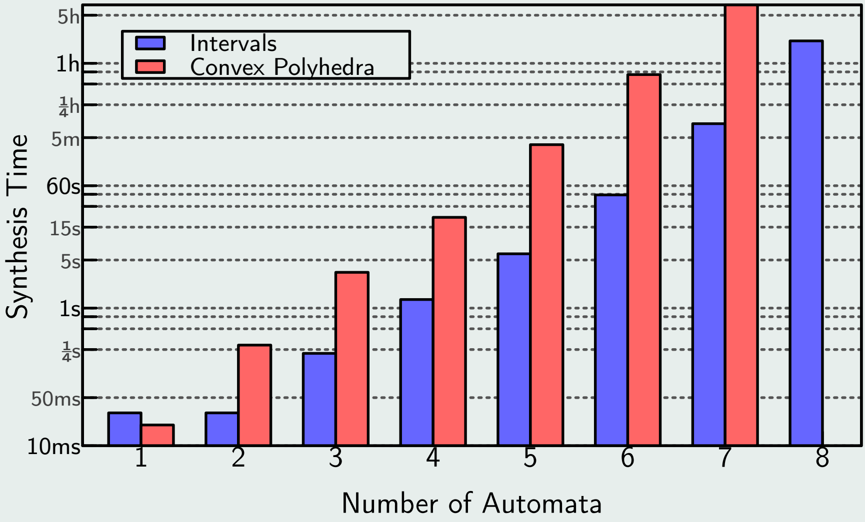
- $X = \langle t_1 \dots t_n, x_1 \dots x_n, a_1 \dots a_n \rangle$ $\qquad \mathcal{D}_X = \{\text{Idle}, \text{Active}\}^n \times \mathbb{Z}^n \times \mathbb{B}^n$
- $I_{uc} = \langle r_1 \dots r_n, s_1 \dots s_n \rangle, I_c = \langle c_1 \dots c_n \rangle$ $\qquad \mathcal{D}_{I_{uc}} = \mathbb{B}^{2n}, \mathcal{D}_{I_c} = \mathbb{B}^n$
- Enforcing Mutual Exclusion Between Active States & Bounds on x_i's & Relational Constraints on x_i's

$$\Phi(\langle t_1 \dots t_n, x_1 \dots x_n, \dots \rangle) =$$

$$\bigoplus_{i \in [1,n]} (t_i = \text{Active}) \wedge \bigwedge_{i \in [1,n]} (x_i \leqslant 10) \wedge \bigwedge_{i \in [1,n[} (x_i \leqslant x_{i+1})$$

Performance of Over-approximating Synthesis (cont'd)

Performance Results: Mutual Exclusion & Bounds & Relational Constraints



Outline

- Discrete Controller Synthesis for Logico-numerical Programs
- State of the Art
- Discrete Controller Synthesis Principles
- ReaX: Technical Choices, Implementation & Evaluations
- Conclusions

Conclusion & Further Works

ReaX

- Discrete Controller Synthesis for Logico-numerical Synchronous/Reative Programs
- Efficient Exact Synthesis
- Over-approximating Synthesis with Abstract Interpretation Techniques
- Heptagon/BZR Backend

$\rightsquigarrow$ Favorably Replaces Sigali

Conclusion & Further Works

ReaX

- Discrete Controller Synthesis for Logico-numerical Synchronous/Reative Programs
- Efficient Exact Synthesis
- Over-approximating Synthesis with Abstract Interpretation Techniques
- Heptagon/BZR Backend

$\leadsto$ Favorably Replaces Sigali

Forthcoming Challenges

- Enforcement of Quantitative Properties
- Improving Precision
- Synthesis Failure Diagnosis
- Avoiding Deadlocks

Thanks!

Outline

- Discrete Controller Synthesis for Logico-numerical Programs
- State of the Art
- Discrete Controller Synthesis Principles
- ReaX: Technical Choices, Implementation & Evaluations
- Conclusions

Outline

- Backup
  - On the Necessary Convexity of Bad
  - Producing an Executable Controller
  - Triangularization of the Controller
  - Performance Comparison with Sigali for a Realistic Model

On the Necessary Convexity of *Bad*

Example Problem

With Usual Numerical Abstract Domains:

$$\alpha(\{x \in \mathbb{Z} \mid x \leqslant 0 \wedge 10 \leqslant x\}) = \top$$

A Solution

- Split *Bad* into a Disjunction of n Convex Clauses Bad_i
- Compute Every I'_{Bad_i}
- Compute the Controller using $\bigcup_{i \in [1,n]} I'_{Bad_i}$

$\rightsquigarrow$ Deadlocks!

Producing an *Executable* Controller

- New Constraint $\mathcal{A}_\Phi$ $(\subseteq \mathcal{X} \times \mathcal{U} \times \mathcal{C})$
  - Relating States with *Allowed Inputs*

$$\mathcal{A}_\Phi = \mathcal{T}_S^{-1}(I_{Bad}^c) \cap \mathcal{A}_\Phi$$

- Controller $(\mathcal{K}_\Phi : \mathcal{X} \times \mathcal{U} \to \wp(\mathcal{C}))$

$$\mathcal{K}_\Phi = \lambda(x, \upsilon).\, \{\iota \,|\, (x, \upsilon, \iota) \in \mathcal{A}_\Phi\}$$

Resulting System

$$[S'] = \langle \mathcal{X}, \mathcal{U}, \lambda(\sigma, \upsilon).\mathcal{T}_S(\sigma, \upsilon, \text{choose} \circ \mathcal{K}_\Phi(\sigma, \upsilon)), \mathcal{A}_\Phi, \mathcal{X}_0 \rangle$$

- $\text{choose} : \wp(\mathcal{C}) \to \mathcal{C}$: Non-Deterministic Choice!

Triangularization of the Controller

- Code Generation for $\lambda(\sigma, \upsilon).\mathsf{choose}(\mathcal{K}_{\Phi}(\sigma, \upsilon))$
- Using Oracles ("Phantom Variables")
  - "Preferred Value" for the Controllable Inputs

Performance Comparison with Sigali for a Realistic Model

What About A Realistic Model?

	Sigali	ReaX
2-tiers	6s	3.2s
4-tiers	105s	12s