Logic in Software, Dynamical and Biological Systems

Ashish Tiwari
SRI International
Menlo Park, CA 94025
tiwari@csl.sri.com
Problem Classes

From a logical perspective, we have three classes of problems:
Given description $E$, find/check some desired description $E'$ such that

1. $E \Leftrightarrow E'$
   Example: Linear equation solving, Gröbner basis, theorem proving, computer algebra

2. $E \Rightarrow E'$
   Example: verification, abstraction, abstract interpretation, bounded synthesis

3. $E' \Rightarrow E$
   Example: learning, synthesis, diagnosis
Formal Methods

Model and analyze systems formally

Two aspects:

- Formal model of dynamical system
- Formal property specification language
Formal Models of Dynamical Systems

Modeling formalisms: **Time and state space**

**Time** $T$ domain:
- discrete-time: $\mathbb{N}$
- continuous-time: $\mathbb{R}$
- hybrid-time: $\mathbb{N} \times \mathbb{R}$

**State space** $SS$ domain:
- discrete space: $2^n \times \mathbb{N}^m$
- continuous space: $\mathbb{R}^n$
- hybrid space: $2^n \times \mathbb{R}^m$

Semantics: $T \mapsto SS$
Outline

I. Continuous dynamical system verification $\mathrel{\rightarrow} \exists \forall$ solving

II. Hybrid system verification $\mathrel{\rightarrow} \exists \forall$ solving + discrete system verification

III. Component-based Synthesis $\mathrel{\rightarrow} \exists \forall$ solving

IV. \(\exists \forall\) Solvers

V. Systems Biology $\mathrel{\rightarrow} \forall$ solving

VI. Program verification $\mathrel{\rightarrow}$ Approximating logical operators
Continuous Dynamical Systems

Tuple: $\langle X, f, Inv \rangle$ where

- $X$: set of $n$ real-valued variables
- $f$: vector field; mapping $\mathbb{R}^n \mapsto \mathbb{R}^n$
- $Inv$: invariant region, subset of $\mathbb{R}^n$

Example: CDS with

- $X := \{ x_1, x_2 \}$
- $f(x_1, x_2) := (\, -x_1 - x_2, x_1 - x_2 \,)$
- $Inv := \mathbb{R}^2$

Example CDS’s dynamics are given by:

- $\frac{dx_1}{dt} = -x_1 - x_2$
- $\frac{dx_2}{dt} = x_1 - x_2$

Semantics: A structure $\langle \mathbb{R}^n, \rightarrow \rangle$ where $\rightarrow$ is

$\{(F(0), F(t_1)) \mid \forall 0 \leq t \leq t_1 : \frac{dF(t)}{dt} = f(F(t)), F(t) \in Inv\}$
Continuous Dynamical Systems Reachability

Linear systems: \( \frac{d\vec{x}}{dt} = A\vec{x} + b \)

Exact reachable sets can be computed when either

- \( A \) is diagonalizable with all rational eigenvalues
- \( A \) is diagonalizable with all purely imaginary rational eigenvalues
- \( A \) is nilpotent

In these cases, after suitable change of variables, reachable sets are semi-algebraic and can be obtained using quantifier elimination
Certificate-Based Verification

A certificate for $M \models \phi$ is $\Phi$ such that

1. $\models \Phi \Rightarrow \phi$

2. $M \models \Phi$ is locally checkable
   $M \models \Phi$ reduces to a formula in the (underlying FO) logic

Examples:

<table>
<thead>
<tr>
<th>Property $\phi$</th>
<th>Certificate $\Phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>safety</td>
<td>inductive invariant</td>
</tr>
<tr>
<td>stability</td>
<td>Lyapunov function</td>
</tr>
<tr>
<td>termination</td>
<td>ranking function</td>
</tr>
<tr>
<td>controlled safety</td>
<td>controlled inductive invariant</td>
</tr>
</tbody>
</table>
Certificate-based verification reduces the verification problem to an $\exists \forall$ formula.

\[
M \models \phi \\
\uparrow \\
\exists \Phi : ((M \models \Phi) \land (\Phi \Rightarrow \phi)) \\
\uparrow \\
\exists \Phi : \forall \bar{x} : \text{quantifier-free FO formula} \\
\uparrow \\
\exists \bar{a} : \forall \bar{x} : \text{quantifier-free FO formula}
\]

The last step performed by choosing a template for $\Phi$
Inductive Invariants for CDSs

Used to prove safety of CDSs

How to define inductiveness?

A set $I$ is inductive if

\[ \forall \vec{x} : \vec{x} \in I \land \vec{x} \rightarrow \vec{y} \Rightarrow \vec{y} \in I \]

Recall semantics of CDS has uncountably infinite $\rightarrow$-successors for every state, not defined constructively

([T.2003], [Prajna and Jadbabaie 2004], [Sankaranarayanan et al. 2004])
Inductiveness for CDSs

Example:

\[
\begin{align*}
\frac{dx_1}{dt} &= -x_1 - x_2 \\
\frac{dx_2}{dt} &= x_1 - x_2
\end{align*}
\]

Is \(x_1^2 + x_2^2 \leq 0.5\) inductive?

**Intuition:** Ensure vector field points inwards at all points on the boundary of the set.
Let \( p := x_1^2 + x_2^2 - 0.5 \)

The set \( p \leq 0 \) is inductive if

\[
\frac{dp}{dt} < 0 \\
\lor \frac{dp}{dt} = 0 \land \frac{d^2p}{dt^2} < 0 \\
\lor \frac{dp}{dt} = \frac{d^2p}{dt^2} = 0 \land \frac{d^3p}{dt^3} < 0 \\
\ldots
\]

where \( \frac{dp}{dt} := \vec{\nabla} p \cdot f \) is Lie derivative of \( p \) wrt \( f \).

Several sound checks, but no complete check in general

For special cases, finite complete checks exist
Example: Certificate-Based Safety

Example: \[
\frac{dx_1}{dt} = -x_1 - x_2 \quad \frac{dx_2}{dt} = x_1 - x_2
\]

Problem: If \(x_1 \leq 0.5\) and \(x_2 \leq 0.5\) initially, prove \(G(x_2 \leq 1)\)

Let us find a certificate of the form \(p \leq 0\) where \(p := ax_1^2 + bx_2^2 + c\)

We need to solve

\[
\exists a, b, c : \forall x_1, x_2 : \quad (p = 0 \Rightarrow \frac{dp}{dt} < 0) \land \\
(x_1 \leq 0.5 \land x_2 \leq 0.5 \Rightarrow p \leq 0) \land \\
(p \leq 0 \Rightarrow x_2 \leq 1)
\]

We get \(p := x_1^2 + x_2^2 - 0.5\). Proved.
Certification-based Verification

Without Solving $\exists \forall$

A Lyapunov function is a certificate for stability

We can discover Lyapunov functions by solving $\exists \forall$ formulas

But even without solving $\exists \forall$ formulas, we can determine stability of linear systems

Can we find useful invariants without solving $\exists \forall$ formulas?
Inductive Sets of Linear Systems

Without solving $\exists \forall$ formulas

Consider $\frac{d\vec{x}}{dt} = A\vec{x}$

If $\vec{c}$ is a left eigenvector of $A$ corr to $\lambda$, then

$$\vec{c}^T A = \lambda \vec{c}^T$$

Let $p := \vec{c}^T \vec{x}$, we have

$$\frac{dp}{dt} = \frac{d\vec{c}^T \vec{x}}{dt} = \vec{c}^T \frac{d\vec{x}}{dt} = \vec{c}^T A \vec{x} = \lambda \vec{c}^T \vec{x} = \lambda p$$

Hence, $p \geq 0$ and $p \leq 0$ are inductive sets

The surface $p = 0$ is called a barrier certificate

Inductive sets for linear systems can be obtained by analyzing matrix $A$
Example: Certificate-based Verification w/o $\exists \forall$

Example. Consider a cruise control:

\[
\begin{align*}
\dot{v} &= a \\
\dot{a} &= -4v + 3v_f - 3a + \text{gap} \\
gap &= -v + v_f
\end{align*}
\]

where \(v, a\) is the velocity and acceleration of this car, \(v_f\) is the velocity of car in front, and \(\text{gap}\) is the distance between the two cars.

Prove that the cars will not crash when ACC mode is initiated in given set of states.

Solution: Use inductive invariant corr to the negative real eigenvalue of \(A\).
Hybrid Automata

A powerful modeling language

A finite collection of CDS with switching between them

Tuple \( \langle Q, (\text{CDS}_q)_{q \in Q}, E \rangle \) where

- \( Q \): finite set of modes
- \( \text{CDS}_q \): CDS \( \langle X, f_q, Inv_q \rangle \) within state \( q \)
- \( E \): subset of \( (Q \times \mathbb{R}^n) \times (Q \times \mathbb{R}^n) \)

Semantics: A structure \( \langle Q \times \mathbb{R}^n, \rightarrow \rangle \) where \( \rightarrow \) is

\[
E \cup \{(q, F(0), q, F(t_1)) \mid \forall 0 \leq t \leq t_1 : \frac{dF(t)}{dt} = f_q(F(t)), F(t) \in Inv_q\}
\]
**Example: Hybrid Automata**

**Bouncing Ball:** Ball under vertical free fall that loses 10% of its velocity when it bounces off the ground

**One mode** $q$ with variables $X := \{y, v\}$ and dynamics:

\[
\frac{dy}{dt} = v \quad \quad \frac{dv}{dt} = -9.8
\]

so, $f_q(y, v) := (v, -9.8)$ is the **vector field**

**Discrete transition** given by:

$$(q, (0, v), q, (0, -0.9 \times v))$$
Hybrid Automata Verification Problem

Semantics of hybrid automata are given as discrete state transition system (with uncountably infinite state space)

Therefore, we can ask about the complexity of the model checking problem

Even reachability is undecidable
Classes of Hybrid Automata

Several subclasses of HA have been studied

Restrictions on the continuous dynamics and the discrete dynamics

Timed Automata: $\frac{dx}{dt} = 1$ for all $x$, in all modes

- Guards of the form $x - y \leq c$ (Boolean combination)
- Some clocks $x$ can be reset $x := 0$

Linear Hybrid Automata: $\frac{dx}{dt} = c_x$ for all $x$, in all modes there are linear constraints among the $c_x$ variables

- Guards are linear constraint over $X$

Model checking problems are decidable for timed automata, but undecidable for linear hybrid automata

Boundary is well studied
Analyzing Hybrid Automata

These decidable subclasses are too restrictive

Need sound, but incomplete, techniques for $M \models \phi$

Generic approaches:

- Abstraction
- Deductive Methods

Concrete approaches:

- certificate-based verification: $M \models \Phi$ and $\Phi \Rightarrow \phi$
- relational abstraction: $M \Rightarrow M'$ and $M' \models \phi$
Relational Abstraction

Replace continuous dynamics by its relational abstraction

Relational abstraction of a dynamical system \((X, \rightarrow)\) is another dynamical system \((X, \rightarrow)\) such that

\[
\text{TransitiveClosure}(\rightarrow) \subseteq \rightarrow
\]

**Benefit:**

Eliminates need for iterative fixpoint computation
Useful for proving safety properties, and establishing conservative safety bounds
Example: Relational Abstraction

For the continuous-time continuous-space dynamical system:

\[
\frac{dx}{dt} = -x
\]

we have the following continuous-space discrete-time relational abstraction:

\[
x \rightarrow x' := 0 < x' \leq x \lor x \leq x' < 0 \lor x = x' = 0
\]
Computing Relational Abstractions

We can compute good quality relational abstractions of linear systems.

<table>
<thead>
<tr>
<th>Dynamics</th>
<th>Relational Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\dot{x} = 1, \dot{y} = 1$</td>
<td>$x' - x = y' - y \land x' \geq x$</td>
</tr>
<tr>
<td>$\dot{x} = 2, \dot{y} = 3$</td>
<td>$(x' - x)/2 = (y' - y)/3 \land x' \geq x$</td>
</tr>
<tr>
<td>$\ddot{x} = A\vec{x}$</td>
<td>$(0 &lt; p' \leq p) \lor (p \leq p' &lt; 0) \lor (p = p' = 0)$, where $p = \vec{c}^T \vec{x}$, $\vec{c}$ eigenvector of $A^T$ corr. to negative eigenvalue</td>
</tr>
<tr>
<td></td>
<td>Similarly for eigenvector corr. to positive eigenvalue</td>
</tr>
<tr>
<td></td>
<td>Coarser abstraction for complex eigenvalues</td>
</tr>
</tbody>
</table>

Complete for timed, multirate, linear hybrid automata
Using Relational Abstraction

- Replace all continuous dynamics by its relational abstraction
- Result is uncountably infinite state discrete state transition system
- Use bounded model checker, or k-induction prover, or ...

Key summary points:
- Differential equations induce uncountably-infinite successors
- Fixpoint approaches unsuitable
- Certificate-based verification for CDSs eliminates need for fixpoint
- Relational abstraction = lifting certificate-based methods from CDSs to Hybrid Systems
- Fixpoint only on the discrete structure of the model
- In general, require $\exists \forall$ solving, which can be avoided for linear ODE dynamics
Problem: How to *wire* the components to synthesize a *desired* system?

Given $E$, find $E'$ s.t. $E' \Rightarrow E$
### Synthesis: Concrete Examples

<table>
<thead>
<tr>
<th>Desired System $F_{\text{spec}}$</th>
<th>Components $f_i$’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort an array</td>
<td>comparators</td>
</tr>
<tr>
<td>compute $\frac{x + y}{2}$</td>
<td>modulo arithmetic ops</td>
</tr>
<tr>
<td>find rightmost one</td>
<td>bitwise ops, arithmetic ops</td>
</tr>
<tr>
<td>compute $x^{243}$</td>
<td>multiplication</td>
</tr>
<tr>
<td>accept $\omega$-regular language</td>
<td>Buchi automata</td>
</tr>
<tr>
<td>safe hybrid system</td>
<td>multiple operating modes</td>
</tr>
<tr>
<td>geometry construction</td>
<td>ruler-compass steps</td>
</tr>
<tr>
<td>deobfuscated code</td>
<td>parts of obfuscated code</td>
</tr>
<tr>
<td>verification proof</td>
<td>verification inference rules</td>
</tr>
</tbody>
</table>

**Question:** $\exists C : \forall x : C(f_1, f_2, \ldots)(x) \Rightarrow F_{\text{spec}}(x)$
Synthesis Problem Classes

\[ \exists C : \forall x : C(f_1, f_2, \ldots)(x) \Rightarrow F_{\text{spec}}(x) \]

Parameters that define the synthesis problem:

- composition operator \( C \)
- class of specifications \( F_{\text{spec}} \)
- class of component specifications \( f_i \)

Fixing the synthesis problem:
fix these parameters, fix representation of \( F_{\text{spec}}, f_i \)
Bounded Synthesis

The synthesis problem is still hard

We make it feasible by replacing the unbounded quantifier, $\exists C$, by a bounded quantifier

$$\exists C : \forall x : C(f_1, f_2, \ldots)(x) \Rightarrow F_{\text{spec}}(x)$$

\[ \Downarrow \]

$$\exists c : \forall x : c(f_1, f_2, f_3)(x) \Rightarrow F_{\text{spec}}(x), \text{ } c \text{ in some finite set}$$

This bounded synthesis problem is solved by deciding the $\exists \forall$ formula

Examples: straight-line program synthesis, loop-free program synthesis, geometry constructions synthesis
Examples: Synthesized Programs

RoundUpToTheNextHighestPowerOf2(x):

1. \( o_1 := (x - 1) \)
2. \( o_2 := (o_1 \gg 1) \)
3. \( o_3 := o_1 | o_2 \)
4. \( o_4 := o_3 \gg 2 \)
5. \( o_5 := o_3 | o_4 \)
6. \( o_6 := o_5 \gg 4 \)
7. \( o_7 := o_5 | o_6 \)
8. \( o_8 := o_7 \gg 8 \)
9. \( o_9 := o_7 | o_8 \)
10. \( o_{10} := o_9 \gg 16 \)
11. \( o_{11} := o_9 | o_{10} \)
12. \( res := o_{10} + 1 \)
Examples: Synthesized Programs

HigherOrderHalfOfxy(x, y):

1. $o_1 := x \ & \ 0xFFFF$
2. $o_2 := x \gg 16$
3. $o_3 := y \ & \ 0xFFFF$
4. $o_4 := y \gg 16$
5. $o_5 := o_1 \ast o_3$
6. $o_6 := o_2 \ast o_3$
7. $o_7 := o_1 \ast o_4$
8. $o_8 := o_2 \ast o_4$
9. $o_9 := o_5 \gg 16$
10. $o_{10} := o_6 + o_9$
11. $o_{11} := o_{10} \ & \ 0xFFFF$
12. $o_{12} := o_{10} \gg 16$
13. $o_{13} := o_7 + o_{11}$
14. $o_{14} := o_{13} \gg 16$
15. $o_{15} := o_{14} + o_{12}$
16. $res := o_{15} + o_8$
Solving $\exists \forall$ Problems

When dynamics are not linear, and when dealing with other domains/synthesis, we need $\exists \forall$ solvers

Approaches:

- eliminating quantifiers, e.g. qepcad, virtual substitution
- replacing $\forall$ quantifiers by $\exists$ using duality theorems, such as Farkas Lemma and Positivstellensatz
- cleverly enumerating instances of the $\exists$ quantifier, CEG-$\forall \exists$ Solving
- using numerical methods based on semidefinite programming
∃∀ Solving: Semidefinite Programming

Special class of ∃∀ problems:

minimize \( c^T x \)
subject to \( F_0 + \sum_{i=1}^{m} x_i F_i \geq 0 \)

where \( c \in \mathbb{R}^m \) and \( F_0, \ldots, F_m \in \mathbb{R}^{n \times n} \) are symmetric matrices.

Logical reading of the feasibility instance:

\( \exists x \forall y : y^T (F_0 + \sum_{i=1}^{m} x_i F_i)y \geq 0 \)

Convex optimization/Interior point methods

Abstract to these solvable classes

Ashish Tiwari, SRI Intl. Logic in Software, Dynamical and Biological Systems: 33
Another class of $\exists \forall$ problems that reduce to SDP programming:

minimize $c^T x$

subject to $P_0(y) + \sum_{i=1}^{m} x_i P_i(y)$ is 0 (or SOS), . . . ,

where $c \in \mathbb{R}^m$ and $P_0, \ldots, P_m \in \mathbb{R}[y]$

Approximate logical reading of the feasibility instance:

$\exists x \forall y : (P_0 + \sum_{i=1}^{m} x_i P_i) \geq 0 \land \cdots$

Not applicable to $\exists x \forall y : (P_0(x, y) \geq 0 \land P_1(x, y) \geq 0 \Rightarrow P_2(x, y) \geq 0)$
Solving: Counter-Example Guided Solver

CE guided iterative procedure for solving $\exists \vec{u} : \forall \vec{x} : \phi(\vec{u}, \vec{x})$

1. Guess $\vec{u}_0$ for $\vec{u}$

2. (Verification) Check if

$$\forall \vec{x} : \phi(\vec{u}_0, \vec{x})$$

3. If true, then return $\vec{u}_0$

4. Get counterexample $\vec{x}_0$, add it to $X$

5. (Finite Synthesis) Find new $\vec{u}_0$ such that

$$\exists \vec{u}_0 : \bigwedge_{\vec{x}_0 \in X} \phi(\vec{u}_0, \vec{x}_0)$$

6. If unsatisfiable, return False, else goto Step 2
Solving $\exists \vec{u} : \forall \vec{x} : \phi(\vec{u}, \vec{x})$

1. $X :=$ some finite set of choices for $\vec{x}$

2. Find two values $\vec{u}_1, \vec{u}_2$ that work for $X$, but differ on some $\vec{x}_0$

$$\exists \vec{u}_1, \vec{u}_2, \vec{x}_0 : \left( \bigwedge_{\vec{x} \in X} (\phi(\vec{u}_1, \vec{x}) \land \phi(\vec{u}_2, \vec{x})) \right) \land (\phi(\vec{u}_1, \vec{x}_0) \not\Leftrightarrow \phi(\vec{u}_2, \vec{x}_0))$$

3. If satisfiable, we add $\vec{x}_0$ to $X$ and go to (2)

4. If unsatisfiable, then find one program that works for $X$

$$\exists \vec{u}_1 : \bigwedge_{\vec{x} \in X} \phi(\vec{u}_1, \vec{x})$$

5. If satisfiable, verify and return $\vec{u}_1$

6. Otherwise, return “unsatisfiable”
A third algorithm for solving $\exists \vec{u} : \forall \vec{x} : \phi(\vec{u}, \vec{x})$

1. Find finite set $X$ of good values for $\vec{x}$
2. Synthesize $\vec{u}_0$ that works for finite set $X$
3. Verify that $\vec{u}_0$ works on randomly sampled inputs

We can perform Step (2) using intelligently enumerating values for $\vec{u}$

Geometry synthesis
Enormous amounts of data being generated

- DNA sequencing: Fully sequencing genomes is rapid and easy
- DNA microarray: Which genes are being transcribed
- Proteomics: Which proteins are present
- Flow cytometry: Concentration in individual cells

And how to use it to predict clinical observations and phenotypes?
Model-based development

Also, a common feature in embedded system design

Goal: Models can help

- perform \textit{in-silico} experiments
- guide \textit{wet lab} experiments
- suggest novel \textit{drug} targets
Nutrient Sets

Goal: Starting from the genome, find nutrient sets on which that organism will grow

- Sequence genome of the organism
- Extract genes
- Predict metabolic network
- Predict growth on nutrient sets
Metabolic Network: Rewriting-based Modeling

Petri nets: Ground AC rewrite systems with 1 AC symbol

Example:

\[ a_1 : \quad A + B \rightarrow C + D \]
\[ a_2 : \quad C + A \rightarrow E \]

The numeric parameters \( a_1, a_2 \) capture relative affinity/preference/likelihood.

Typical metabolic networks have 1000’s of reactions and metabolites.

Also used to model other biochemical reactions: cell signaling.
Stochastic Firing: Chemical Master Equation

Strategy for firing rewrite rules: stochastic

Physics-based models of biochemical reaction networks: stochastic Petrinets

Semantics is given using the CME

\[ \frac{dP(X, t)}{dt} = \sum_{r \in R} a(P(X - r, t), r) \]

\( X \): set of metabolites, \(|X| = n\); e.g. \( X = \{A, B, C, D, E\} \)

\( R \): set of reactions

\( r \): a reaction, element of \( \mathbb{N}^n \); e.g. \( A + C \rightarrow E \leftrightarrow [-1, 0, -1, 0, 1] \)

\( P \): map from \( \mathbb{N}^+ \times \mathbb{R}^+ \rightarrow [0, 1] \)
**Stochastic Firing: Example**

\[ a_1 : \quad A + B \rightarrow C + D \quad a_2 : \quad C + A \rightarrow E \]

Evolving probability distribution:

<table>
<thead>
<tr>
<th>A=2,B=1,C=D=E=0</th>
<th>A=1,B=0,C=1,D=1,E=0</th>
<th>A=0,B=0,C=0,D=1,E=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>3</td>
<td>1/4</td>
<td>1/2</td>
</tr>
<tr>
<td>4</td>
<td>1/8</td>
<td>3/8</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Difficulty:** Not enough data to know how to compute \( a \)

Does not scale
Deterministic Firing: Mass Action Dynamics

Approximation of CME using ordinary differential equations

\[ a_1 : \quad A + B \rightarrow C + D \quad \quad a_2 : \quad C + A \rightarrow E \]

ODE model using mass action dynamics:

\[
\frac{dA(t)}{dt} = -a_1 \cdot A(t) \cdot B(t) - a_2 \cdot A(t) \cdot C(t)
\]
\[
\frac{dB(t)}{dt} = -a_1 \cdot A(t) \cdot B(t)
\]
\[
\frac{dC(t)}{dt} = -a_2 \cdot A(t) \cdot C(t) + a_1 \cdot A(t) \cdot B(t)
\]
\[
\frac{dD(t)}{dt} = a_1 \cdot A(t) \cdot B(t)
\]
\[
\frac{dE(t)}{dt} = a_2 \cdot A(t) \cdot C(t)
\]

Issue: (i) approximate (ii) Still need \( a_1, a_2 \)
Nondeterministic Firing: Rewriting

Preferable because we do not need extra parameters

Organism grows if it can produce biomass compounds starting from nutrients

This is a reachability question

Petri net reachability is decidable, but inefficient

Example: If $A$, $B$ are nutrients, and $E$ is a biomass compound, then:

$$2A + B \rightarrow A + C + D \rightarrow E + D$$
Reachability: Via Constraint Solving

We can perform approximate reachability via constraint solving

Example:

\[ A + B \implies C + D \quad C + A \implies E \]

Constraints: Suppose initial state is \(2A + B\), we want to reach \(D + E\)

\[
egin{align*}
A : & \quad -r_1 - r_2 + 2 = 0 \\
B : & \quad -r_1 + 1 = 0 \\
C : & \quad r_1 - r_2 = 0 \\
D : & \quad r_1 - 1 = 0 \\
E : & \quad r_2 - 1 = 0
\end{align*}
\]

If \(D + E\) is reachable from \(2A + B\), then above constraints are satisfiable

This is called Flux Balance Analysis
Nutrient Sets for E.Coli

We have used constraint solving for finding (minimal) nutrient sets for E.Coli.

Exact Reachability is defined as the least fixpoint.

**Flux Balance Analysis:** an overapproximation of the reachability relation.

We developed a constraint-based approach that captures reachability more accurately than FBA.

**Results:**

1. About **75% accuracy** with experimental results.
2. Predicted growth of E.Coli on **cynate** as both Carbon and Nitrogen source, which was **experimentally verified**.
3. Can compute **all** minimal nutrient sets for E.Coli.
Logic in Software Verification

1 \( x := 0; \ y := 0; \ z := n; \)
2 while (*) {
3     if (*) {
4         x := x+1;
5         z := z-1;
6     } else {
7         y := y+1;
8         z := z-1;
9     }
10 }

Ashish Tiwari, SRI Intl. Logic in Software, Dynamical and Biological Systems: 48
Traditional Approach: Annotate & Check

1 x := 0; y := 0; z := n;
   [ z + x + y == n ]
2 while (*) {
3   if (*) {
4     x := x+1;
5     z := z-1;
6     [ z + x + y == n ]
7   } else {
8     y := y+1;
9     z := z-1;
10    [ z + x + y == n ]
11   }
12 }
Ashish Tiwari, SRI Intl. Logic in Software, Dynamical and Biological Systems: 49
Traditional Approach: Annotate & Check

Proof obligation generated:

\[
z + x + y = n \land x' = x + 1 \land z' = z - 1 \land y' = y
\]

\[
\Rightarrow z' + x' + y' = n
\]

\[
z + x + y = n \land y' = y + 1 \land z' = z - 1 \land x' = x
\]

\[
\Rightarrow z' + x' + y' = n
\]

The theory \( T \) determined by semantics of the programming language.
Example: Abstract Interpretation

[ true ]
1 \( x := 0; y := 0; z := n; \)
\[ x = 0 \land y = 0 \land z = n \] \( \exists x, y, z : x = 0 \land y = 0 \land z = n \)
2 \text{while} (\ast) \{ \\
3 \quad \text{if} (\ast) \{ \\
4 \quad \quad x := x+1; \\
5 \quad \quad z := z-1; \ [ (x = 1 \land y = 0 \land z = n - 1) ] \\
6 \quad \} \text{else} \{ \\
7 \quad \quad y := y+1; \\
8 \quad \quad z := z-1; \ [ (x = 0 \land y = 1 \land z = n - 1) ] \\
9 \quad \}
\ [ (x = 1 \land y = 0 \land z = n - 1) \lor (x = 0 \land y = 1 \land z = n - 1) \] 
10 \}
Example: Abstract Interpretation

Suppose we can only use conjunctions of atomic facts

We need to overapproximate

- the $\exists$ quantifier
- the $\lor$ operator

We need to find a conjunction of atomic formulas that is implied by

- $\exists x, y, z : x = 0 \land y = 0 \land z = n \land x = x + 1 \land z = z - 1 \land y = y \rightarrow x = 1 \land y = 0 \land z = n - 1$

- $(x = 1 \land y = 0 \land z = n - 1) \lor (x = 0 \land y = 1 \land z = n - 1) \rightarrow x + y = 1 \land z = n - 1$
Example: Abstract Interpretation

\[
\begin{align*}
\text{true} & \quad \text{[ true ]} \\
1 & \quad x := 0; \ y := 0; \ z := n; \\
\quad & \quad [ \ x = 0 \land y = 0 \land z = n \ ] \\
2 & \quad \text{while } (\ast) \ {\{} \\
\quad & \quad \quad [ (x = 0 \land y = 0 \land z = n) \lor (x + y = 1 \land z = n - 1) ] \\
3 & \quad \text{if } (\ast) \ {\{} \\
4 & \quad \quad x := x + 1; \\
5 & \quad \quad z := z - 1; \quad [ (x = 1 \land y = 0 \land z = n - 1) ] \\
6 & \quad \} \ \text{else } {\{} \\
7 & \quad \quad y := y + 1; \\
8 & \quad \quad z := z - 1; \quad [ (x = 0 \land y = 1 \land z = n - 1) ] \\
9 & \quad \} \\
10 & \quad [ (x + y = 1 \land z = n - 1) ]
\end{align*}
\]
Hence, we need to over-approximate

\[
((x + y = 1 \land z = n - 1) \lor x = 0 \land y = 0 \land z = n) \\
\]

\[
(x + y = 1 \land z = n - 1) \quad \text{T} \quad \Rightarrow \quad z + x + y = n \\
(x = 0 \land y = 0 \land z = n) \quad \text{T} \quad \Rightarrow \quad z + x + y = n
\]

We get the loop invariant \(z + x + y = n\).
Abstract Interpretation over logical lattices

Lattices defined by

- elements: some subset of formulas in \( T \) closed under \( \wedge \)
- partial order: some subset of \( \Rightarrow \)

A common class is strictly logical lattices:

- elements: conjunction \( \phi \) of atomic formulas in \( T \)
- partial order: \( \phi \sqsubseteq \phi' \) if \( T \models \phi \Rightarrow \phi' \)
In any logical lattice

- **meet** $\sqcap$ $\mapsto$ (over-approximation of) logical and $\land \ (\lceil \land \rceil )$
- **join** $\sqcup$ $\mapsto$ over-approximation of logical or $\lor \ (\lceil \lor \rceil )$
- **partial order** $\sqsubseteq$ $\mapsto$ under-approximation of logical implies $\Rightarrow \ (\lfloor \Rightarrow \rfloor )$
- **projection** $\mapsto$ over-approximation of logical exists $\exists \ (\lceil \exists \rceil )$

In strictly logical lattices:

- **meet** $\sqcap$ $\mapsto$ $\land$
- **join** $\sqcup$ $\mapsto$ $\phi_1 [\lor] \phi_2$ is the strongest $\phi \in \Phi$ s.t. $\phi_i \Rightarrow \phi$ for $i = 1, 2$
- **partial order** $\sqsubseteq$ $\mapsto$ $\Rightarrow$
- **projection** $\mapsto$ $\exists U.\phi$ is the strongest $\phi' \in \Phi$ s.t. $(\exists U.\phi) \Rightarrow \phi'$

**Challenge:** For what domains can we efficiently compute these operations?
Over-Approximation of $\lor$: Examples

- **Linear arithmetic with equality** (Karr 1976)
  Eg. $\{x = 0, y = 1\} \lor \{x = 1, y = 0\} = \{(x + y = 1)\}$

- **Linear arithmetic with inequalities** (Cousot and Halbwachs 1978)
  Eg. $\{x = 0\} \lor \{x = 1\} = \{0 \leq x, x \leq 1\}$

- **Nonlinear equations** (polynomials) (Rodriguez-Carbonell and Kapur 2004)
  Eg. $\{x = 0\} \lor \{x = 1\} = \{x(x - 1) = 0\}$

- **Term Algebra** (Gulwani, T. and Necula 2004)
  Eg. $\{x = a, y = f(a)\} \lor \{x = b, y = f(b)\} = \{y = f(x)\}$
UFS does not define a logical lattice.

The $\left[ \lor \right]$ of two finite sets of facts need not be finitely presented. [Gulwani, T. and Necula 2004]

\[
\begin{align*}
\phi_1 & \equiv \{a = b\} \\
\phi_2 & \equiv \{fa = a, \ fb = b, \ ga = gb\} \\
\phi_1 \lor \phi_2 & \equiv \bigwedge_i gf^i a = gf^i b
\end{align*}
\]

The formula $\bigwedge_i gf^i a = gf^i b$ can not be represented by finite set of ground equations.

Proof. It induces infinitely many congruence classes with more than one signature.
Combining Logical Interpreters: Motivation

\begin{align*}
  x &:= 0; \ y := 0; \quad x := c; \ y := c; \quad x := 0; \ y := 0; \\
  u &:= 0; \ v := 0; \quad u := c; \ v := c; \quad u := 0; \ v := 0; \\
  \text{while (*) } \{ \quad &\text{while (*) } \{ \quad \text{while (*) } \{ \\
  \quad x := u + 1; \quad x := G(u, 1); \quad x := u + 1; \\
  \quad y := 1 + v; \quad y := G(1, v); \quad y := 1 + v; \\
  \quad u := F(x); \quad u := F(x); \quad u := *; \\
  \quad v := F(y); \quad v := F(y); \quad v := *; \\
  \} \quad } \quad } \quad \\
  \text{assert( } x = y \text{ )} \quad \text{assert( } x = y \text{ )} \quad \text{assert( } x = y \text{ )} \\
  \Sigma &= \Sigma_{LA} \cup \Sigma_{UFS} \quad \Sigma &= \Sigma_{UFS} \quad \Sigma = \Sigma_{LA} \\
  T &= T_{LA} + T_{UFS} \quad T = T_{UFS} \quad T = T_{LA}
\end{align*}
Combining abstract interpreters is not easy [Cousot76]

For combining logical interpreters (over strictly logical lattices), we need to combine:

- $\lceil \lor \rceil$
- $\lceil \exists \rceil$
- $T$
- $\Rightarrow$

Example:

$$(x = 0 \land y = 1) \lceil \lor \rceil (x = 1 \land y = 0)$$

$$= x + y = 1 \land C[x] + C[y] = C[0] + C[1]$$
Logical Product

Given two logical lattices, we define the logical product $L_1 \ast L_2$ as:

- **elements**: conjunction $\phi$ of atomic formulas in $T_1 \cup T_2$
- $E \subseteq E'$ : $E \Rightarrow_{T_1 \cup T_2} E'$ and $\text{AlienTerms}(E') \subseteq \text{Terms}(E)$

$\text{AlienTerms}(E) = \text{subterms in } E\text{ that belong to different theory}$

$\text{Terms}(E) = \text{all subterms in } E\text{, plus all terms equivalent to these subterms (in } T_1 \cup T_2 \cup E)$

Eg. $\{x = F(a+1), y = a\} \lor \{x = F(b+1), y = b\} = \{x = F(y+1)\}$ since:

\[
\begin{align*}
x &= F(a+1) \land y = a \Rightarrow x = F(y+1) \\
x &= F(b+1) \land y = b \Rightarrow x = F(y+1) \\
x &= F(a+1) \land y = a \Rightarrow y+1 = a+1 \\
x &= F(b+1) \land y = b \Rightarrow y+1 = b+1
\end{align*}
\]
Combining the ⇒ Test

Combining satisfiability procedures

Nelson-Oppen combination method
Combining \( \lor \) Operators

Given procedures:

\[ L_1(E_l, E_r) \]
\[ L_2(E_l, E_r) \]

We wish to compute \( E_l \lor E_r \) in the logical product \( L_1 \ast L_2 \)

Example.

\[ \{ z = a - 1, y = f(a) \} \lor \{ z = b - 1, y = f(b) \} = \{ y = f(1 + z) \} \]
Combining $\bigvee$ Operators

$z = a - 1, \, y = f(a)$  \quad $z = b - 1, \, y = f(b)$

Purify+NOSat  \quad $z = a - 1 \quad y = f(a) \quad z = b - 1 \quad y = f(b)$

LR-Exchange  \quad $a = \langle a, b \rangle \quad a = \langle a, b \rangle \quad b = \langle a, b \rangle \quad b = \langle a, b \rangle$

Base  \quad $\bigvee$  \quad $\bigvee$  \quad $\bigvee$

$\langle a, b \rangle = 1 + z \quad y = f(\langle a, b \rangle)$

Quant Elim  \quad $\exists$  \quad $UF*LA$

Return  \quad $y = f(1 + z)$
The $\exists$ Operator

Required to compute transfer function for assignments

$E = \exists_L V : (E')$ if $E$ is the least element in lattice $L$ s.t.

- $E' \sqsubseteq_L E$
- $\text{Vars}(E) \cap V = \emptyset$

Examples:

- $\exists_L a : (x < a \land a < y) = (x < y)$
- $\exists_U F a : (x = f(a) \land y = f(f(a))) = (y = f(x))$
- $\exists_{LA*UF} a, b, c : (a < b < y \land z = c + 1 \land a = f f b \land c = f b) = (f(z - 1) < y)$

How to construct $\exists_{LA*UF}$ using $\exists_L A$ and $\exists_U F$?
## Combining $\exists$ Operators

### Problem

\[
a < b < y, \ z = c + 1, \ a = f f b, \ c = fb \quad \{a, b, c\}
\]

### Purify+NOSat

\[
a < b < y, \ z = c + 1 \quad a = f f b, \ c = fb
\]

### QSat

\[
a \mapsto fc \quad c \mapsto z - 1
\]

### Base $\exists$

\[
\exists LA \quad \exists UF
\]

\[
a < y, \ z = c + 1 \quad a = fc
\]

### Substitute

\[
c \mapsto z - 1, \ a \mapsto fc
\]

### Return

\[
f(z - 1) < y
\]
Quantified Abstract Domain

Lifting base logical domains to quantified domains

array-init(A,n)

1 for (i = 0; i < n; i++) {
2 \( A[i] = 0 \)
3 } \[ \forall k (0 \leq k < n \Rightarrow A[k] = 0) \]
Array Initialization

array-init(A, n)
1  for (i = 0; i < n; i++) {
    (i = 1 ∧ A[0] = 0) ∨ (i = 2 ∧ A[0] = 0 ∧ A[1] = 0)
2  A[i] = 0
3  }

Let us write it out as a quantified fact.
Array Initialization

\begin{verbatim}
array-init(A, n)
  1 for (i = 0; i < n; i++) {
      (i = 1 ∧ ∀k(k = 0 ⇒ A[k] = 0)) ∨
      (i = 2 ∧ ∀k(k = 0 ⇒ A[k] = 0) ∧ ∀k(k = 1 ⇒ A[k] = 0))
    2 A[i] = 0
    3 }

Too many quantified facts...let us merge them into one.

i = 2 ∧ ∀k(--- ⇒ A[k] = 0)

--- should be k = 0 [∨] k = 1:

0 ≤ k ≤ 1 ⇒ (k = 0 ∨ k = 1)
\end{verbatim}
Array Initialization

\[\text{array-init}(A,n)\]

1 \hspace{1em} \text{for} \hspace{1em} (i = 0; \hspace{0.5em} i < n; \hspace{0.5em} i++) \hspace{1em} \{ \\
\hspace{1em} i = 1 \land \forall k (k = 0 \Rightarrow A[k] = 0) \lor \\
\hspace{1em} i = 2 \land \forall k (0 \leq k < 2 \Rightarrow A[k] = 0) \\
2 \hspace{1em} A[i] = 0 \\
3 \hspace{1em} \} \\

Now we need to \(\bigvee\) of two quantified facts.
Array Initialization

\[
i = 1 \quad \quad [\vee] \quad \quad i = 2
\]

\[
\forall k (k = 0 \Rightarrow A[k] = 0)
\]

\[
1 \leq i \leq 2
\]

\[
\forall k (0 \leq k < 2 \Rightarrow A[k] = 0)
\]

Obviously, \(_-\) should be \(k = 0 \quad [\wedge] \quad 0 \leq k < 2\).

\(k = 0\) is no good.
Array Initialization

\[
i = 1 \quad \begin{array}{c} [\lor] \end{array} \quad i = 2 \\
\forall k(k = 0 \Rightarrow A[k] = 0) \quad \forall k(0 \leq k < 2 \Rightarrow A[k] = 0) \\
1 \leq i \leq 2 \\
\forall k(\_\_\_ \Rightarrow A[k] = 0)
\]

Actually, \_\_\_ should be

\[
i = 1 \Rightarrow k = 0 \quad [\land] \quad i = 2 \Rightarrow 0 \leq k < 2
\]

Let us see if the answer satisfies this.

\[
0 \leq k < i \Rightarrow (i = 1 \Rightarrow k = 0 \land i = 2 \Rightarrow 0 \leq k < 2)
\]
The Quantified Domain

\[ E \land \bigwedge \forall U_i (F_i \implies e_i) \]

where \( E, F, e \) are members of three base domains, requires

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_1 \uparrow \lor E_2 )</td>
<td>join of ( E_1 ) and ( E_2 )</td>
</tr>
<tr>
<td>( E_1 \uparrow \land E_2 )</td>
<td>meet of ( E_1 ) and ( E_2 )</td>
</tr>
<tr>
<td>( \exists \ x. E )</td>
<td>eliminate ( x ) from ( E )</td>
</tr>
<tr>
<td>( E_1 \Rightarrow E_2 )</td>
<td>partial order test comparing ( E_1 ) and ( E_2 )</td>
</tr>
<tr>
<td>( (E_1 \lor E_2)/E )</td>
<td>under-approximate ( E \Rightarrow (E_1 \lor E_2) )</td>
</tr>
<tr>
<td>( (E_1 \Rightarrow E'_1) \land (E_2 \Rightarrow E'_2) )</td>
<td>underapprox. ( (E_1 \Rightarrow E'_1) \land (E_2 \Rightarrow E'_2) )</td>
</tr>
<tr>
<td>( \forall \ x.(E \Rightarrow E') )</td>
<td>underapproximate ( \forall x(E \Rightarrow E') )</td>
</tr>
</tbody>
</table>
Logical Interpretation: Summary

- **Logical lattices** are good candidates for thinking about and building abstract interpreters

  Logical Interpretation: $\llbracket \lor \rrbracket$, $\llbracket \exists \rrbracket$, $\Rightarrow$

  Logical Product: Combination Algorithms

  Quantified Extension: $\llbracket \lor \rrbracket$, $\llbracket \land \rrbracket$, $\llbracket \forall \rrbracket$, abduction

- The **assertion checking** problem for program classes:
  - Is related to T-unification
  - **Unification type** determines complexity
  - Interprocedural analysis needs context unification
<table>
<thead>
<tr>
<th>CDS</th>
<th>HS</th>
<th>Synthesis</th>
<th>Syst Bio.</th>
<th>S/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>( (M \models \phi) ? )</td>
<td>( (M \models \phi) ? )</td>
<td>( (M? \models \phi) )</td>
<td>( (M \models \phi) ? )</td>
<td>( (M \models \phi) ? )</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>( M \models \phi' ), ( M \Rightarrow M' ), ( \phi' \Rightarrow \phi )</td>
<td>( M' \Rightarrow \phi )</td>
<td>( \exists \forall \psi )</td>
<td>( \forall \psi )</td>
<td>( M \models \phi' ), ( \phi' \Rightarrow \phi )</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>( \exists \forall \psi )</td>
<td>( \exists \forall \psi, M' \models \phi )</td>
<td>( \exists \forall \psi )</td>
<td>( \forall \psi )</td>
<td>Logical Interp.</td>
</tr>
</tbody>
</table>

\[ \exists \forall \text{solver} \]  \[ \forall \text{solver} \]  \[ \text{Approx. ops} \]
Conclusion

SMT Solvers have revolutionized solving of $\forall$ formulas

Possible directions of evolution:

- $\exists \forall$ SMT Solvers
- Approximating SMT Solvers
- SMT+ and SMT- Solvers
- Probabilistic SMT Solvers