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 $\mapsto$ $\exists\forall$ solving

II. Hybrid system verification $\mapsto$ $\exists\forall$ solving + discrete system verification

III. Component-based Synthesis $\mapsto$ $\exists\forall$ solving

IV. $\exists\forall$ Solvers

V. Systems Biology $\mapsto$ $\forall$ solving

VI. Program verification $\mapsto$ Approximating logical operators

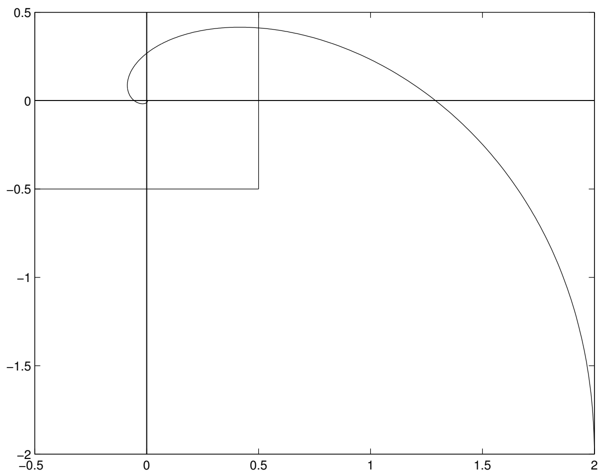
Continuous Dynamical Systems

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

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

Example CDS's dynamics are given by:

$$\begin{aligned} \frac{dx_1}{dt} &= -x_1 - x_2 \\ \frac{dx_2}{dt} &= x_1 - x_2 \end{aligned}$$

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 Φ 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:

Property ϕ	Certificate Φ
safety	inductive invariant
stability	Lyapunov function
termination	ranking function
controlled safety	controlled inductive invariant

Certificate-Based Verification

Certificate-based verification reduces the verification problem to an $\exists\forall$ formula.

$$M \models \phi$$

$$\Uparrow$$

$$\exists \Phi : ((M \models \Phi) \;\wedge\; (\Phi \Rightarrow \phi))$$

$$\Uparrow$$

$$\exists \Phi : \forall \vec{x} : \text{ quantifier-free FO formula}$$

$$\Uparrow$$

$$\exists \vec{a} : \forall \vec{x} : \text{ quantifier-free FO formula}$$

The last step performed by choosing a template for Φ

Inductive Invariants for CDSs

Used to prove safety of CDSs

How to define inductiveness ?

A set I is inductive if

$$\forall \vec{x} : \vec{x} \in I \ \wedge \ \vec{x} \rightarrow \vec{y} \Rightarrow \vec{y} \in I$$

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:

$$\begin{aligned}\frac{dx_1}{dt} &= -x_1 - x_2 \\ \frac{dx_2}{dt} &= x_1 - x_2\end{aligned}$$

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

Lie Derivative

Let $p := x_1^2 + x_2^2 - 0.5$

The set $p \leq 0$ is inductive if

$$\begin{aligned} p = 0 \quad \Rightarrow \quad & \frac{dp}{dt} < 0 \\ & \vee \frac{dp}{dt} = 0 \wedge \frac{d^2p}{dt^2} < 0 \\ & \vee \frac{dp}{dt} = \frac{d^2p}{dt^2} = 0 \wedge \frac{d^3p}{dt^3} < 0 \\ & \ldots \end{aligned}$$

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

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

$$\exists a, b, c : \forall x_1, x_2 : \qquad \begin{array}{l} (p = 0 \Rightarrow \frac{dp}{dt} < 0) \;\wedge \\ (x_1 \leq 0.5 \wedge x_2 \leq 0.5 \Rightarrow p \leq 0) \;\wedge \\ (p \leq 0 \Rightarrow x_2 \leq 1) \end{array}$$

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

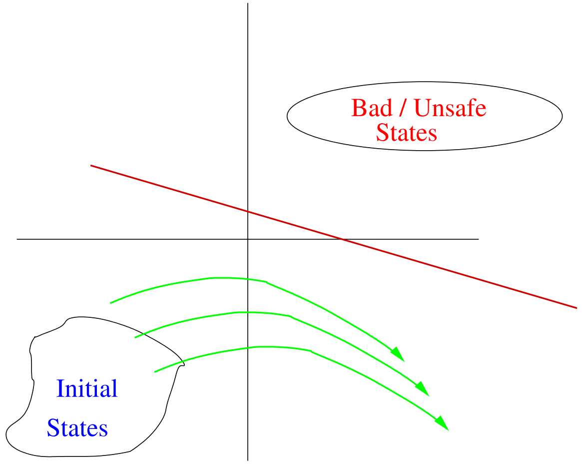
Without solving $\exists\forall$ formulas

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

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

$$\begin{aligned} \dot{v} &= a \\ \dot{a} &= -4v + 3v_f - 3a + gap \\ \dot{gap} &= -v + v_f \end{aligned}$$

where v, a is the velocity and acceleration of this car, v_f is the velocity of car in front, and 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 $\langle Q, (\texttt{CDS}_q)_{q \in Q}, E \rangle$ where

Q: finite set of modes

$\texttt{CDS}_q$: 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:

$$\frac{dy}{dt} = v \qquad\qquad \frac{dv}{dt} = -9.8$$

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

Discrete transition given by:

$$(q, (0, v), q, (0, -0.9 * v))$$

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:

$$x \rightarrow x' \quad := \quad 0 < x' \leq x \ \vee \ x \leq x' < 0 \ \vee \ x = x' = 0$$

Computing Relational Abstractions

We can compute good quality relational abstractions of linear systems

Dynamics	Relational Abstraction
$\dot{x} = 1, \dot{y} = 1$	$x' - x = y' - y \ \wedge \ x' \geq x$
$\dot{x} = 2, \dot{y} = 3$	$(x' - x)/2 = (y' - y)/3 \ \wedge \ x' \geq x$
$\dot{\vec{x}} = A\vec{x}$	$(0 < p' \leq p) \ \vee \ (p \leq p' < 0) \ \vee \ (p = p' = 0)$, where $p = \vec{c}^T \vec{x}$, $\vec{c}$ eigenvector of A^T corr. to negative eigenvalue Similarly for eigenvector corr. to positive eigenvalue Coarser abstraction for complex eigenvalues

Complete for timed, multirate, linear hybrid automata

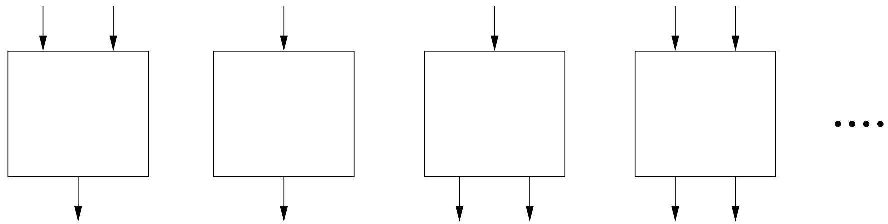
Using Relational Abstraction

- 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

Component-Based Synthesis



Problem: How to wire the components to synthesize a desired system ?

Given E, find E' s.t. $E' \Rightarrow E$

Synthesis: Concrete Examples

Desired System $F_{\mathtt{spec}}$	Components f_i's
sort an array	comparators
compute $\frac{x+y}{2}$	modulo arithmetic ops
find rightmost one	bitwise ops, arithmetic ops
compute x^{243}	multiplication
accept ω-regular language	Buchi automata
safe hybrid system	multiple operating modes
geometry construction	ruler-compass steps
deobfuscated code	parts of obfuscated code
verification proof	verification inference rules

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

Synthesis Problem Classes

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

Parameters that define the synthesis problem:

- composition operator C
- class of specifications $F_{\mathtt{spec}}$
- class of component specifications f_i

Fixing the synthesis problem:
fix these parameters, fix representation of $F_{\mathtt{spec}}, f_i$

Bounded Synthesis

The synthesis problem is still hard

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

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

$$\Downarrow$$

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

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. $o_1 := x \ \& \ \texttt{0xFFFF}$
2. $o_2 := x \gg 16$
3. $o_3 := y \ \& \ \texttt{0xFFFF}$
4. $o_4 := y \gg 16$
5. $o_5 := o_1 * o_3$
6. $o_6 := o_2 * o_3$
7. $o_7 := o_1 * o_4$
8. $o_8 := o_2 * o_4$
9. $o_9 := o_5 \gg 16$
10. $o_{10} := o_6 + o_9$
11. $o_{11} := o_{10} \ \& \ \texttt{0xFFFF}$
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-$\exists\forall$ Solving
- using numerical methods based on semidefinite programming

$\exists\forall$ Solving: Semidefinite Programming

Special class of $\exists\forall$ 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

$\exists\forall$ Solving: Sum-of-Squares Programming

Another class of $\exists\forall$ problems that reduce to SDP programming:

minimize $\quad c^T x$

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

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

Approximate logical reading of the feasibility instance:

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

Not applicable to $\exists x \forall y : (P_0(x, y) \geq 0 \wedge P_1(x, y) \geq 0 \Rightarrow P_2(x, y) \geq 0)$

$\exists\forall$ 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

$\exists\forall$ Solving: Distinguishing Input

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 $\vec{x}_0$

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

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”

$\exists\forall$ Solving: A Nonsymbolic Solver

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

Biology

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?

Systems Biology

Model-based development

Also, a common feature in embedded system design

Goal: Models can help

- perform *in-silico* experiments
- guide *wet lab* experiments
- suggest novel 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

Petrinets: Ground AC rewrite systems with 1 AC symbol

Example:

$$\begin{array}{lrcl} a_1 : & A + B & \to & C + D \\ a_2 : & C + A & \to & E \end{array}$$

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

- 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. $A + C \to E \mapsto [-1, 0, -1, 0, 1]$
- P: map from ${N^+}^n \times \mathbb{R}^+ \mapsto [0, 1]$

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

Stochastic Firing: Example

$$a_1 : \quad A + B \rightarrow C + D \qquad a_2 : \quad C + A \rightarrow E$$

Evolving probability distribution:

	A=2,B=1,C=D=E=0	A=1,B=0,C=1,D=1,E=0	A=0,B=0,C=0,D=1,E=1
1	1	0	0
2	1/2	1/2	0
3	1/4	1/2	1/4
4	1/8	3/8	1/2
5	...	...	...
6	0	0	1

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

$$a_1: \quad A + B \;\rightarrow\; C + D \qquad\qquad a_2: \quad C + A \;\rightarrow\; E$$

ODE model using mass action dynamics:

$$
\begin{aligned}
\frac{dA(t)}{dt} &= -a_1 * A(t) * B(t) - a_2 * A(t) * C(t) \\
\frac{dB(t)}{dt} &= -a_1 * A(t) * B(t) \\
\frac{dC(t)}{dt} &= -a_2 * A(t) * C(t) + a_1 * A(t) * B(t) \\
\frac{dD(t)}{dt} &= a_1 * A(t) * B(t) \\
\frac{dE(t)}{dt} &= a_2 * A(t) * C(t)
\end{aligned}
$$

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

Petrinet 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:

$$A + B \;\rightarrow\; C + D \qquad\qquad C + A \;\rightarrow\; E$$

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

$$\begin{aligned}
A: &\quad -r_1 - r_2 + 2 = 0 \\
B: &\quad -r_1 + 1 = 0 \\
C: &\quad r_1 - r_2 = 0 \\
D: &\quad r_1 - 1 = 0 \\
E: &\quad r_2 - 1 = 0
\end{aligned}$$

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

```
x := 0; y := 0; z := n;
while (*) {
    if (*) {
        x := x+1;
        z := z-1;
    } else {
         y := y+1;
         z := z-1;
    }
}
```

Traditional Approach: Annotate & Check

```
x := 0; y := 0; z := n;
[ z + x + y == n ]
while (*) {
    if (*) {
        x := x+1;
        z := z-1;
        [ z + x + y == n ]
    } else {
         y := y+1;
         z := z-1;
         [ z + x + y == n ]
    }
}
```

Traditional Approach: Annotate & Check

Proof obligation generated:

$$z + x + y = n \;\wedge\; x' = x + 1 \;\wedge\; z' = z - 1 \wedge y' = y$$
$$\stackrel{\mathbf{T}}{\Rightarrow} \quad z' + x' + y' = n$$

$$z + x + y = n \;\wedge\; y' = y + 1 \;\wedge\; z' = z - 1 \wedge x' = x$$
$$\stackrel{\mathbf{T}}{\Rightarrow} \quad z' + x' + y' = n$$

The theory $\mathbf{T}$ determined by semantics of the programming language.

Example: Abstract Interpretation

```
  [ true ]
x := 0; y := 0; z := n;
```

$[\ x = 0 \wedge y = 0 \wedge z = n\ ]$ $\qquad \exists \overline{x}, \overline{y}, \overline{z} : x = 0 \wedge y = 0 \wedge z = n$

```
while (*) {
    if (*) {
        x := x+1;
        z := z-1;
```

$[\ (x = 1 \wedge y = 0 \wedge z = n - 1)\ ]$

```
    } else {
        y := y+1;
        z := z-1;
```

$[\ (x = 0 \wedge y = 1 \wedge z = n - 1)\ ]$

```
    }
```

$[\ (x = 1 \wedge y = 0 \wedge z = n - 1) \vee (x = 0 \wedge y = 1 \wedge z = n - 1)\ ]$

```
}
```

Example: Abstract Interpretation

Suppose we can only use conjunctions of atomic facts

We need to overapproximate

- the $\exists$ quantifier
- the $\vee$ operator

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

- $\exists \overline{x}, \overline{y}, \overline{z} : \overline{x} = 0 \wedge \overline{y} = 0 \wedge \overline{z} = n \wedge x = \overline{x} + 1 \wedge z = \overline{z} - 1 \wedge y = \overline{y}$
  $\longrightarrow \quad x = 1 \wedge y = 0 \wedge z = n - 1$
- $(x = 1 \wedge y = 0 \wedge z = n - 1) \vee (x = 0 \wedge y = 1 \wedge z = n - 1)$
  $\longrightarrow \quad x + y = 1 \wedge z = n - 1$

Example: Abstract Interpretation

```
  [ true ]
x := 0; y := 0; z := n;
  [ x = 0 ∧ y = 0 ∧ z = n ]
while (*) {
    [ (x = 0 ∧ y = 0 ∧ z = n) ∨ (x + y = 1 ∧ z = n − 1) ]
    if (*) {
        x := x+1;
        z := z-1;  [ (x = 1 ∧ y = 0 ∧ z = n − 1) ]
    } else {
        y := y+1;
        z := z-1;  [ (x = 0 ∧ y = 1 ∧ z = n − 1) ]
    }
    [ (x + y = 1 ∧ z = n − 1) ]
}
```

Hence, we need to over-approximate

$$((x + y = 1 \wedge z = n - 1) \ \vee \ x = 0 \wedge y = 0 \wedge z = n)$$

$$(x + y = 1 \wedge z = n - 1) \quad \stackrel{\mathbf{T}}{\Rightarrow} \quad z + x + y = n$$
$$(x = 0 \wedge y = 0 \wedge z = n) \quad \stackrel{\mathbf{T}}{\Rightarrow} \quad z + x + y = n$$

We get the loop invariant $z + x + y = n$.

Logical Interpretation

Abstract Interpretation over logical lattices

Lattices defined by

elements	:	some subset of formulas in $\mathbf{T}$ closed under $\wedge$
partial order	:	some subset of $\stackrel{\mathbf{T}}{\Rightarrow}$

A common class is strictly logical lattices:

elements	:	conjunction ϕ of atomic formulas in $\mathbf{T}$
partial order	:	$\phi \sqsubseteq \phi'$ if $\mathbf{T} \models \phi \Rightarrow \phi'$

In any logical lattice

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

In strictly logical lattices:

meet $\sqcap$	$\mapsto$	$\wedge$
join $\sqcup$	$\mapsto$	$\phi_1 \lceil \vee \rceil \phi_2$ is the strongest $\phi \in \Phi$ s.t. $\phi_i \overset{\mathbf{T}}{\Rightarrow} \phi$ for $i = 1, 2$
partial order $\sqsubseteq$	$\mapsto$	$\overset{\mathbf{T}}{\Rightarrow}$
projection	$\mapsto$	$\lceil \exists \rceil U.\phi$ is the strongest $\phi' \in \Phi$ s.t. $(\exists U.\phi) \overset{\mathbf{T}}{\Rightarrow} \phi'$

Challenge: For what domains can we efficiently compute these operations?

Over-Approximation of $\vee$: Examples

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

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

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

- Term Algebra (Gulwani, T. and Necula 2004)
  Eg. $\{x = a, y = f(a)\} \lceil\vee\rceil \{x = b, y = f(b)\} = \{y = f(x)\}$

UFS does not define a logical lattice

The $\lceil\vee\rceil$ of two finite sets of facts need not be finitely presented. [Gulwani, T. and Necula 2004]

$$
\begin{aligned}
\phi_1 &\equiv \{a = b\} \\
\phi_2 &\equiv \{fa = a,\ fb = b,\ ga = gb\} \\
\phi_1 \lceil\vee\rceil \phi_2 &\equiv \bigwedge_i gf^i a = gf^i b
\end{aligned}
$$

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

```
x :=0; y := 0;
u := 0; v := 0;
while (*) {
    x := u + 1;
    y := 1 + v;
    u := F(x);
    v := F(y);
}
assert( x = y )
```

$\Sigma = \Sigma_{LA} \cup \Sigma_{UFS}$

$\mathbf{T} = \mathbf{T}_{LA} + \mathbf{T}_{UFS}$

```
x := c; y := c;
u := c; v := c;
while (*) {
    x := G(u, 1);
    y := G(1, v);
    u := F(x);
    v := F(y);
}
assert( x = y )
```

$\Sigma = \Sigma_{UFS}$

$\mathbf{T} = \mathbf{T}_{UFS}$

```
x :=0; y := 0;
u := 0; v := 0;
while (*) {
    x := u + 1;
    y := 1 + v;
    u := *;
    v := *;
}
assert( x = y )
```

$\Sigma = \Sigma_{LA}$

$\mathbf{T} = \mathbf{T}_{LA}$

Combining Logical Interpreters

Combining abstract interpreters is not easy [Cousot76]

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

- $\lceil \vee \rceil$
- $\lceil \exists \rceil$
- $\stackrel{\mathbf{T}}{\Rightarrow}$

Example:

$$\begin{aligned} & (x = 0 \wedge y = 1) \;\; \lceil \vee \rceil \;\; (x = 1 \wedge y = 0) \\ = \;\; & x + y = 1 \wedge C[x] + C[y] = C[0] + C[1] \end{aligned}$$

Logical Product

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

$$
\begin{array}{lcl}
\text{elements} & : & \text{conjunction } \phi \text{ of atomic formulas in } \mathbf{T}_1 \cup \mathbf{T}_2 \\
E \sqsubseteq E' & : & E \Rightarrow_{\mathbf{T}_1 \cup \mathbf{T}_2} E' \text{ and } \mathit{AlienTerms}(E') \subseteq \mathit{Terms}(E)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{AlienTerms}(E) & = & \text{subterms in } E \text{ that belong to different theory} \\
\mathit{Terms}(E) & = & \text{all subterms in } E\text{, plus all terms equivalent} \\
 & & \text{to these subterms (in } \mathbf{T}_1 \cup \mathbf{T}_2 \cup E)
\end{array}
$$

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

$$
\begin{array}{rcl}
x = F(a+1) \wedge y = a & \Rightarrow & x = F(y+1) \\
x = F(b+1) \wedge y = b & \Rightarrow & x = F(y+1) \\
x = F(\underline{a+1}) \wedge y = a & \Rightarrow & y+1 = \underline{a+1} \\
x = F(\underline{b+1}) \wedge y = b & \Rightarrow & y+1 = \underline{b+1}
\end{array}
$$

Combining the $\Rightarrow$ Test

Combining satisfiability procedures

Nelson-Oppen combination method

Combining $\lceil \vee \rceil$ Operators

Given procedures:

$\lceil \vee \rceil_{L_1}(E_l, E_r)$

$\lceil \vee \rceil_{L_2}(E_l, E_r)$

We wish to compute $E_l \lceil \vee \rceil E_r$ in the logical product $L_1 * L_2$

Example.

$$\{z = a - 1, y = f(a)\} \lceil \vee \rceil \{z = b - 1, y = f(b)\} \quad = \quad \{y = f(1 + z)\}$$

Combining $\lceil \vee \rceil$ Operators

$\boxed{z = a - 1, y = f(a)}$ $\boxed{z = b - 1, y = f(b)}$

Purify+NOSat	$z = a - 1$	$y = f(a)$	$z = b - 1$	$y = f(b)$
LR-Exchange	$a = \langle a, b \rangle$	$a = \langle a, b \rangle$	$b = \langle a, b \rangle$	$b = \langle a, b \rangle$

Base $\lceil \vee \rceil$ $\boxed{\lceil \vee \rceil_{LA}}$ $\boxed{\lceil \vee \rceil_{UF}}$

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

Quant Elim $\boxed{\lceil \exists \rceil_{UF*LA}}$

Return $\boxed{y = f(1 + z)}$

The $\lceil \exists \rceil$ Operator

Required to compute transfer function for assignments

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

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

Examples:

- $\lceil \exists \rceil_{LA} a : (x < a \wedge a < y) = (x < y)$
- $\lceil \exists \rceil_{UF} a : (x = f(a) \wedge y = f(f(a))) = (y = f(x))$
- $\lceil \exists \rceil_{LA*UF} a, b, c : (a < b < y \wedge z = c + 1 \wedge a = ffb \wedge c = fb) = (f(z-1) < y)$

How to construct $\lceil \exists \rceil_{LA*UF}$ using $\lceil \exists \rceil_{LA}$ and $\lceil \exists \rceil_{UF}$?

Combining $\lceil\exists\rceil$ Operators

Problem	$\boxed{a < b < y, z = c+1, a = ffb, c = fb}$		$\boxed{\{a, b, c\}}$
Purify+NOSat	$\boxed{a < b < y, z = c+1}$		$\boxed{a = ffb, c = fb}$
QSat		$\rightarrow$	$c \mapsto z-1$
QSat	$a \mapsto fc$	$\leftarrow$	
Base $\lceil\exists\rceil$	$\boxed{\lceil\exists\rceil_{LA}}$		$\boxed{\lceil\exists\rceil_{UF}}$
	$\boxed{a < y, z = c+1}$		$\boxed{a = fc}$
Substitute	$c \mapsto z-1, a \mapsto fc$		
Return	$\boxed{f(z-1) < y}$		

Quantified Abstract Domain

Lifting base logical domains to quantified domains

```
array-init(A, n)
  for (i = 0; i < n; i++) {
        A[i] = 0
  }
  [ ∀k(0 ≤ k < n ⇒ A[k] = 0) ]
```

Array Initialization

`array-init`(A, n)

`for` $(i = 0$; $i < n$; `i++)` $\{$

$(i = 1 \;\wedge\; A[0] = 0) \;\vee\; (i = 2 \;\wedge\; A[0] = 0 \;\wedge\; A[1] = 0)$

$A[i] = 0$

$\}$

Let us write it out as a quantified fact.

Array Initialization

array-init(A, n)

for $(i = 0;\ i < n;$ i++) {

$(i = 1 \;\wedge\; \forall k(k = 0 \;\Rightarrow\; A[k] = 0)) \vee$

$(i = 2 \;\wedge\; \forall k(k = 0 \;\Rightarrow\; A[k] = 0) \;\wedge\; \forall k(k = 1 \;\Rightarrow\; A[k] = 0))$

$A[i] = 0$

}

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

$$i = 2 \wedge \forall k(\_\_\_\_ \Rightarrow A[k] = 0)$$

$\_\_\_\_$ should be $k = 0 \;\; \lfloor \vee \rfloor \;\; k = 1$:

$$0 \leq k \leq 1 \Rightarrow (k = 0 \vee k = 1)$$

Array Initialization

array-init(A, n)

for $(i = 0;\ i < n;\ $ i++) $\{$

$$i = 1 \wedge \forall k(k = 0 \;\Rightarrow\; A[k] = 0) \vee$$

$$i = 2 \wedge \forall k(0 \leq k < 2 \;\Rightarrow\; A[k] = 0)$$

$A[i] = 0$

$\}$

Now we need to $\lceil \vee \rceil$ of two quantified facts.

Array Initialization

$i = 1$ $\lceil\vee\rceil$ $i = 2$

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

$1 \le i \le 2$

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

Obviously, $\_\_\_\_$ should be $k = 0$ $\lfloor\wedge\rfloor$ $0 \le k < 2$.

$k = 0$ is no good.

Array Initialization

$i = 1$ $\quad\lceil\vee\rceil\quad$ $i = 2$

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

$$1 \leq i \leq 2$$

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

Actually, $\_\_\_\_$ should be

$$i = 1 \Rightarrow k = 0 \lfloor\wedge\rfloor\; i = 2 \Rightarrow 0 \leq k < 2$$

Let us see if the answer satisfies this.

$$0 \leq k < i \Rightarrow (i = 1 \Rightarrow k = 0 \wedge i = 2 \Rightarrow 0 \leq k < 2)$$

The Quantified Domain

$$E \wedge \bigwedge_i \forall U_i (F_i \Rightarrow e_i)$$

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

Function	Description
$E_1 \; \lceil\vee\rceil \; E_2$	join of E_1 and E_2
$E_1 \; \lceil\wedge\rceil \; E_2$	meet of E_1 and E_2
$\lceil\exists\rceil \; x.E$	eliminate x from E
$E_1 \; \lfloor\Rightarrow\rfloor \; E_2$	partial order test comparing E_1 and E_2
$(E_1 \; \lfloor\vee\rfloor \; E_2)/E$	under-approximate $E \Rightarrow (E_1 \vee E_2)$
$(E_1 \Rightarrow E_1') \; \lfloor\wedge\rfloor \; (E_2 \Rightarrow E_2')$	underapprox. $(E_1 \Rightarrow E_1') \wedge (E_2 \Rightarrow E_2')$
$\lfloor\forall\rfloor \; x.(E \Rightarrow E')$	underapproximate $\forall x(E \Rightarrow E')$

Logical Interpretation: Summary

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

Logical Interpretation	:	$\lceil\vee\rceil$, $\lceil\exists\rceil$, $\Rightarrow$
Logical Product	:	Combination Algorithms
Quantified Extension	:	$\lfloor\vee\rfloor$, $\lfloor\wedge\rfloor$, $\lfloor\forall\rfloor$, abduction

- The assertion checking problem for program classes:
  - Is related to $\mathbf{T}$-unification
  - Unification type determines complexity
  - Interprocedural analysis needs context unification

Summary

CDS	HS	Synthesis	Syst Bio.	S/W
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$(M \models \phi)?$	$(M \models \phi)?$	$(M? \models \phi)$	$(M \models \phi)?$	$(M \models \phi)?$
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$M \models \phi'$, $\phi' \Rightarrow \phi$	$M \Rightarrow M'$, $M' \Rightarrow \phi$	$\exists\forall\psi$	$\forall\psi$	$M \models \phi'$, $\phi' \Rightarrow \phi$
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$\exists\forall\psi$	$\exists\forall\psi,\ M' \models \phi$			Logical Interp.
$\searrow$	$\downarrow$	$\swarrow$	$\downarrow$	$\downarrow$
	$\boxed{\exists\forall \text{ solver}}$		$\boxed{\forall \text{ solver}}$	$\boxed{\text{Approx. ops}}$

Conclusion

SMT Solvers have revolutionalized solving of $\forall$ formulas

Possible directions of evolution:

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