Control from Computer Science

Oded Maler

CNRS-VERIMAG

Grenoble, France
Model-based System Design

Formal Model

Analysis

Design

Abstract Controller

Experiments

Thinking

World

Implementation

Controller

I

O
The Coffee Machine

![Diagram of the coffee machine]

<table>
<thead>
<tr>
<th>Port</th>
<th>From→To</th>
<th>Event types</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E → M₁</td>
<td>coin-in</td>
<td>a coin was inserted</td>
</tr>
<tr>
<td>2</td>
<td>E → M₁</td>
<td>cancel</td>
<td>cancel button pressed</td>
</tr>
<tr>
<td>3</td>
<td>M₁ → E</td>
<td>coin-out</td>
<td>release the coin</td>
</tr>
<tr>
<td>4</td>
<td>M₁ → M₂</td>
<td>ok</td>
<td>sufficient money inserted</td>
</tr>
<tr>
<td>5</td>
<td>M₁ → M₂</td>
<td>reset</td>
<td>money returned to user</td>
</tr>
<tr>
<td>6</td>
<td>M₂ → M₁</td>
<td>done</td>
<td>drink distribution ended</td>
</tr>
<tr>
<td>7</td>
<td>E → M₂</td>
<td>req-coffee</td>
<td>coffee button pressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>req-tea</td>
<td>tea button pressed</td>
</tr>
<tr>
<td>8</td>
<td>E → M₂</td>
<td>drink-ready</td>
<td>drink preparation ended</td>
</tr>
<tr>
<td>9</td>
<td>M₂ → E</td>
<td>st-coffee</td>
<td>start preparing coffee</td>
</tr>
<tr>
<td></td>
<td></td>
<td>st-tea</td>
<td>start preparing tea</td>
</tr>
</tbody>
</table>
The Two Sub-Machines

\[ M_1 \]

\[ M_2 \]
Normal behaviors:

0A coin-in 1B cancel coin-out 0A

0A coin-in 1B req-coffee st-coffee
1C drink-ready 0A
An Unexpected Behavior

0A coin-in 1B req-coffee st-coffee 1C cancel coin-out 0C drink-ready 0A
Fixing the Bug

\[ M_1 \]

\[ \text{coin-in/ ok} \quad \text{lock/} \]

\[ 0 \quad 1 \rightarrow 2 \]

\[ \text{cancel/coin-out, reset} \]

\[ \text{done/} \]

\[ M_2 \]

\[ \text{drink-ready/done} \]

\[ A \rightarrow B \rightarrow C \rightarrow D \]

\[ \text{ok/} \]

\[ \text{reset/} \]

\[ \text{req-coffee/st-coffee,lock} \]

\[ \text{req-tea/st-tea,lock} \]

\[ \text{drink-ready/done} \]
Fixing the Bug – the Global Model

drink-ready/

coin-in/

0A → 1B

cancel/coin-out  req-tea/st-tea

2D

req-coffee/st-coffee

2C

drink-ready/
The Moral of the Story

1) Many systems can be modeled as a composition of interacting automata (transition systems, discrete event systems).

2) Potential behaviors of the system correspond to paths in the global transition graph of the system.

3) These paths are labeled by input events. Each input sequence might generate a different behavior.

4) We want to make sure that a system responds correctly to all conceivable inputs.

5) For every individual input sequence we can simulate the reaction of the system. But we cannot do it exhaustively due to the huge number of input sequences.

6) Verification is a collection of automatic and semi-automatic methods to analyze all the paths in the graph.

7) This is hard for humans to do and even for computers.
Model I: Closed Systems

A transition system is $S = (X, \delta)$ where $X$ is finite and $\delta : X \rightarrow X$ is the transition function.

The state-space $X$ has no numerical meaning and no interesting structure.

$X^k$ is the set of all sequences of length $k$; $X^*$ the set of all sequences.

Behavior: The behavior of $S$ starting from an initial state $x_0 \in X$, is

$$\xi = \xi[0], \xi[1], \ldots \in X^*$$

such that $\xi[0] = x_0$ and for every $i$,

$$\xi[i + 1] = \delta(\xi[i])$$

Basic Reachability Problem: Given $x_0$ and a set $P \subseteq X$, does the behavior of $S$ starting at $x_0$ reach $P$?
Solution by Forward Simulation

\[ \xi[0] := x_0 \]
\[ F^0 := \{ x_0 \} \]

**repeat**

\[ \xi[k + 1] := \delta(\xi[k]) \]
\[ F^{k+1} := F^k \cup \{ \xi[i + 1] \} \]

**until** \( F^{k+1} = F^k \)

\[ F_* := F^k \]

\[ \{ x_1 \}, \{ x_1, x_2 \}, \{ x_1, x_2, x_3 \}, \{ x_1, x_2, x_3, x_5 \} \]

How to do it for continuous system defined by \( \dot{x} = f(x) \)?
Model II: Systems with One Input

A one-input transition system is $S = (X, V, \delta)$ where $X$ and $V$ are finite $\delta : X \times V \rightarrow X$ is the transition function.

Behavior Induced by Input: Given an input sequence $\psi \in V^*$, the behavior of $S$ starting from $x_0 \in X$ in the presence of $\psi$ is a sequence

$$\xi(\psi) = \xi[0], \xi[1], \ldots \in X^*$$

such that

$$\xi[i + 1] = \delta(\xi[i], \psi[i]).$$
The reachability problem: Is there some input sequence $\psi \in V^*$ such that $\xi(\psi)$ reaches $P$?

For every given $\psi$ we can use the previous algorithm, simulate and obtain $F_*(\psi)$.

For an automaton with $n$ states all states are reachable by sequences of length $< n$.

$$F_* = \bigcup_{\xi \in V^n} F_*(\psi)$$
A More Efficient Way

Many different inputs lead to the same state.

Immediate successors: \( \delta(x) = \{x' : \exists u \ \delta(x, u) = x'\} \)

Successors of a set \( F \): \( \delta(F) = \{\delta(x) : x \in F\} \)

Forward reachability algorithm (breadth-first):

\[
F^0 := \{x_0\}
\]

\[\text{repeat}\]

\[F^{k+1} := F^k \cup \delta(F^k)\]

\[\text{until } F^{k+1} = F^k\]

\[F_* := F^k\]

Complexity: only \( O(n \cdot \log n \cdot |V|) \)
Variations: Depth-First and Backwards

Depth-first:

Backwards: find all states from which there is an input leading to $P$.

Immediate predecessors:
\[ \delta^{-1}(x) = \{ x' : \exists u \; \delta(x', u) = x \} \]

\[
\begin{align*}
F^0 & := P \\
\text{repeat} & \quad F^{k+1} := F^k \cup \delta^{-1}(F^k) \\
\text{until} & \quad F^{k+1} = F^k \\
F_* & := F^k
\end{align*}
\]
Admissible Inputs

So far we have assumed that the external environment can generate all sequences in $V^*$. Sometimes we have a more restricted environment, e.g. it will never produce $v_1 v_1$. We can build an automaton which models the environment and compose it with the model of the system.
There are algorithms that take a description of any open system and verify whether any of the admissible inputs drives the system into a set $P$. Such algorithms always terminate after a finite number of steps.

This is essentially what verification is all about.

The result is general: it is valid for every discrete finite-state system. Of course, finite systems can be very large and special tricks are needed to verify them.

The analogue for continuous systems: do the same for a system defined by $\dot{x} = f(x, u)$. 
Systems with two Inputs

A two-input transition system is $S = (X, U, V, \delta)$ where $X$, $U$ and $V$ are finite sets and $\delta : X \times U \times V \to X$ is the transition function.

The behavior in the presence of two inputs, $\eta \in U^*$ and $\psi \in V^*$: a sequence $\xi(\eta, \psi)$ s.t.

$$\xi[i + 1] = \delta(\xi[i], \eta[i], \psi[i])$$
Games and Strategies

Interpretation of inputs:

$U$: we, the good guys, the controller.

$V$: they, the bad guys, disturbances.

An antagonist game situation. Our goal is to choose each time an element of $U$ such that the behaviors induces by all possible disturbances are good.

**Strategy**: a function $c : X^* \rightarrow U$

**State strategy**: a function $c : X \rightarrow U$.

Each strategy $c$ converts a type III system into a type II system $S_c = (X, V, \delta_c)$ s.t. $\delta_c(x, v) = \delta(x, c(x), v)$.

**Synthesis for Reachability**: Let $S = (X, U, V, \delta)$ let $P \subseteq X$ be a set of “bad” states. The controller synthesis problem is: find a strategy $c$ such that all the behaviors of the derived system $S_c = (X, V, \delta_c)$ never reach $P$. 
Finding Winning States and Strategies

Controllable Predecessors: For $S = (X, U, V, \delta)$ and $F \subseteq X$, the set of controllable predecessors of $F$ is

$$\pi(F) = \{ x : \exists u \in U \ \forall v \in V \ \delta(x, u, v) \in F \}$$

The states from which the controller, by properly selecting $u$, can force the system into $P$ in the next step.

The following backward algorithm finds the set $F_*$ of “winning states” from which $P$ can be avoided forever.

\[
F^0 := X - P \\
\text{repeat} \\
\quad F^{k+1} := F^k \cap \pi(F^k) \\
\text{until} \ F^{k+1} = F^k \\
F_* := F^k
\]

Remark: this is similar to the Ramadge-Wonham theory of discrete event control.
Synthesis Example

We want to avoid $x_5$.

$F^0 = \{x_1, x_2, x_3, x_4\}$

$F^1 = \{x_1, x_2, x_3\} = F_*$

The resulting “closed-loop” system always remains in $\{x_1, x_2, x_3\}$. 
Discrete Infinite-State Systems

Computer program are syntactic representation of dynamical systems with infinite state-space.

\[
\text{repeat} \quad y := y + 1 \quad \text{until } y = 4
\]

State space: \( \{x_1, x_2\} \times \mathbb{Z} \)

Forward reachability algorithm will terminate if started from \((x_1, 2)\) but not from \((x_1, 5)\).

The reachability problem is unsolvable: there is no general algorithm that solves every instance of it.

“Deductive” approach: prove properties “analytically”.

“Symbolic” approach: reachability using formulae to represent sets of states, e.g. \( x = x_1 \land y \geq 5 \).
Continuous (and Hybrid) Systems

Why? ...

Problems: state space $\mathbb{R}^n$, infinite even when bounded, time domain $\mathbb{R}$. Mathematical $\mathbb{R}$ vs. numerical $\mathbb{R}$ in the computer.

Reachability for $\dot{x} = f(x)$: When we have a closed-form solution, e.g. for $\dot{x} = Ax$, the reachable set can be written as $F_* = \{x_0e^{At} : t \geq 0\}$ but how to test whether $F_* \cap P = \emptyset$?

Forward simulation: discretize time and replace the system with $\xi'[(n + 1)\Delta] = \xi'[n\Delta] + h(\xi'[n\Delta], \Delta)$.

![Diagram of forward simulation](image)

This is not the “real” thing and it is not guaranteed to converge but that’s life.
Continuous Systems with Input

Systems of the form $\dot{x} = f(x, v)$. Admissible inputs are signals of the form $\psi : T \rightarrow V$.

Problem: show that no admissible input drives the system into a set $P$.

For every $\psi$ we can simulate and “compute” $F_*(\psi)$, but there is no finite subset of inputs that covers all reachable states.

The set of all inputs is a doubly-dense tree, both vertically (time) and horizontally ($V$).
Incremental Reachability Computation

Breadth-first computation of reachable states.

\[ x \xrightarrow{t} x' \] denotes the existence of an input signal \( \psi : [0, t] \rightarrow V \) that drives the system from \( x \) to \( x' \) in \( t \) time.

Let \( F \) be a subset of \( X \) and let \( I \) be a time interval. The \( I \)-successors of \( F \) are all the states that can be reached from \( F \) within that time interval, i.e.

\[ \delta_I(F') = \{ x' : \exists x \in F \ \exists t \in I \ x \xrightarrow{t} x' \}. \]

Semigroup property:

\[ \delta_{[0,r_2]}(\delta_{[0,r_1]}(F)) = \delta_{[0,r_1+r_2]}(F). \]

\[
\begin{align*}
F^0 & := \{ x_0 \} \\
\text{repeat} & \\
F^{k+1} & := F^k \cup \delta_{[0,r]}(F^k) \\
\text{until} & \\
F^{k+1} & = F^k \\
F_* & := F^k
\end{align*}
\]
But $\delta_{[0,r]}(F)$ cannot be computed exactly. We can over-approximate it by $\delta'$ such that for every $F$

$$\delta_{[0,r]}(F') \subseteq \delta'_{[0,r]}(F')$$

and $\delta'_{[0,r]}(F')$ belongs to some effective sub-class of $\mathbb{R}^n$, e.g. polyhedra.

The result of the algorithm is a set $F'$ s.t. $F_* \subseteq F'$ and hence $F' \cap P = \emptyset$ implies the correctness of the system.
Conclusion

We have developed a system called \( \frac{d}{dt} \) which accepts as input a description of a continuous or a hybrid system and computes automatically an over-approximation of the reachable states.

More about it in the special session on reachability.

Challenge: use more knowledge on the system dynamics in order to increase the performance and treat systems with higher dimensions.

Challenge: develop algorithms for automatic synthesis of strategies for systems with two inputs, \( \dot{x} = f(x, u, v) \).