Verification and Performance in Networked Embedded Systems

Eric Klavins (University of Washington)
J.M. McNew (Toyota Research)
David Thorsley (University of Washington)
Steve Safarik (University of Washington)
Setting

Heterogeneity
Local communication
Motion
Changing environment
Stochasticity
Complex Tasks / Subtasks

Metabolobotics

Programmable Parts
Challenges

- Abstraction
- Programming and design
- Specification
- Verification
- Performance

Send(strTestMessage.GetBuffer(),strTestMessage.GetLength()+1)
Overview of UW Progress

1. Testbeds: PPT, EFRI
2. Suite of examples specified and verified
3. Lyapunov function co-design
4. Separation of continuous and discrete design
5. Separation of verification and performance tuning
6. Pseudometrics to compare, diagnose and optimize stochastic systems
7. Diagnosis / observation of stochastic process with low-dimensional, stochastic output
Abstraction: Concurrency

A concurrent program: Each guards and rule can be evaluated by small groups of agents without global knowledge.

Non-determinism: Which rule? Where to apply it? When to apply it?
Abstraction: How is the rule applied? – That’s an implementation issue!
In general, each node carries a data structure, and rules operate on pairs of data structures.

$$\Phi = \begin{cases} 
  a \ a \rightarrow b - b, \\
  a \ b \rightarrow b - c, \\
  b \ b \rightarrow c - c 
\end{cases}$$
Graph Grammar Rules More Generally

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Unbound node indices</th>
<th>Existential variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>Precondition on data structures maintained by i, j and k</td>
<td>Precondition local network</td>
</tr>
<tr>
<td>Postcondition</td>
<td>Postcondition on network (there may also be postconditions on data)</td>
<td></td>
</tr>
</tbody>
</table>

### Rule $r_3$ (Merge Branches)

**Vertices:** $i$, $j$, $k$

**Variables:** $b_i$

**Precondition:**

\[
i.\text{mode} = t \land j.\text{mode} = a \land k.\text{mode} = a \land ij.\text{order}(j) < b_i \land ik.\text{order}(k) < b_i
\]

**Effect:**

\[
\begin{array}{c}
E = \\
\begin{tikzpicture}
  \node (i) at (0,0) {$i$};
  \node (j) at (1,0) {$j$};
  \node (k) at (2,0) {$k$};
  \draw (i) -- (j);
  \draw (j) -- (k);
\end{tikzpicture}
\end{array}
\]

\[
E := \\
\begin{tikzpicture}
  \node (i) at (0,0) {$i$};
  \node (j) at (1,0) {$j$};
  \node (k) at (2,0) {$k$};
  \draw (i) -- (j);
  \draw (j) -- (k);
\end{tikzpicture}
\]
Example: Embedded Graph Grammars

\[ u_i(\gamma) = \begin{cases} 
1 & \text{if } i.mode = \text{east} \\
-1 & \text{if } i.mode = \text{west} \\
\sum_{j \in N(i)} x_j - x_i & \text{if } i.mode = \text{follow}
\end{cases} \]

\( \Phi: \)

<table>
<thead>
<tr>
<th>Rule ( r_1 ) (Join)</th>
<th>Rule ( r_2 ) (Close Cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition:</strong></td>
<td><strong>Precondition:</strong></td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>(e \quad w)</td>
<td>(</td>
</tr>
<tr>
<td><strong>Effect:</strong></td>
<td><strong>Effect:</strong></td>
</tr>
<tr>
<td>(w \leftarrow f)</td>
<td>(w \leftarrow f)</td>
</tr>
</tbody>
</table>

Example: Metabolobotics

NSF EFRI: Build it and program it!
MURI: Verification tools should apply!

Action name: $\text{manipulate}(i, j) :$
Least Specific Guard: $(r_i = \text{CARRIER} \lor r_i = \text{BONDER} \lor r_i = \text{BREAKER}) \land (x_i, y_i) = (x_j, y_j)$
Update: $m'_i = m_i \cup j$

Other actions: move, weld, break, label, communicate
Example:

Metabolobotics

Animation
Sources of Nondeterminism (Due to Abstraction)

- Rule application order
- Rule application time
- Rule application place

Metabolobotics example produces many different structures (in many different ways) – but they are (a) connected and (b) contain all parts.

EGG example produces different results depending on timing and order.

**Specification:** Make statements about desired sets of possible trajectories.

**Verification:** Check whether our programs always produce trajectories contained within those sets.
Verification: States, Updates and Trajectories

\[ \sigma = \langle s_0, s_1, \ldots \rangle \quad (\text{or } \sigma : \mathbb{R}^+ \cup \{0\} \rightarrow \mathcal{S}) \]

\[ s[f] \iff s \text{ is a state contained in the set of states specified by the formula } f. \]

\[ s[a]t \iff (s,t) \text{ is a pair of states contained in the set of pairs specified by the action } a. \]

\[ \sigma[F] \iff \sigma \text{ is a trajectory contained in the set of trajectories specified by the formula } F. \]
Lamport’s Temporal Logic of Actions
(Lamport, ACM Toplas 16, 3 (May 1994) 872-923)

Semantics

\[ s[f] \triangleq f(\forall 'v' : s[v]/v) \]
\[ \sigma[F \land G] \triangleq \sigma[F] \land \sigma[G] \]
\[ s[A]t \triangleq A(\forall 'v' : s[v]/v, t[v]/v') \]
\[ \sigma[\neg F] \triangleq \neg \sigma[F] \]
\[ \models A \triangleq \forall s, t \in St : s[A]t \]
\[ \models F \triangleq \forall \sigma \in St^\infty : \sigma[F] \]
\[ s[Enabled A] \triangleq \exists t \in St : s[A]t \]
\[ \langle s_0, s_1, \ldots \rangle[F'] \triangleq \forall n \in Nat : \langle s_n, s_{n+1}, \ldots \rangle[F'] \]
\[ \langle s_0, s_1, \ldots \rangle[A] \triangleq s_0[A]s_1 \]

Additional notation

\[ p' \triangleq p(\forall 'v' : v'/v) \]
\[ \Diamond F \triangleq \neg \Box \neg F \]
\[ [A]_f \triangleq A \lor (f' = f) \]
\[ F \leadsto G \triangleq \Box (F \Rightarrow \Diamond G) \]
\[ (A)_f \triangleq A \land (f' \neq f) \]
\[ WF_f(A) \triangleq \Box \Diamond (A)_f \lor \Box \Diamond \neg \text{Enabled } (A)_f \]
\[ Unchanged \ f \triangleq f' = f \]
\[ SF_f(A) \triangleq \Box \Diamond (A)_f \lor \Diamond \Box \neg \text{Enabled } (A)_f \]

Guard: Rule pairs, programs, refinements, implementations and specification can all be written in TLA.
**Verification: Progress via Lyapunov Co-Design**

Lamport’s TLA inference rule for progress uses a *discrete Lyapunov function* $H_C$.

\[
F \land (c \in S) \Rightarrow (H_C \leadsto (G \lor \exists d \in S : (c > d) \land H_d))
\]

\[
F \Rightarrow ((\exists c \in S : H_C) \leadsto G)
\]

**McNew’s Lyapunov Codesign Method:**
1) Build individual behaviors, each decreasing it’s own LF $U_i$
2) Compose behaviors, noting priority
3) Verify that $U_i$ increasing $\rightarrow$ $U_j$ decreases for some $j<i$
4) $U=(U_1,...,U_k)$ is a Lyapunov function under the lexicographic ordering
Verification: Progress via LLFs

\[ \Phi = \begin{cases} 
(w, x) \rightarrow (w - 1, x) \quad (y, z + x) \\
(0, x) \quad (y, z) \rightarrow (0, x) - (y, z) \\
(0, x) - (y, z) \rightarrow (0, x) - (\max(y - 1, 0), \max(z - 1, 0)) 
\end{cases} \]

Let \( U = (U_1, U_2, U_3) \) where

1. \( U_1 = \sum_{i \in V} i.a \)
2. \( U_2 = (|V|^2 - |V|)/2 - |E| \)
3. \( U_3 = \sum_{i \in V} i.b \)

This grammar results in a fully connected graph all of whose vertices are labeled (0,0).
Tree-tree Reconfiguration

Assumptions:
• Local communication.
• Unknown initial state.

Goals:
• Convergence to an isomorphism of the desired tree formation (progress).
• All intermediate states are tree formations (safety).
• Correct behavior from all initial conditions.

Verification: Separation of Discrete and Continuous

1. Write a program for only the discrete part of the problem (e.g. network reconfiguration).

2a) Lift the discrete solution to the hybrid (discrete/continuous state).

2b) Determine a specification, C, that if met by u and ψ guarantees safety and progress in discrete solution.

3) Design u and ψ to meet the specification C.
Tree-tree Reconfiguration (Discrete Part)

**Rule r₁**
Vertices: i, j
Precondition:
- \( i \text{.mode} = t \land j \text{.mode} = a \) 
- Denote \( i \text{.role} \) by \( v \), \( \exists v \in N_i (v) \) such that 
  \( (w \text{.mode} = a \land w \text{.order}(w) = i \text{.order}(j)) \)
- \( G[\{i, j\}] = i - j \)
Effect:
- \( w \text{.mode} := t \), \( j \text{.tree} := i \text{.tree} \)
- \( j \text{.mode} := t \), \( j \text{.role} := w \)
- \( ij \text{.offset} := vw \text{.offset} \), \( ij \text{.head} := j \)

**Rule r₂**
Vertices: i, j, k
Precondition:
- \( i \text{.mode} = t \land j \text{.mode} = a \land k \text{.mode} = a \)
- \( ij \text{.order}(j) > b_i \land \neg ij \text{.order}(j) > b_i \land ij \text{.order}(j) < b_i \)
- \( jk \text{.order}(k) < b_i \land \neg jk \text{.order}(k) < b_i \land jk \text{.order}(k) > b_i \)
- \( G[\{i, j, k\}] = i - j - k \)
Effect:
- \( G[\{i, j, k\}] := i - j - k \)

**Rule r₃**
Vertices: i, j, k
Precondition:
- \( i \text{.mode} = t \land j \text{.mode} = a \land k \text{.mode} = a \)
- \( ij \text{.order}(j) < b_i \land \neg ij \text{.order}(j) < b_i \land ij \text{.order}(j) > b_i \)
- \( G[\{i, j, k\}] = i - j - k \)
Effect:
- \( G[\{i, j, k\}] := i - j - k \)

Local network reconfigurations are guaranteed to preserve tree

---

Verification: Progress

$U_1$—The number of vertices yet to be matched to the target graph.

$U_2$—The number of t-a edges to which $r_2$ might apply.

$U_3$—The average distance from $b_i$ on t-a edges to which $r_2$ might apply.

$U_4$—The summed distance from $b_i$ on t-a edges to which $r_3$ applies.

$U_5$—The number of subgraphs to which $r_4$ or $r_5$ apply.

$U_6$—The number of subgraphs to which $r_5$ applies.

Tree-tree Reconfiguration: Continuous Part

Intermediate specification on \( u \) and \( \psi \)

Safety: \( \square (ij \in E \rightarrow ij \in E_\psi) \)

Progress: \( \dot{x} = u \rightsquigarrow (\dot{x} \neq u \lor \exists r. x \in \text{cont\_guard}(r)) \)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Merge Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>( i, j, k )</td>
</tr>
<tr>
<td>Variables</td>
<td>( b_i )</td>
</tr>
<tr>
<td>Precondition:</td>
<td>( i.\text{mode} = m \land j.\text{mode} = a \land k.\text{mode} = a \land</td>
</tr>
<tr>
<td>( E = )</td>
<td>( i \rightarrow j \leftarrow k )</td>
</tr>
<tr>
<td>Effect:</td>
<td>( E := )</td>
</tr>
<tr>
<td>( i.\text{mode} := a )</td>
<td></td>
</tr>
</tbody>
</table>

\[
\dot{x}_i = -\sum_{j \in N_\psi(i)} \frac{2(\Delta - ||\alpha|| - ||x_i - x_j - \alpha||)}{(\Delta - ||\alpha|| - ||x_i - x_j - \alpha||)^2} (x_i - x_j - \alpha)
\]

Simple implementation (can prove it satisfies C)

Continuous predicate updates to rules.
Simulation
The Separation of Verification and Performance Tuning

**Code for a non-deterministic walk**

```plaintext
mode = search : move_lt
mode = search : move_rt
mode = search : move_up
mode = search : move_dn
```

**TLA spec with fairness on movement**

\[
\Pi = \square(x_{\min} \leq x \leq x_{\max} \land y_{\min} \leq y \leq y_{\max})
\]

\[
= \bigwedge_{x_{\min} \leq w \leq x_{\max}} SF(x = w \land move_{\text{Lt}})
\]

\[
= \bigwedge_{x_{\min} \leq w \leq x_{\max}} SF(x = w \land move_{\text{Rt}})
\]

\[
= \bigwedge_{y_{\min} \leq z \leq y_{\max}} SF(y = z \land move_{\text{Dn}})
\]

\[
= \bigwedge_{y_{\min} \leq w \leq y_{\max}} SF(y = z \land move_{\text{Up}})
\]

\[
\vdash \Diamond (x = x^{*} \land y = y^{*})
\]

**Inference rule**

\[
\text{true} \\
\hline
mode = \text{search} \leadsto (\text{mode} \neq \text{search} \lor (x = x^{*} \land y = y^{*}))
\]

Safarik, Napp and Klavins, In progress.
The Separation of Verification and Performance Tuning

**Code for a feedback controlled random walk**

\[
\begin{align*}
\text{mode = search : move_lt} & \quad u_f(x,y,t)dt \\
\text{mode = search : move_rt} & \quad u_r(x,y,t)dt \\
\text{mode = search : move_up} & \quad u_u(x,y,t)dt \\
\text{mode = search : move_dn} & \quad u_d(x,y,t)dt \\
\end{align*}
\]

Probability that the rule will fire in the next \( dt \) seconds given the state is \((x,y)\)

**Thm:** Set of unfair trajectories of the non-deterministic system has measure zero in any probabilistic implementation in which \( u_*(x,y,t) > 0 \) for all \( x, y \) and \( t \).

Thus, even an (almost) deterministic search algorithm is guaranteed to be correct.
An Example: Tuning Metabolism While Preserving Correctness

Reduced model ($k_i$ depends on allocation of welders, breakers, carriers).

\[
\begin{align*}
B & \xrightarrow{k_1} F \\
F & \xrightarrow{k_2} P \\
P & \xrightarrow{k_3} D
\end{align*}
\]

Safarik, Napp and Klavins, In progress.
Summary

• Examples/Testbeds
  – Next: EFRI testbed available / simulator enhanced

• Lyapunov Co-Design
  – Next: Lyapunov co-design in stochastic systems (e.g. Parrilo)

• Separation of continuous and discrete design
  – Next: Formalize in TLA

• Separation of correctness and performance
  – Next: Formalize in TLA
  – Next: Feedback controlled performance that does not affect correctness

• Stochastic diagnosis, model reduction
  – David’s talk