Computational Bottlenecks in Symbolic Control Synthesis

Example: Vehicle Dynamics

3-dimensional state \((x_1, x_2, x_3) \in X\) and 2-dimensional input \((u_1, u_2) \in U\):

\[
\begin{align*}
\dot{x}_1 &= u_1 \cos(\alpha + u_2) / \cos(\alpha) \\
\dot{x}_2 &= u_1 \sin(\alpha + u_2) / \sin(\alpha) \\
\dot{x}_3 &= u_1 \tan(u_2)
\end{align*}
\]

where \(\alpha = \arctan(\tan(u_2)/2)\).

Sparsity-Aware Abstraction

Each next state \(x_1^+, x_2^+, x_3^+\) in discrete time vehicle dynamics only depends on a subset of \((x_1, x_2, x_3, u_1, u_2)\).

\[
\Sigma(\mathbf{x}, \mathbf{x}^+) = \bigwedge_{i=1}^{3} \left( \Sigma_i(x_1, x_2, x_3, u_1, u_2, x_1^+, x_2^+, x_3^+) \right)
\]

Discretization:

- 93 states: \(51 \times 51 \times 36\) grid
- 49 inputs: \(7 \times 7\) grid

Preparations for Efficient Parallel Execution

- \(X\) and \(U\): bounded, quantized and then flattened.
- Multi-precision flat spaces.
- \((\hat{x}, \hat{u})\) is an element of the 2D flat space \(X \times U\).
- 2D task scheduling problem.
- Devices are tuned with sample slices of \(X \times U\).
- Each \((\hat{x}, \hat{u})\) is assigned to a processing element (PE).

Sparsity-Aware Abstraction computes and combines abstractions of lower dimensional components \(\Sigma_1, \Sigma_2, \Sigma_3\) and eliminates redundant computations.

- **Equality**: Yields same abstraction as the regular algorithm.
- **Efficiency**: Linear with respect to state dimension, exponential with respect to sparsity parameter.
- **Generality**: Only assumption is Cartesian product state space and component-wise dynamics.

Decomposed Controller Predecessor

Controllable predecessor operator is a key subroutine in formal control synthesis:

\[
\text{CPRE}(Z) = \exists u_i. (\exists x_i. \Sigma(x_i, u_i, x_i^+) \wedge \forall x^+. (\Sigma(x_i, u_i, x^+) \Rightarrow Z(x^+)))
\]

**Goal**: Create a controllable predecessor without constructing monolithic system \(\Sigma(\mathbf{x}, \mathbf{u}, \mathbf{x}^+)\). Substituting decomposed representation of \(\Sigma\) into red term yields

\[
\begin{align*}
\forall x^+. (\Sigma(x_i, u_i, x^+) \Rightarrow Z(x^+)) \\
= \forall x^+. (\Sigma_1 \wedge \Sigma_2 \wedge \Sigma_3 \Rightarrow Z(x^+)) \\
= \forall x^+_1. \forall x^+_2. \forall x^+_3. (\neg \Sigma_1 \vee \neg \Sigma_2 \vee \Sigma_3 \vee Z(x^+))
\end{align*}
\]

**Obstacle**: The universal quantifier \(\forall x^+\) doesn’t distribute over disjunctions.

**Solution**: Iteratively eliminate \(x_1^+, x_2^+, x_3^+\) variables over smaller formulas.

Equation (1): \(\forall x^+_1. \forall x^+_2. (\neg \Sigma_1 \vee \Sigma_2 \vee x^+_3. (\neg \Sigma_3 \vee Z))\)

Equation (2): \(\forall x^+_1. (\neg \Sigma_1 \vee \Sigma_2 \vee \forall x^+_3. (\neg \Sigma_3 \vee Z))\)

Equation (3): \(\forall x^+_1. (\neg \Sigma_1 \vee \forall x^+_2. (\neg \Sigma_2 \vee \forall x^+_3. (\neg \Sigma_3 \vee Z))\)

Visualization of equations (1), (2), (3):

\[
\begin{align*}
\Sigma_1 &\quad \Sigma_2 &\quad \Sigma_3 \\
x_1 &\quad x_2 &\quad x_3 \\
\mathbf{Z} &\quad \mathbf{V} &\quad \mathbf{W}
\end{align*}
\]

\[
\begin{align*}
\text{Equation (1)} &\quad \text{Equation (2)} &\quad \text{Equation (3)}
\end{align*}
\]

Initial Results and Future Work

- **OARS** in a lock-free fast-to-query data structure.
- Combining sparsity and parallel implementations.
- Testing on FPGAs and the Cloud.
- Python wrappers or domain specific language.

Parallel Construction of Symbolic Controllers

- Currently, a Fixed-point (FP) implementation is considered.
- Efficient on-the-fly memory-less discretization of OARS is used.
- Distributed bit-based storage of results reduces controller’s size.
- Parallel convergence check is applied after some FP iterations.

Parallel Benchmarks

<table>
<thead>
<tr>
<th>Ex. / Trans.</th>
<th>SCOTs0.2</th>
<th>SCOTs v0.2</th>
<th>SCOTs0.2</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCDC / 1M</td>
<td>30</td>
<td>2</td>
<td>99</td>
<td>8.3</td>
</tr>
<tr>
<td>Vehicle / 4M</td>
<td>739</td>
<td>203</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>Unicycle / 10SM</td>
<td>5989</td>
<td>2797</td>
<td>8.3</td>
<td></td>
</tr>
</tbody>
</table>

Table: Results with NVIDIA P5000 GPU. Time in sec and includes abstraction and synthesis. SCOTs0.2 and SCOTs v0.2 use FP and run on Intel Xeon E5-2630.