Automatic Synthesis of Control Software from System Level Formal Specifications

Vadim Alimguzhin  Federico Mari  Igor Melatti
Ivano Salvo  Enrico Tronci

Model Checking Group http://mclab.di.uniroma1.it/
Computer Science Department
Sapienza University of Rome

Oct 25th, 2013
Outline

1 Motivations

2 Our Method

3 Experimental Results

4 Conclusion and Theses
The Problem

How to write correct software?

• Software is usually written by humans, thus it always contains errors
• In fact, after a program has been written (and sometimes while it is written) it is always tested to check for errors
• This is an unavoidable step in any software engineering design model
The Problem

How to write correct software?

- Software is usually written by humans, thus it always contains errors
- In fact, after a program has been written (and sometimes while it is written) it is always tested to check for errors
- This is an unavoidable step in any software engineering design model
The Problem

How to write correct software?

- Software is usually written by humans, thus it always contains errors.
- In fact, after a program has been written (and sometimes while it is written) it is always tested to check for errors.
- This is an unavoidable step in any software engineering design model.
The Problem
How to write correct software?

- Software is usually written by humans, thus it always contains errors
- In fact, after a program has been written (and sometimes while it is written) it is always tested to check for errors
- This is an unavoidable step in any software engineering design model
An approximate answer
BUG HUNTING: Testing + Simulation

Input sequence (stimulus)
... u(3) u(2) u(1) u(0)

System (Model)
Compute output by Simulation or by running the actual system when possible

Output sequence
y(0) y(1) y(2) y(3) ...

Define initial state + parameters

Observer checks that output sequence ok

Automatic Synthesis of Control Software from System Level Formal Specifications
I. Melatti, Sapienza University
The Problem

Testing/Simulation: Drawbacks

• However, testing and simulation have both a main drawback: they may only reveal error presence, and not prove error absence

• For a word processor, it is not a so great problem: if an error is discovered after the testing/simulation was completed (or even after deploying), one may release a new version of the software

• What about safety critical or mission critical software?
The Problem
When you cannot fail

Spanair Flight JK 5022, 20 August 2008
Two (Intertwined) Solutions

1. Software may be *automatically verified* to be *correct* (absence of errors)

2. Software may be *automatically generated* from high-level specifications, so that it is *correct by construction*

Methods used to achieve such results have many common points
**Verification**

**Sys**
(VHDL, Verilog, C, C++, Java, MathLab, Simulink, ...)

**BAD**
(CTL, CTL*, LTL, ...)
Synthesis

Specification \rightarrow Synthesizer \rightarrow Generated Software

\textbf{Sol}
\textbf{NoSol}
no solution exists
Synthesis

Specification \rightarrow \text{Synthesizer} \rightarrow \text{Generated Software}

- \text{SOL}
- \text{UNKNOWN}
- \text{NoSol}

no solution exists
Embedded Systems

Our target

Wikipedia: An *embedded system* is a computer system designed to do a few dedicated functions with real-time computing constraints.
Motivations

Our Method

Experimental Results

Conclusion and Theses

Embedded Systems
Software Based Control Systems

1. Every \( T \) seconds (sampling time) do
2. Read AD conversion \( \hat{x} \) of plant sensor outputs \( x \)
3. If (\( \hat{x} \) is not in the Controllable Region)
4. Then // Exception (Fault Detected):
5. Start Fault Isolation and Recovery (FDIR)
6. Else // Nominal case:
7. Compute (Control Law) command \( \hat{u} \) from \( \hat{x} \)
8. Send DA conversion \( u \) of \( \hat{u} \) to plant actuators

Automatic Synthesis of Control Software from System Level Formal Specifications

I. Melatti, Sapienza University
Current methodology

Writing Control Law and Controllable Region

- Plant Model
- Closed Loop System Specifications

Control Engineering

Control Law, Sampling Time T, Quantization Q

Verify Control Law by Simulation (e.g., Simulink)

PASS

Software Engineering

Software Size <= RAM, Quantization OK?

Control Software

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL

FAIL
Current methodology
Many Drawbacks

- Must check many different properties:
  - Is Control Law correct (wrt closed loop specifications)?
  - Is the Controllable region big enough?
  - Are safety properties satisfied?
  - All checked by simulation
  - Is the whole Control Software correct (often referred to as System Level Correctness)?
  - Checked by Hardware In the Loop Simulation (a simulator is used for the system)
Current methodology

Many Drawbacks

- How to effectively write the Control Software code?
  - Hard real-time requirement: Control Software WCET < Sampling Time ($T$).
  - Schedulability analysis requires knowledge of Control Software WCET.
  - But WCET is known very late (after software implementation) in the design cycle.

- Design space exploration is difficult
  - there may exist many control laws meeting the given closed loop specs
  - only one control law is provided to the software designer
  - This limits a priori the software design space (e.g., to save on RAM, CPU, Energy)
Outline

1. Motivations
2. Our Method
3. Experimental Results
4. Conclusion and Theses

Automatic Synthesis of Control Software from System Level Formal Specifications
I. Melatti, Sapienza University
QKS: Controller Synthesis for DTLHSs

CAV 2010

**Input** \( (H, I, G, A/D + D/A) \)
- plant modeled by DTLHS \( H \)
- desired controllable region \( I \) (linear constraints)
- goal region \( G \) (linear constraints)
- conversion A/D and D/A

**Output** Controller \( K \)

1. function `Controllable.Region()`
2. function `Control.Law()`

With
- **robustness** w.r.t. parameters variations
- **guaranteed Worst Case Execution Time** (\( WCET = Time_{IFTE} \times State\_bits \times Actuators\_bits \))
CAV 2010 Solution
Algorithm and Tool QKS

Control Synthesis Problem \((\mathcal{H}, I, G, AD)\)

QKS

\(K +\) controllable region \(D\)

\(\text{SOL}\)

\(\text{UNKNOWN}\)

\(\text{I} \quad \text{D}\)

\(\text{NoSol}\)

no solution exists

\text{UNKNOWN} stems from undecidability of the problem
The Plant Model

Discrete Time Linear Hybrid Systems (DTLHS)

Roughly, any hybrid system which dynamics can be described using linear differential equation can be modelled as a DTLHS (using a suitable sampling time $T$).

$$N(X, U, Y, X') = (i_L' = (1 + T_{a_1,1})i_L + T_{a_1,2}v_O + T_{a_1,3}v_D) \land$$
$$\land (v_O' = T_{a_2,1}i_L + (1 + T_{a_2,2})v_O + T_{a_2,3}v_D) \land$$
$$\land (v_u - v_D \leq (1 + \rho)V_i) \land (v_u - v_D \leq (1 - \rho)V_i) \land$$
$$\land (i_D = i_L - i_u) \land (q \to v_D = 0) \land (\bar{q} \to i_D \geq 0) \land$$
$$\land (\bar{q} \to v_D \leq 0) \land (\bar{q} \to v_D = R_{off}i_D) \land$$
$$\land (\bar{u} \to v_u = 0) \land (\bar{u} \to v_u = R_{off}i_u)$$

$\rightarrow$ stands for if-then

- $\rho = 0.25$ (25\%) $V_i$
- $tolerance$
- $X = [i_L, v_O]$
- $U = [u]$
- $Y = [v_D, v_u, i_D, i_u, q]$
QKS: Other Inputs

- $I$ and $G$ are linear predicates over state variables.
- The typical goal of a controller for the buck DC-DC converter is keeping the output voltage $v_O$ close enough to a given reference value $V_{\text{ref}}$.
- In formulas, $G \equiv (|v_O - V_{\text{ref}}| \leq \theta) \land (|i_L| \leq 2)$ (where $\theta = 0.01$ is the desired buck precision).
- Liveness constraint: the goal must be touched infinitely often.
- And one typically wants to be able to control all states in a big enough square region, e.g. $I \equiv (|i_L| \leq 2) \land (0 \leq v_O \leq 6.5)$. 

Automatic Synthesis of Control Software from System Level Formal Specifications

I. Melatti, Sapienza University
QKS: Quantization

- Here we will deal with A/D conversion (D/A is simpler)
- Control Software is discrete, the plant (DTLHS) is real
- Thus, the control software will have to manage different (real) states in the same way
- In fact, sensors read real-value values and sample them into a given admissible interval, using a fixed number of bits for the result
QKS: Quantization

Two consequences:

- We have a finite state labelled transition system (LTS) induced by the quantization $Q$: thus we call it a $Q$ control abstraction.
- Our $Q$ control abstraction is highly non-deterministic (if I buy a new sensor, with more bits, I can decrease the non-determinism).
QKS Algorithm
Control Abstraction

• The control abstraction $\hat{\mathcal{H}}$ of $\mathcal{H}$ is the plant as it can be seen from the control software after AD conversion and before DA conversion.
• We only consider uniform quantizations, thus it is enough to give the number of bits $b$ available on the sensor to be used.
• It is again a linear predicate: $x \in \hat{x}$ iff $\inf A_x + \delta_x \hat{x} \leq x \leq \inf A_x + \delta_x (\hat{x} + 1)$, with $\delta_x = (\sup A_x - \inf A_x)/2^b$. 

Automatic Synthesis of Control Software from System Level Formal Specifications
I. Melatti, Sapienza University
QKS Flow

Step 1: Control Abstraction Computation

Finite LTS Control Problem

Step 2: Symbolic Strong Controller Synthesis

Most General Optimal Controller

Step 3: C Code Generation from OBDD

Control Software

Specifications

Plant Model (DTLHS)

Implementation Specification (Quantization Schema)

System Level Formal Specification (Liveness and Safety)
QKS Algorithm

- Once $\hat{H}$ has been computed, it is passed to known Dijkstra-based algorithms to generate the controller
  - of course, also abstractions $\hat{I}$, $\hat{G}$ of $I$, $G$ are needed
  - starting from any initial abstract state, drives $\hat{H}$ to $\hat{G}$ regardless of possible nondeterminism
  - nondeterminism may be already present in $H$ or introduced when computing $\hat{H}$

- The controller is then translated into control software (as a C function), which reads the quantized state and outputs a quantized action
  - the control software is correct-by-construction
  - an upper bound for control software WCET (Worst Case Execution Time) is computed
How to Compute Control Abstractions

**Require:** DTLHS $\mathcal{H}$, quantization $Q$, abstract state $\hat{x}$, partial control abstraction $\hat{N}$.

**Ensure:** $\text{ctrAbsAux} (\mathcal{H}, Q, \hat{x}, \hat{N})$

1. **for all** $\hat{u} \in \Gamma(A_U)$ **do**
2.  **if** $\hat{u}$ does not lead outside $A$ **then**
3.   **if** $\text{selfLoop}(\mathcal{H}, Q, \hat{x}, \hat{u})$ **then** $\hat{N} \leftarrow \hat{N} \cup \{(\hat{x}, \hat{u}, \hat{x})\}$
4. **for all** $\hat{x}' \in \Gamma(O)$ **do**
5.  **if** $\hat{x} \neq \hat{x}' \land \text{existsTrans}(\mathcal{H}, Q, \hat{x}, \hat{u}, \hat{x}')$ **then**
6.   $\hat{N} \leftarrow \hat{N} \cup \{(\hat{x}, \hat{u}, \hat{x}')\}$
7. **return** $\hat{N}$

This must be repeated for each of the $2^b$ abstract states, that is...
How to Compute Control Abstractions

**Require:** DTLHS $\mathcal{H} = (X, U, Y, N)$, quantization $Q = (A, \Gamma)$.

**Ensure:** $\text{ctrAbs} (\mathcal{H}, Q)$

\[
\hat{N} \leftarrow \emptyset \\
\text{for all } \hat{x} \in \Gamma(A_X) \text{ do} \\
\quad \hat{N} \leftarrow \text{ctrAbsAux}(\mathcal{H}, Q, \hat{x}, \hat{N}) \\
\text{return } (\Gamma(A_X), \Gamma(A_U), \hat{N})
\]
How to Compute Strong Controllers

Require: LTS control problem \((S, I, G)\), with LTS \(S = (S, A, T)\).
Ensure: \(strongCtr(S, I, G)\)

1: \(K(s, a) \leftarrow 0\), \(D(s) \leftarrow G(s)\), \(\tilde{D}(s) \leftarrow 0\)
2: while \(D(s) \neq \tilde{D}(s)\) do
3: \(F(s, a) \leftarrow \exists s' : T(s, a, s') \land \forall s' [T(s, a, s') \Rightarrow D(s')]\)
4: \(K(s, a) \leftarrow K(s, a) \lor (F(s, a) \land \exists a : K(s, a))\)
5: \(\tilde{D}(s) \leftarrow D(s)\), \(D(s) \leftarrow D(s) \lor \exists a : K(s, a)\)
6: return \(\langle \forall s [I(s) \Rightarrow \exists a : K(s, a)], \exists a : K(s, a), K(s, a)\rangle\)
Deciding on Sol, NoSol, Unk

**Require:** DTLHS control problem \((\mathcal{H}, I, G)\), quantization \(Q = (A, \Gamma)\)

**Ensure:** \(qCtrSyn(\mathcal{H}, Q, I, G)\)

1. \(\hat{I} \leftarrow \Gamma(I), \hat{G} \leftarrow \Gamma(G)\)
2. \(\hat{M} \leftarrow \text{minCtrAbs}(\mathcal{H}, Q)\)
3. \((b, \hat{D}, \hat{K}) \leftarrow \text{strongCtr}(\hat{M}, \hat{I}, \hat{G})\)
4. **if** \(b\) **then** **return** \((\text{SOL}, \hat{D}, \hat{K})\)
5. \(\hat{W} \leftarrow \text{minFullCtrAbs}(\mathcal{H}, Q)\)
6. **if** \(\text{existsWeakCtr}(\hat{W}, \hat{I}, \hat{G})\) **then** **return** \((\text{UNK}, \hat{D}, \hat{K})\)
7. **else** **return** \((\text{NO SOL}, \hat{D}, \hat{K})\)
Writing C Code

Require: COBDD $\rho$, node $v$, boolean $b$

Ensure: $\text{Synthesize}(\rho, v, b)$:
1. $\langle v_1, b_1, \ldots, v_r, b_r \rangle \leftarrow \text{SolveFunctionalEq}(\rho, v, b)$
2. $\text{GenerateCCode}(\rho, v_1, b_1, \ldots, v_r, b_r)$
Writing C Code

Require: COBDD $\rho$, $v_1, \ldots, v_r$, boolean values $b_1, \ldots, b_r$
Ensure: $\text{GenerateCCode}(\rho, v_1, b_1, \ldots, v_r, b_r)$:

1: print “int K_bits(int *x, int action) { int ret_b;
   switch(action) {
2: for all $i \in [r]$ do
3:   print “case ”, $i-1$, “: ret_b = ”, $\overline{b_i}$, “; goto L”,
   $v_i$,”;”
4:   print “}” /* end of the switch block */
5: $W \leftarrow \emptyset$
6: for all do $i \in [r]$ $W \leftarrow \text{Translate}(\rho, v_i, W)$ done
7: print “} K(int *x, int *u){int i; for(i = 0; i < ”, $r$, “; i++) u[i] = K_bits(x, i);}”

Automatic Synthesis of Control Software from System Level Formal Specifications

I. Melatti, Sapienza University
Writing C Code

Require: COBDD $\rho$, node $v$, nodes set $W$

Ensure: $\text{Translate}(\rho, v, W)$:

1: if $v \in W$ then return $W$
2: $W \leftarrow W \cup \{v\}$, print "L_", $v$, ":"
3: if $v = 1$ then
4: print "return ret_b;"
5: else
6: let $i$ be s.t. var($v$) = $x_i$
7: print "if(x[","i-1","]==1)goto L_", high($v$)
8: if flip($v$) then print "else {ret_b=!ret_b; goto L_", low($v$), ";};""
9: else print "else goto L_", low($v$)
10: $W \leftarrow \text{Translate}(\rho, \text{high}(v), W)$
11: $W \leftarrow \text{Translate}(\rho, \text{low}(v), W)$
12: return $W$
From OBDD for $K(\hat{x}, \hat{a})$...
... to C Code

```c
int K_bits(int *x, int action) { int ret_b;
    switch(action) {
      case 0: ret_b = 0; goto L_0x15;
      case 1: ret_b = 0; goto L_0x10;
    }
    L_0x15: if (x[0] == 1) goto L_0x13;
      else { ret_b = !ret_b; goto L_0x14; }
    L_0x13: if (x[1] == 1) goto L_0xe;
      else { ret_b = !ret_b; goto L_1; }
    L_0xe: if (x[2] == 1) goto L_1;
      else { ret_b = !ret_b; goto L_1; }
    L_0x14: if (x[1] == 1) goto L_0xe;
      else goto L_1;
    L_0x10: if (x[0] == 1) goto L_0xe;
      else { ret_b = !ret_b; goto L_0xf; }
    L_0xf: if (x[1] == 1) goto L_0xe;
      else { ret_b = !ret_b; goto L_0xe; }
    L_1: return ret_b; }

void K(int *x, int *u) { int i;
    for(i = 0; i < 2; i++) u[i] = K_bits(x, i); }
```
Outline

1 Motivations
2 Our Method
3 Experimental Results
4 Conclusion and Theses
Experimental Setting

- Algorithms implemented in C (some precomputation is in Python)
- External libraries: GLPK (to solve MILPs) and CUDD (to manage OBDDs)
- 3.0 GHz Intel hyperthreaded Quad Core Linux PC with 8 GB of RAM
Case Study 1
Buck DC-DC Converter

Applications: Consumer Electronics, Airplanes, Satellites, Switching power suppliers (off-chip), On-chip power suppliers for multicore processors (energy saving)
## Experimental Results

### QKS Performance

| $b$ | CPU   | MEM    | Arcs   | CPU | $|K|$ |
|-----|-------|--------|--------|-----|------|
| 8   | 1.95e+03 | 4.41e+07 | 6.87e+05 | 2.10e-01 | 1.39e+02 |
| 9   | 9.55e+03 | 5.67e+07 | 3.91e+06 | 2.64e+01 | 3.24e+03 |
| 10  | 1.42e+05 | 8.47e+07 | 2.61e+07 | 7.36e+01 | 1.05e+04 |
| 11  | 8.76e+05 | 1.11e+08 | 2.15e+08 | 2.94e+02 | 2.88e+04 |

Automatic Synthesis of Control Software from System Level Formal Specifications

I. Melatti, Sapienza University
## Experimental Results

### QKS Performance

<table>
<thead>
<tr>
<th>b</th>
<th>CPU</th>
<th>MEM</th>
<th>μ</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1.96e+03</td>
<td>4.46e+07</td>
<td>UNK</td>
</tr>
<tr>
<td>9</td>
<td>9.58e+03</td>
<td>7.19e+07</td>
<td>SOL</td>
</tr>
<tr>
<td>10</td>
<td>1.42e+05</td>
<td>1.06e+08</td>
<td>SOL</td>
</tr>
<tr>
<td>11</td>
<td>8.76e+05</td>
<td>2.47e+08</td>
<td>SOL</td>
</tr>
</tbody>
</table>
Experimental Results

QKS Performance: MILPs

Most called MILP type

Average of MILP calls
Control Software Performances

Controlled region
Part of the Generated C Code

```c
int Controllable_Region(int *x) {
    int ret_b = 0;
    L_2af64a1: if (x[2] == 1) goto L_2b001e0;
    else { ret_b = !ret_b; goto L_2afff40; }
    L_21f95e0: return ret_b;
    L_2b07f00: if (x[14] == 1) goto L_21f95e0;
    else goto L_2b07ee0;
    /* ... */
}

int Control_Law(int *x, int *u) {
    int i;
    for (i = 0; i < 1; i++)
        u[i] = Aux_Bits(x,i);
    return 0;
}

int Aux_Bits(int *x, int b) {
    int ret_b;
    switch (b) {
        case 0: ret_b = 0; goto L_2af6081; }
    L_2af6081: if (x[2] == 1) goto L_2a6d2e0;
    else { ret_b = !ret_b; goto L_2af6060; }
    L_21f95e0: return ret_b;
    /* ... */
}
```
Control Software Performances

Startup: $v_O$ from $i_L = 0$, $v_O = 0$

State of the art: $\approx 2$ ms

Automatic Synthesis: $\approx 0.3$ ms

Ripple for $v_O$ ($b = 11$)

State of the art: $\approx 50$ mV

Automatic Synthesis: $\approx 4$ mV
Case Study 2
Inverted Pendulum

- Used as a benchmark for control engineering methods
- Nonlinear dynamics, but we have a method to effectively linearize non-linear behaviours

\[
\begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \frac{g}{l} \sin x_1 + \frac{1}{ml^2} u
\end{align*}
\]

with \( x_1 = \theta, \quad x_2 = \dot{\theta} \)
## Experimental Results

### QKS Performance

| $b$ | $T$ | $\rho$ | $|K|$ | CPU   | MEM   |
|-----|-----|--------|------|-------|-------|
| 8   | 0.1 | 0.1    | 2.73e+04 | 2.56e+03 | 7.72e+04 |
| 9   | 0.1 | 0.1    | 5.94e+04 | 1.13e+04 | 1.10e+05 |
| 10  | 0.1 | 0.1    | 1.27e+05 | 5.39e+04 | 1.97e+05 |
| 11  | 0.01| 0.05   | 4.12e+05 | 1.47e+05 | 2.94e+05 |
Control Software Performances

Controlled region
Control Software Performances

27 swings, each time decreasing angular speed \((b = 11)\)

\[\text{Ripple for } x_1 = \theta \ (b = 10)\]
Outline

1. Motivations
2. Our Method
3. Experimental Results
4. Conclusion and Theses

Automatic Synthesis of Control Software from System Level Formal Specifications

I. Melatti, Sapienza University
About 10 publications related to QKS till now
Most of such publications deal with particular aspects of QKS
On each of such aspects, improvements are possible
This both theoretical (new algorithms) and practical (implementation and evaluation through experiments)
Improvement 1: Parallel Algorithm

- Control Abstraction generation is responsible for 99% of computation time
- May require over a month of computation
- We developed a parallel map-and-reduce algorithm
- Works well, but there are some bottlenecks to be overcome
Improvement 2: Small Size Controllers

- The Control Software generated by QKS may be too big for a microcontroller
- We have developed a synthesis algorithm which trades optimality with size
- Works well, but may be improved
Improvement 3: On-the-Fly Synthesis

- In this variant, control abstraction computation and controller synthesis are carried out together.
- Essentially, failures (i.e., NoSol or Unk) of QKS are recognized faster.
- This enables design space exploration, i.e. exploration of different designs.
- A parallel version should be designed.
Improvement 4: Non-linear Systems

• Methodology to effectively linearize systems with non-linear dynamics
• Essentially, we use a linear dynamics which overapproximates the original one
• Thus, if a controller is found, it is a controller also for the original system
Other Works on Synthesis

- We are currently running two FP7 European Projects as coordinators
- Such projects sum up to 5.7 million Euros
  - **SmartHG** ([http://smarthg.di.uniroma1.it/](http://smarthg.di.uniroma1.it/)): develop services to improve electricity distribution both from distributors and users point of view
  - **PAEON** ([http://paeon.di.uniroma1.it/](http://paeon.di.uniroma1.it/)): develop automatic treatments for human infertility
- Both such projects have parts regarding synthesis (though tackled via different methods)
Any questions?