QKS

Quantized Controller Software Generator

DTLHS

Desired Controllable Region

Goal Region

AD/DA n. of bits

Unknown

NoSol

SOL

Control Software

Actual Controllable Region

Robustness by Construction

Guaranteed WCET

Control Software Synthesis for DTLHS

Stems from undecidability of DTLHS control problem

[ICTAC 12]

Idea: Halting problem for two counter machine reduced to DTLHS control problem

mclab qks

D/A

Action

Plant (physical system)

A/D

Observable state of plant

Microcontroller
QKS Workflow

1. **Input**
   - Implementation Specification (Quantization Schema)
   - Plant Model (DTLHS)

2. **Step 1: Control Abstraction Computation**

3. **Finite LTS Control Problem**

4. **Step 2: Symbolic Strong Controller Synthesis**

5. **Step 3: C Code Generation from OBDD**

6. **Most General Optimal Controller**

7. **Control Software**

---

Control Software Synthesis for DTLHS
With 12 bits on the inverted pendulum QKS generated control software has size greater than 8MB.

Micro-controllers typically have few MB of memory.

Is it possible to synthesize control software with a smaller size?
Small Size Control Software

Modify controller synthesis to have small size controllers
Example FSM A

```plaintext
Control Software Synthesis for DTLHS
```
Controllers for A

Most general optimal controller as output by QKS

Small size controller

Idea: try to use the same action as much as possible
OBDDs for A

OBDD-based computation of the mgo controller for a finite state machine (control abstraction) [Cimatti, 98]

Increase sharing
Same height
C Code for OBDDs for A

```
int ctrlLaw(unsigned char *x) {
    int act=0;
    L_v1: if (x[2]==1) goto L_v3;
        else { act = !act;
              goto L_v2; }
    L_v2: if (x[1]==1) goto L_v4;
        else { act = !act;
              goto L_v4; }
    L_v3: if (x[1]==1) return act;
        else goto L_v4;
    L_v4: if (x[0]==1) return act;
        else { act = !act;
              return act; }
}
```

```
int ctrlLaw(unsigned char *x) {
    int act=0;
    L_v1: if (x[2]==1) goto L_v2;
        else return act;
    L_v2: if (x[1]==1) return act;
        else goto L_v3;
    L_v3: if (x[0]==1) return act;
        else { act = !act;
              return act; }
}
```
Controller Synthesis Algorithm
Controller Synthesis Algorithm
Controller Synthesis Algorithm

- Pre-Images
  - $\text{Pre}(a_0)$

Idea: finding maximal regions

- Pre-Images
  - $\text{Pre}(a_0)$
  - $\text{Pre}(a_1)$

Idea: finding maximal regions

- Pre-Images
  - Pre$(a_0)$
  - Pre$(a_1)$
  - Pre$(a_2)$

Idea: finding maximal regions

- Pre-Images
  - Pre($a_0$)
  - Pre($a_1$)
  - Pre($a_2$)

- Merge ($a_0$, $a_1$, $a_2$)
  - Pre($a_0$)

Idea: finding maximal regions

- Pre-Images
  - $\text{Pre}(a_0)$
  - $\text{Pre}(a_1)$
  - $\text{Pre}(a_2)$

- Merge $(a_0, a_1, a_2)$
  - $\text{Pre}(a_0)$
  - $\cup (\text{Pre}(a_1) \setminus \text{Pre}(a_0))$

Idea: finding maximal regions

- Pre-Images
  - \(\text{Pre}(a_0)\)
  - \(\text{Pre}(a_1)\)
  - \(\text{Pre}(a_2)\)

- Merge \((a_0, a_1, a_2)\)
  - \(\text{Pre}(a_0)\)
  - \(\cup (\text{Pre}(a_1) \setminus \text{Pre}(a_0))\)

Idea: finding maximal regions

- **Pre-Images**
  - $\text{Pre}(a_0)$
  - $\text{Pre}(a_1)$
  - $\text{Pre}(a_2)$

- **Merge** $(a_0, a_1, a_2)$
  - $\text{Pre}(a_0)$
  - $\cup (\text{Pre}(a_1) \setminus \text{Pre}(a_0))$
  - $\cup (\text{Pre}(a_2) \setminus \text{Pre}(a_0) \setminus \text{Pre}(a_1))$

Idea: finding maximal regions

- Pre-Images
  - $\text{Pre}(a_0)$
  - $\text{Pre}(a_1)$
  - $\text{Pre}(a_2)$

- Merge $(a_0, a_1, a_2)$
  - $\text{Pre}(a_0)$
  - $\bigcup (\text{Pre}(a_1) \setminus \text{Pre}(a_0))$
  - $\bigcup (\text{Pre}(a_2) \setminus \text{Pre}(a_0) \setminus \text{Pre}(a_1))$

Idea: finding maximal regions

- **Pre-Images**
  - ✓ Pre($a_0$)
  - ✓ Pre($a_1$)
  - ✓ Pre($a_2$)

- **Merge ($a_0$, $a_1$, $a_2$)**
  - ✓ Pre($a_0$)
  - ✓ $\bigcup (\text{Pre}(a_1) \setminus \text{Pre}(a_0))$
  - ✓ $\bigcup (\text{Pre}(a_2) \setminus \text{Pre}(a_0) \setminus \text{Pre}(a_1))$

Idea: finding maximal regions
Differences

$K_{mg}$

$K_{sc}$
## Under-actuated Inverted Pendulum

| $b$ | $T$ | $|K^\text{mgo}|$ | $|K^\text{sc}|$ | $\frac{|K^\text{sc}|}{|K^\text{mgo}|}$ | Path$^\text{mgo}$ | Path$^\text{sc}$ | $\frac{\text{Path}^\text{sc}}{\text{Path}^\text{mgo}}$ |
|-----|-----|-----------------|-----------------|-------------------------------|----------------|----------------|-----------------|
| 8   | 0.1 | 163             | 44              | 27.4%                         | 132.96        | 234.35        | 1.76            |
| 9   | 0.1 | 352             | 92              | 26.3%                         | 69.64         | 147.74        | 2.12            |
| 10  | 0.1 | 752             | 206             | 27.5%                         | 59.16         | 133.70        | 2.26            |
| 11  | 0.01| 2467            | 213             | 8.6%                          | 1315.69       | 1898.50       | 1.44            |
| 12  | 0.01| 8329            | 439             | 5.3%                          | 674.39        | 1280.32       | 1.90            |

- **pros**
  - code size in kB of .o file after gcc compilation
  - average length of (worst case) paths to the goal region

- **cons**
  - `Path^sc / Path^mgo`
Under-actuated Inverted Pendulum

Setup Time

- mgo $x_1$ (red line)
- mgo $x_2$ (green dashed line)
- sc $x_1$ (blue dotted line)
- sc $x_2$ (pink square line)

- mgo: ~10 secs
- sc: ~14 secs
Under-actuated Inverted Pendulum

Ripple

mgo: ~0.0001 rad

sc: ~0.0002 rads
Under-actuated Inverted Pendulum

Enabled Actions in Controlled Regions

mgo

sc: more compact
On QKS Off-line Computation Time

• Off-line computation time can be **very large**

• Not affordable for **design space exploration**

• **Is it possible to speed-up at least negative answers?** (When no controller exists?)
On-the-Fly Control Software Generation

1. Control Abstraction Computation

2. Symbolic Strong Controller Synthesis

3. C Code Generation from OBDD

Generate control abstraction computation on demand when needed by controller synthesis.
On-the-Fly Control Software Generation

Input
- Implementation Specification (Quantization Schema)
- Plant Model (DTLHS)
- System Level Formal Specification (Liveness and Safety)

Step 1: On-The-Fly Strong Controller Synthesis

Most General Optimal Controller

Step 2: C Code Generation from OBDD

Control Software

Generate control abstraction computation on demand when needed by controller synthesis
Control Software Synthesis for DTLHS

Control Abstraction as Computed by Exhaustive Method
On-the-Fly Control Software Generation

Control Abstraction as Computed by Exhaustive Method

Computing an Abstraction of the Goal Region
On-the-Fly Control Software Generation

Control Abstraction as Computed by Exhaustive Method

States Reaching the Goal in One Step
On-the-Fly Control Software Generation

Control Abstraction as Computed by Exhaustive Method

States Reaching the Controllable Region in One Step
### Inverted Pendulum with Fixed Pivot Point

| b  | n | CPU\(^{exh}\) | RAM\(^{exh}\) | CPU\(^{otf}\) | RAM\(^{otf}\) | | | |
|---|---|---|---|---|---|---|---|
| 8 | 10 | 9.90e+04 | 1.70e+08 | 4.58e+02 | 3.03e+07 | 1.25e+02 | 99.54 | 216.16 | FAIL |
| 8 | 8  | 4.41e+04 | 1.68e+08 | 3.06e+02 | 3.05e+07 | 2.06e+02 | 99.31 | 144.12 | FAIL |
| 8 | 6  | 2.28e+04 | 1.65e+08 | 2.77e+04 | 9.12e+07 | 6.40e+03 | -21.49 | 0.82  | PASS |
| 8 | 4  | 1.17e+04 | 1.63e+08 | 1.47e+04 | 8.68e+07 | 7.53e+03 | -25.64 | 0.80  | PASS |
| 8 | 2  | 4.91e+03 | 1.63e+08 | 1.35e+01 | 2.98e+07 | 1.53e+02 | 99.73 | 363.70 | FAIL |
| 8 | 1  | 2.69e+03 | 1.53e+08 | 4.72e+00 | 2.98e+07 | 1.61e+02 | 99.82 | 569.92 | FAIL |
| 9 | 10 | 4.95e+05 | 2.39e+08 | 2.70e+03 | 3.16e+07 | 1.88e+02 | 99.45 | 183.33 | FAIL |
| 9 | 8  | 2.31e+05 | 2.31e+08 | 2.40e+05 | 2.70e+08 | 1.06e+04 | -3.90 | 0.96  | PASS |
| 9 | 6  | 1.20e+05 | 2.18e+08 | 1.19e+05 | 2.71e+08 | 1.25e+04 | 0.83  | 1.01  | PASS |
| 9 | 4  | 5.66e+04 | 1.98e+08 | 5.34e+04 | 2.50e+08 | 1.55e+04 | 5.65  | 1.06  | PASS |
| 9 | 2  | 2.18e+04 | 1.91e+08 | 2.29e+04 | 2.43e+08 | 2.16e+04 | -5.06 | 0.95  | PASS |
| 9 | 1  | 1.16e+04 | 1.78e+08 | 1.97e+01 | 3.02e+07 | 2.11e+02 | 99.83 | 588.83 | FAIL |
| 10 | 10 | 3.82e+06 | 6.08e+08 | 1.45e+04 | 3.65e+07 | 2.87e+02 | 99.62 | 263.45 | FAIL |
| 10 | 8  | 1.71e+06 | 5.40e+08 | 6.74e+03 | 3.83e+07 | 6.01e+02 | 99.61 | 253.71 | FAIL |
| 10 | 6  | 7.45e+05 | 4.72e+08 | 6.67e+05 | 8.81e+08 | 2.45e+04 | 10.47 | 1.12  | PASS |
| 10 | 4  | 3.05e+05 | 4.13e+08 | 2.77e+05 | 8.31e+08 | 2.99e+04 | 9.18  | 1.10  | PASS |
| 10 | 2  | 1.05e+05 | 3.29e+08 | 9.96e+04 | 8.12e+08 | 4.52e+04 | 5.14  | 1.05  | PASS |
| 10 | 1  | 5.29e+04 | 2.64e+08 | 5.09e+04 | 8.07e+08 | 6.31e+04 | 3.78  | 1.04  | PASS |

**Overall**: $7.85e+06$ 6.08e+08 1.60e+06 8.81e+08 79.62 4.91

Sampling time $T = nt$ where $t$ is the system time step
Design Space Exploration Speed-up

Percentage Gain of On-the-Fly w.r.t. Exhaustive

![Percentage Gain of On-the-Fly w.r.t. Exhaustive](chart.png)
Schedulability & Realizability Analysis

• In our case, control software WCET is linear in the number of quantization bits

• For any \( n \) (sampling time \( T = nt \), \( t \) time step) and for any \( b \) we consider the architecture in the ATMEL family that has smallest fit flash memory

• We compute the WCET (less or equal than \( 4bT_B \), where \( T_B \) is an upper bound to compute an if-then-else block)

• Referring to a Rate Monotonic Schedule (RMS), we estimate the number of processes that can run in parallel with the control software (assuming 100 clock cycles as WCET for such processes)
### Schedulability & Realizability Analysis

$k$ is the number of processes running together with the control software.

<table>
<thead>
<tr>
<th>$b$</th>
<th>$n$</th>
<th>Code size (bytes)</th>
<th>Arch</th>
<th>WCET (secs)</th>
<th>WCET $T$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
<td>5.00e+03</td>
<td>atmega8</td>
<td>3.20e-04</td>
<td>6.40e-04</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>1.45e+05</td>
<td>atmega16</td>
<td>1.92e-04</td>
<td>6.40e-04</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4.85e+03</td>
<td>atmega8</td>
<td>6.40e-05</td>
<td>6.40e-04</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>4.31e+03</td>
<td>atmega8</td>
<td>3.20e-05</td>
<td>6.40e-04</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>7.66e+03</td>
<td>atmega8</td>
<td>3.60e-04</td>
<td>7.20e-04</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>2.80e+05</td>
<td>atmega16</td>
<td>2.16e-04</td>
<td>7.20e-04</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>9.50e+05</td>
<td>ARM</td>
<td>4.32e-06</td>
<td>4.32e-05</td>
<td>344</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>5.98e+03</td>
<td>atmega8</td>
<td>3.60e-05</td>
<td>7.20e-04</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1.20e+04</td>
<td>atmega8</td>
<td>4.00e-04</td>
<td>8.00e-04</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>1.06e+06</td>
<td>ARM</td>
<td>1.44e-05</td>
<td>4.80e-05</td>
<td>344</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1.96e+06</td>
<td>ARM</td>
<td>4.80e-06</td>
<td>4.80e-05</td>
<td>344</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2.63e+06</td>
<td>ARM</td>
<td>2.40e-06</td>
<td>4.80e-05</td>
<td>344</td>
</tr>
</tbody>
</table>
Inverted Pendulum

Setup Time

Decreasing sampling time... decreases setup time

Increasing quantization bits... decreases setup time
Inverted Pendulum

Ripple

Decreasing sampling time…

Increasing quantization bits…

… decreases ripple

… decreases ripple
On QKS Off-line Computation Time

• Off-line computation time can be very large

• Not affordable for design space exploration

• Is it possible to speed-up at least negative answers? (When no controller exists?)

• Is it possible to exploit parallelism?

yes: on-the-fly
Map-Reduce Control Abstraction

Step 1: Control Abstraction Computation

Map-reduce approach to generate control abstraction

Finite LTS Control Problem

Step 2: Symbolic Strong Controller Synthesis

Most General Optimal Controller

Step 3: C Code Generation from OBDD

Control Software
Control Abstraction Computation

Standard Algorithm

Original system -> MaxCtrlAbs -> MinCtrlAbs

Arcs are placed by solving MILPs which are independent from each other.
Example: 3 workers & 16 states (2 state vars, 2 bits each)

- Distributed memory model
- Use MPI_barrier to synchronize processes
- Use shared filesystem to exchange data between processes

Map-Reduce Control Abstraction
Inverted Pendulum

Parallel vs Sequential

Speed-up

Scaling Efficiency

\[
\text{Speedup} = \frac{\text{Sequential Time}}{\text{Parallel Time}}
\]

\[
\text{Scaling Efficiency} = \frac{\text{Speedup}}{\text{Number of Processes}} \times 100\%
\]
Inverted Pendulum

Parallel

Communication time

I/O time

30 processes have low comm. time also for good distribution of “hard” states as shown in the following

Comm Time = Waiting Time + I/O Time
Inverted Pendulum

Parallel

30 workers — 9 bits

40 workers — 9 bits

Hard states. Their distribution influences comm. time (see prev. slide)

With 40 processes hard states are not well distributed: higher comm. time
Linearizing Discrete Time Hybrid Systems

Prof. Enrico Tronci
Computer Science Department

Ph.D. Programme "Automatica, Bioengineering and Operations Research"
Course on "Hybrid systems: Computation and Control"

September 12–22, 2018
Department of Computer, Control, and Management Engineering
Sapienza University of Rome (Italy)
On QKS Input Models

- QKS can deal only with **linear systems**
- Many interesting systems are **nonlinear**
- Is it possible to deal with nonlinear systems?
Over-approx a nonlinear DTHS with a DTLHS in such a way that controllers for the DTLHS are guaranteed to be controllers also for the DTHS.
We over-approximate DTHS *syntactically* transforming its transition relation.

For each nonlinear function* $f$ occurring in transition relation:

1. we compute its piecewise affine under- and over-approximations $f$ and $f^+$ (*linearisation*)

2. we replace $f$ by linear constraints satisfied by all values in the interval $[f^-, f^+]$

Resulting DTLHS has more behaviour than original DTHS.

Over-approximation can be made as tight as we want.

Tighter => more chances to find controller, but harder to compute it.

*We require $f$ to be Lipschitz continuous
Input:

- Closed bounded region \( D = [a, b] \subset \mathbb{R}^n \)
- Lipschitz continuous function \( f : D \rightarrow \mathbb{R} \)
- Number of sampling points \( m \)
- Maximum linearisation error \( \varepsilon \)

**Linearisation Algorithm**

```
linearize(D, f, m, \varepsilon)
```

1. Compute values of \( f \) in \( m^n \) points from \( D \)
2. Construct LP which parameters are coefficients of linearisation of \( f \) and objective is to minimise linearisation error \( err \)
3. Solve constructed LP

   - If \( err \leq \varepsilon \), return linearisation defined by LP solution parameter values.
   - If not, split \( D \) into \( 2^n \) subregions \( D_i \) and call recursively \( linearize(D_i, f, m, \varepsilon) \)

4. Return linearisation \( D \) composed by linearisations of \( D_i \)
Control Software Synthesis for Discrete Time Nonlinear Hybrid Systems

https://bitbucket.org/mclab/linearizer – general library for linearisation of a function
https://bitbucket.org/mclab/lin4qks – tool built upon linearizer library for automatic overapproximation of DTHS by means of DTLHS

Discrete Time Hybrid System

Safety and Liveness

Quantization Sampling Time

lin4qks

Discrete Time Linear Hybrid System

QKS

Correct-By-Construction Control Software

WCET \leq \text{Sampling Time}
Experimental Evaluation: Linearisation

\[ y \cos x \]

\( \varepsilon \): linearisation error
Experimental Evaluation: Linearisation

\[ \sum_{i=1}^{t} x_i x_{i+1} \]

<table>
<thead>
<tr>
<th></th>
<th>( t=2, \ m=40, \ x_i \in [-5,5] )</th>
<th></th>
<th>( t=3, \ m=10, \ x_i \in [-2,2] )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( N )</td>
<td>( \text{CPU} )</td>
<td>( N )</td>
</tr>
<tr>
<td>monolithic</td>
<td>4768</td>
<td>5d7h49m51s</td>
<td>45436</td>
</tr>
<tr>
<td>term-wise</td>
<td>536</td>
<td>1m15s</td>
<td>660</td>
</tr>
</tbody>
</table>

\( n \): number of linearisation intervals  
\( m \): number of sampling points

Term-wise approach helps counteracting curse of dimensionality
Experimental Evaluation: Control Synthesis

Inverted Pendulum with different frictions

\[ \ddot{\theta} = \frac{g}{l} \sin \theta - \frac{\mu_1}{ml^2} \dot{\theta} + \frac{1}{ml^2} u \quad \text{above} \]

\[ \theta \geq -\frac{\pi}{2} \land \theta \leq \frac{\pi}{2} \]

\[ \ddot{\theta} = \frac{g}{l} \sin \theta - \frac{\mu_2}{ml^2} \dot{\theta} + \frac{1}{ml^2} u \quad \text{below} \]

\[ \theta \leq -\frac{\pi}{2} \lor \theta \geq \frac{\pi}{2} \]

Hybrid system with inter-sampling mode jumps
## Experimental Evaluation: Control Synthesis

### Inverted Pendulum with different frictions

<table>
<thead>
<tr>
<th>b</th>
<th>ε</th>
<th>τ</th>
<th>T</th>
<th>n</th>
<th>CPU&lt;sub&gt;lin&lt;/sub&gt;</th>
<th>CPU&lt;sub&gt;ctr&lt;/sub&gt;</th>
<th>RAM&lt;sub&gt;ctr&lt;/sub&gt;</th>
<th>RES</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.5</td>
<td>0.01</td>
<td>0.1</td>
<td>4</td>
<td>&lt; 1s</td>
<td>27m40s</td>
<td>76MB</td>
<td>FAIL</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>0.01</td>
<td>0.1</td>
<td>4</td>
<td>&lt; 1s</td>
<td>8d22h13m35s</td>
<td>466MB</td>
<td>PASS</td>
</tr>
<tr>
<td>9</td>
<td>1.1</td>
<td>0.01</td>
<td>0.1</td>
<td>2</td>
<td>&lt; 1s</td>
<td>1h01m04s</td>
<td>75MB</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

- b: number of AD bits
- ε: linearisation error
- τ: system time step
- T: sampling time
- n: number of intervals in the linearisation
Experimental Evaluation: Control Synthesis

Inverted Pendulum with different frictions

Controllable Region

Simulation starting from downwards position
Experimental Evaluation: Control Synthesis
Comparison with state of the art

lin4qks + QKS  

VS  

state-of-the-art tool PESSOA  
https://sites.google.com/a/cyphylab.ee.ucla.edu/pessoa/
Since PESSOA can only handle switched hybrid systems, we set frictions to be the same in our inverted pendulum benchmark to perform comparison.

\[
\ddot{\theta} = \frac{g}{l} \sin \theta - \frac{\mu}{ml^2} \dot{\theta} + \frac{1}{ml^2} u
\]
Experimental Evaluation: Control Synthesis
Comparison with state of the art

<table>
<thead>
<tr>
<th>Tool</th>
<th>b</th>
<th>( \tau )</th>
<th>T</th>
<th>CPU</th>
<th>MEM</th>
<th>IKI</th>
<th>h(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PESSOA</td>
<td>9</td>
<td>ode45* (0.1)</td>
<td>0.1</td>
<td>2h00m07s</td>
<td>111MB</td>
<td>19994</td>
<td>40</td>
</tr>
<tr>
<td>QKS</td>
<td>9</td>
<td>0.1</td>
<td>0.1</td>
<td>1h36m07s</td>
<td>467MB</td>
<td>21629</td>
<td>38</td>
</tr>
<tr>
<td>QKS</td>
<td>9</td>
<td>0.01</td>
<td>0.1</td>
<td>23h14m24s</td>
<td>504MB</td>
<td>21961</td>
<td>38</td>
</tr>
</tbody>
</table>

For QKS experiments we use linearisation computed with \( \varepsilon = 0.1 \) (8 intervals)

- b: number of AD bits
- \( \tau \): system time step
- T: sampling time
- \(|K|\): controller OBDD size
- h(K): controller OBDD height

*PESSOA uses variable-step integrator ode45 from MATLAB®
Experimental Evaluation: Control Synthesis
Comparison with state of the art

Simulation starting from downwards position
Experimental Evaluation: Control Synthesis
Comparison with state of the art

Controllable Regions

PESSOA

QKS
Experimental Evaluation

All benchmarks are available online:

https://bitbucket.org/mclab/linearizer-benchmark