Continuous Verification of Large Embedded Software using SMT-Based Bounded Model Checking

Lucas Cordeiro, Bernd Fischer, Joao Marques-Silva
lcc08r@ecs.soton.ac.uk
Agenda

- Continuous Verification
- SMT-based Bounded Model Checking (BMC)
- Case Study and Experimental Results
- Conclusions and Future Work
Continuous Verification

• based on Fowler’s **continuous integration (CI):** build and test full system after each change
• complement testing by verification (SMT-based bounded model checking)
  – assertions
  – language-specific properties
• exploit existing information
  – development history (SCM)
  – test cases
• limit change propagation
  – equivalence checks
Functional Equivalence Checking

• determine whether modified functions need to be re-verified
  – no need to re-verify properties if functions are equivalent
  – **less expensive** than re-verifying the function
  – **undecidable** due to unbounded memory usage
Functional Equivalence Checking

• determine whether modified functions need to be re-verified
  – no need to re-verify properties if functions are equivalent
  – less expensive than re-verifying the function
  – undecidable due to unbounded memory usage

• goal: compare input-output relation

```c
unsigned Inv(int signal) {
    unsigned inverter;
    if (signal >= 0)
        inverter = signal;
    else
        inverter = -1*signal;
    return inverter;
}
```

```c
unsigned Inv(int signal) {
    if (signal < 0)
        return -signal;
    else
        return signal;
}
```
Functional Equivalence Checking

- determine whether modified functions need to be re-verified
  - no need to re-verify properties if functions are equivalent
  - less expensive than re-verifying the function
  - undecidable due to unbounded memory usage

- goal: compare input-output relation
  - remove variables and returns

```c
unsigned Inv(int signal) {
    unsigned inverter;
    if (signal >= 0)
        inverter = signal;
    else
        inverter = -1*signal;
    return inverter;
}
```

```c
unsigned Inv(int signal) {
    if (signal < 0)
        return -signal;
    else
        return signal;
}
```
Functional Equivalence Checking

• determine whether modified functions need to be re-verified
  – no need to re-verify properties if functions are equivalent
  – less expensive than re-verifying the function
  – undecidable due to unbounded memory usage

• goal: compare input-output relation
  – remove variables and returns
  – convert the function bodies into SSA

\[
\alpha_1 = \begin{bmatrix}
inverter_1 = \text{signal}_1 \\
\land inverter_2 = -1 \ast \text{signal}_1 \\
\land inverter_3 = (\text{signal}_1 \geq 0 ? \text{inverter}_1 : \text{inverter}_2)
\end{bmatrix}
\]

\[
\alpha_2 = [\text{signal}'_2 = (\text{signal}'_1 < 0 ? -\text{signal}'_1 : \text{signal}'_1)]
\]

```c
unsigned Inv(int signal) {
    unsigned inverter;
    if (signal >= 0)
        inverter = signal;
    else
        inverter = -1*signal;
    return inverter;
}
```

```c
unsigned Inv(int signal) {
    if (signal < 0)
        return -signal;
    else
        return signal;
}
```
Functional Equivalence Checking

- determine whether modified functions need to be re-verified
  - no need to re-verify properties if functions are equivalent
  - less expensive than re-verifying the function
  - undecidable due to unbounded memory usage
- goal: compare input-output relation
  - remove variables and returns
  - convert the function bodies into SSA
  - show that the input and output variables coincide

\[
(\alpha_1 \land \alpha_2 \land (\text{signal}_1 = \text{signal}'_1)) \rightarrow (\text{inverter}_3 = \text{signal}'_2)
\]
Generalizing Test Cases

• use **existing test cases** to reduce the state space
  – run the unit tests, keep track of inputs
  – guide model checker to visit states not yet visited
• test stubs break the **global model** into **local models**
  – use test case as initial state
  – generate reachable states on-demand
⇒ reduces the number of paths and variables

```
main

sensor ⇔ timer ⇔ serial

sensor

Test cases

a=nondet_int(); assume(a>10 && a<200);
```
Generalizing Test Cases: Example

Simple circular FIFO buffer:

```c
static char buffer[BUFFER_MAX];
void initLog(int max) {
    buffer_size = max;
    first = next = 0;
}

int removeLogElem(void) {
    first++;
    return buffer[first-1];
}

void insertLogElem(int b) {
    if (next < buffer_size) {
        buffer[next] = b;
        next = (next+1)%buffer_size;
    }
}
```

**Test case:**
check whether messages are added to and removed from the circular buffer

```c
static void testCircularBuffer(void) {
    int senData[] = {1, -128, 98, 88, 59, 1, -128, 90, 0, -37};
    int i;
    initLog(5);
    for(i=0; i<10; i++)
        insertLogElem(senData[i]);
    for(i=5; i<10; i++)
        ASSERT_EQUAL_INT(senData[i], removeLogElem());
}
```
Generalizing Test Cases: Example

Simple circular FIFO buffer:

```c
static char buffer[BUFFER_MAX];
void initLog(int max) {
    buffer_size = max;
    first = next = 0;
}

int removeLogElem(void) {
    first++;
    return buffer[first-1];
}

void insertLogElem(int b) {
    if (next < buffer_size) {
        buffer[next] = b;
        next = (next+1)%buffer_size;
    }
}
```

BUT: implementation is flawed!

- The array buffer is of type char[]
- Assign an integer variable
Generalizing Test Cases: Example

Simple circular FIFO buffer:

```c
static char buffer[BUFFER_MAX];
void initLog(int max) {
    buffer_size = max;
    first = next = 0;
}

int removeLogElem(void) {
    first++;
    return buffer[first-1];
}

void insertLogElem(int b) {
    if (next < buffer_size) {
        buffer[next] = nondet_int();
        next = (next+1)%buffer_size;
    }
}
```

BUT: implementation is flawed!

The array buffer is of type char[]

Assign an integer variable

We can detect the error by assigning a non-deterministic value

This can lead to false results
Rather than modifying the program we *modify the test stubs*

```c
static void testCircularBuffer(void) {
    int senData[] = {nondet_int(), ..., nondet_int()};

    assume(senData[0] <= 1 && senData[0] >= 42);
    assume(senData[1] <= -128 && senData[1] >= -28);
    ...
    int i;
    initLog(5);
    for(i=0; i<10; i++)
        insertLogElem(senData[i]);
    for(i=5; i<10; i++)
        ASSERT_EQUAL_INT(senData[i],
                         removeLogElem());
}
```

⇒ detects two bugs related to buffer over- and underflow

Block larger parts of the search space (combine respective values into a single interval)

- force the model checker towards the “unobvious” errors
Agenda

- Continuous Verification

- SMT-based Bounded Model Checking (BMC)

- Case Study and Experimental Results

- Conclusions and Future Work
Bounded Model Checking (BMC)

Basic Idea: check negation of given property up to given depth

- transition system $M$ unrolled $k$ times
  - for programs: unroll loops, unfold arrays, ...
- translated into verification condition $\psi$ such that
  $$\psi \text{ satisfiable iff } \varphi \text{ has counterexample of max. depth } k$$
- has been applied successfully to verify (embedded) software
Satisfiability Modulo Theories (1)

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories** (⇒ building-in operators).

<table>
<thead>
<tr>
<th>Theory</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>$x_1 = x_2 \land \neg (x_1 = x_3) \Rightarrow \neg (x_1 = x_3)$</td>
</tr>
<tr>
<td>Bit-vectors</td>
<td>$(b &gt;&gt; i) \land 1 = 1$</td>
</tr>
<tr>
<td>Linear arithmetic</td>
<td>$(4y_1 + 3y_2 \geq 4) \lor (y_2 - 3y_3 \leq 3)$</td>
</tr>
<tr>
<td>Arrays</td>
<td>$(j = k \land a[k]=2) \Rightarrow a[j]=2$</td>
</tr>
<tr>
<td>Combined theories</td>
<td>$(j \leq k \land a[j]=2) \Rightarrow a[k] &lt; 3$</td>
</tr>
</tbody>
</table>
Satisfiability Modulo Theories (2)

• Given
  – a decidable $\Sigma$-theory $T$
  – a quantifier-free formula $\varphi$

$\varphi$ is $T$-satisfiable iff $T \cup \{\varphi\}$ is satisfiable, i.e., there exists a structure that satisfies both formula and sentences of $T$

• Given
  – a set $\Gamma \cup \{\varphi\}$ of first-order formulae over $T$

$\varphi$ is a $T$-consequence of $\Gamma$ ($\Gamma \vDash_T \varphi$) iff every model of $T \cup \Gamma$ is also a model of $\varphi$

• Checking $\Gamma \vDash_T \varphi$ can be reduced in the usual way to checking the $T$-satisfiability of $\Gamma \cup \{\neg \varphi\}$
Software BMC using ESBMC

- program modelled as state transition system
  - state: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unrolled up to given bounds
  - number of loop iterations
  - size of arrays
- unrolled program optimized to reduce blow-up
  - constant folding
  - forward substitutions

```c
int main() {
    int a[2], i, x;
    if (x==0)
        a[i]=0;
    else
        a[i+2]=1;
    assert(a[i+1]==1);
}
```
Software BMC using ESBMC

• program modelled as state transition system
  – *state*: program counter and program variables
  – derived from control-flow graph
  – checked safety properties give extra nodes
• program unrolled up to given bounds
  – number of loop iterations
  – size of arrays
• unrolled program optimized to reduce blow-up
  – constant folding
  – forward substitutions\[\text{crucial}\]
• front-end converts unrolled and optimized program into SSA

```c
int main() {
    int a[2], i, x;
    if (x==0)
        a[i]=0;
    else
        a[i+2]=1;
    assert(a[i+1]==1);
}
```

\[g_1 = x_1 == 0 \]
\[a_1 = a_0 \text{ WITH } [i_0:=0] \]
\[a_2 = a_0 \]
\[a_3 = a_2 \text{ WITH } [2+i_0:=1] \]
\[a_4 = g_1 \ ? \ a_1 : a_3 \]
\[t_1 = a_4 [1+i_0] == 1 \]
Software BMC using ESBMC

- program modelled as state transition system
  - state: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unrolled up to given bounds
  - number of loop iterations
  - size of arrays
- unrolled program optimized to reduce blow-up
  - constant folding
  - forward substitutions
- front-end converts unrolled and optimized program into SSA
- extraction of constraints \( C \) and properties \( P \)
  - specific to selected SMT solver, uses theories
- satisfiability check of \( C \land \neg P \)

```c
int main() {
    int a[2], i, x;
    if (x==0)
        a[i]=0;
    else
        a[i+2]=1;
    assert(a[i+1]==1);
}
```

\[
\begin{align*}
C & := \begin{cases}
g_1 := (x_1 = 0) \\
\land a_1 := store(a_0, i_0, 0) \\
\land a_2 := a_0 \\
\land a_3 := store(a_2, 2 + i_0, 1) \\
\land a_4 := ite(g_1, a_1, a_3)
\end{cases} \\
\end{align*}
\]

\[
\begin{align*}
P & := \begin{cases}
i_0 \geq 0 \land i_0 < 2 \\
\land 2 + i_0 \geq 0 \land 2 + i_0 < 2 \\
\land 1 + i_0 \geq 0 \land 1 + i_0 < 2 \\
\land select(a_4, i_0 + 1) = 1
\end{cases} \\
\end{align*}
\]
Extending ESBMC

- SMT solvers provide different encodings for numbers:
  - abstract domains ($\mathbb{Z}$, $\mathbb{R}$)
  - fixed-width bit vectors ($\text{unsigned int}$, ...)
- majority of VCs solved faster if numeric types are modelled by abstract domains but possible loss of precision
  - default solver: Z3 (using AUFLIRA logic)
  - switch to Boolector and encode as bit-vectors (when using bit operations or typecasts but no pointers)
- encoding of floating-point arithmetic leads to large formulae
  - approximate by fixed-point arithmetic
- we check two properties for dynamic memory allocation
  - whether argument to malloc / free is a dynamic object
  - whether argument to free is still a valid object
Agenda

- Continuous Verification
- SMT-based Bounded Model Checking (BMC)
- Case Study and Experimental Results
- Conclusions and Future Work
Experimental Evaluation

- goal: check efficiency and effectiveness of ESBMC
  - check error-detection capability on different benchmarks
  - evaluate ESBMC’s performance relative to CBMC
  - evaluate scalability of the CV approach

- set-up:
  - Intel Pentium Dual CPU, 2GHz / 4GB RAM, Linux OS
  - time limit 3600 seconds / individual property
## Error-Detection Capability

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>Enc.</th>
<th>Solver</th>
<th>Time</th>
<th>Passed</th>
<th>Fail</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERISEC</td>
<td>590</td>
<td>2140</td>
<td>190.9</td>
<td>228.4</td>
<td>419.2</td>
<td>1949</td>
<td>202</td>
<td>0</td>
</tr>
<tr>
<td>NE</td>
<td>3102</td>
<td>184</td>
<td>5.2</td>
<td>188</td>
<td>20</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNU-RT</td>
<td>3430</td>
<td>726</td>
<td>72.7</td>
<td>8.3</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POWERSTONE</td>
<td>2957</td>
<td>2053</td>
<td>127.4</td>
<td>913.6</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **SMT solver time**
- **Encoding time**
- **Number of properties checked**
- **Lines of code**
- **Error detected in module – GOOD THING**
- **Error occurred in tool – BAD THING**
## Error-Detection Capability

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>Time</th>
<th>#P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enc.</td>
<td>Solver</td>
</tr>
<tr>
<td>VERISEC</td>
<td>9090</td>
<td>2148</td>
<td>190.9</td>
<td>228.4</td>
</tr>
<tr>
<td>NECLA</td>
<td>1011</td>
<td>208</td>
<td>59.3</td>
<td>89</td>
</tr>
<tr>
<td>SNU-RT</td>
<td>1102</td>
<td>790</td>
<td>3167</td>
<td>12.2</td>
</tr>
<tr>
<td>WCET</td>
<td></td>
<td></td>
<td>8.3</td>
<td>81</td>
</tr>
<tr>
<td>POWERSTONE</td>
<td></td>
<td></td>
<td>913.6</td>
<td>1041</td>
</tr>
</tbody>
</table>

*VERISEC and NECLA are not specialized to embedded software*
## Error-Detection Capability

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>Time</th>
<th>#P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enc.</td>
<td>Solver</td>
</tr>
<tr>
<td>VERISEC</td>
<td>9090</td>
<td>2148</td>
<td>190.9</td>
<td>228.4</td>
</tr>
<tr>
<td>NECLA</td>
<td>1011</td>
<td>208</td>
<td>59.3</td>
<td>89</td>
</tr>
<tr>
<td>SNU-RT</td>
<td>3102</td>
<td>790</td>
<td>3167</td>
<td>12.2</td>
</tr>
<tr>
<td>WCET</td>
<td>3430</td>
<td>726</td>
<td>72.7</td>
<td>8.3</td>
</tr>
<tr>
<td>POWERSTONE</td>
<td>2977</td>
<td>2053</td>
<td>127.4</td>
<td>913.6</td>
</tr>
</tbody>
</table>

*Contain ANSI-C constructs commonly found in embedded software*
## Error-Detection Capability

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>Time</th>
<th>#P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enc.</td>
<td>Solver</td>
</tr>
<tr>
<td>VERISEC</td>
<td>9090</td>
<td>2148</td>
<td>190.9</td>
<td>228.4</td>
</tr>
<tr>
<td>NECLA</td>
<td>1011</td>
<td>208</td>
<td>59.3</td>
<td>88</td>
</tr>
<tr>
<td>SNU-RT</td>
<td>3102</td>
<td>790</td>
<td>316</td>
<td>62</td>
</tr>
<tr>
<td>WCET</td>
<td>3430</td>
<td>726</td>
<td>72</td>
<td>22</td>
</tr>
<tr>
<td>POWERSTONE</td>
<td>2957</td>
<td>2053</td>
<td>12</td>
<td>43</td>
</tr>
</tbody>
</table>

*string manipulation, aliasing, dynamic memory allocation, interprocedural dataflow*
Comparison to CBMC [D. Kroening]

- SAT-based BMC for full ANSI-C
  - SMT-based version does not seem to support full ANSI-C
  - mature tool (V3.3.2)
  - not recent SMT-based version
- goal: compare efficiency of CBMC vs. ESBMC
  - on identical verification problems
# Comparison to SAT-CBMC

[D. Kroening]

<table>
<thead>
<tr>
<th>Module</th>
<th>SAT-CBMC</th>
<th>ESBMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>#P</td>
</tr>
<tr>
<td></td>
<td>Fail</td>
<td>Error</td>
</tr>
<tr>
<td>exStbKey</td>
<td>3.1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>exStbHDMI</td>
<td>SF</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>exStbLE</td>
<td>MO</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>exStbHwAcc</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>exStbResolution</td>
<td>TO</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>exStbFb</td>
<td>MO</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>exStbCc</td>
<td>174.5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>exStbDemo</td>
<td>MO</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>267</td>
</tr>
</tbody>
</table>

**Notes:**
- **Segmentation fault**
- **Memory out**
- **Time out**
### Comparison to SAT-CBMC

[D. Kroening]

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>Time</th>
<th>SAT-CBMC</th>
<th>ESBMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#P</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fail</td>
<td>Error</td>
</tr>
<tr>
<td>exStbKey</td>
<td>558</td>
<td>22</td>
<td>3.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>exStbHDMI</td>
<td>1508</td>
<td>41</td>
<td>SF</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>exStbLED</td>
<td>128</td>
<td>59</td>
<td>MO</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>exStbHwAcc</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>exStbResolution</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>exStbFb</td>
<td>18</td>
<td>18</td>
<td>MO</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>exStbCc</td>
<td>331</td>
<td>5</td>
<td>174.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>exStbDemo</td>
<td>30902</td>
<td>267</td>
<td>MO</td>
<td>0</td>
<td>267</td>
</tr>
</tbody>
</table>

Both tools fail to model check the module `exStbDemo`.

---

Both tools fail to model check the module `exStbDemo`. 

---

Both tools fail to model check the module `exStbDemo`.
Comparison to SMT-CBMC [D. Kroening]

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>SMT-CBMC</th>
<th>ESBMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td>#P</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fail</td>
<td>Error</td>
</tr>
<tr>
<td>exStbKey</td>
<td>558</td>
<td>22</td>
<td>3.1</td>
<td>0</td>
</tr>
<tr>
<td>exStbHDMI</td>
<td>1508</td>
<td>41</td>
<td>SF</td>
<td>0</td>
</tr>
<tr>
<td>exStbLED</td>
<td>430</td>
<td>59</td>
<td>TO</td>
<td>0</td>
</tr>
<tr>
<td>exStbHwAcc</td>
<td>1432</td>
<td>115</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>exStbResolution</td>
<td>353</td>
<td>32</td>
<td>TO</td>
<td>0</td>
</tr>
<tr>
<td>exStbFb</td>
<td>689</td>
<td>48</td>
<td>SF</td>
<td>0</td>
</tr>
<tr>
<td>exStbCc</td>
<td>331</td>
<td>5</td>
<td>TO</td>
<td>0</td>
</tr>
<tr>
<td>exStbDemo</td>
<td>30902</td>
<td>267</td>
<td>MO</td>
<td>0</td>
</tr>
</tbody>
</table>
Scalability

• To model check the exStbDemo module, we apply the continuous verification approach
  – we use EmbUnit test framework for dynamic verification
  – we use subversion as SCM system
• goal: apply the CV approach to large embedded software used in a commercial product
## Scalability

<table>
<thead>
<tr>
<th>Module</th>
<th>#TC</th>
<th>#P</th>
<th>SMT-CBMC</th>
<th>Subversion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#P</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fail</td>
<td>Error</td>
</tr>
<tr>
<td>getCommand</td>
<td>18</td>
<td>237</td>
<td>4.4</td>
<td>1</td>
</tr>
<tr>
<td>commandLoop</td>
<td>26</td>
<td>237</td>
<td>128.4</td>
<td>0</td>
</tr>
<tr>
<td>checkEndOfPvStream</td>
<td>4</td>
<td>229</td>
<td>161.2</td>
<td>0</td>
</tr>
<tr>
<td>checkEndOfIPStream</td>
<td>3</td>
<td>228</td>
<td>4.4</td>
<td>0</td>
</tr>
<tr>
<td>checkEndOfMediaStream</td>
<td>4</td>
<td>228</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>setupFrameBuffers</td>
<td>2</td>
<td>228</td>
<td>4.2</td>
<td>0</td>
</tr>
<tr>
<td>setupFBResolution</td>
<td>2</td>
<td>228</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total verification time in seconds for each PR</strong></td>
<td></td>
<td></td>
<td></td>
<td>314.4</td>
</tr>
</tbody>
</table>
## Scalability

<table>
<thead>
<tr>
<th>Module</th>
<th>#TC</th>
<th>#P</th>
<th>SMT-CBMC</th>
<th>Subversion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fail</td>
<td>#P</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>R12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PR13</td>
<td></td>
</tr>
<tr>
<td>getCommand</td>
<td>18</td>
<td>237</td>
<td>4.4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commandLoop</td>
<td>26</td>
<td>237</td>
<td>128.4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkCommandParams</td>
<td>4</td>
<td>229</td>
<td>161.2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkEndOfPvrStream</td>
<td>3</td>
<td>228</td>
<td>4.4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkEndOfIPStream</td>
<td>3</td>
<td>228</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkEndOfMediaStream</td>
<td>4</td>
<td>228</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setupFrameBuffers</td>
<td>2</td>
<td>228</td>
<td>4.2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setupFBResolution</td>
<td>2</td>
<td>228</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total verification time in seconds for each PR</strong></td>
<td>314.4</td>
<td>169.8</td>
<td>169.5</td>
<td>298.1</td>
</tr>
</tbody>
</table>
## Scalability

<table>
<thead>
<tr>
<th>Module</th>
<th>#TC</th>
<th>#P</th>
<th>SMT-CBMC Time</th>
<th>#P</th>
<th>PR10</th>
<th>PR11</th>
<th>PR12</th>
<th>PR13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fail</td>
<td>Error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getCommand</td>
<td>18</td>
<td>237</td>
<td>4.4</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>commandLoop</td>
<td>26</td>
<td>237</td>
<td>128.4</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>checkCommandParams</td>
<td>4</td>
<td>229</td>
<td>161.2</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>checkEndOfPvrStream</td>
<td>3</td>
<td>228</td>
<td>4.4</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>checkEndOfIPStream</td>
<td>3</td>
<td>228</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkEndOfMediaStream</td>
<td>4</td>
<td>228</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setupFrameBuffers</td>
<td>2</td>
<td>228</td>
<td>4.2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setupFBResolution</td>
<td>2</td>
<td>228</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Total verification time in seconds for each PR: 314.4, 169.8, 169.5, 298.1

- Reduces verification time by up to 50%
- But not always
Agenda

• Continuous Verification

• SMT-based Bounded Model Checking

• Case Study and Experimental Results

• Conclusions and Future Work
Conclusions

- introduced **continuous verification** approach
- evaluated on large embedded software
- described a new set of encodings that allow us to reason accurately about embedded software.
  - provided encodings for typical ANSI-C constructs not directly supported by SMT-solvers
- available at [users.ecs.soton.ac.uk/lcc08r/esbmc/](users.ecs.soton.ac.uk/lcc08r/esbmc/)

Future work

- concurrency (based on Pthread library)
- termination analysis