Numerical Static Analysis of Interrupt-Driven Programs via Sequentialization

Xueguang Wu¹  Liqian Chen¹  Antoine Miné²  Wei Dong¹  Ji Wang¹

¹National University of Defense Technology, Changsha, China
²CNRS & LIP6, UPMC, Paris, France

05/10/2015 – EMSOFT 2015
Overview

• Motivation

• Interrupt-driven programs (IDPs)

• Sequentialization of IDPs

• Analysis of sequentialized IDPs via abstract interpretation

• Implementation and experiments

• Conclusion
Interrupts in Embedded Software

• **Interrupts** are a commonly used technique that introduce **concurrency** in embedded software.

• Embedded software may contain intensive **numerical** computations which are **error prone**.
Motivation

• Without considering the interleaving, sequential program analysis results may be **unsound**

```c
int x, y, z;
void TASK(){
    if(x<y){           //❶
        z = 1/(x-y);   //❷
    }
    return;
}

void ISR(){
    x++;            
    y--;            
    return;
}
```

Interrupt semantics:
Given \(x=1,y=3\), if ISR fires at **❶**, there is a division-by-zero error at **❷**

Sequential program analysis: no division-by-zero

**UNSOUND !**
Existing Work

• Sequentialization methods for concurrent programs
  - KISS [PLDI’04], Kidd et al. [SPIN’10], REKH [VMCAI’13], Cseq [ASE’13], …

• Numerical static analysis of concurrent embedded software
  - cXprop [LCTES’06], Monniaux [EMSOFT’07], AstréeA[ESOP’11] …

Few existing numerical static analysis methods consider interrupts
Our Goal

• Challenges of analyzing IDPs
  • interleaving state space can grow exponentially with the number of interrupts (scalability)
  • interrupts are controlled by hardware (precision)
    • e.g., periodic interrupts, interrupt mask register (IMR)

• Goal
  • a sound approach for numerical static analysis of embedded C programs with interrupts
Basic Idea

IDPs → Seq → Sequential Programs

Numerical static analysis via abstract interpretation
Overview

- Motivation
- Interrupt-driven programs (IDPs)
- Sequentialization of IDPs
- Analysis of sequentialized IDPs via abstract interpretation
- Implementation and experiments
- Conclusion
Interrupt-Driven Programs

• Our target interrupt-driven programs (IDPs)
  • an IDP consists of a fixed finite set of tasks and interrupts
  • tasks are scheduled cooperatively, while interrupts are scheduled preemptively by priority

• Application scenarios

Satellite  Wireless network OS  LEGO robotics (OSEK)
Interrupt-Driven Programs

- Model of interrupt-driven programs
  - 1 task + N interrupts
    - each interrupt priority with at most one interrupt
  - only 2 forms of statements accessing shared variables
    - \texttt{l=g}  //read from a shared variable g
    - \texttt{g=l}  //write to a shared variable g

\begin{itemize}
  \item \textit{Expr} := \texttt{l | C | E_1 \circ E_2} (where \texttt{l} \in \textit{NV}, \texttt{C} is a constant, \texttt{E_1, E_2} \in \textit{Expr} and \circ \in \{+,-,\times,\div\})
  \item \textit{Stmt} := \texttt{l = g | g = l} | \texttt{l = e | S_1 ; S_2 | \textbf{skip} | enableISR(i)}
                        | \texttt{disableISR(i) | if e then S_1 else S_2}
                        | \texttt{while e do S}
                        (where \texttt{l} \in \textit{NV}, \texttt{g} \in \textit{SV}, \texttt{e} \in \textit{Expr}, \texttt{i} \in [1,N],
                        \texttt{S_1, S_2, S} \in \textit{Stmt})
  \item \textit{Task} := \texttt{entry} (where \texttt{entry} \in \textit{Stmt})
  \item \textit{ISR} := \texttt{\langle entry, p \rangle} (where \texttt{entry} \in \textit{Stmt}, \texttt{p} \in [1,N])
  \item \textit{Prog} := \texttt{Task || ISR_1 || \ldots || ISR_N}
\end{itemize}
Interrupt-Driven Programs

- Model of interrupt-driven programs
  - 1 task + N interrupts
    - each interrupt priority with at most one interrupt
  - only 2 forms of statements accessing shared variables
    - \( l=g \) //read from a shared variable \( g \)
    - \( g=l \) //write to a shared variable \( g \)

This model simplifies IDPs without losing generality

\[
\text{Stmt} := l = g \mid g = l \mid l = e \mid S_1 ; S_2 \mid \text{skip} \mid \text{enableISR}(i) \\
\mid \text{disableISR}(i) \mid \text{if } e \text{ then } S_1 \text{ else } S_2 \\
\mid \text{while } e \text{ do } S \\
\text{(where } l \in NV, g \in SV, e \in Expr, i \in [1,N], \\
S_1, S_2, S \in \text{Stmt})
\]

\[
\text{Task} := \text{entry} \ (\text{where } \text{entry} \in \text{Stmt})
\]

\[
\text{ISR} := \langle \text{entry} , p \rangle \ (\text{where } \text{entry} \in \text{Stmt}, \ p \in [1,N])
\]

\[
\text{Prog} := \text{Task} \parallel \text{ISR}_1 \parallel \ldots \parallel \text{ISR}_N
\]
Interrupt-Driven Programs

• Assumptions over the model

  1. all accesses to shared variables \((l=g\) and \(g=l\)) are atomic.

  this assumption exists in most of concurrent program analysis, e.g., Cseq [ASE’13], AstréeA[ESOP’11], KISS [PLDI’04]

  2. the IMR is intact inside an ISR, i.e. \(IMR_{ISR_i}^{entry} = IMR_{ISR_i}^{exit}\)

  keeping IMR intact holds for practical IDPs, e.g., satellite control programs
Overview

• Motivation

• Interrupt-driven programs (IDPs)

• Sequentialization of IDPs

• Analysis of sequentialized IDPs via abstract interpretation

• Implementation and experiments

• Conclusion
Basic Idea of Sequentialization

- **Observation:** firing of interrupts can be simulated by function calls

- **Basic idea:** add a `schedule()` function before each (atomic) program statement of the task and interrupts
  - the `schedule()` function non-deterministically schedules higher priority interrupts

Original:

\[ st_1; \ldots; st_k \]

Sequentialized:

\[ st_1'; \ldots; st_k' \]

where \( st_i' = schedule(); st_i \)
Example

```c
int x, y, z;
void task(){
    if(x<y){
        z = 1/(x-y);
    }
    return;
}
void ISR(){
    x++; // Modified
    y--; // Modified
    return; // Modified
}
```

only allow l=g and g = l

```c
int x, y, z;
void task’(){
    int tx, ty;
    tx = x;
    ty = y;
    if(tx < ty){
        tx = x;
        ty = y;
        z = 1/(tx-ty);
    }
    return;
}
void ISR’(){
    int tx, ty;
    tx = x;
    tx = tx + 1;
    x = tx;
    ty = y;
    ty = ty + 1;
    y = ty;
    return;
}
```
# Example

```c
int x, y, z;
void task() {
    if (x < y) {
        z = 1 / (x - y);
    }
    return;
}

void ISR() {
    x++;
    y--;
    return;
}

void task_seq() {
    int tx, ty;
    schedule(); tx = x;
    schedule(); ty = y;
    schedule();
    if (tx < ty) {
        schedule(); tx = x;
        schedule(); ty = y;
        schedule();
        z = 1 / (tx - ty);
    }
    schedule(); return;
}

void ISR_seq() {
    int tx, ty;
    schedule(); tx = x;
    schedule(); tx = tx + 1;
    schedule(); x = tx;
    schedule(); ty = y;
    schedule(); ty = ty + 1;
    schedule(); y = ty;
    schedule(); return;
}

void schedule() {
    int prevPrio = Prio;
    for (int i <= 1; i <= N; i++) {
        if (i <= Prio) continue;
        if (nondet()) {
            Prio = i;
            ISR_seq[i].entry();
        }
    }
    Prio = prevPrio;
}
```

Add schedule() before each program statement.
Example

```c
int x, y, z;
void task(){
    if(x<y){
        z = 1/(x-y);
    }
    return;
}
void ISR(){
    x++;  // Non-deterministically schedule higher priority interrupts
    y--;  
    return ;
}

int Prio=0;
//current priority
ISR ISRseqs[N];  
//ISR entry
void task_seq(){
    int tx, ty;
    schedule(); tx = x;
    schedule(); ty = y;
    schedule();
    if(tx < ty){
        schedule(); tx = x;
        schedule(); ty = y;
    }
    schedule(); return ;
}

void schedule(){
    int prevPrio = Prio;
    for(int i<=1;i<=N;i++){
        if(i<=Prio) continue;
        if(nondet()){
            Prio = i;
            ISRseqs[i].entry();
        }
    }
    Prio = prevPrio;

void ISR_seq(){
    int tx, ty;
    schedule();tx = x;
    schedule(); tx = tx + 1;
    schedule();x = tx;
    schedule(); ty = y;
    schedule(); ty = ty + 1;
    schedule(); y = ty;
    schedule(); return ;
}
```

Basic Idea of Sequentialization

• The disadvantage of the basic sequentialization method
  • the resulting sequentialized program becomes large
    • too many schedule() functions are invoked

• Further observation
  • interrupts and tasks communicate with each other by shared variables
    • interrupts only affect those statements which access shared variables

Further idea: utilize data flow dependency to reduce the size of sequentialized programs
Sequentialization by Considering Data Flow Dependency

- Example: Program \{ St_1; St_2; \ldots; St_n \}, where only St_n reads shared variables (SVs)

Basic Sequentialization

\[
\{ \text{schedule(); } St_1; \text{schedule(); } St_2; \ldots; \text{schedule(); } St_n \}\]

Consider SVs

\[
\{ St_1; St_2; \ldots; St_{n-1}; \text{for(int i=0; i<K; i++)}\}
\text{Schedule(); }
St_n
\}
\]
Sequentialization by Considering Data Flow Dependency

• Key idea: schedule **relevant** interrupts only for those statements **accessing shared variables**
  
  • before \( l = g \) (i.e., reading a shared variable)
    
    • schedule those interrupts which may affect the value of shared variable \( g \)
  
  • after \( g = l \) (i.e., writing a shared variable)
    
    • schedule those interrupts of which the execution results may be affected by shared variable \( g \)
Sequentialization by Considering Data Flow Dependency

• Need to consider the firing number of interrupts, otherwise the analysis results may be not sound

```c
void scheduleG_K(group: int set){
    for(int i=1;i<=K;i++)
        scheduleG(group);
}
```

K is the upper bound of the firing times of each ISR, which can be a specific value or +∞
Example

```c
int x, y, z;
void task()
{
    int t, tx, ty, tz;
    x = 10;
    y = 0;
    tx = x;
    ty = y;
    t = tx + ty;
    ty = y;
    tx = t - ty;
    x = tx;
    tz = t * 2;
    z = tz;
    ty = y;
    ty = t - ty;
    y = ty;
}
void ISR1()
{
    int tx, ty;
    ty = y;
    ty = ty + 1; y = ty;
    tx = x;
    tx = tx - 1; x = tx;
}
void ISR2()
{
    int tz;
    tz = z;
    tz = tz + 1; z = tz;
}
```

These statements access shared variables
Example

```c
int x, y, z;
void task(){
    int t, tx, ty, tz;
    x = 10; scheduleG_K({1});
    y = 0; scheduleG_K({1});
    tx = x; ty = y;
    t = tx+ty;
    ty=y;
    tx = t-ty;
    x = tx;
    tz = t*2;
    z = tz;
    ty = y;
    ty = t-ty;
    y = ty;
}
void ISR1(){
    int tx, ty;
    ty = y; ty = ty + 1; y = ty;
    tx = x; tx = tx -1; x = tx;
}
void ISR2(){
    int tz;
    tz = z; tz = tz+1; z=tz;
}
```

only invoke `scheduleG_K()` before reading or after writing SVs

```c
int x, y, z;
void task(){
    int t, tx, ty, tz;
    x = 10; scheduleG_K({1});
    y = 0; scheduleG_K({1});
    tx = x; ty = y;
    t = tx+ty;
    ty=y;
    tx = t-ty;
    x = tx;
    tz = t*2;
    z = tz;
    ty = y;
    ty = t-ty;
    y = ty;
}
void ISR1_seq(){
    //Same as ISR1
}
void ISR2_seq(){
    //Same as ISR2
    //scheduleG_K({1}) gives:
    for(int i=0;i<K;i++)
        if(nondet()) ISR1_seq();
    //scheduleG_K({2}) gives:
    for(int i=0;i<K;i++)
        if(nondet()) ISR2_seq();
```
int x, y, z;
void task()
{
    int t, tx, ty, tz;
    x = 10;       // Example
    y = 0;
    tx = x;
    ty = y;
    t = tx + ty;
    ty = y;
    tx = t - ty;
    x = tx;
    tz = t * 2;
    z = tz;
    ty = y;
    ty = t - ty;
    y = ty;
}

void ISR1()
{
    int tx, ty;
    ty = y;         // only invoke relevant ISRs
    ty = ty + 1;    // Seq
    y = ty;
    tx = x;
    tx = tx - 1;    // scheduleG_K({1})
    x = tx;
}

void ISR2()
{
    int tz;
    tz = z;
    tz = tz + 1;    // scheduleG_K({2})
    z = tz;
}
Overview

• Motivation
• Interrupt-driven programs (IDPs)
• Sequentialization of IDPs
• Analysis of sequentialized IDPs via abstract interpretation
• Implementation and experiments
• Conclusion
Analysis of Sequentialized IDPs via Abstract Interpretation

IDPs

Seq

Sequential Programs

Numerical static analysis via abstract interpretation
Analysis of Sequentialized IDPs via Abstract Interpretation

• Analysis of sequentialized IDPs
  • using generic numerical abstract domains

• Need to consider **specific** features of sequentialized IDPs
  • firing number of interrupts affects the analysis result
  • interrupts with period

Need specific abstract domains to consider interrupt features
A Specific Abstract Domain for IDPs

• At-most-once firing periodic interrupts
  • periodic interrupts: firing with a fixed time interval
  • the period of interrupts is larger than one task period

• An abstract domain for at-most-once firing periodic interrupts
  • use boolean flag variables to distinguish whether ISRs have happened or not
A Specific Abstract Domain for IDPs

• Example of boolean flag abstract domain

```c
int x;
void task(){
    void ISR1(){
        int tx,z;
        int tx;
        x=0;
        tx = x;
        tx = tx+10;
        x = tx;
        tx=tx+1;
        tx=tx+1;
        x=tx;
        } } /* xnf ∈ [0,0], xf ∈ [0,0] */
    ISR1 hasn’t fired
    ISR1 has fired
    z=1/(x-5);
    } } /* xnf ∈ [0,0], xf ∈ [10,10] */
    ISR1 hasn’t fired
    ISR1 has fired
    x=tx;
    } } /* xnf ∈ [1,1], xf ∈ [11,11] */
    ISR1 hasn’t fired
    ISR1 has fired
    z=1/(x-5);
    } } /* division is safe */
```

If only using interval domain: x ∈ [1,21] and there will be a division by zero false alarm
Overview

- Motivation
- Interrupt-driven programs (IDPs)
- Sequentialization of IDPs
- Analysis of sequentialized IDPs via abstract interpretation
- Implementation and experiments
- Conclusion
Implementation and Experiments

• Implementation
  • frontend: CIL
  • numerical abstract domain library: Apron

• Benchmarks
  • OSEK programs from Goblint [Schwarz et al. POPL 11]
  • LEGO robotic control program (Nxt_gs)
  • universal asynchronous receive and transmitter (UART)
    • ping pong buffer program from satellite application program
    • ADC controller from satellite application program
    • a satellite control program
Implementation and Experiments

• Aims of the experiments
  • check run time errors of IDPs
  • compare the generated program size and the time consumption of sequentialization methods with and without considering data flow dependency
  • compare the scalability and precision of numerical static analysis for sequentialization methods with and without considering data flow dependency
### Implementation and Experiments

- Experiments of sequentialization

<table>
<thead>
<tr>
<th>Program</th>
<th>Name</th>
<th>Loc_task</th>
<th>Loc_ISR</th>
<th>#Vars</th>
<th>#ISR</th>
<th>SEQ LOC</th>
<th>SEQ Time (s)</th>
<th>DF_SEQ LOC</th>
<th>DF_SEQ Time (s)</th>
<th>DF_SEQ Q/SEQ (%LOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Motv_Ex</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>158</td>
<td>0.004</td>
<td>134</td>
<td>0.006</td>
<td>84.81</td>
</tr>
<tr>
<td></td>
<td>DataRace_Ex</td>
<td>20</td>
<td>40</td>
<td>9</td>
<td>2</td>
<td>385</td>
<td>0.004</td>
<td>242</td>
<td>0.005</td>
<td>62.86</td>
</tr>
<tr>
<td></td>
<td>Privatize</td>
<td>25</td>
<td>37</td>
<td>7</td>
<td>2</td>
<td>393</td>
<td>0.006</td>
<td>168</td>
<td>0.004</td>
<td>42.75</td>
</tr>
<tr>
<td></td>
<td>Nxt_gs</td>
<td>23</td>
<td>154</td>
<td>27</td>
<td>1</td>
<td>1199</td>
<td>0.005</td>
<td>552</td>
<td>0.006</td>
<td>46.04</td>
</tr>
<tr>
<td></td>
<td>UART</td>
<td>129</td>
<td>15</td>
<td>47</td>
<td>1</td>
<td>5940</td>
<td>0.010</td>
<td>1215</td>
<td>0.010</td>
<td>20.45</td>
</tr>
<tr>
<td></td>
<td>PingPong_Sate</td>
<td>130</td>
<td>53</td>
<td>21</td>
<td>1</td>
<td>3159</td>
<td>0.006</td>
<td>842</td>
<td>0.006</td>
<td>26.65</td>
</tr>
<tr>
<td></td>
<td>ADC_Sate</td>
<td>1870</td>
<td>2989</td>
<td>312</td>
<td>1</td>
<td>123K</td>
<td>0.449</td>
<td>23K</td>
<td>0.8</td>
<td>18.70</td>
</tr>
<tr>
<td></td>
<td>Satellite_Control</td>
<td>33885</td>
<td>1227</td>
<td>1352</td>
<td>1</td>
<td>10M</td>
<td>16.1</td>
<td>534K</td>
<td>1.6</td>
<td>5.34</td>
</tr>
</tbody>
</table>

The scale of sequentialized program by DF_SEQ is smaller than SEQ.
## Implementation and Experiments

- Experiment of numerical static analysis

<table>
<thead>
<tr>
<th>Program</th>
<th>Analysis of SEQ (s)</th>
<th>Analysis of DF_SEQ (s)</th>
<th>Warnings &amp; Proved Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>BOX</td>
<td>OCT</td>
<td>BOX</td>
</tr>
<tr>
<td>Motv_Ex</td>
<td>0.007</td>
<td>0.011</td>
<td>0.006</td>
</tr>
<tr>
<td>DataRace_Ex</td>
<td>0.042</td>
<td>0.053</td>
<td>0.011</td>
</tr>
<tr>
<td>Privatize</td>
<td>0.029</td>
<td>0.036</td>
<td>0.005</td>
</tr>
<tr>
<td>Nxt_gs</td>
<td>0.113</td>
<td>0.140</td>
<td>0.040</td>
</tr>
<tr>
<td>UART</td>
<td>0.732</td>
<td>5.782</td>
<td>0.128</td>
</tr>
<tr>
<td>Ping_Pong</td>
<td>0.429</td>
<td>2.434</td>
<td>0.054</td>
</tr>
<tr>
<td>ADC_Sate</td>
<td>MemOut</td>
<td>MemOut</td>
<td>80.5</td>
</tr>
<tr>
<td>Satellite Control</td>
<td>MemOut</td>
<td>MemOut</td>
<td>5190</td>
</tr>
</tbody>
</table>

Precision of SEQ&DF_SEQ is the same and the scalability of DF_SEQ is much better.
Overview

• Motivation

• Interrupt-driven programs

• Sequentialization of IDPs

• Analysis of sequentialized IDPs via abstract interpretation

• Implementation and experiments

• Conclusion
Conclusion

- **Contribution:** a *sound* approach for numerical static analysis of embedded C software with interrupts

![Diagram](image-url)
Conclusion

• **Contribution:** a sound approach for numerical static analysis of embedded C software with interrupts

![Diagram](image-url)

- IDPs
  - Seq
  - Sequential Programs

- a simple model with restrictions and assumptions
- static analysis explicit interpretation
Conclusion

• **Contribution:** a sound approach for numerical static analysis of embedded C software with interrupts

```
IDPs
```

```
Seq
```

```
Sequential Programs
```

consider data flow dependency to sequentialize IDPs (scalability)
Conclusion

• **Contribution:** a *sound* approach for numerical static analysis of embedded C software with interrupts

- *IDPs*
- *Seq*
- *Sequential Programs*
- a specific abstract domain for sequentialized IDPs (precision)

- Numerical static analysis via abstract interpretation
Conclusion

• Future work
  • extending the model to support IDPs with tasks preemption tasks
  • designing more specific abstract domains that fit IDPs

IDPs

Seq

Sequential Programs

Numerical static analysis via abstract interpretation
Thank you
Any Questions?