Handling Overflow in MLton

2012 ACM SIGPLAN Workshop on ML

Alexander Bjerremand Hansen ¹
Jan Midtgaard ²

¹ Mjølner Informatics A/S
² Aarhus University

September 13, 2012
The MLton compiler

- MLton is an Standard ML compiler
  - Open source
  - Industrial strength (Intel, MathWorks, etc.)
  - Whole-program
  - Optimizing

- It has been developed over the last 15 years
Handling overflow

In an intermediate representation integer arithmetic can take the form:

\[ \ell(x_1 + x_2) \quad \text{Overflow} \Rightarrow \ell' \]

If \( x_1 + x_2 \) does not overflow this could be optimized to

\[ x \leftarrow \text{Word32_add}(x_1, x_2) \]
\[ \ell(x) \]
Handling overflow

In an intermediate representation integer arithmetic can take the form:

\[ \ell(x_1 + x_2) \text{ Overflow} \Rightarrow \ell' \]

If \( x_1 + x_2 \) does not overflow this could be optimized to

\[ x \leftarrow \text{Word32_add}(x_1, x_2) \]

\[ \ell(x) \]

Bad news!
Handling overflow

In an intermediate representation integer arithmetic can take the form:

\[ \ell(x_1 + x_2) \text{ Overflow } \Rightarrow \ell' \]

If \( x_1 + x_2 \) does not overflow this could be optimized to

\[ x \leftarrow \text{Word32_add}(x_1, x_2) \]
\[ \ell(x) \]

Bad news!
MLton produces sub-optimal code regarding overflow checks.
Dead code - example

When compiling hamlet.sml we find this snippet among the 110k lines of intermediate code:

```plaintext
L_25120 (x_16013: word32)
  x_10771: bool = WordU32_lt (x_10769, global_33)
  case x_10771 of
    true => L_22171 | false => L_24271
L_22171 ()
  loop_355 (x_10769 + global_5) Overflow => L_24272()
```

Relevant constants:

```
global_5:  word32 = 0x1
global_33: word32 = 0x3B7
```
Dead code - example

The same code typeset as a flow graph:

\[
x_{10771} : \text{bool} = x_{10769} < \text{global}_33
\]

\[
\text{case } x_{10771} \text{ of }
\]

\[
\text{true} \\
\]

\[
x_{10769} + \text{global}_5
\]

\[
\text{false} \\
\]

\[
\text{Overflow}
\]

\[
\text{raise Overflow}_0
\]

(on 32 bit)
Dead code - example

With global constants inlined:

```plaintext
x_10771: bool = x_10769 < 951

case x_10771 of
  true: x_10769 + 1
  false: raise Overflow_0

(overflow on 32 bit)
```

Alexander Bjerremand Hansen (alx@cs.au.dk) 6/29
Dead code - example

With variables $\alpha$-renamed:

\[
y : \text{bool} = x < 951
\]

\[
\text{case } y \text{ of}
\]

\[
\begin{align*}
\text{true} & \rightarrow \ x + 1 \\
\text{false} & \rightarrow \ \text{Overflow}
\end{align*}
\]

raise Overflow_0

(on 32 bit)
Motivation

Bad news!
MLton produces sub-optimal code regarding overflow checks.

The question is: Can we formulate a static analysis to improve the generated code?
Motivation

Bad news! MLton produces sub-optimal code regarding overflow checks.

The question is: Can we formulate a static analysis to improve the generated code?

Good news!
Motivation

Bad news!
MLton produces sub-optimal code regarding overflow checks.

The question is: Can we formulate a static analysis to improve the generated code?

Good news!
Yes we can!
Agenda

- Motivation
- MLton in more detail
- Detecting dead code in integer arithmetic with interval analysis
- Analysis
- Results
MLton in more detail

Source → AST → CoreML

Lexing & parsing → Elaborate & defunctorize

Match compilation & datatype lifting

Monomorphize

XML → SXML

Closure conversion

ToSSA

SSA

ToSSA2

ToRSSA

RSSA

ToMachine

Machine
MLton in more detail - SSA form

- Static Single-Assignment form
- Each variable has one definition in the program text
  - Static property
- Simply-typed
  - Polymorphism eliminated by monomorphization
- First-order
  - Higher order functions eliminated by defunctionalization (closure conversion)
- Most optimizations at SSA level
Interval Analysis

The standard way of analysing integer variables is Interval/Range analysis as proposed by Cousot and Cousot [1976].

Lattice of intervals (IL):

\[
\top = \langle \text{MIN}, \text{MAX} \rangle \\
\langle \text{MIN}, \text{MAX} - 1 \rangle \quad \langle \text{MIN} + 1, \text{MAX} \rangle \\
\langle \text{MIN}, \text{MAX} - 2 \rangle \quad \langle \text{MIN} + 1, \text{MAX} - 1 \rangle \quad \langle \text{MIN} + 2, \text{MAX} \rangle \\
\vdots \quad \vdots \quad \vdots \\
\langle \text{MIN}, \text{MIN} \rangle \ldots \langle -1, -1 \rangle \quad \langle 0, 0 \rangle \quad \langle 1, 1 \rangle \ldots \langle \text{MAX}, \text{MAX} \rangle \\
\bot
\]

Lattice relating variables with intervals (I):

\[ I : \text{Var} \rightarrow IL \]
Initial lookup function

When we need the abstract value of a variable $x$ in block $\ell$:

$$L(x, \ell) = l[x]$$

This version ignores the block and returns the value from the $l$ lattice.
Constraints

We express the analysis using constraints. Examples:

\[ (X \leftarrow v) \in P \quad v : \text{word} \]
\[ I[X] = \langle v, v \rangle \]

Constant Definition

\[ \ell(X) \in P \quad \ell(x_1 \text{ op } x_2) = S \]
\[ S \in P \quad \text{label}(S) = \ell' \]
\[ (L(x_1, \ell') \circ_{\text{op}} L(x_2, \ell')) \sqsubseteq I[X] \]

Arithmetic

\[ f(X_1, \ldots, X_n) \in P \quad \ell'(X_r) \in P \]
\[ (f(x_1, \ldots, x_n) \text{ NonTail \{cont=\ell', handler=H\}}) = S \]
\[ S \in P \quad i \in \{1, \ldots, n\} \quad \text{label}(S) = \ell'' \]
\[ x_i : \text{word} \]
\[ L(x_i, \ell'') \sqsubseteq I[X_i] \]

NonTail Call
Constraints

We express the analysis using constraints. Examples:

\[
\begin{align*}
(X \leftarrow v) \in P & \quad \Rightarrow \\
I[X] = \langle v, v \rangle & \quad \text{Constant Definition}
\end{align*}
\]

\[
(L(x_1, \ell') \circ p L(x_2, \ell')) \subseteq I[X] \quad \text{Arithmetic}
\]

\[
(f(X_1, \ldots, X_n) \in P \quad \ell'(X_i) \in P)
\]

\[
(f(x_1, \ldots, x_n) \text{ NonTail } \{\text{cont}=\ell', \text{handler}=H\}) = S
\]

\[
S \in P \quad \Rightarrow \\
i \in \{1, \ldots, n\} \quad \Rightarrow \\
\text{label}(S) = \ell''
\]

\[
X_i : \text{word}
\]

\[
L(x_i, \ell'') \subseteq I[X_i] \quad \text{NonTail Call}
\]

Alexander Bjerremand Hansen (alx@cs.au.dk) 14/29
What do we need? 1/2

We need conditional information from boolean comparisons of integers.

\[
y: \text{bool} = x < 951 \\
\text{case } y \text{ of }
\]

\[
\text{true}
\]

\[
x + 1
\]

\[
\text{false}
\]

Overflow

raise Overflow_0
What do we need? 2/2

The information from the comparison must be saved for later utilization.

```
y:bool = x < 951
...
```

```
case y of
  true
  false
```

```
x + 1
```

```
raise Overflow_0
```

```
Overflow
```

Alexander Bjerremand Hansen (alx@cs.au.dk) 16/29
Flow sensitivity - 1/2

\( A_{T,y}[X] \) is the interval for \( y \) if \( X \) is true
Flow sensitivity - 1/2

$$A_{T,y}[X]$$ is the interval for $$y$$ if $$X$$ is true

$$\prec$$ and $$\succeq$$ bound intervals from above or below.
Flow sensitivity - 1/2

$A_{T,y}[X]$ is the interval for $y$ if $X$ is true

$\prec$ and $\succeq$ bound intervals from above or below.

\[
\begin{align*}
(X \leftarrow \text{Word}_{-}\text{lt}(x_1, x_2)) &= S \quad S \in P \\
X : \text{bool} &\quad \text{label}(S) = \ell \\
A_{T,x_1}[X] &= L(x_1, \ell) \prec L(x_2, \ell) \\
A_{T,x_2}[X] &= L(x_2, \ell) \succeq L(x_1, \ell) \\
A_{F,x_1}[X] &= L(x_1, \ell) \succeq L(x_2, \ell) \\
A_{F,x_2}[X] &= L(x_2, \ell) \prec L(x_1, \ell)
\end{align*}
\]

Primitive Definition - Less-Than

Alexander Bjerremand Hansen (alx@cs.au.dk) 17/29
A fact lattice $F$ stores optional local extra information for variables.

$$F : B \to Var \to IL$$
Flow sensitivity - 2/2

A fact lattice $F$ stores optional local extra information for variables.

$$F : B \rightarrow Var \rightarrow IL$$

A case transfer can propagate the four intervals down through the control paths:

$$(\text{case } x \text{ of true} => \ell \mid \text{false} => \ell') \in P \quad x : \text{bool}$$

\[
\begin{align*}
A_{T,y}[x] &= i & A_{T,z}[x] &= i' \\
A_{F,y}[x] &= i'' & A_{F,z}[x] &= i''' \\
\text{Case Transfer - Bool}
\end{align*}
\]

\[
\begin{align*}
i &\sqsubseteq F[\ell][y] & i' &\sqsubseteq F[\ell][z] \\
i'' &\sqsubseteq F[\ell'][y] & i''' &\sqsubseteq F[\ell'][z]
\end{align*}
\]
Improved lookup function

Search the ancestor tree for facts about the input variable first.
Let $\text{anc}(\ell)$ be the ancestor to the block $\ell$.

$$L(x, \ell) = \begin{cases} F[\ell][x] & \text{, if } F[\ell][x] \neq \bot \\ I[x] & \text{, if } F[\ell][x] = \bot \land \text{anc}(\ell) = \emptyset \\ L(x, \text{anc}(\ell)) & \text{, otherwise} \end{cases}$$
The implementation

We have implemented the analysis and transformation as an additional SSA optimization pass in MLton.

It takes approximately 1500 lines of SML.

The pass is followed by the pass RemoveUnused which takes care of removing potential dead blocks after our transformation.
Transforms the simplified, $\alpha$-renamed hamlet.sml example from earlier:

```
L
  y: bool = WordU32_lt (x, 951)
  case y of
    true => L' | false => ...
L'
  L'' (x + 1) Overflow => L'''' ()
```

into

```
L
  y: bool = WordU32_lt (x, 951)
  case y of
    true => L' | false => ...
L'
  z: word32 = Word32_add (x, 1)
  L'' (z)
```
Benchmarks - Hits

Benchmarks are from fib to hamlet.

Using the MLton benchmark suite:

<table>
<thead>
<tr>
<th>Handlers</th>
<th>Transformed</th>
</tr>
</thead>
<tbody>
<tr>
<td>4240</td>
<td>913</td>
</tr>
</tbody>
</table>

So around 22% of the overflow-checking arithmetic in the benchmark suite is transformed into primitive operations.
## Benchmarks - Size 1/2

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>without (KB)</th>
<th>with (KB)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>mandelbrot</td>
<td>5</td>
<td>117,383</td>
<td>104,643</td>
<td>10.9%</td>
</tr>
<tr>
<td>flat-array</td>
<td>4</td>
<td>117,119</td>
<td>104,631</td>
<td>10.7%</td>
</tr>
<tr>
<td>zern</td>
<td>32</td>
<td>141,925</td>
<td>140,501</td>
<td>1.0%</td>
</tr>
<tr>
<td>barnes-hut</td>
<td>25</td>
<td>158,891</td>
<td>157,407</td>
<td>0.9%</td>
</tr>
<tr>
<td>smith-normal-form</td>
<td>53</td>
<td>249,054</td>
<td>247,326</td>
<td>0.7%</td>
</tr>
<tr>
<td>fft</td>
<td>12</td>
<td>136,160</td>
<td>135,264</td>
<td>0.7%</td>
</tr>
<tr>
<td>raytrace</td>
<td>69</td>
<td>308,684</td>
<td>307,084</td>
<td>0.5%</td>
</tr>
</tbody>
</table>
Total reduction of the binary sizes in the 43 benchmarks is 52 kB or 0.6% of the total size.

Average reduction: 1.2 kB or 0.85%.

The reduction ranges from 80 B to 12.7 kB.

Every benchmark has reduced binary size.
## Benchmarks - Run-time 1/2

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>without (s)</th>
<th>with (s)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix-multiply</td>
<td>8</td>
<td>6.25</td>
<td>4.56</td>
<td>-27%</td>
</tr>
<tr>
<td>imp-for</td>
<td>4</td>
<td>8.88</td>
<td>6.55</td>
<td>-26%</td>
</tr>
<tr>
<td>tailfib</td>
<td>5</td>
<td>10.54</td>
<td>8.45</td>
<td>-20%</td>
</tr>
<tr>
<td>mandelbrot</td>
<td>5</td>
<td>8.70</td>
<td>8.20</td>
<td>-6%</td>
</tr>
<tr>
<td>md5</td>
<td>20</td>
<td>11.65</td>
<td>11.38</td>
<td>-2%</td>
</tr>
<tr>
<td>zern</td>
<td>32</td>
<td>6.03</td>
<td>5.89</td>
<td>-2%</td>
</tr>
</tbody>
</table>
## Benchmarks - Run-time 1/2

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>without (s)</th>
<th>with (s)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix-multiply</td>
<td>8</td>
<td>6.25</td>
<td>4.56</td>
<td>-27%</td>
</tr>
<tr>
<td>imp-for</td>
<td>4</td>
<td>8.88</td>
<td>6.55</td>
<td>-26%</td>
</tr>
<tr>
<td>tailfib</td>
<td>5</td>
<td>10.54</td>
<td>8.45</td>
<td>-20%</td>
</tr>
<tr>
<td>mandelbrot</td>
<td>5</td>
<td>8.70</td>
<td>8.20</td>
<td>-6%</td>
</tr>
<tr>
<td>md5</td>
<td>20</td>
<td>11.65</td>
<td>11.38</td>
<td>-2%</td>
</tr>
<tr>
<td>zern</td>
<td>32</td>
<td>6.03</td>
<td>5.89</td>
<td>-2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>without (s)</th>
<th>with (s)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>hamlet</td>
<td>41</td>
<td>9.60</td>
<td>9.59</td>
<td>0%</td>
</tr>
<tr>
<td>lexgen</td>
<td>29</td>
<td>5.46</td>
<td>5.45</td>
<td>0%</td>
</tr>
</tbody>
</table>
## Benchmarks - Run-time 1/2

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>without (s)</th>
<th>with (s)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix-multiply</td>
<td>8</td>
<td>6.25</td>
<td>4.56</td>
<td>-27%</td>
</tr>
<tr>
<td>imp-for</td>
<td>4</td>
<td>8.88</td>
<td>6.55</td>
<td>-26%</td>
</tr>
<tr>
<td>tailfib</td>
<td>5</td>
<td>10.54</td>
<td>8.45</td>
<td>-20%</td>
</tr>
<tr>
<td>mandelbrot</td>
<td>5</td>
<td>8.70</td>
<td>8.20</td>
<td>-6%</td>
</tr>
<tr>
<td>md5</td>
<td>20</td>
<td>11.65</td>
<td>11.38</td>
<td>-2%</td>
</tr>
<tr>
<td>zern</td>
<td>32</td>
<td>6.03</td>
<td>5.89</td>
<td>-2%</td>
</tr>
<tr>
<td>hamlet</td>
<td>41</td>
<td>9.60</td>
<td>9.59</td>
<td>0%</td>
</tr>
<tr>
<td>lexgen</td>
<td>29</td>
<td>5.46</td>
<td>5.45</td>
<td>0%</td>
</tr>
<tr>
<td>vector-rev</td>
<td>5</td>
<td>9.08</td>
<td>9.42</td>
<td>4%</td>
</tr>
<tr>
<td>tak</td>
<td>4</td>
<td>6.16</td>
<td>6.69</td>
<td>9%</td>
</tr>
</tbody>
</table>
Average reduction in run-time: 1.8%.
## Compile time

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>CT without</th>
<th>CT with</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>hamlet</td>
<td>41</td>
<td>10.42</td>
<td>29.57</td>
<td>184%</td>
</tr>
<tr>
<td>mlyacc</td>
<td>79</td>
<td>5.92</td>
<td>14.29</td>
<td>141%</td>
</tr>
<tr>
<td>model-elimination</td>
<td>67</td>
<td>5.44</td>
<td>9.21</td>
<td>70%</td>
</tr>
<tr>
<td>vliw</td>
<td>44</td>
<td>4.16</td>
<td>6.75</td>
<td>62%</td>
</tr>
<tr>
<td>raytrace</td>
<td>69</td>
<td>3.31</td>
<td>4.20</td>
<td>27%</td>
</tr>
</tbody>
</table>

Average increase in compile time: 15%.}

Alexander Bjerremand Hansen (alx@cs.au.dk) 27/29
Compile time

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>CT without</th>
<th>CT with</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>hamlet</td>
<td>41</td>
<td>10.42</td>
<td>29.57</td>
<td>184%</td>
</tr>
<tr>
<td>mlyacc</td>
<td>79</td>
<td>5.92</td>
<td>14.29</td>
<td>141%</td>
</tr>
<tr>
<td>model-elimination</td>
<td>67</td>
<td>5.44</td>
<td>9.21</td>
<td>70%</td>
</tr>
<tr>
<td>vliw</td>
<td>44</td>
<td>4.16</td>
<td>6.75</td>
<td>62%</td>
</tr>
<tr>
<td>raytrace</td>
<td>69</td>
<td>3.31</td>
<td>4.20</td>
<td>27%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># hits</th>
<th>CT without</th>
<th>CT with</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>zern</td>
<td>32</td>
<td>2.08</td>
<td>2.16</td>
<td>4%</td>
</tr>
<tr>
<td>nucleic</td>
<td>4</td>
<td>3.38</td>
<td>3.49</td>
<td>3%</td>
</tr>
<tr>
<td>tsp</td>
<td>19</td>
<td>2.06</td>
<td>2.12</td>
<td>3%</td>
</tr>
<tr>
<td>knuth-bendix</td>
<td>25</td>
<td>2.28</td>
<td>2.33</td>
<td>2%</td>
</tr>
</tbody>
</table>

Average increase in compile time: 15%.
Conclusion

- With relatively straightforward application of static analysis we improve the generated code by MLton.

- Our improvement:
  - 22% of overflow checks transformed
  - 0%-10% reduced binary size
  - Up to 27% reduction in run-time.

- There is still room for improvement though:
  - in the choice of data structures.
  - in analysis precision.
  - in the compile-time for the analysis.

Thank you.

Alexander Bjerremand Hansen (alx@cs.au.dk)
Conclusion

- With relatively straightforward application of static analysis we improve the generated code by MLton.
- Our improvement:
  - 22% of overflow checks transformed
  - 0%-10% reduced binary size
  - Up to 27% reduction in run-time.
- There is still room for improvement though:
  - in the choice of data structures.
  - in analysis precision.
  - in the compile-time for the analysis.

Master’s Thesis:
http://cs.au.dk/~alx/
Conclusion

- With relatively straightforward application of static analysis we improve the generated code by MLton.
- Our improvement:
  - 22% of overflow checks transformed
  - 0%-10% reduced binary size
  - Up to 27% reduction in run-time.
- There is still room for improvement though:
  - in the choice of data structures.
  - in analysis precision.
  - in the compile-time for the analysis.

Master’s Thesis:
http://cs.au.dk/~alx/

Thank you.
Widening

When a join of intervals coarsen the bounds widening is performed.

Widening consist of jumping the bounds of the resulting interval to the nearest integer from a set $B$.

\[ w(l, h) = \langle \max\{x \in B | x \leq l\}, \min\{x \in B | x \geq h\} \rangle \]

$B$ is all integer constants in the program.