Static Analysis of Endian Portability by Abstract Interpretation

SAS 2021

David Delmas[1,2] Abdelraouf Ouadjaout[2] Antoine Miné[2]

[1]Airbus – Avionics Software

[2]Sorbonne Université – LIP6

17 October 2021

MOPSA

Endianness

No consensus

Representation of multi-byte scalar values in memory

- Little-endian systems
  - least-significant byte at lowest address
  - Intel processors
- Big-endian systems
  - least-significant byte at highest address
  - internet protocols, legacy or embedded processors (e.g. SPARC, PowerPC)

Which bit should travel first? The bit from the big end or the bit from the little end? Can a war between Big Endians and Little Endians be avoided?

On Holy Wars and a Plea for Peace

Danny Cohen
Information Sciences Institute

This article was written in an attempt to stop a war. I hope it is not too late for peace to prevail again. Many believe that the central question of this war is, What is the proper byte order in messages? More specifically, the question is, Which bit should travel first—the bit from the little end of the word or the bit from the big end of the word?

Followers of the former approach are called Little Endians, or Lilliputians; followers of the latter are called Big Endians, or Blefuscuians. I employ these Swiftian terms because this modern conflict is so reminiscent of the holy war described in *Gulliver's Travels.*[1]

Approaches to serialization

The above question arises as a result of the serialization process performed on messages to allow them to be sent through communication media. If the unit of communication is a message, this question has no meaning. If the units are computer words, one must determine their size and the order in which they are sent.

Since they are sent virtually at once, there is no need to determine the order of the elements of these words.

If the unit of transmission is an eight-bit byte, questions about bytes are meaningful but questions about the order of the elementary particles that constitute these bytes are not.

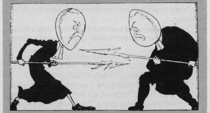
If the units of communication are bits, the atoms (quarks?) of computation, the only meaningful question concerns the order in which the bits are sent. Most modern communication is based on a single stream of information, the bit-stream. Hence, bits, rather than bytes or words, are the units of information that are actually transmitted over channels such as wires and satellites.

Notes on Swift's *Gulliver's Travels*

Swift's hero, Gulliver, is shipwrecked and washed ashore on Lilliput, whose six-inch inhabitants are required by law to break their eggs only at the little ends. Of course, all those citizens who habitually break their eggs at the big ends are angered by the proclamation. Civil war breaks out between the Little Endians and the Big Endians, resulting in the Big Endians taking refuge on a nearby island, the kingdom of Blefuscu. The controversy is ethically and politically important for the Lilliputians. In fact, Swift has 11,000 Lilliputian rebels die over the egg question. The issue might seem silly, but Swift is satirizing the actual causes of religious or holy wars.

Swift's point is that the difference between breaking an egg at the little end and breaking it at the big end is trivial. He suggests that everyone do it in his preferred way.

Of course, we are making the opposite point. We agree that the difference between sending information with the little or the big end first is trivial, but insist that everyone must do it in the same way to avoid anarchy.



Reproduced from *The Annotated Gulliver's Travels* by Isaac Asimov. Published by Crown Publishers, Inc.

Endianness

No consensus

Representation of multi-byte scalar values in memory

- Little-endian systems
  - least-significant byte at lowest address
  - Intel processors
- Big-endian systems
  - least-significant byte at highest address
  - internet protocols, legacy or embedded processors (e.g. SPARC, PowerPC)

Endianness versus portability

Low-level C programs

- typically rely on **assumptions** on endianness.

$\Rightarrow$ **Porting** to platform with opposite endianness is **challenging**.

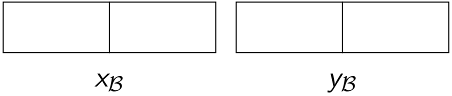
Agenda

1. Motivating example
2. Syntax and concrete semantics
3. Memory model
4. Evaluation

Reading multi-byte input in network byte-order

Big-endian version

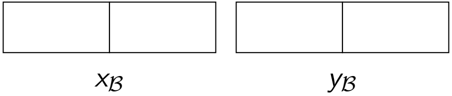
```
    u16 x, y;  // or u32, or u64
    read_from_network((u8 *)&x, sizeof(x));




    y = x;

  // read y
```



Reading multi-byte input in network byte-order

Big-endian version

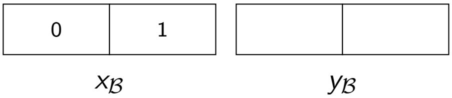
```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));




y = x;

// read y
```



Reading multi-byte input in network byte-order

Big-endian version

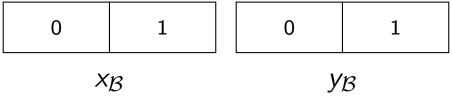
```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));




y = x;

// read y
```



Reading multi-byte input in network byte-order

Big-endian version

```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));




y = x;

// read y
```



Reading multi-byte input in network byte-order

Big-endian version

```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));




y = x;

// read y
```

0	1	0	1

$x_{\mathcal{B}}$ $y_{\mathcal{B}}$

$$1 = 0 \times 2^8 + 1 = y_{\mathcal{B}}$$

Reading multi-byte input in network byte-order

Big-endian version on little-endian machine

```
    u16 x, y;  // or u32, or u64
    read_from_network((u8 *)&x, sizeof(x));




    y = x;

  // read y
```

0	1

$x_{\mathcal{L}}$

0	1

$y_{\mathcal{L}}$

0	1

$x_{\mathcal{B}}$

0	1

$y_{\mathcal{B}}$

$$y_{\mathcal{L}} = 0 + 1 \times 2^8 = 256$$

$$1 = 0 \times 2^8 + 1 = y_{\mathcal{B}}$$

Reading multi-byte input in network byte-order

Big-endian version on little-endian machine

```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));




y = x;

// read y
```

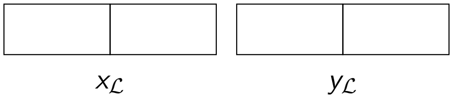
0	1	0	1	0	1	0	1
$x_\mathcal{L}$		$y_\mathcal{L}$		$x_\mathcal{B}$		$y_\mathcal{B}$	

$$y_\mathcal{L} = 0 + 1 \times 2^8 = 256 \quad \neq \quad 1 = 0 \times 2^8 + 1 = y_\mathcal{B}$$

Reading multi-byte input in network byte-order

Porting to little-endian

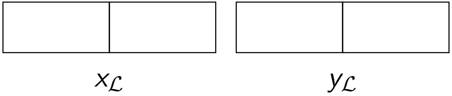
```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));

u8 *px = (u8 *)&x, *py = (u8 *)&y;
for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];



// read y
```



Reading multi-byte input in network byte-order

Porting to little-endian

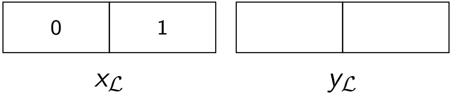
```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));

u8 *px = (u8 *)&x, *py = (u8 *)&y;
for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];



// read y
```



Reading multi-byte input in network byte-order

Porting to little-endian

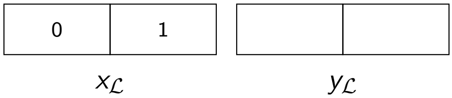
```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));

u8 *px = (u8 *)&x, *py = (u8 *)&y;
for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];



// read y
```



Reading multi-byte input in network byte-order

Porting to little-endian

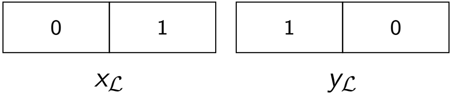
```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));

u8 *px = (u8 *)&x, *py = (u8 *)&y;
for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];



// read y
```



Reading multi-byte input in network byte-order

Porting to little-endian

```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));

u8 *px = (u8 *)&x, *py = (u8 *)&y;
for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];



// read y
```



Porting to little-endian

```
u16 x, y;  // or u32, or u64
read_from_network((u8 *)&x, sizeof(x));

u8 *px = (u8 *)&x, *py = (u8 *)&y;
for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];



// read y
```

0	1	1	0
$x_{\mathcal{L}}$		$y_{\mathcal{L}}$	

$$y_{\mathcal{L}} = 1 + 0 \times 2^8 = 1$$

Reading multi-byte input in network byte-order

Both versions, with conditional inclusion

```
  u16 x, y;  // or u32, or u64
  read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  u8 *px = (u8 *)&x, *py = (u8 *)&y;
  for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];
# else
  y = x;
# endif
```

// read y: $y_{\mathcal{L}} \stackrel{?}{=} y_{\mathcal{B}}$

$x_{\mathcal{L}}$		$y_{\mathcal{L}}$		$x_{\mathcal{B}}$		$y_{\mathcal{B}}$	
0	1	1	0	0	1	0	1

$$y_{\mathcal{L}} = 1 + 0 \times 2^8 = 1$$

$$1 = 0 \times 2^8 + 1 = y_{\mathcal{B}}$$

Reading multi-byte input in network byte-order

Both versions, with conditional inclusion

```
  u16 x, y;  // or u32, or u64
  read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  u8 *px = (u8 *)&x, *py = (u8 *)&y;
  for (int i=0; i<sizeof(x); i++) py[i] = px[sizeof(x)-i-1];
# else
  y = x;
# endif
```

// read y: $y_{\mathcal{L}} \stackrel{?}{=} y_{\mathcal{B}}$

$x_{\mathcal{L}}$		$y_{\mathcal{L}}$		$x_{\mathcal{B}}$		$y_{\mathcal{B}}$	
0	1	1	0	0	1	0	1

$$y_{\mathcal{L}} = 1 + 0 \times 2^8 = 1 \quad = \quad 1 = 0 \times 2^8 + 1 = y_{\mathcal{B}}$$

Reading multi-byte input in network byte-order

Both versions, with bitwise arithmetics

```c
  u16 x, y;  // or u32, or u64
  read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  y = (((x >> 8) & 0xff) | ((x & 0xff) << 8));      // see paper

# else
  y = x;
# endif
```

// read y: $y_{\mathcal{L}} \stackrel{?}{=} y_{\mathcal{B}}$

0	1	1	0	0	1	0	1

$x_{\mathcal{L}}$ $\quad y_{\mathcal{L}}$ $\quad x_{\mathcal{B}}$ $\quad y_{\mathcal{B}}$

$$y_{\mathcal{L}} = 1 + 0 \times 2^8 = 1 \qquad = \qquad 1 = 0 \times 2^8 + 1 = y_{\mathcal{B}}$$

Endian portability analysis

Endian portability

A program is called **endian portable** if two endian-specific versions thereof

- compute equal outputs
- when run on equal inputs
- on their respective platforms.

This talk

We present

a **static analysis** by abstract interpretation
to infer the **endian portability**
of **large** real-world **low-level** C programs.

Agenda

Agenda

1. Motivating example
2. Syntax and concrete semantics
3. Memory model
4. Evaluation

$$
\begin{array}{lcll}
\textit{dstat} & ::= & \textit{stat} & \\
& \mid & \textit{stat} \parallel \textit{stat} & \\
& \mid & \textbf{if } \textit{dcond} \textbf{ then } \textit{dstat} \textbf{ else } \textit{dstat} & \\
& \mid & \ldots & \\
& \mid & \textbf{assert\_sync}(\textit{expr}) &
\end{array}
$$

$$
\begin{array}{lcll}
\textit{stat} & ::= & *_{\textit{scalar-type}} \ \textit{expr} \leftarrow \textit{expr} & \\
& \mid & \textbf{if } \textit{cond} \textbf{ then } \textit{stat} \textbf{ else } \textit{stat} & \\
& \mid & \ldots & \\
\textit{cond} & ::= & \textit{expr} \bowtie 0 & \bowtie \in \{\leq, \geq, =, \neq, <, >\} \\
\textit{dcond} & ::= & \textit{cond} & \\
& \mid & \textit{cond} \parallel \textit{cond} &
\end{array}
$$

$$
\begin{array}{lcll}
\textit{expr} & ::= & *_{\textit{scalar-type}} \ \textit{expr} & \\
& \mid & \& V & \\
& \mid & [c_1, c_2] & c_1, c_2 \in \mathbb{Z} \\
& \mid & \circ \ \textit{expr} & \\
& \mid & \textit{expr} \diamond \textit{expr} &
\end{array}
$$

$$
\begin{array}{lcl}
\textit{scalar-type} & ::= & \textit{int-type} \mid \textbf{ptr} \\
\textit{int-type} & ::= & \textbf{u8} \mid \textbf{u16} \mid \textbf{u32} \mid \textbf{u64} \\
& \mid & \textbf{s8} \mid \textbf{s16} \mid \textbf{s32} \mid \textbf{s64}
\end{array}
$$

$$
\begin{array}{lcl}
\circ & ::= & \texttt{-} \mid \texttt{\~{}} \mid (\textit{scalar-type}) \\
\diamond & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{\&} \mid \texttt{|} \mid \texttt{\^{}} \mid \texttt{>>} \mid \texttt{<<}
\end{array}
$$

Lifting simple program semantics to double programs

Double program P

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{V}$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$

Endianness w.l.o.g. $(\mathcal{L}, \mathcal{B})$

Semantics $\mathbb{D}[\![\, s \,]\!] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

Lifting simple program semantics to double programs

Double program P

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{V}$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$

Endianness w.l.o.g. $(\mathcal{L}, \mathcal{B})$

Semantics $\mathbb{D}[\![\, s \,]\!] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

Transfer functions Delmas and Miné [2019a,b]

$$\mathbb{D}[\![\, s_1 \parallel s_2 \,]\!]X \triangleq \bigcup_{(\rho_\mathcal{L}, \rho_\mathcal{B}) \in X} (\mathbb{S}_\mathcal{L}[\![\, s_1 \,]\!]\{\, \rho_\mathcal{L} \,\} \times \mathbb{S}_\mathcal{B}[\![\, s_2 \,]\!]\{\, \rho_\mathcal{B} \,\})$$

Lifting simple program semantics to double programs

Double program P

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{V}$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$

Endianness w.l.o.g. $(\mathcal{L}, \mathcal{B})$

Semantics $\mathbb{D}[\![s]\!] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

Transfer functions Delmas and Miné [2019a,b]

$$\mathbb{D}[\![s_1 \parallel s_2]\!]X \triangleq \bigcup_{(\rho_\mathcal{L}, \rho_\mathcal{B}) \in X} (\mathbb{S}_\mathcal{L}[\![s_1]\!]\{\rho_\mathcal{L}\} \times \mathbb{S}_\mathcal{B}[\![s_2]\!]\{\rho_\mathcal{B}\})$$

$$\begin{aligned}\mathbb{D}[\![\textbf{if } e_1 \bowtie 0 \parallel e_2 \bowtie 0 \textbf{ then } s \textbf{ else } t]\!] &\triangleq \mathbb{D}[\![\ ]\!] \circ \mathbb{F}[\![e_1 \bowtie 0 \parallel e_2 \bowtie 0]\!] \\ &\dot{\cup}\ \mathbb{D}[\![\ ]\!] \circ \mathbb{F}[\![e_1 \not\bowtie 0 \parallel e_2 \not\bowtie 0]\!] \\ &\dot{\cup}\ \mathbb{D}[\![\ ]\!] \circ \mathbb{F}[\![e_1 \bowtie 0 \parallel e_2 \not\bowtie 0]\!] \\ &\dot{\cup}\ \mathbb{D}[\![\ ]\!] \circ \mathbb{F}[\![e_1 \not\bowtie 0 \parallel e_2 \bowtie 0]\!]\end{aligned}$$

where $\mathbb{F}[\![e_1 \bowtie 0 \parallel e_2 \bowtie 0]\!]X \triangleq \{ (\rho_\mathcal{L}, \rho_\mathcal{B}) \in X \mid \exists v_1 \in \mathbb{E}_\mathcal{L}[\![e]\!]\rho_\mathcal{L} : \exists v_2 \in \mathbb{E}_\mathcal{B}[\![e]\!]\rho_\mathcal{B} : v_1 \bowtie 0 \wedge v_2 \bowtie 0 \}$

Lifting simple program semantics to double programs

Double program P

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{V}$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$

Endianness w.l.o.g. $(\mathcal{L}, \mathcal{B})$

Semantics $\mathbb{D}[\![\, s \,]\!] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

Transfer functions — Delmas and Miné [2019a,b]

$$
\begin{array}{rcl}
\mathbb{D}[\![\, s_1 \parallel s_2 \,]\!]X & \triangleq & \bigcup_{(\rho_\mathcal{L}, \rho_\mathcal{B}) \in X} (\mathbb{S}_\mathcal{L}[\![\, s_1 \,]\!]\{\, \rho_\mathcal{L} \,\} \times \mathbb{S}_\mathcal{B}[\![\, s_2 \,]\!]\{\, \rho_\mathcal{B} \,\}) \\
\mathbb{D}[\![\, \textbf{if } e_1 \bowtie 0 \parallel e_2 \bowtie 0 \textbf{ then } s \textbf{ else } t \,]\!] & \triangleq & \mathbb{D}[\![\, \ldots \,]\!] \circ \mathbb{F}[\![\, e_1 \bowtie 0 \parallel e_2 \bowtie 0 \,]\!] \\
 & \dot{\cup} & \mathbb{D}[\![\, \ldots \,]\!] \circ \mathbb{F}[\![\, e_1 \not\bowtie 0 \parallel e_2 \not\bowtie 0 \,]\!] \\
 & \dot{\cup} & \mathbb{D}[\![\, \ldots \,]\!] \circ \mathbb{F}[\![\, e_1 \bowtie 0 \parallel e_2 \not\bowtie 0 \,]\!] \\
 & \dot{\cup} & \mathbb{D}[\![\, \ldots \,]\!] \circ \mathbb{F}[\![\, e_1 \not\bowtie 0 \parallel e_2 \bowtie 0 \,]\!] \\
\text{where } \mathbb{F}[\![\, e_1 \bowtie 0 \parallel e_2 \bowtie 0 \,]\!]X & \triangleq & \{\, (\rho_\mathcal{L}, \rho_\mathcal{B}) \in X \mid \exists v_1 \in \mathbb{E}_\mathcal{L}[\![\, e \,]\!]\rho_\mathcal{L} : \exists v_2 \in \mathbb{E}_\mathcal{B}[\![\, e \,]\!]\rho_\mathcal{B} : v_1 \bowtie 0 \wedge v_2 \bowtie 0 \,\}
\end{array}
$$

Lifting simple program semantics to double programs

Double program P

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{V}$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$

Endianness w.l.o.g. $(\mathcal{L}, \mathcal{B})$

Semantics $\mathbb{D}[\![s]\!] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

Transfer functions — Delmas and Miné [2019a,b]

$$
\begin{array}{lcl}
\mathbb{D}[\![s_1 \mid\mid s_2]\!] X & \triangleq & \bigcup_{(\rho_\mathcal{L}, \rho_\mathcal{B}) \in X} (\mathbb{S}_\mathcal{L}[\![s_1]\!] \{ \rho_\mathcal{L} \} \times \mathbb{S}_\mathcal{B}[\![s_2]\!] \{ \rho_\mathcal{B} \}) \\
\mathbb{D}[\![\textbf{if } e_1 \bowtie 0 \mid\mid e_2 \bowtie 0 \textbf{ then } s \textbf{ else } t]\!] & \triangleq & \mathbb{D}[\![s]\!] \circ \mathbb{F}[\![e_1 \bowtie 0 \mid\mid e_2 \bowtie 0]\!] \\
 & \dot\cup & \mathbb{D}[\![t]\!] \circ \mathbb{F}[\![e_1 \not\bowtie 0 \mid\mid e_2 \not\bowtie 0]\!] \\
 & \dot\cup & \mathbb{D}[\![\ldots]\!] \circ \mathbb{F}[\![e_1 \bowtie 0 \mid\mid e_2 \not\bowtie 0]\!] \\
 & \dot\cup & \mathbb{D}[\![\ldots]\!] \circ \mathbb{F}[\![e_1 \not\bowtie 0 \mid\mid e_2 \bowtie 0]\!] \\
\text{where } \mathbb{F}[\![e_1 \bowtie 0 \mid\mid e_2 \bowtie 0]\!] X & \triangleq & \{ (\rho_\mathcal{L}, \rho_\mathcal{B}) \in X \mid \exists v_1 \in \mathbb{E}_\mathcal{L}[\![e]\!] \rho_\mathcal{L} : \exists v_2 \in \mathbb{E}_\mathcal{B}[\![e]\!] \rho_\mathcal{B} : v_1 \bowtie 0 \wedge v_2 \bowtie 0 \}
\end{array}
$$

Lifting simple program semantics to double programs

Double program P

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{V}$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$

Endianness w.l.o.g. $(\mathcal{L}, \mathcal{B})$

Semantics $\mathbb{D}[\![\, s \,]\!] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

Transfer functions Delmas and Miné [2019a,b]

$$
\begin{array}{rcl}
\mathbb{D}[\![\, s_1 \parallel s_2 \,]\!]X & \triangleq & \bigcup_{(\rho_\mathcal{L}, \rho_\mathcal{B}) \in X} (\mathbb{S}_\mathcal{L}[\![\, s_1 \,]\!]\{\, \rho_\mathcal{L} \,\} \times \mathbb{S}_\mathcal{B}[\![\, s_2 \,]\!]\{\, \rho_\mathcal{B} \,\}) \\
\mathbb{D}[\![\, \textbf{if } e_1 \bowtie 0 \parallel e_2 \bowtie 0 \textbf{ then } s \textbf{ else } t \,]\!] & \triangleq & \mathbb{D}[\![\, s \,]\!] \circ \mathbb{F}[\![\, e_1 \bowtie 0 \parallel e_2 \bowtie 0 \,]\!] \\
 & \dot\cup & \mathbb{D}[\![\, t \,]\!] \circ \mathbb{F}[\![\, e_1 \not\bowtie 0 \parallel e_2 \not\bowtie 0 \,]\!] \\
 & \dot\cup & \mathbb{D}[\![\, s_\mathcal{L} \parallel t_\mathcal{B} \,]\!] \circ \mathbb{F}[\![\, e_1 \bowtie 0 \parallel e_2 \not\bowtie 0 \,]\!] \\
 & \dot\cup & \mathbb{D}[\![\, t_\mathcal{L} \parallel s_\mathcal{B} \,]\!] \circ \mathbb{F}[\![\, e_1 \not\bowtie 0 \parallel e_2 \bowtie 0 \,]\!] \\
\text{where } \ \mathbb{F}[\![\, e_1 \bowtie 0 \parallel e_2 \bowtie 0 \,]\!]X & \triangleq & \{\, (\rho_\mathcal{L}, \rho_\mathcal{B}) \in X \mid \exists v_1 \in \mathbb{E}_\mathcal{L}[\![\, e \,]\!]\rho_\mathcal{L} : \exists v_2 \in \mathbb{E}_\mathcal{B}[\![\, e \,]\!]\rho_\mathcal{B} : v_1 \bowtie 0 \wedge v_2 \bowtie 0 \,\} \\
s_\mathcal{L}, s_\mathcal{B} & \triangleq & \text{little-endian and big-endian versions of } s
\end{array}
$$

Lifting simple program semantics to double programs

Double program P

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{V}$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$

Endianness w.l.o.g. $(\mathcal{L}, \mathcal{B})$

Semantics $\mathbb{D}[\![s]\!] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

Transfer functions Delmas and Miné [2019a,b]

$$
\begin{array}{rcl}
\mathbb{D}[\![s_1 \parallel s_2]\!]X & \triangleq & \bigcup_{(\rho_\mathcal{L}, \rho_\mathcal{B}) \in X} (\mathbb{S}_\mathcal{L}[\![s_1]\!]\{\rho_\mathcal{L}\} \times \mathbb{S}_\mathcal{B}[\![s_2]\!]\{\rho_\mathcal{B}\}) \\
\mathbb{D}[\![\textbf{if } e_1 \bowtie 0 \parallel e_2 \bowtie 0 \textbf{ then } s \textbf{ else } t]\!] & \triangleq & \mathbb{D}[\![s]\!] \circ \mathbb{F}[\![e_1 \bowtie 0 \parallel e_2 \bowtie 0]\!] \\
 & \dot\cup & \mathbb{D}[\![t]\!] \circ \mathbb{F}[\![e_1 \not\bowtie 0 \parallel e_2 \not\bowtie 0]\!] \\
 & \dot\cup & \mathbb{D}[\![s_\mathcal{L} \parallel t_\mathcal{B}]\!] \circ \mathbb{F}[\![e_1 \bowtie 0 \parallel e_2 \not\bowtie 0]\!] \\
 & \dot\cup & \mathbb{D}[\![t_\mathcal{L} \parallel s_\mathcal{B}]\!] \circ \mathbb{F}[\![e_1 \not\bowtie 0 \parallel e_2 \bowtie 0]\!] \\
\text{where } \mathbb{F}[\![e_1 \bowtie 0 \parallel e_2 \bowtie 0]\!]X & \triangleq & \{ (\rho_\mathcal{L}, \rho_\mathcal{B}) \in X \mid \exists v_1 \in \mathbb{E}_\mathcal{L}[\![e]\!]\rho_\mathcal{L} : \exists v_2 \in \mathbb{E}_\mathcal{B}[\![e]\!]\rho_\mathcal{B} : v_1 \bowtie 0 \land v_2 \bowtie 0 \} \\
s_\mathcal{L}, s_\mathcal{B} & \triangleq & \text{little-endian and big-endian versions of } s \\
\mathbb{D}[\![\textbf{assert\_sync}(e)]\!]X & \triangleq & \{ (\rho_\mathcal{L}, \rho_\mathcal{B}) \in X \mid \exists v \in \mathbb{V} : \mathbb{E}_\mathcal{L}[\![e]\!]\rho_\mathcal{L} = \{ v \} = \mathbb{E}_\mathcal{B}[\![e]\!]\rho_\mathcal{B} \}
\end{array}
$$

Agenda

Low-level memory abstraction

Memory model

- Concrete level

  each program holds values for individual bytes

- Low-level C programs

$$\left.\begin{array}{l}\text{multi-byte access to memory}\\ \text{numerical invariants}\end{array}\right\} \Rightarrow \quad \text{need for scalar cells}$$

$$\left.\begin{array}{l}\text{byte-level access to encoding}\\ \text{abuse unions and pointers}\end{array}\right\} \Rightarrow \quad \text{cells may overlap}$$

Low-level memory abstraction

Memory model

- Concrete level

  each program holds values for individual bytes

- Low-level C programs

  multi-byte access to memory
  numerical invariants
  } ⇒ need for scalar cells

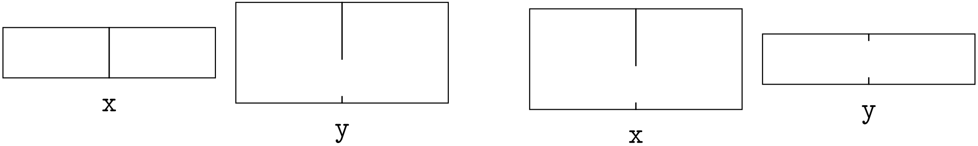
  byte-level access to encoding
  abuse unions and pointers
  } ⇒ cells may overlap

The Cells abstract domain Miné [2006, 2013]

- Memory as a dynamic collection of cells
  - synthetic scalar variables
    $\langle V, o, \tau, \alpha, k\rangle \in \widetilde{Cell} \triangleq \mathcal{V} \times \mathbb{N} \times \textit{scalar-type} \times \{\mathcal{L}, \mathcal{B}\} \times \{1, 2\}$
  - holding values for memory dereferences discovered during analysis
- Analysis with numerical domain (1 dimension / cell)

Analyzing the motivating example with cells

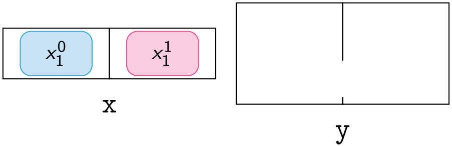
```
   u16 x, y;
   read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
   ((u8 *)&y)[0] = ((u8 *)&x)[1];
   ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
   y = x;
# endif
   assert_sync(y);   // y1 ?= y2
```

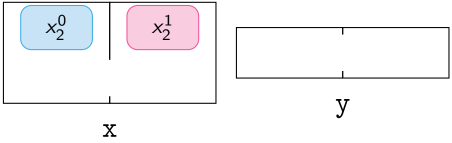


Analyzing the motivating example with cells

```c
  u16 x, y;
  read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif
  assert_sync(y);  // y1 =? y2
```

$x_1^0 = x_2^0 \;\wedge\; x_1^1 = x_2^1$





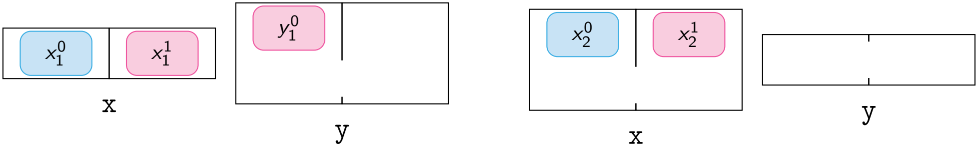
$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1\rangle$ $\quad$ $x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2\rangle$

Analyzing the motivating example with cells

```
  u16 x, y;
  read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif

  assert_sync(y);   // y1 ?= y2
```

$$x_1^0 = x_2^0 \ \wedge \ x_1^1 = x_2^1$$

$$y_1^0 = x_1^1$$



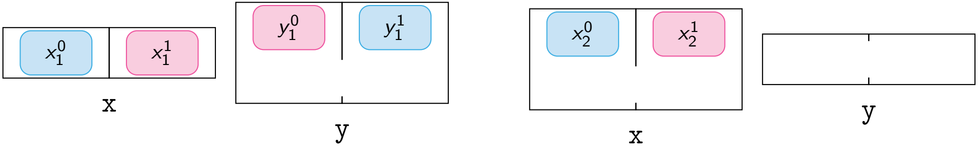
$$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1 \rangle \qquad x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2 \rangle$$

Analyzing the motivating example with cells

```
u16 x, y;
read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif

  assert_sync(y);  // y1 ?= y2
```

$$x_1^0 = x_2^0 \;\wedge\; x_1^1 = x_2^1$$

$$y_1^0 = x_1^1$$
$$y_1^1 = x_1^0$$



$$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1\rangle \qquad x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2\rangle$$

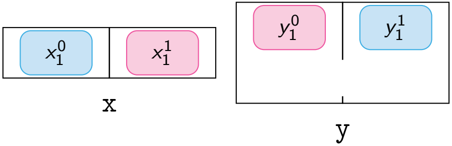
Analyzing the motivating example with cells

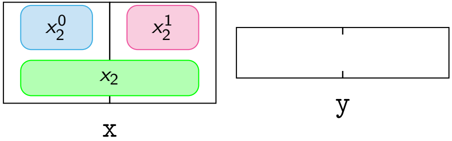
```
  u16 x, y;
  read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif

  assert_sync(y);  // y₁ =? y₂
```

$$x_1^0 = x_2^0 \;\wedge\; x_1^1 = x_2^1$$

$$y_1^0 = x_1^1$$
$$y_1^1 = x_1^0$$

$$x_2 = 2^8 \times x_2^0 + x_2^1$$





$$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1\rangle$$

$$x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2\rangle$$
$$x_2 \triangleq \langle x, 0, \mathbf{u16}, \mathcal{B}, 2\rangle$$

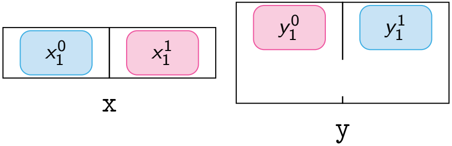
Analyzing the motivating example with cells

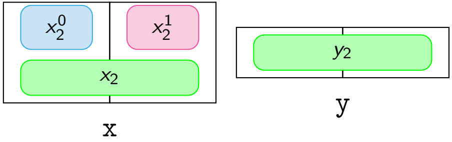
```
u16 x, y;
read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif
assert_sync(y);  // y1 ?= y2
```

$$x_1^0 = x_2^0 \;\wedge\; x_1^1 = x_2^1$$

$$y_1^0 = x_1^1$$
$$y_1^1 = x_1^0$$

$$x_2 = 2^8 \times x_2^0 + x_2^1 \;\wedge\; y_2 = x_2$$





$$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1 \rangle$$

$$x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2 \rangle$$
$$x_2 \triangleq \langle x, 0, \mathbf{u16}, \mathcal{B}, 2 \rangle$$

Analyzing the motivating example with cells

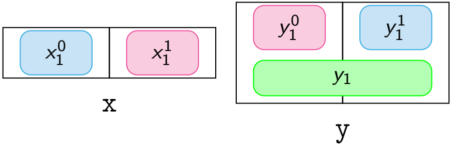
```c
u16 x, y;
read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif
assert_sync(y); // y1 ?= y2
```

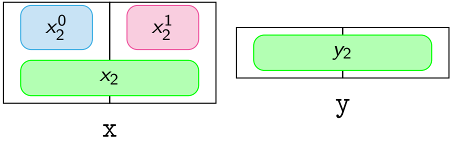
$$x_1^0 = x_2^0 \ \wedge \ x_1^1 = x_2^1$$

$$y_1^0 = x_1^1$$
$$y_1^1 = x_1^0$$

$$x_2 = 2^8 \times x_2^0 + x_2^1 \ \wedge \ y_2 = x_2$$

$$y_1 = y_1^0 + 2^8 \times y_1^1$$





$$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1\rangle$$
$$y_1 \triangleq \langle y, 0, \mathbf{u16}, \mathcal{L}, 1\rangle$$

$$x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2\rangle$$
$$x_2 \triangleq \langle x, 0, \mathbf{u16}, \mathcal{B}, 2\rangle$$

Optimizing the memory model for the common case

Complex invariants $\Longrightarrow$ expressive numerical domain?

- Program invariants and cell constraints

$$\mathbf{x_1^0} = \mathbf{x_2^0} = y_1^1 \qquad \mathbf{x_1^1} = \mathbf{x_2^1} = y_1^0 \qquad y_2 = x_2 \qquad \mathbf{y_1} \stackrel{?}{=} \mathbf{y_2}$$

$$x_1 = x_1^0 + 2^8 x_1^1 \qquad y_1 = y_1^0 + 2^8 y_1^1 \qquad x_2 = 2^8 x_2^0 + x_2^1 \qquad y_2 = 2^8 y_2^0 + y_2^1$$

- Common case: most multi-byte cells hold **equal values** in the little- and big-endian memories

Optimizing the memory model for the common case

Complex invariants $\Longrightarrow$ expressive numerical domain?

- Program invariants and cell constraints

$$\begin{array}{llll} \mathbf{x_1^0} = \mathbf{x_2^0} = y_1^1 & \mathbf{x_1^1} = \mathbf{x_2^1} = y_1^0 & y_2 = x_2 & \mathbf{y_1} \stackrel{?}{=} \mathbf{y_2} \\ x_1 = x_1^0 + 2^8 x_1^1 & y_1 = y_1^0 + 2^8 y_1^1 & x_2 = 2^8 x_2^0 + x_2^1 & y_2 = 2^8 y_2^0 + y_2^1 \end{array}$$

- Common case: most multi-byte cells hold **equal values** in the little- and big-endian memories

Sharing cells in the memory environment

- **Single** representation for **two** cells
  - from **different** program **versions**
  - at **same** memory **location**
  - holding **equal values**
- A bi-cell is $\mathcal{B}icell \triangleq \widetilde{\mathcal{C}ell} \cup (\widetilde{\mathcal{C}ell} \times \widetilde{\mathcal{C}ell})$

either a cell

or a pair of cells holding equal values (shared bi-cell)

Analyzing the motivating example: **from cells** to bi-cells

```
u16 x, y;
read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif

assert_sync(y); // y1 =? y2
```

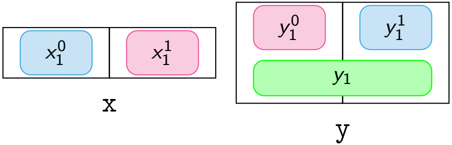
$x_1^0 = x_2^0 \;\wedge\; x_1^1 = x_2^1$

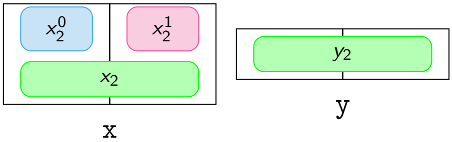
$y_1^0 = x_1^1$

$y_1^1 = x_1^0$

$x_2 = 2^8 \times x_2^0 + x_2^1 \;\wedge\; y_2 = x_2$

$y_1 = y_1^0 + 2^8 \times y_1^1$





$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1\rangle$

$y_1 \triangleq \langle y, 0, \mathbf{u16}, \mathcal{L}, 1\rangle$

$x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2\rangle$

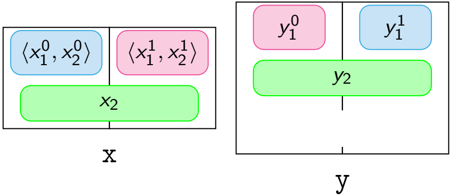
$x_2 \triangleq \langle x, 0, \mathbf{u16}, \mathcal{B}, 2\rangle$

Analyzing the motivating example: from cells **to bi-cells**

```
u16 x, y;
read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif

assert_sync(y); // y1 ?= y2
```

$$y_1^0 = \langle x_1^1, x_2^1 \rangle$$
$$y_1^1 = \langle x_1^0, x_2^0 \rangle$$

$$y_2 = x_2 \ \wedge \ x_2 = \ldots$$



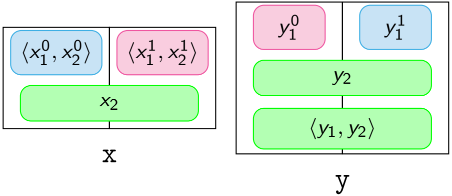
$$x_1^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1 \rangle$$
$$x_2^n \triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2 \rangle$$
$$x_2 \triangleq \langle x, 0, \mathbf{u16}, \mathcal{B}, 2 \rangle$$
$$y_1 \triangleq \langle y, 0, \mathbf{u16}, \mathcal{L}, 1 \rangle$$
$$y_2 \triangleq \langle y, 0, \mathbf{u16}, \mathcal{B}, 2 \rangle$$

Analyzing the motivating example: from cells **to bi-cells**

```c
u16 x, y;
read_from_network((u8 *)&x, sizeof(x));
# if __BYTE_ORDER == __LITTLE_ENDIAN
  ((u8 *)&y)[0] = ((u8 *)&x)[1];
  ((u8 *)&y)[1] = ((u8 *)&x)[0];
# else
  y = x;
# endif

assert_sync(y); // y1 =? y2
```

$$
\begin{aligned}
y_1^0 &= \langle x_1^1, x_2^1 \rangle \\
y_1^1 &= \langle x_1^0, x_2^0 \rangle
\end{aligned}
$$

$$y_2 = x_2 \ \wedge \ x_2 = \ldots$$



$$
\begin{aligned}
x_1^n &\triangleq \langle x, n, \mathbf{u8}, \mathcal{L}, 1 \rangle \\
x_2^n &\triangleq \langle x, n, \mathbf{u8}, \mathcal{B}, 2 \rangle \\
x_2 &\triangleq \langle x, 0, \mathbf{u16}, \mathcal{B}, 2 \rangle \\
y_1 &\triangleq \langle y, 0, \mathbf{u16}, \mathcal{L}, 1 \rangle \\
y_2 &\triangleq \langle y, 0, \mathbf{u16}, \mathcal{B}, 2 \rangle
\end{aligned}
$$

Agenda

1. Motivating example
2. Syntax and concrete semantics
3. Memory model
4. Evaluation

Implementation

MOPSA
analyzer

http://mopsa.lip6.fr/

Mopsa platform

- Modular development
- Precise static analyses
- Multiple languages
- Multiple properties

Prototype abstract interpreter

- 3,000 lines of OCaml source code
  - 19% double program management and iterators
  - 45% memory domain
  - 36% symbolic predicate domain (see paper)
- leverages 31,000 lines of Mopsa (excluding parsers)

Benchmarks

Origin	Name	LOC	Time	Revision	Result
Open Source	GENEVE	218	1 s	2014-1	🐞
				2014-2	✓
				2016	🐞
				2017	✓
	MLX5	125	155 ms	2017	🐞
				2020-1	🐞
				2020-2	✓
	Squashfs	110	150 ms	2020-1	🐞
				2020-2	✓
Industrial	Module S	300 K	9.7 h	2020	✓
	Module A	1 M	20.4 h	2020	🐞
				2021	✓

Disclaimer:

- Modules A and S are part of an early prototype, not in production yet.
- All findings have been incorporated into the development cycle.

Conclusion

Static analysis of Endian portability

1. Novel concrete collecting semantics
   - two versions of a program
   - platforms with different endiannesses
2. Joint memory abstraction

   relations between little- and big-endian memories
3. Prototype static analyzer
   - scale to large industrial software
   - with zero false alarms

Conclusion

Static analysis of Endian portability

1. Novel concrete collecting semantics
   - two versions of a program
   - platforms with different endiannesses
2. Joint memory abstraction

   relations between little- and big-endian memories
3. Prototype static analyzer
   - scale to large industrial software
   - with zero false alarms

More in the paper: Symbolic predicate domain

- Relations between bytes of variables of the two programs
- Established by bitwise arithmetics
- Near-linear cost

Conclusion

Static analysis of Endian portability

1. Novel concrete collecting semantics
   - two versions of a program
   - platforms with different endiannesses
2. Joint memory abstraction

   relations between little- and big-endian memories
3. Prototype static analyzer
   - scale to large industrial software
   - with zero false alarms

Future work

- **Industrialize** (certification of avionics simulation fidelity)
- Extend
  - **Portability** (layouts of C types, sizes of machine integers)
  - **Patches** modifying data-types

Backup slides

References

D. Delmas and A. Miné. Analysis of Program Differences with Numerical Abstract Interpretation. In *PERR 2019*, Prague, Czech Republic, Apr. 2019a.

D. Delmas and A. Miné. Analysis of Software Patches Using Numerical Abstract Interpretation. In B.-Y. E. Chang, editor, *Proc. of the 26th International Static Analysis Symposium (SAS'19)*, volume 11822 of *Lecture Notes in Computer Science*, pages 225–246, Porto, Portugal, Oct. 2019b. Bor-Yuh Evan Chang, Springer.

A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *Proc. of the ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'06)*, pages 54–63. ACM, June 2006.

A. Miné. Static analysis by abstract interpretation of concurrent programs. Technical report, École normale supérieure, May 2013.