The role of software and the cost of bugs

Safety-critical software
- flight-by-wire
- engine and breaks
- power plants
- pacemakers
- inertial systems
The role of software and the cost of bugs

**Safety-critical software**
- flight-by-wire
- engine and breaks
- power plants
- pacemakers
- inertial systems

**Software bugs**
- serious consequences
The role of software and the cost of bugs

Safety-critical software
- flight-by-wire
- engine and breaks
- power plants
- pacemakers
- inertial systems

Evolving software
Bugs can be introduced in
- initial development
- later version
- new environment

Software bugs
- serious consequences
The role of software and the cost of bugs

Safety-critical software
- flight-by-wire
- engine and breaks
- power plants
- pacemakers
- inertial systems

Evolving software
Bugs can be introduced in
- initial development
- later version
- new environment

Ariane 5.01 maiden flight
- reuse of Ariane 4 software
- different environment

Software bugs
serious consequences
The role of software and the cost of bugs

Ariane 5.01 maiden flight failure
- reuse of Ariane 4 software
- different environment
- direct cost: 500,000,000 $
The role of software and the cost of bugs

Safety-critical software
- flight-by-wire
- engine and breaks
- power plants
- pacemakers
- inertial systems

Evolving software
Bugs can be introduced in
- initial development
- later version
- new environment
  regression
  portability error

Software bugs
serious consequences

Software verification
is mandatory

Ariane 5.01 maiden flight failure
- reuse of Ariane 4 software
- different environment
- direct cost: 500,000,000 $
Program verification techniques

Automatic
Program verification techniques

Sound: all errors are detected

Automatic
Program verification techniques

Complete:
all warnings are true errors

Sound:
all errors are detected

Automatic

Complete

Sound
Program verification techniques

**Complete:**
all warnings are true errors

**Sound:**
all errors are detected

**Automatic**
Program verification techniques

Complete:
all warnings are true errors

Sound:
all errors are detected

Automatic

Test

Complete

Sound
Program verification techniques

**Complete:**
all warnings are true errors

**Sound:**
all errors are detected
Complete: all warnings are true errors

Sound: all errors are detected

Program verification techniques

Complete: all warnings are true errors

Sound: all errors are detected

Automatic

Test

Static analysis

Program proof
Aircraft functions transferred from hardware

de Havilland DH 106 Comet - 1949
Aircraft functions transferred from hardware to software

A350 Flight Deck
Software inside civil aircraft

Avionics software
- critical components of embedded systems
- e.g. flight-by-wire control systems
- major impact on safety
- widely used inside modern aircraft

Certification
- by third parties on behalf of Authorities (FAA, EASA)
- stringent rules on development and verification processes
- DO-178/ED-12 international standard
Traditional process-based assurance

informal verification

Large verification effort

- intellectual **reviews**
- unit and integration **tests**

Flowchart:
- System Requirements
  - High-level Requirements
    - Software Architecture
      - Low-level Requirements
        - Source code
          - Executable Object code

- Development activity
- Review or analysis
- Test activity
- Integration Testing
- Unit Testing

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.
Traditional process-based assurance  informal verification

Large verification effort
- intellectual **reviews**
- unit and integration **tests**

© V. Soumier
Traditional process-based assurance

Informal verification

Large verification effort
- intellectual reviews
- unit and integration tests

Development 30%
Verification 70%

System Requirements
High-level Requirements
Software Architecture
Low-level Requirements
Source code
Executable Object code

Development activity
Review or analysis
Test activity

Integration Testing
Unit Testing

A320 Family
A330
A380
A350XWB
Automated process leveraging formal verification

**Static analysis by AI**
- absence of *run-time error*
- numerical accuracy
- stack usage
- WCET

**Program proof**
- to replace unit testing

**Source code verification**
- formally verified compiler

**Industrial efficiency**
- cost savings in LLR processes
## Principle of formal verification by abstract interpretation

### Define the concrete semantics of your program

| Concrete semantics ≡ mathematical model of the set of all its possible behaviours in all possible environments can be constructed from semantics of commands of the programming language |
|---|---|

Define the concrete semantics of your program
Principle of formal verification by abstract interpretation

Define the concrete semantics of your program

concrete semantics $\equiv$ mathematical model of the set of all its possible behaviours in all possible environments

$can$ $be$ $constructed$ $from$ $semantics$ $of$ $commands$ $of$ $the$ $programming$ $language$

Define a specification

specification $\equiv$ subset of possible behaviours
Principle of formal verification by abstract interpretation

Define the concrete semantics of your program

**concrete semantics** ≡ mathematical **model** of the set of all its possible behaviours in all possible environments

*can be constructed from semantics of commands of the programming language*

Define a specification

**specification** ≡ subset of possible behaviours

Conduct a **formal proof**

that the concrete semantics meets the specification

use computers to **automate the proof**
Concrete semantics of program $P$

The concrete semantics of a program formalizes (is a mathematical model of) the set of all its possible executions in all possible execution environments.

Undecidability

– The concrete mathematical semantics of a program is an "infinite" mathematical object, not computable;
– All nontrivial questions on the concrete program semantics are undecidable.

Example: Kurt Gödel argument on termination

– Assume termination($P$) would always terminates and returns true iff $P$ always terminates on all input data;
– The following program yields a contradiction $P$ ·

\[
\text{while } \text{termination}(P) \text{ do skip od}
\]
II) Define which specification must be checked

Formalize what you are interested to prove about program behaviors

Specification of $P$ (e.g. safety property) © Cousot and Cousot [2010]
Formal proof of $P$

The safety properties of a program express that no possible execution in any possible execution environment can reach an erroneous state.

Safety proofs:
- A safety proof consists in proving that the intersection of the program concrete semantics and the forbidden zone is empty;
- Undecidable problem (the concrete semantics is not computable);
- It is impossible to provide completely automatic answers with finite computer resources and neither human interaction nor uncertainty on the answer.

Test/debugging:
- Consists in considering a subset of the possible executions;
- Not a correctness proof;
- Absence of coverage is the main problem.

Semantics$[\text{P}] \subseteq \text{Specification}[\text{P}]$
Abstract semantics for $P$

Semantics[$P$] is uncomputable

Abstraction of the trajectories

$\text{Abstraction}(\text{Semantics}[$P$])$
Abstraction(Semantics\[ P \]) \subseteq \text{Specification}[ P ]
Soundness of abstract interpretation

Semantics\[ P \] \subseteq \text{Abstraction}(\text{Semantics}[ P ]) \subseteq \text{Specification}[ P ]
Static analysis by abstract interpretation © Cousot and Cousot [2010]

Alarms

When abstract proofs may fail while concrete proofs would succeed, an alarm must be raised for this overapproximation!

Error or false alarm?
True error

The abstract alarm may correspond to a concrete error.

Forbidden zone

Alarm !!!

Error
Static analysis by abstract interpretation © Cousot and Cousot [2010]

Incompleteness ⇒ false alarms

Forbidden zone

Alarm !!!

False alarm
Static analysis by abstract interpretation

Numerical abstract domains

Bertrane et al. [2010]

Concrete values

Intervals
\[ x, y \in [a, b] \]
linear cost

Polyhedra
\[ \bigwedge \sum_i a_i x_i \leq b \]
exponential cost

Octagons
\[ \bigwedge \pm x \pm y \leq c \]
cubic cost

Abstract domains

- **sound** approximations of the concrete semantics
- trade-off between **cost** and **precision**
Goal of the thesis
Apply static analysis to two program equivalence problems

Regression verification

Objective program change does not add undesirable behaviors.
Patch analysis inferring that two syntactically close versions of a program compute equal outputs when run on equal inputs in the same environment.

Portability verification

Objective environment change does not add undesirable behaviors.
Portability analysis inferring that two syntactically close versions of a program compute equal outputs when run on equal inputs in their respective environments.
1. Introduction

2. Patch analysis for numerical programs

3. Patch analysis for C and structure layout portability

4. Endian portability analysis for C programs

5. Conclusion
1 Introduction

2 Patch analysis for numerical programs

3 Patch analysis for C and structure layout portability

4 Endian portability analysis for C programs

5 Conclusion
Running example  
Unchloop from Trostanetski et al. [2017]

Original program $P_1$

```plaintext
a = input(0,10);
b = input(0,10);
c = 1;
i=0;
while (i<a) {
    c=c+b;
i=i+1;
} 

r = c;
output(r);
```
Running example

Original and patched program versions $P_1$ and $P_2$

```
a = input(0,10);
b = input(0,10);
c = 1;
i=0;
while (i<a) {
    c=c+b;
    i=i+1;
}
r = c;
output(r);
```

```
a = input(0,10);
b = input(0,10);
c = 0;
i=0;
while (i<a) {
    c=c+b;
    i=i+1;
}
r = c+1;
output(r);
```
Running example

Unchloop from Trostanetski et al. [2017]

Original and patched program versions $P_1$ and $P_2$

**assume:**

$\begin{align*}
a &= \text{input}(0, 10) \\
b &= \text{input}(0, 10) \\
c &= 1;
\end{align*}$

$\begin{align*}
i &= 0; \\
\textbf{while} \ (i < a) \ {} \\
&\quad \{ \\
&\quad \quad c = c + b; \\
&\quad \quad i = i + 1; \\
&\quad \} \\
r &= c; \\
\text{output}(r);
\end{align*}$

$\begin{align*}
a &= \text{input}(0, 10) \\
b &= \text{input}(0, 10) \\
c &= 0;
\end{align*}$

$\begin{align*}
i &= 0; \\
\textbf{while} \ (i < a) \ {} \\
&\quad \{ \\
&\quad \quad c = c + b; \\
&\quad \quad i = i + 1; \\
&\quad \} \\
r &= c + 1; \\
\text{output}(r);
\end{align*}$

$\begin{align*}
a_1 &= a_2 \land b_1 = b_2 \\
(\text{equal inputs})
\end{align*}$
### Running example

Original and patched program versions $P_1$ and $P_2$

**assume:** \[ a_1 = a_2 \land b_1 = b_2 \] (equal inputs)

<table>
<thead>
<tr>
<th>$P_1$</th>
<th>$P_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = \text{input}(0,10);$</td>
<td>$a = \text{input}(0,10);$</td>
</tr>
<tr>
<td>$b = \text{input}(0,10);$</td>
<td>$b = \text{input}(0,10);$</td>
</tr>
<tr>
<td>$c = 1;$</td>
<td>$c = 0;$</td>
</tr>
<tr>
<td>$i=0;$</td>
<td>$i=0;$</td>
</tr>
<tr>
<td><strong>while</strong> $(i &lt; a)$ {</td>
<td><strong>while</strong> $(i &lt; a)$ {</td>
</tr>
<tr>
<td>$c = c + b;$</td>
<td>$c = c + b;$</td>
</tr>
<tr>
<td>$i = i + 1;$</td>
<td>$i = i + 1;$</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>$r = c;$</td>
<td>$r = c + 1;$</td>
</tr>
<tr>
<td><strong>output</strong>$(r);$</td>
<td><strong>output</strong>$(r);$</td>
</tr>
</tbody>
</table>

**prove:** \[ r_1 = r_2 \] (equal outputs)
## Running example

Unchloop from Trostanetski et al. [2017]

Invariants of program versions $P_1$ and $P_2$

<table>
<thead>
<tr>
<th>Assume:</th>
<th>$a_1 = a_2 \land b_1 = b_2$ (equal inputs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = \text{input}(0,10);$</td>
<td>$a_1 \in [0,10]$</td>
</tr>
<tr>
<td>$b = \text{input}(0,10);$</td>
<td>$b_1 \in [0,10]$</td>
</tr>
<tr>
<td>$c = 1;$</td>
<td>$c_1 = b_1 \times i_1 + 1$</td>
</tr>
<tr>
<td>$i=0;$</td>
<td>$c_2 = b_2 \times i_2$</td>
</tr>
<tr>
<td>while (i&lt;a) {}</td>
<td>$c_1 = a_1 \times b_1 + 1$</td>
</tr>
<tr>
<td>$c=c+b;$</td>
<td>$r_1 = a_1 \times b_1 + 1$</td>
</tr>
<tr>
<td>$i=i+1;$</td>
<td>$r = c+1;$</td>
</tr>
<tr>
<td>$r = c;$</td>
<td>output(r);</td>
</tr>
<tr>
<td>output(r);</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prove:</th>
<th>$r_1 = r_2$ (equal outputs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = \text{input}(0,10);$</td>
<td>$a_2 \in [0,10]$</td>
</tr>
<tr>
<td>$b = \text{input}(0,10);$</td>
<td>$b_2 \in [0,10]$</td>
</tr>
<tr>
<td>$c = 0;$</td>
<td>$c_2 = a_2 \times b_2$</td>
</tr>
<tr>
<td>$i=0;$</td>
<td></td>
</tr>
<tr>
<td>while (i&lt;a) {}</td>
<td>$c_2 = a_2 \times b_2 + 1$</td>
</tr>
<tr>
<td>$c=c+b;$</td>
<td></td>
</tr>
<tr>
<td>$i=i+1;$</td>
<td></td>
</tr>
<tr>
<td>$r = c+1;$</td>
<td></td>
</tr>
<tr>
<td>output(r);</td>
<td></td>
</tr>
</tbody>
</table>
Running example

Unchloop from Trostanetski et al. [2017]

Invariants of program versions $P_1$ and $P_2$

**assume:**

$$a_1 = a_2 \land b_1 = b_2$$  \hspace{1cm} (equal inputs)

<table>
<thead>
<tr>
<th></th>
<th>$a_1 = a_2 \land b_1 = b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>input(0,10); $a_1 \in [0,10]$</td>
</tr>
<tr>
<td>$b$</td>
<td>input(0,10); $b_1 \in [0,10]$</td>
</tr>
<tr>
<td>$c$</td>
<td>$c_1 = b_1 \times i_1 + 1$</td>
</tr>
<tr>
<td>$i$</td>
<td>$i_0$; \hspace{1cm} $c_1 = a_1 \times b_1 + 1$</td>
</tr>
<tr>
<td>$r$</td>
<td>$r_1 = c_1 = a_1 \times b_1 + 1$</td>
</tr>
</tbody>
</table>

prove:

$$r_1 = r_2$$  \hspace{1cm} (equal outputs)

<table>
<thead>
<tr>
<th></th>
<th>$a_2 \in [0,10]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>input(0,10); $a_2 \in [0,10]$</td>
</tr>
<tr>
<td>$b$</td>
<td>input(0,10); $b_2 \in [0,10]$</td>
</tr>
<tr>
<td>$c$</td>
<td>$c_2 = b_2 \times i_2$</td>
</tr>
<tr>
<td>$i$</td>
<td>$i_0$; \hspace{1cm} $c_2 = a_2 \times b_2$</td>
</tr>
<tr>
<td>$r$</td>
<td>$r_2 = c_1 + 1$</td>
</tr>
</tbody>
</table>

output(r);
Running example

Proving the equivalence of program versions $P_1$ and $P_2$

**assume:**

\[ a_1 = a_2 \land b_1 = b_2 \]  
(equal inputs)

\[
\begin{align*}
\text{output}(r); & \quad r_1 = a_1 \times b_1 + 1 \\
\text{output}(r); & \quad r_2 = a_2 \times b_2 + 1
\end{align*}
\]

**prove:**

\[ r_1 = r_2 \]  
(equal outputs)

Proof of equivalence

from **separate** analyses of $P_1$ and $P_2$

requires inferring **expressive** relational invariants  
(non linear)

\[ \implies \text{costly} \text{ numerical abstraction} \quad (\text{beyond polyhedra}) \]
Our approach
Joint analysis of program versions $P_1$ and $P_2$

First construct a double program $P$
from the AST of $P_1$ and $P_2$
using edit distance algorithms
with dynamic programming

```python
a = input(0,10);
b = input(0,10);
c = 1;
i=0;
while (i<a) {
    c=c+b;
i=i+1;
}
```

```python
r = c;
output(r);
```

```python
a = input(0,10);
b = input(0,10);
c = 0;
i=0;
while (i<a) {
    c=c+b;
i=i+1;
}
```

```python
r = c+1;
output(r);
```
Our approach
Joint analysis of program versions $P_1$ and $P_2$

First construct a double program $P$

from the AST of $P_1$ and $P_2$

using edit distance algorithms

with dynamic programming

```plaintext
a = input(0,10);
b = input(0,10);
c = 1;
i=0;
while (i<a) {
    c=c+b;
i=i+1;
}

r = c;
output(r);
```

```plaintext
a = input(0,10);
b = input(0,10);
c = 1 || 0;
i=0;
while (i<a) {
    c=c+b;
i=i+1;
}

r = c || c+1;
output(r);
```

```plaintext
a = input(0,10);
b = input(0,10);
c = 0;
i=0;
while (i<a) {
    c=c+b;
i=i+1;
}

r = c+1;
output(r);
```
**Our approach**

Joint analysis of program versions $P_1$ and $P_2$

First construct a double program $P$

*from* the AST of $P_1$ and $P_2$

*using* edit distance algorithms

*with* dynamic programming

<table>
<thead>
<tr>
<th>Left version: $P_1 = \pi_1(P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_1(s_1 \parallel s_2) \triangleq s_1$</td>
</tr>
<tr>
<td>$\pi_1(c = 1 \parallel 0) = c = 1$</td>
</tr>
<tr>
<td>$\pi_1(r = c \parallel c+1) = r = c$</td>
</tr>
</tbody>
</table>

```plaintext
a = input(0,10); b = input(0,10); c = 1; i=0; while (i<a) {
    c=c+b; i=i+1;
} r = c; output(r);
```

```plaintext
a = input(0,10); b = input(0,10); c = 1 \parallel 0; i=0; while (i<a) {
    c=c+b; i=i+1;
} r = c \parallel c+1; output(r);
```
Our approach
Joint analysis of program versions $P_1$ and $P_2$

First construct a double program $P$

from the AST of $P_1$ and $P_2$

using edit distance algorithms

with dynamic programming

Right version: $P_2 = \pi_2(P)$

\[\pi_2(s_1 \parallel s_2) \triangleq s_2\]
\[\pi_2(c = 1 \parallel 0) = c = 0\]
\[\pi_2(r = c \parallel c+1) = r = c+1\]

\[
\begin{align*}
a &= \text{input}(0,10); \\
b &= \text{input}(0,10); \\
c &= 1 \parallel 0; \\
i &= 0; \\
\text{while} \ (i < a) \ {\{}
  & c = c + b; \\
  & i = i + 1; \\
\} \\
r &= c \parallel c+1; \\
\text{output}(r);
\end{align*}
\]

\[
\begin{align*}
a &= \text{input}(0,10); \\
b &= \text{input}(0,10); \\
c &= 0; \\
i &= 0; \\
\text{while} \ (i < a) \ {\{}
  & c = c + b; \\
  & i = i + 1; \\
\} \\
r &= c + 1; \\
\text{output}(r);
\end{align*}
\]
Our approach
Joint analysis of program versions $P_1$ and $P_2$

**First** construct a double program $P$
from the AST of $P_1$ and $P_2$
using edit distance algorithms
with dynamic programming

**Then** analyze the double program $P$
using double program semantics
relating variables of $P_1$ and $P_2$
with less expressive invariants \( \text{(linear)} \)

```plaintext
a = input(0,10);
b = input(0,10);
c = 1 ∥ 0;
i=0;
while (i<a) {
    c=c+b;
i=i+1;
}

r = c ∥ c+1;
output(r);
```

\[
a_1 = a_2 \in [0, 10] \\
b_1 = b_2 \in [0, 10] \\
c_1 = 1 \land c_2 = 0 \\
c_1 = c_2 + 1 \\
r_1 = r_2
\]
Lifting simple program semantics to double programs

Concrete domain of simple programs

**Simple programs** $P_1$ and $P_2$

**Simple states** in $\mathcal{E} \triangleq \mathcal{V} \rightarrow \mathbb{Z}$

**Semantics** $\mathcal{S}[s] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$
Lifting simple program semantics to double programs

Concrete domain of simple programs and double programs

**Simple programs** $P_1$ and $P_2$

- **Simple states** in $\mathcal{E} \triangleq \mathcal{V} \rightarrow \mathbb{Z}$
- **Semantics** $S[s] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

**Double program** $P$

- **Double states** in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$
- **Semantics** $D[s] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$
Lifting simple program semantics to double programs
Patch, input, output, assignment and bloc statements

Simple programs $P_1$ and $P_2$

Simple states in $\mathcal{E} \triangleq \mathcal{V} \to \mathbb{Z}$
Semantics $\mathcal{S}[s] \in \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$

Double program $P$

Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$
Semantics $\mathcal{D}[s] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

$\mathcal{D}[s_1 \parallel s_2]X \triangleq \bigcup_{(\rho_1,\rho_2) \in X} \{ (\rho'_1, \rho'_2) \mid \rho'_1 \in \mathcal{S}[s_1] \{ \rho_1 \} \land \rho'_2 \in \mathcal{S}[s_2] \{ \rho_2 \} \}$
Lifting simple program semantics to double programs
Patch, input, output, assignment and bloc statements

<table>
<thead>
<tr>
<th>Simple programs $P_1$ and $P_2$</th>
<th>Double program $P$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simple states</strong> in $\mathcal{E} \triangleq \mathit{V} \rightarrow \mathbb{Z}$</td>
<td><strong>Double states</strong> in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$</td>
</tr>
<tr>
<td>Semantics $\mathcal{S}[s] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$</td>
<td>Semantics $\mathcal{D}[s] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$</td>
</tr>
</tbody>
</table>

$$
\begin{align*}
\mathcal{D}[s_1 \parallel s_2] & \triangleq \bigcup_{(\rho_1,\rho_2) \in X} \{ (\rho'_1, \rho'_2) \mid \rho'_1 \in \mathcal{S}[s_1] \{ \rho_1 \} \land \rho'_2 \in \mathcal{S}[s_2] \{ \rho_2 \} \} \\
\mathcal{D}[V \leftarrow e_1 \parallel e_2] & \triangleq \mathcal{D}[V \leftarrow e_1 \parallel V \leftarrow e_2] \\
\mathcal{D}[V \leftarrow e] & \triangleq \mathcal{D}[V \leftarrow e \parallel V \leftarrow e]
\end{align*}
$$
Lifting simple program semantics to double programs
Patch, input, output, assignment and bloc statements

Simple programs $P_1$ and $P_2$

Simple states in $E \triangleq V \rightarrow \mathbb{Z}$
Semantics $S[s] \in \mathcal{P}(E) \rightarrow \mathcal{P}(E)$

Double program $P$

Double states in $D \triangleq E \times E$
Semantics $D[s] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$

\[
\begin{align*}
\mathcal{D}[s_1 || s_2] X & \triangleq \bigcup_{(\rho_1, \rho_2) \in X} \{ (\rho'_1, \rho'_2) \mid \rho'_1 \in S[s_1] \{ \rho_1 \} \land \rho'_2 \in S[s_2] \{ \rho_2 \} \} \\
\mathcal{D}[V \leftarrow e_1 || e_2] & \triangleq \mathcal{D}[V \leftarrow e_1 || V \leftarrow e_2] \\
\mathcal{D}[V \leftarrow e] & \triangleq \mathcal{D}[V \leftarrow e || V \leftarrow e] \\
\mathcal{D}[V \leftarrow \text{input}(a, b)] X & \triangleq \{ (\rho_1[V \mapsto v], \rho_2[V \mapsto v]) \mid v \in [a, b] \land (\rho_1, \rho_2) \in X \} \\
\mathcal{D}[\text{output}(V)] X & \triangleq \{ (\rho_1, \rho_2) \in X \mid \rho_1(V) = \rho_2(V) \}
\end{align*}
\]
Lifting simple program semantics to double programs
Patch, input, output, assignment and bloc statements

Simple programs $P_1$ and $P_2$

Simple states in $E \triangleq V \rightarrow \mathbb{Z}$

Semantics $\mathbb{S}[s] \in \mathbb{P}(E) \rightarrow \mathbb{P}(E)$

Double program $P$

Double states in $D \triangleq E \times E$

Semantics $\mathbb{D}[s] \in \mathbb{P}(D) \rightarrow \mathbb{P}(D)$

\[
\begin{align*}
\mathbb{D}[s_1 \parallel s_2]X & \triangleq \bigcup_{(\rho_1, \rho_2) \in X} \{(\rho'_1, \rho'_2) \mid \rho'_1 \in \mathbb{S}[s_1]\{\rho_1\} \land \rho'_2 \in \mathbb{S}[s_2]\{\rho_2\}\} \\
\mathbb{D}[V \leftarrow e_1 \parallel e_2] & \triangleq \mathbb{D}[V \leftarrow e_1 \parallel V \leftarrow e_2] \\
\mathbb{D}[V \leftarrow e] & \triangleq \mathbb{D}[V \leftarrow e \parallel V \leftarrow e] \\
\mathbb{D}[V \leftarrow \text{input}(a, b)]X & \triangleq \{(\rho_1[V \mapsto v], \rho_2[V \mapsto v]) \mid v \in \{a, b\} \land (\rho_1, \rho_2) \in X\} \\
\mathbb{D}[\text{output}(V)]X & \triangleq \{(\rho_1, \rho_2) \in X \mid \rho_1(V) = \rho_2(V)\} \\
\mathbb{D}[s_1; s_2] & \triangleq \mathbb{D}[s_2] \circ \mathbb{D}[s_1]
\end{align*}
\]
Lifting simple program semantics to double programs

**if statement**

<table>
<thead>
<tr>
<th><strong>Simple programs</strong> $P_1$ and $P_2$</th>
<th><strong>Double program</strong> $P$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simple states</strong> in $E \triangleq V \rightarrow \mathbb{Z}$</td>
<td><strong>Double states</strong> in $D \triangleq E \times E$</td>
</tr>
<tr>
<td>Semantics $\text{S}[s] \in \mathcal{P}(E) \rightarrow \mathcal{P}(E)$</td>
<td>Semantics $\text{D}[s] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$</td>
</tr>
<tr>
<td>Conditions $\text{C}[c] \in \mathcal{P}(E) \rightarrow \mathcal{P}(E)$</td>
<td>Conditions $\text{F}[c_1</td>
</tr>
</tbody>
</table>
Lifting simple program semantics to double programs

**if statement**

<table>
<thead>
<tr>
<th>Simple programs $P_1$ and $P_2$</th>
<th>Double program $P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple states in $\mathcal{E} \triangleq \mathcal{V} \rightarrow \mathbb{Z}$</td>
<td>Double states in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$</td>
</tr>
<tr>
<td>Semantics $\mathbb{S}[s] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$</td>
<td>Semantics $\mathbb{D}[s] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$</td>
</tr>
<tr>
<td>Conditions $\mathbb{C}[c] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$</td>
<td>Conditions $\mathbb{F}[c_1</td>
</tr>
</tbody>
</table>

\[
\mathbb{F}[c_1 || c_2]X \triangleq \{ (\rho_1, \rho_2) \in X \mid \mathbb{C}[c_1]\{ \rho_1 \} \neq \emptyset \neq \mathbb{C}[c_2]\{ \rho_2 \} \}
\]
Lifting simple program semantics to double programs

**if statement**

---

**Simple programs** $P_1$ and $P_2$

- **Simple states** in $E \triangleq \mathcal{V} \rightarrow \mathbb{Z}$
- **Semantics** $\mathbb{S}[s] \in \mathcal{P}(E) \rightarrow \mathcal{P}(E)$
- **Conditions** $\mathbb{C}[c] \in \mathcal{P}(E) \rightarrow \mathcal{P}(E)$

---

**Double program** $P$

- **Double states** in $D \triangleq \mathcal{E} \times \mathcal{E}$
- **Semantics** $\mathbb{D}[s] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$
- **Conditions** $\mathbb{F}[c_1 \parallel c_2] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$

---

\[
\mathbb{F}[c_1 \parallel c_2] X \triangleq \{ (\rho_1, \rho_2) \in X \mid \mathbb{C}[c_1]\{\rho_1\} \neq \emptyset \neq \mathbb{C}[c_2]\{\rho_2\} \}
\]

\[
\mathbb{D}[\text{if } c_1 \parallel c_2 \text{ then } s \text{ else } t] \triangleq \mathbb{D}[ \ldots ] \circ \mathbb{F}[c_1 \parallel c_2] \circ \mathbb{D}[\ldots] \circ \mathbb{F}[\neg c_1 \parallel \neg c_2] \circ \mathbb{D}[\ldots] \circ \mathbb{F}[c_1 \parallel \neg c_2] \circ \mathbb{D}[\ldots] \circ \mathbb{F}[\neg c_1 \parallel c_2]
\]
Lifting simple program semantics to double programs

if statement

Simple programs $P_1$ and $P_2$

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

Semantics $\mathcal{S}[s] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

Conditions $\mathcal{C}[c] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

Double program $P$

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

Semantics $\mathcal{D}[s] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$

Conditions $\mathcal{F}[c_1 \parallel c_2] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$

\[
\mathcal{F}[c_1 \parallel c_2]X \triangleq \{(\rho_1, \rho_2) \in X | \mathcal{C}[c_1]\{\rho_1\} \neq \emptyset \neq \mathcal{C}[c_2]\{\rho_2\}\}
\]

\[
\mathcal{D}[\text{if } c_1 \parallel c_2 \text{ then } s \text{ else } t] \triangleq \mathcal{D}[s] \cup \mathcal{D}[t] \cup \mathcal{F}[c_1 \parallel c_2] \cup \mathcal{F}[\neg c_1 \parallel \neg c_2] \cup \mathcal{F}[c_1 \parallel \neg c_2] \cup \mathcal{F}[\neg c_1 \parallel c_2]
\]
Lifting simple program semantics to double programs

if statement

Simple programs $P_1$ and $P_2$

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

Semantics $\mathcal{S}[s] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

Conditions $\mathcal{C}[c] \in \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

Double program $P$

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

Semantics $\mathcal{D}[s] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$

Conditions $\mathcal{F}[c_1 \parallel c_2] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$

\[
\mathcal{F}[c_1 \parallel c_2] X \triangleq \{ (\rho_1, \rho_2) \in X | \mathcal{C}[c_1] \{ \rho_1 \} \neq \emptyset \neq \mathcal{C}[c_2] \{ \rho_2 \} \}
\]

\[
\mathcal{D}[\text{if } c_1 \parallel c_2 \text{ then } s \text{ else } t] \triangleq \begin{array}{ll}
\mathcal{D}[s] & \circ \mathcal{F}[c_1 \parallel c_2] \\
\hat{\cup} & \mathcal{D}[t] & \circ \mathcal{F}[\neg c_1 \parallel \neg c_2] \\
\hat{\cup} & \mathcal{D}[\pi_1(s) \parallel \pi_2(t)] & \circ \mathcal{F}[c_1 \parallel \neg c_2] \\
\hat{\cup} & \mathcal{D}[\pi_1(t) \parallel \pi_2(s)] & \circ \mathcal{F}[\neg c_1 \parallel c_2] \\
\end{array}
\]
Lifting simple program semantics to double programs

**while statement**

**Double program** $P$

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

**Semantics** $\mathcal{D}[s] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$

**Conditions** $\mathcal{F}[c_1 \parallel c_2] \in \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{P}(\mathcal{D})$
Lifting simple program semantics to double programs

while statement

Double program $P$

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

Semantics $D[s] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$

Conditions $F[c_1 \parallel c_2] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$

$D[\textbf{while } c_1 \parallel c_2 \textbf{ do } s]X \triangleq F[\neg c_1 \parallel \neg c_2](\text{lfp } H^X)$
Lifting simple program semantics to double programs

**while** statement

**Double program** $P$

- **Double states** in $\mathcal{D} \triangleq \mathcal{E} \times \mathcal{E}$
- **Semantics** $\mathcal{D}[s] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$
- **Conditions** $\mathcal{F}[c_1 || c_2] \in \mathcal{P}(\mathcal{D}) \to \mathcal{P}(\mathcal{D})$

\[
\mathcal{D}[\text{while } c_1 || c_2 \text{ do } s] X \triangleq \mathcal{F}[\neg c_1 || \neg c_2](\text{lfp } H^X)
\]

\[
H^X(Y) \triangleq X \cup \left( \mathcal{D}[s] \circ \mathcal{F}[c_1 || c_2] Y \cup \mathcal{D}[\pi_1(s) || \text{skip}] \circ \mathcal{F}[c_1 || \neg c_2] Y \cup \mathcal{D}[\text{skip} || \pi_2(s)] \circ \mathcal{F}[\neg c_1 || c_2] Y \right)
\]
Construct a double program from a pair of program versions
First merge identical statements

\[
\begin{align*}
\text{first} & \leftarrow \text{input}(0, 100); \\
\text{last} & \leftarrow \text{input}(0, 100); \\
\text{break} & \leftarrow \text{false}; \\
\end{align*}
\]

\[
\begin{align*}
i & \leftarrow 0; \\
\text{while} \ (\neg \text{break}) \ {\{} \\
\quad \text{x} & \leftarrow \text{first} + i \times 2; \\
\quad \text{if} \ (\text{last} < \text{x}) \\
\quad \quad \text{then} \ \text{break} & \leftarrow \text{true} \\
\quad \quad \text{else} \ \text{r} & \leftarrow \text{x}; \\
\quad \quad \quad \text{i} & \leftarrow \text{i} + 1 \\
\quad \} \\
\text{output}(\text{r})
\end{align*}
\]
Construct a double program from a pair of program versions

Then align similar control structures

```plaintext
first ← input(0, 100);
last ← input(0, 100);
break ← false;

out ← (last < first);
if (¬out) {
x ← first;
i ← 1;
while (¬break) {
    while (¬break) {
        x ← first + i × 2;
        if (last < x)
            then break ← true
        else r ← x;
        break ← true
    }
    i ← i + 1
}
output(r)
```

```plaintext
output(r)
```

```
```
```
Construct a double program from a pair of program versions

Then align similar control structures

\[
\begin{align*}
\text{\textit{first}} & \leftarrow \text{\textit{input}}(0, 100); \\
\text{\textit{last}} & \leftarrow \text{\textit{input}}(0, 100); \\
\text{\textit{break}} & \leftarrow \text{false}; \\
i & \leftarrow 0; \| \text{\textit{out}} \leftarrow (\text{\textit{last}} < \text{\textit{first}}); \\
\end{align*}
\]

\begin{align*}
\text{if } (\neg \text{\textit{out}}) \{ \\
\quad x & \leftarrow \text{\textit{first}}; \\
\quad i & \leftarrow 1; \\
\text{while } (\neg \text{\textit{break}}) \{ \\
\quad r & \leftarrow x; \\
\text{if } (\text{\textit{out}}) \\
\text{then } \text{\textit{break}} & \leftarrow \text{true} \\
\text{else} \quad r & \leftarrow x; \\
\text{else} \{ \\
\quad x & \leftarrow \text{\textit{first}} + i \times 2; \text{ \ \textit{out}} \leftarrow (\text{\textit{last}} < x); \\
\text{if } (\text{\textit{out}} \land \neg \text{\textit{more}}) \text{ then } \text{\textit{break}} & \leftarrow \text{true} \\
\quad i & \leftarrow i + 1 \\
\}
\}
\text{output}(r)
\end{align*}
Construct a double program from a pair of program versions

Then align similar control structures using simple program transformations

```plaintext
first ← input(0, 100);
last ← input(0, 100);
break ← false;
i ← 0; || out ← (last < first);

if (true) {
    if (¬out) {
        x ← first;
i ← 1;
        while (¬break) {
            x ← first + i × 2;
            if (last < x)
                then break ← true
            else r ← x;

            i ← i + 1
        }
    }

    output(r)

} if (¬out) {
    x ← first;
i ← 1;
    while (¬break) {
        r ← x;
        if (out)
            then break ← true
        else { x ← first + i × 2; out ← (last < x);
            if (out ∧ ¬more) then break ← true }

        i ← i + 1
    }
```

Construct a double program from a pair of program versions

The double program obtained allows for successful patch analysis with linear invariants

```plaintext
first ← input(0, 100);
last ← input(0, 100);
break ← false;
i ← 0  || out ← (last < first);
if (true  || ¬out) {
    skip
    x ← first;
i ← 1;
while (¬break) {
x ← first + i × 2  || r ← x;
if (last < x  || out)
    then break ← true
else
    r ← x  || x ← first + i × 2; out ← (last < x);
    if (out ∧ ¬more) then break ← true
i ← i + 1
}
}
output(r)
```
1. Introduction

2. Patch analysis for numerical programs

3. Patch analysis for C and structure layout portability

4. Endian portability analysis for C programs

5. Conclusion
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p += sizeof(u16);

output(*p);
Low-level C programs

```c
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p += sizeof(u16);

output(*p);
```
Low-level C programs

Patching a C data structure

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);
u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```

Removing unused field a

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);
u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```
Low-level C programs

Patching a C data structure

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```

Removing unused field `a`

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```
Low-level C programs

Patching a C data structure

```c
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p += sizeof(u16);

output(*p);
```

Removing unused field `a`

```c
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p += sizeof(u16);

output(*p);
```
Low-level C programs

Patching a C data structure

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```

Removing unused field `a`

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```
Low-level C programs

Patching a C data structure

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);

u8 *p = (u8*) &s + 1;
p += sizeof(u16);
output(*p);
```

Removing unused field `a`

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);

u8 *p = (u8*) &s + 1;
p += sizeof(u16);
output(*p);
```
Low-level C programs

Patching a C data structure

```
struct { u16 a; u16 b; } s;
s.b = input(0,1000);
u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```

Removing unused field a

```
struct { u16 a; u16 b; } s;
s.b = input(0,1000);
u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```
Low-level C programs
The Cell memory model

```c
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p += sizeof(u16);

output(*p);
```
Low-level C programs

The Cell memory model

```c
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *)&s + 1;
p += sizeof(u16);

output(*p);
```

Memory model

- Concrete level
  - the program holds values for individual bytes
- Low-level C programs
  - multi-byte access to memory
  - numerical invariants
  - byte-level access to encoding
  - abuse unions and pointers

\[ \Rightarrow \text{need for scalar cells} \]
\[ \Rightarrow \text{cells may overlap} \]
Low-level C programs

The Cell memory model

```c
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p += sizeof(u16);

output(*p);
```

Memory model

- **Concrete level**
  - the program holds values for individual bytes

- **Low-level C programs**
  - multi-byte access to memory
  - numerical invariants
  - byte-level access to encoding
  - abuse unions and pointers

⇒ need for scalar cells

⇒ cells may overlap

The Cells abstract domain

- Memory as a dynamic collection of cells
  - synthetic scalar variables \( \langle V, o, \tau \rangle \in Cell \subseteq V \times \mathbb{N} \times scalar-type \)
  - holding values for memory dereferences discovered during analysis

- Analysis with numerical domain (1 dimension / cell)
Low-level C programs

The Cell memory model

```c
struct { u16 a; u16 b; } s;
s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p += sizeof(u16);
output(*p);
```

The Cells abstract domain

- Memory as a dynamic collection of cells
  - synthetic scalar variables \( \langle V, o, \tau \rangle \in Cell \subseteq V \times \mathbb{N} \times scalar\text{-}type \)
  - holding values for memory dereferences discovered during analysis
- Analysis with numerical domain
  (1 dimension / cell)
Low-level C programs

The Cell memory model

```
struct { u16 a; u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p += sizeof(u16);

output(*p);
```

\[ \text{byte}(n, k) = \lfloor n/2^{8k} \rfloor \mod 2^8 \]

The Cells abstract domain

- Memory as a dynamic collection of cells
  - synthetic scalar variables \( \langle V, o, \tau \rangle \in Cell \subseteq V \times \mathbb{N} \times \text{scalar-type} \)
  - holding values for memory dereferences discovered during analysis
- Analysis with numerical domain \((1 \text{ dimension} / \text{cell})\)

Miné [2006a, 2013]
Patch analysis for low-level C programs
Lifting the Cell memory model

```c
struct { u16 a; u16 b; } s; ||
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p+=sizeof(u16) || skip;

output(*p);
```
struct { u16 a; u16 b; } s; 
struct { u16 b; } s;

s.b = input(0,1000);
u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;
output(*p);
struct { u16 a; u16 b; } s;  
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p+=sizeof(u16) || skip;

output(*p);
Patch analysis for low-level C programs

Lifting the Cell memory model

\[ \text{byte}(n, k) = \left\lfloor \frac{n}{2^k} \right\rfloor \mod 2^8 \]

\[
\begin{align*}
\text{struct} \ { \{ \ u16 \ a; \ u16 \ b; \} \ s; \} \\
\text{struct} \ { \{ \ u16 \ b; \} \ s; } \\
\text{s.b = input}(0,1000); \\
\text{u8 } *p = (\text{u8 }*) \&s + 1; \\
p += \text{sizeof}(\text{u16}) \| \text{skip}; \\
\text{output}(*p);
\end{align*}
\]

Program invariants and cell constraints

\[
\begin{align*}
c_1 &= c_2 \in [0, 1000] \\
c'_1 &= \text{byte}(c_1, 1) \\
c'_2 &= \text{byte}(c_2, 1) \\
c'_1 &\equiv c'_2
\end{align*}
\]
Optimizing the memory model for the common case

### Complex invariants \(\implies\) expressive numerical domain?

- **Program invariants and cell constraints**
  
  \[
  c_1' = \left\lfloor \frac{c_1}{2^8} \right\rfloor \mod 2^8 \quad \land \quad c_1 = c_2 \quad \implies \quad c_1' = c_2'
  \]

- **Common case**: most multi-byte cells hold **equal values** in the memories of \(P_1\) and \(P_2\)

---

**Sharing cells in the memory environment**

- **Single representation for two cells** - from different program versions - holding equal values
- A bi-cell is \(\text{Bicell} \equiv \text{Cell} \cup (\text{Cell} \times \text{Cell})\)
- **either a single cell** \(\text{Cell} \equiv \text{Cell}_1 \cup \text{Cell}_2\)
- **or a pair of cells holding equal values (shared bi-cell)\**

---

35
Optimizing the memory model for the common case

### Complex invariants ➞ expressive numerical domain?

- Program invariants and cell constraints
  
  \[
  \begin{align*}
  c_1' &= \lfloor c_1 / 2^8 \rfloor \mod 2^8 \\
  c_2' &= \lfloor c_2 / 2^8 \rfloor \mod 2^8
  \end{align*}
  \]

  \( \land \quad c_1 = c_2 \quad \Rightarrow \quad c_1' = c_2' \)

- **Common case**: most multi-byte cells hold equal values in the memories of \( P_1 \) and \( P_2 \)

### Sharing cells in the memory environment

- **Single** representation for two cells
  - from different program versions
  - holding equal values

- A bi-cell is
  
  \[ Bicell \triangleq \text{\( \tilde{\text{Cell}} \)} \cup (\text{\( \tilde{\text{Cell}} \) \times \( \tilde{\text{Cell}} \)}) \]

  either a single cell  
  \[ \text{\( \tilde{\text{Cell}} \)} \triangleq \text{\( Cell_1 \) \cup Cell_2 \)} \]

  or a pair of cells holding equal values (shared bi-cell)
Patch analysis for low-level C programs
From cells to bi-cells

```c
struct { u16 a; u16 b; } s; ||
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;

output(*p);
```

S1

S2
Patch analysis for low-level C programs
From cells to bi-cells

```c
struct { u16 a; u16 b; } s;
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;
output(*p);
```

S1

S2
Patch analysis for low-level C programs

From cells to bi-cells

```
struct { u16 a; u16 b; } s;  
struct { u16 b; } s;
```

\[ s.b = \text{input}(0,1000); \]

```
u8 *p = (u8 *) &s + 1;
p+=\text{sizeof}(u16) \parallel \text{skip};
output(*p);
```

Program invariants and bi-cell constraints

\[ c_1 \overset{?}{=} c_2 \]
Patch analysis for low-level C programs
From cells to bi-cells

```c
struct { u16 a; u16 b; } s;
struct { u16 b; } s;

s.b = input(0,1000); //

u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;

output(*p);
```

Program invariants and bi-cell constraints

\[ \langle c_1, c_2 \rangle \in [0, 1000] \]
Patch analysis for low-level C programs

From cells to bi-cells

```c
struct { u16 a; u16 b; } s;  \
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;

p+=sizeof(u16)  \| skip;

output(*p);
```

Program invariants and bi-cell constraints

\[ \langle c_1, c_2 \rangle \in [0,1000] \]

\[ c_1' \equiv c_2' \]
Patch analysis for low-level C programs

From cells to bi-cells

```c
struct { u16 a; u16 b; } s; \\
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;

output(*p);
```

Program invariants and bi-cell constraints

\[
\langle c_1, c_2 \rangle \in [0, 1000] \\
c_1' \equiv c_2'
\]

Shared bi-cell synthesis
Patch analysis for low-level C programs
From cells to bi-cells

```c
struct { u16 a; u16 b; } s;
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8*) &s + 1;
p+=sizeof(u16) || skip;

output(*p);
```

Program invariants and bi-cell constraints

\[ \langle c_1, c_2 \rangle \in [0, 1000] \]

\[ c_1' \equiv c_2' \]

Shared bi-cell synthesis

\[ \exists \langle c_1', c_2' \rangle \neq X \]
Patch analysis for low-level C programs

From cells to bi-cells

\[
\text{byte}(n, k) = \lfloor \frac{n}{2^k} \rfloor \mod 2^8
\]

\[
\begin{align*}
\text{struct} & \{ \text{u16 a; u16 b; } \} \ s; \\
\text{struct} & \{ \text{u16 b; } \} \ s;
\end{align*}
\]

s.b = \text{input}(0,1000);

\[
\text{u8 *p = (u8 *) &s + 1;}
\]

p+=\text{sizeof(u16)} \parallel \text{skip};

\text{output(*p);}

Program invariants and bi-cell constraints

\[
\begin{align*}
\langle c_1, c_2 \rangle & \in [0,1000] \\
c_1' & \equiv c_2'
\end{align*}
\]

Shared bi-cell synthesis

\[
\begin{align*}
\exists \langle c_1', c_2' \rangle & \ ? X \\
\forall \rho : \rho(c_1') & = \rho(c_2') \ ? \$ > \text{polyhedra} \\
& \s_1 \\
& c_1' = \text{byte}(c_1, 1) \\
& c_2' = \text{byte}(c_2, 1)
\end{align*}
\]
Patch analysis for low-level C programs
From cells to bi-cells

```c
struct { u16 a; u16 b; } s;
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;

output(*p);
```

Program invariants and bi-cell constraints

 ⟨c₁, c₂⟩ ∈ [0, 1000]

 c'₁ ≠ c'₂

Shared bi-cell synthesis

∃⟨c'₁, c'₂⟩ ? X

∀ρ : ρ(c₁') = ρ(c₂') ? $ > polyhedra

∃(x₁, x₂, o) : x₁ = x₂ ∧
    c'ᵢ at offset o inside xᵢ ? ✓ { xᵢ = cᵢ; o = 1

S₁

S₂

⟨c₁, c₂⟩

c₁' c₂'
Patch analysis for low-level C programs
From cells to bi-cells

```c
struct { u16 a; u16 b; } s;
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;

output(*p);
```

Program invariants and bi-cell constraints
\[ \langle c_1, c_2 \rangle \in [0, 1000] \]

Shared bi-cell synthesis
\[ \langle c'_1, c'_2 \rangle \text{ synthesized by pattern-matching} \]
Patch analysis for low-level C programs

From cells to bi-cells

\[
\text{byte}(n, k) = \lceil n / 2^k \rceil \mod 2^8
\]

```
struct { u16 a; u16 b; } s;
struct { u16 b; } s;

s.b = input(0,1000);

u8 *p = (u8 *) &s + 1;
p+=sizeof(u16) || skip;

output(*p);
```

Program invariants and bi-cell constraints

\[
\langle c_1, c_2 \rangle \in [0, 1000]
\]

\[
\langle c_1', c_2' \rangle = \text{byte}(\langle c_1, c_2 \rangle, 1)
\]

Shared bi-cell synthesis
Implementation on top of Mopsa

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

Prototype abstract interpreter
\[ \approx 6,700 \text{ lines of OCaml source code} \]
- 50% bi-cell based memory abstraction
- 33% double program construction
- 17% double program iterators and utilities

The Mopsa leverage effect
\[ \approx 50,000 \text{ lines of Mopsa leveraged} \]
- 38% parsers and utilities
- 27% common framework iterators and numeric domains
- 24% specific for the C language
- 11% generic for all languages
Implementation

Analysis of C programs with cells

Sequence

Reduced product

Cartesian product

Composition

Universal

C specific

Double C
Implementation

Analysis of C programs with cells

Analysis of C patches with cells

- Sequence
- Reduced product
- Cartesian product
- Composition

Universal
C specific
Double C
Implementation

Analysis of C patches with cells

D.program \Downarrow C.program \Downarrow D.builtins
D.interproc \Downarrow C.interproc \Downarrow U.interproc
D.intraproc \Downarrow C.intraproc \Downarrow U.intraproc
D.loops \Downarrow C.loops \Downarrow U.loops
C.libraries \Downarrow C.Aggregates

D.patch

C.cells

C.machineNum \times C.pointers

U.linearRel \wedge U.intervals \wedge U.congruences

Sequence

Reduced product

Cartesian product

Composition

Universal

C specific

Double C
Implementation

Analysis of C patches with cells

- D.program
- C.program
- D.builtins
- D.interproc
- C.interproc
- U.interproc
- D.intraproc
- C.intraproc
- U.intraproc
- D.loops
- C.loops
- U.loops
- D.libraries
- C.libraries
- C.Aggregates

Analysis of C patches with bi-cells

- D.program
- C.program
- D.builtins
- D.interproc
- C.interproc
- U.interproc
- D.intraproc
- C.intraproc
- U.intraproc
- D.loops
- C.loops
- U.loops
- D.libraries
- C.libraries
- C.Aggregates

- Sequence
- Reduced product
- Cartesian product
- Composition
- Universal
- C specific
- Double C

- U.linearRel
- U.intervals
- U.congruences
- U.linearRel
- U.intervals
- U.congruences
## Related works

### Semantic patch analysis

<table>
<thead>
<tr>
<th>Related work</th>
<th>Tool</th>
<th>Characteristics</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symbolic execution</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trostanetski et al. [2017]</td>
<td>ModDiff</td>
<td>Full path enumeration</td>
<td>Approximate fixpoint computation</td>
</tr>
<tr>
<td><strong>Deductive methods</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Godlin and Strichman [2009]</td>
<td>RVT</td>
<td>SMT solvers</td>
<td>Abstract domains</td>
</tr>
<tr>
<td>Lahiri et al. [2012]</td>
<td>SYMDiff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and Klebanov et al. [2018]</td>
<td>RÊVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Abstract interpretation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partush and Yahav [2013]</td>
<td>DIZY</td>
<td>Program transformation</td>
<td>Concrete collecting semantics</td>
</tr>
<tr>
<td>Partush and Yahav [2014]</td>
<td>SCORE</td>
<td>→ correlating program</td>
<td>for double programs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>speculative correlation</td>
<td>double program construction</td>
</tr>
</tbody>
</table>
### Evaluation

Synthetic or simplified benchmarks from the related works

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LOC</th>
<th>#P</th>
<th>Related time</th>
<th>Cell based abstraction</th>
<th>Bi-cell based abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>polyhedra</td>
<td>octagon</td>
</tr>
<tr>
<td><strong>ModDiff</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comp</td>
<td>13</td>
<td>2</td>
<td>539 ms</td>
<td>48 ms</td>
<td>✓</td>
</tr>
<tr>
<td>Const</td>
<td>9</td>
<td>3</td>
<td>541 ms</td>
<td>28 ms</td>
<td>✓</td>
</tr>
<tr>
<td>Fig. 2</td>
<td>14</td>
<td>1</td>
<td>–</td>
<td>31 ms</td>
<td>✓</td>
</tr>
<tr>
<td>LoopMult</td>
<td>14</td>
<td>2</td>
<td>49 s</td>
<td>166 ms</td>
<td>✓</td>
</tr>
<tr>
<td>LoopSub</td>
<td>15</td>
<td>2</td>
<td>1.2 s</td>
<td>60 ms</td>
<td>✓</td>
</tr>
<tr>
<td>UnchLoop</td>
<td>13</td>
<td>2</td>
<td>2.8 s¹</td>
<td>69 ms</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Rêve</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td>11</td>
<td>3</td>
<td>50 ms</td>
<td>43 ms</td>
<td>✓</td>
</tr>
<tr>
<td>while-if</td>
<td>11</td>
<td>3</td>
<td>80 ms</td>
<td>66 ms</td>
<td>✓</td>
</tr>
<tr>
<td>digits10</td>
<td>24</td>
<td>19</td>
<td>1.12 s</td>
<td>312 ms</td>
<td>✓</td>
</tr>
<tr>
<td>barthe</td>
<td>13</td>
<td>2</td>
<td>120 ms</td>
<td>93 ms</td>
<td>✓</td>
</tr>
<tr>
<td>barthe2</td>
<td>11</td>
<td>2</td>
<td>150 ms</td>
<td>81 ms</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Score/Dizy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sign</td>
<td>12</td>
<td>2</td>
<td>–</td>
<td>29 ms</td>
<td>✓</td>
</tr>
<tr>
<td>sum</td>
<td>14</td>
<td>4</td>
<td>4 s</td>
<td>71 ms</td>
<td>✓</td>
</tr>
<tr>
<td>copy²</td>
<td>37</td>
<td>1</td>
<td>2 s</td>
<td>132 ms</td>
<td>✓</td>
</tr>
<tr>
<td>seq²</td>
<td>41</td>
<td>13</td>
<td>11 s</td>
<td>293 ms</td>
<td>✓</td>
</tr>
<tr>
<td>pr²</td>
<td>111</td>
<td>8</td>
<td>1149 s</td>
<td>2.686 s</td>
<td>✓</td>
</tr>
</tbody>
</table>

¹ only 5 loop iterations
² Coreutils (simplified code)
## Evaluation

Real patches from Coreutils and Linux

<table>
<thead>
<tr>
<th>Bench.</th>
<th>LOC</th>
<th>#P</th>
<th>Cell based abstraction</th>
<th>Bi-cell based abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>polyhedra</td>
<td>octagon</td>
</tr>
<tr>
<td>Coreutils</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy</td>
<td>95</td>
<td>1</td>
<td>157 ms ✓</td>
<td>482 ms ✓</td>
</tr>
<tr>
<td>seq</td>
<td>46</td>
<td>16</td>
<td>570 ms ✓</td>
<td>x</td>
</tr>
<tr>
<td>pr</td>
<td>114</td>
<td>8</td>
<td>1.421 s ✓</td>
<td>6.469 s ✓</td>
</tr>
<tr>
<td>test</td>
<td>352</td>
<td>10</td>
<td>9.188 s ✓</td>
<td>x</td>
</tr>
<tr>
<td>Linux</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kvm</td>
<td>248</td>
<td>1/11</td>
<td>2.707 s ✓</td>
<td>4.214 s ✓</td>
</tr>
<tr>
<td>sched</td>
<td>194</td>
<td>7/12</td>
<td>65 ms ✓</td>
<td>x</td>
</tr>
<tr>
<td>dma</td>
<td>270</td>
<td>5/23</td>
<td>285 ms ✓</td>
<td>1.235 s ✓</td>
</tr>
<tr>
<td>block</td>
<td>324</td>
<td>22/6</td>
<td>80 ms ✓</td>
<td>x</td>
</tr>
<tr>
<td>iucv</td>
<td>179</td>
<td>10/9</td>
<td>403 ms ✓</td>
<td>1.757 s ✓</td>
</tr>
<tr>
<td>io_uring</td>
<td>1569</td>
<td>10/14</td>
<td>868.701 s ✓</td>
<td>x</td>
</tr>
</tbody>
</table>

2 simplified Coreutils benchmarks from Score/Dizy
## Evaluation

Real patches from Coreutils and Linux

<table>
<thead>
<tr>
<th>Bench.</th>
<th>LOC</th>
<th>#P</th>
<th>Cell based abstraction</th>
<th>Bi-cell based abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>polyhedra</td>
<td>octagon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy</td>
<td>95</td>
<td>1</td>
<td>157 ms ✓</td>
<td>482 ms ✓</td>
</tr>
<tr>
<td>copy²</td>
<td>37</td>
<td>1</td>
<td>132 ms ✓</td>
<td>373 ms ✓</td>
</tr>
<tr>
<td>seq</td>
<td>46</td>
<td>16</td>
<td>570 ms ✓</td>
<td>X</td>
</tr>
<tr>
<td>seq²</td>
<td>41</td>
<td>13</td>
<td>293 ms ✓</td>
<td>X</td>
</tr>
<tr>
<td>pr</td>
<td>114</td>
<td>8</td>
<td>1.421 s ✓</td>
<td>6.469 s ✓</td>
</tr>
<tr>
<td>pr²</td>
<td>111</td>
<td>8</td>
<td>2.686 s ✓</td>
<td>11.672 s ✓</td>
</tr>
<tr>
<td>test</td>
<td>352</td>
<td>10</td>
<td>9.188 s ✓</td>
<td>X</td>
</tr>
<tr>
<td>Linux</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kvm</td>
<td>248</td>
<td>1/11</td>
<td>2.707 s ✓</td>
<td>4.214 s ✓</td>
</tr>
<tr>
<td>sched</td>
<td>194</td>
<td>7/12</td>
<td>65 ms ✓</td>
<td>X</td>
</tr>
<tr>
<td>dma</td>
<td>270</td>
<td>5/23</td>
<td>285 ms ✓</td>
<td>1.235 s ✓</td>
</tr>
<tr>
<td>block</td>
<td>324</td>
<td>22/6</td>
<td>80 ms ✓</td>
<td>X</td>
</tr>
<tr>
<td>iucv</td>
<td>179</td>
<td>10/9</td>
<td>403 ms ✓</td>
<td>1.757 s ✓</td>
</tr>
<tr>
<td>io_uring</td>
<td>1569</td>
<td>10/14</td>
<td>868.701 s ✓</td>
<td>X</td>
</tr>
</tbody>
</table>

2simplified Coreutils benchmarks from Score/Dizy
Agenda

1. Introduction
2. Patch analysis for numerical programs
3. Patch analysis for C and structure layout portability
4. Endian portability analysis for C programs
5. Conclusion
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)
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.

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.
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.

⇒ **Porting** to platform with opposite endianness is **challenging**.
Reading multi-byte input in network byte-order

Big-endian version

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

```
y = x;
```

```c
// read y
```

![Diagram showing big-endian byte ordering]
Reading multi-byte input in network byte-order

Big-endian version

```c
u16 x, y;  // or u32, or u64

read_from_network((u8 *)&x, sizeof(x));

y = x;

// read y
```

```
x_B
```

```
y_B
```
Reading multi-byte input in network byte-order

Big-endian version

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

- y = x;

// read y

![Diagram showing bytes being read in big-endian order](image)
Reading multi-byte input in network byte-order

Big-endian version

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

y = x;
```

// read y

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x&lt;sub&gt;B&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y&lt;sub&gt;B&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Reading multi-byte input in network byte-order

Big-endian version

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

y = x;
// read y
```

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_B</td>
<td></td>
<td>y_B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

\[ 1 = 0 \times 2^8 + 1 = y_B \]
Reading multi-byte input in network byte-order

Big-endian version on little-endian machine

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

```plaintext
y = x;
```

// read y

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(x_L)</td>
<td>(y_L)</td>
<td>(x_B)</td>
<td>(y_B)</td>
</tr>
</tbody>
</table>

\[
y_L = 0 + 1 \times 2^8 = 256
\]

\[
y_B = 1 = 0 \times 2^8 + 1 = y_B
\]
Reading multi-byte input in network byte-order

Big-endian version on little-endian machine

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

```plaintext

\[
y = x;
\]

```
// read y
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_L)</td>
<td>(y_L)</td>
<td>(x_B)</td>
<td>(y_B)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
y_L = 0 + 1 \times 2^8 = 256 \\
\neqv \\

1 = 0 \times 2^8 + 1 = y_B
\]
Reading multi-byte input in network byte-order

Porting to little-endian

```c
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

\[ x_L \quad y_L \]
Reading multi-byte input in network byte-order

Porting to little-endian

```c
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

```
XML  
```

```
Y<XML?
```

Reading multi-byte input in network byte-order

Porting to little-endian

```c
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ x_L \quad y_L \]
Reading multi-byte input in network byte-order

Porting to little-endian

```c
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
```

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

\[ X_L \] \[ Y_L \]
Reading multi-byte input in network byte-order

Porting to little-endian

```c
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
\text{x}_L \quad \text{y}_L
```
Reading multi-byte input in network byte-order

Porting to little-endian

```c
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

$$y_L = 1 + 0 \times 2^8 = 1$$
Reading multi-byte input in network byte-order

Both versions, with conditional inclusion

```c
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_L = y_B

\[
x_L \quad 0 \quad 1 \\
y_L \quad 1 \quad 0 \\
x_B \quad 0 \quad 1 \\
y_B \quad 0 \quad 1
\]

\[
y_L = 1 + 0 \times 2^8 = 1
\]
\[
1 = 0 \times 2^8 + 1 = y_B
\]
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: \( y_L = y_B \)
```

\[
\begin{array}{c|c|}
0 & 1 & 1 & 0 \\
\end{array}
\]

\[
x_L & y_L & x_B & y_B \\
\end{array}
\]

\[
y_L = 1 + 0 \times 2^8 = 1 = 1 = 0 \times 2^8 + 1 = y_B
\]
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));  // bitwise arithmetic
# else
y = x;
# endif

// read y: \( y_L = y_B \)

\[
\begin{array}{c|c}
0 & 1 \\
\hline
1 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c}
0 & 1 \\
\hline
0 & 1 \\
\end{array}
\]  

\[
y_L = 1 + 0 \times 2^8 = 1
\]

\[
1 = 0 \times 2^8 + 1 = y_B
\]
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**.

**Our approach**

We present
- a **static analysis** by abstract interpretation
to infer the **endian portability** of **large** real-world **low-level** C programs.
Semantics of simple endian-aware low-level C programs
Parameterizing the semantics with endianness

**Memory model**

The semantics of memory reads and writes depends on the endianness of the platform.

\[
x_L^0 \quad x_L^1 \quad y_L^0 \quad y_L^1
\]

\[
x_L = y_L^0 + y_L^1 \times 2^8
\]

\[
x_B^0 \quad x_B^1 \quad y_B^0 \quad y_B^1
\]

\[
y_B = y_B^0 \times 2^8 + y_B^1
\]
Semantics of simple endian-aware low-level C programs
Parameterizing the semantics with endianness

**Memory model**

The semantics of memory reads and writes depends on the endianness of the platform.

\[
x^0_L \quad x^1_L \quad y^0_L \quad y^1_L \\
\]
\[
x_L \quad y_L \\
\]
\[
y_L = y^0_L + y^1_L \times 2^8
\]

\[
x^0_B \quad x^1_B \quad y^0_B \quad y^1_B \\
\]
\[
x_B \quad y_B \\
\]
\[
y_B = y^0_B \times 2^8 + y^1_B
\]

**Endian-aware cell-based memory model**

Cells with endianness encoding \( \varepsilon \)

\[
\langle V, o, \tau, \varepsilon \rangle \in Cell \subseteq V \times \mathbb{N} \times \text{scalar-type} \times \{ L, B \}
\]
Semantics
Lifting ( endian-aware) simple program semantics to ( endian-diverse) double programs

Simple programs \( P_\alpha \) \( \alpha \in \{ L, B \} \)

Simple states in \( E_\alpha \) ( environments over cells)
Statements \( S_\alpha[ s ] \in \mathcal{P}(E_\alpha) \rightarrow \mathcal{P}(E_\alpha) \)
Expressions \( E_\alpha[ e ] \in E_\alpha \rightarrow \mathcal{P}(\forall) \)
Lifting (Endian-aware) simple program semantics to (Endian-diverse) double programs

<table>
<thead>
<tr>
<th>Simple programs $P_\alpha$</th>
<th>$\alpha \in {L, B}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple states in $E_\alpha$ (environments over cells)</td>
<td>$S_\alpha[s] \in \mathcal{P}(E_\alpha) \rightarrow \mathcal{P}(E_\alpha)$</td>
</tr>
<tr>
<td>Statements $E_\alpha[e] \in E_\alpha \rightarrow \mathcal{P}(\forall)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Double program $P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double states in $D \triangleq E_L \times E_B$ (w.l.o.g.)</td>
</tr>
<tr>
<td>Statements $D[s] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$</td>
</tr>
<tr>
<td>Conditions $F[c_L \parallel c_B] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$</td>
</tr>
</tbody>
</table>
Semantics

Lifting ( endian-aware) simple program semantics to ( endian-diverse) double programs

**Simple programs** $P_\alpha$

<table>
<thead>
<tr>
<th>Simple states</th>
<th>in $E_\alpha$ (environments over cells)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statements</td>
<td>$S_\alpha[s] \in \mathcal{P}(E_\alpha) \rightarrow \mathcal{P}(E_\alpha)$</td>
</tr>
<tr>
<td>Expressions</td>
<td>$E_\alpha[e] \in E_\alpha \rightarrow \mathcal{P}(\forall)$</td>
</tr>
</tbody>
</table>

**Simple programs** $P_\alpha$

| $\alpha \in \{L, B\}$ |

**Double program** $P$

<table>
<thead>
<tr>
<th>Double states</th>
<th>in $D \triangleq E_L \times E_B$ (w.l.o.g.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statements</td>
<td>$D[s] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$</td>
</tr>
<tr>
<td>Conditions</td>
<td>$F[c_L \parallel c_B] \in \mathcal{P}(D) \rightarrow \mathcal{P}(D)$</td>
</tr>
</tbody>
</table>

**Transfer functions**

$$D[s_L \parallel s_B] X \triangleq \bigcup_{(\rho_L, \rho_B) \in X} (S_L[s_L] \{ \rho_L \} \times S_B[s_B] \{ \rho_B \})$$

$$D[\text{if } e_L \not\triangleright 0 \parallel e_B \not\triangleright 0 \text{ then } s \text{ else } t] \triangleq D[s] \circ F[e_L \not\triangleright 0 \parallel e_B \not\triangleright 0]$$

$$\cup D[t] \circ F[e_L \not\triangleright 0 \parallel e_B \not\triangleright 0]$$

$$\cup D[\pi_L(s) \parallel \pi_B(t)] \circ F[e_L \not\triangleright 0 \parallel e_B \not\triangleright 0]$$

$$\cup D[\pi_L(t) \parallel \pi_B(s)] \circ F[e_L \not\triangleright 0 \parallel e_B \not\triangleright 0]$$
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
output(y); // y_L = y_B
```

**Invariants and cell constraints**

![Diagram showing the invariants and cell constraints](image)
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
output(y); // y_L = y_B
```

Invariants and cell constraints

\[ x^0_L = x^0_B \land x^1_L = x^1_B \]

\[ x^n_L \triangleq \langle x, n, u8, L \rangle \]

\[ x^n_B \triangleq \langle x, n, u8, B \rangle \]
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
output(y);  // y\_L \overset{?}{=} y_B
```

**Invariants and cell constraints**

\[ x_0^L = x_0^B \land x_1^L = x_1^B \]
\[ y_0^L = x_1^1 \]

\[ x_n^L \overset{\Delta}{=} \langle x, n, \text{u8}, L \rangle \]
\[ x_n^B \overset{\Delta}{=} \langle x, n, \text{u8}, B \rangle \]
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
output(y);  // y_Ł = y_Ł
```

**Invariants and cell constraints**

\[
x_Ł^0 = x_Ł^0 \land x_Ł^1 = x_Ł^1
\]

\[
y_Ł^0 = x_Ł^1
\]

\[
y_Ł^1 = x_Ł^0
\]

\[
x_B^0 \triangleq \langle x, n, u8, B \rangle
\]

\[
x_B^1 \triangleq \langle x, n, u8, \bar{B} \rangle
\]

\[
x_Ł^n \triangleq \langle x, n, u8, Ł \rangle
\]
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
output(y);  // y_L \equiv y_B
```

Invariants and cell constraints

\[
x^0_L = x^0_B \land x^1_L = x^1_B
\]
\[
y^0_L = x^1_C
\]
\[
y^1_L = x^0_C
\]
\[
x_B = 2^8 \times x^0_B + x^1_B
\]
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
output(y); // y_L ≜ y_B
```

Invariants and cell constraints

\[
x^0_L = x^0_B \land x^1_L = x^1_B \\
y^0_L = x^1_L \\
y^1_L = x^0_L \\
x_B = 2^8 \times x^0_B + x^1_B \land y_B = x_B
\]
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

output(y); // y_L = y_B
```

**Invariants and cell constraints**

\[
x^0_L = x^0_B \land x^1_L = x^1_B
\]

\[
y^0_L = y^1_L
\]

\[
y_L = x^0_B + 2^8 \times x^1_B \land y_B = x_B
\]

\[
y_L = y^0_L + 2^8 \times y^1_L
\]
Optimizing the memory model for the common case

Complex invariants \( \implies \) expressive numerical domain?

- Program invariants and cell constraints
  \[
  x^0_L = x^0_B = y^1_L \quad x^1_L = x^1_B = y^0_L \quad y_B = x_B \quad y^?_L = y_B \\
  x_L = x^0_L + 2^8 x^1_L \quad y_L = y^0_L + 2^8 y^1_L \quad x_B = 2^8 x^0_B + x^1_B \quad y_B = 2^8 y^0_B + y^1_B
  \]

- 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 \[\iff\] expressive numerical domain?

- Program invariants and cell constraints
  \[\begin{align*}
  x_0^L &= x_0^B = y_1^L \\
  x_1^L &= x_1^B = y_0^L \\
  y_B &= x_B \\
  x_1^L &= x_1^B = y_1^B \\
  y_0^L &= x_0^B = x_0^L + 2^8 x_1^L \\
  y_B &= x_B = 2^8 y_B + y_1^B
  \end{align*}\]

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

Extension of the bi-cell based memory model

- **Single** representation for two cells
  - from **different** program versions
  - holding **equal values**
  - representing **equalities**, or equalities **modulo byte-swapping**

- A bi-cell is
  \[\text{bi-cell} \triangleq \text{Cell} \cup (\text{Cell} \times \text{Cell})\]
  \[\text{bi-cell} \triangleq \text{Cell}_L \cup \text{Cell}_B\]

either a **single cell**

or a **pair of cells** holding equal values (shared bi-cell)
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
output(y); // y_L = y_B
```

### Invariants and cell constraints

\[
x_L^0 = x_B^0 \land x_L^1 = x_B^1
\]

\[
y_L^0 = x_B^1
\]

\[
y_L^1 = x_L^0
\]

\[
x_B = 2^8 \times x_B^0 + x_B^1 \land y_B = x_B
\]

\[
y_L = y_L^0 + 2^8 \times y_L^1
\]

\[
x_L^n \triangleq \langle x, n, u8, L \rangle
\]

\[
y_L \triangleq \langle y, 0, u16, L \rangle
\]
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
output(y); // y_L = y_B
```

Invariants and bi-cell constraints

- \( y^0_L = \langle x^1_L, x^1_B \rangle \)
- \( y^1_L = \langle x^0_L, x^0_B \rangle \)
- \( y_B = x_B \land x_B = \ldots \)

\[ x^n_L \triangleq \langle x, n, u8, \mathcal{L}, \mathcal{L} \rangle \]
\[ x^n_B \triangleq \langle x, n, u8, B, B \rangle \]
\[ x_B \triangleq \langle x, 0, u16, B, B \rangle \]
\[ y_L \triangleq \langle y, 0, u16, \mathcal{L}, \mathcal{L} \rangle \]
\[ y_B \triangleq \langle y, 0, u16, B, B \rangle \]
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
output(y); // y_L \equiv y_B
```

Invariants and bi-cell constraints

\[ y_0^L = \langle x_1^1, x_0^1 \rangle \]
\[ y_1^L = \langle x_0^L, x_B^0 \rangle \]

\[ y_B = x_B \land x_B = \ldots \]

Shared bi-cell synthesis
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
output(y); // y_L = y_B
```

**Invariants and bi-cell constraints**

\[
y^0_L = \langle x^1_L, x^1_B \rangle \\
y^1_L = \langle x^0_L, x^0_B \rangle \\
y_B = x_B \wedge x_B = \ldots
\]

**Shared bi-cell synthesis**

\[
\exists c : y_L = c = y_B ? \quad x_B \text{ candidate}
\]
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
output(y); // y_L = y_B
```

**Invariants and bi-cell constraints**

\[
y^0_L = \langle x^1_L, x^1_B \rangle \\
y^1_L = \langle x^0_L, x^0_B \rangle \\
y_B = x_B \land x_B = \ldots
\]

**Shared bi-cell synthesis**

\[
\exists c : y_L = c = y_B \land x_B \text{ candidate} \\
y_L = x_B
\]
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

output(y); // y_L = y_B
```

**Invariants and bi-cell constraints**

\[ y^0_L = \langle x^1_L, x^1_B \rangle \]
\[ y^1_L = \langle x^0_L, x^0_B \rangle \]
\[ y_B = x_B \land x_B = \ldots \]

**Shared bi-cell synthesis**

\[ \exists c : y_L = c = y_B \] ? \[ x_B \] candidate
\[ y_L = x_B \] ?
\[ y^0_L = x^1_B ? \checkmark \]
\[ y^1_L = x^0_B ? \checkmark \]
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
output(y); // y_L ? y_B
```

**Invariants and bi-cell constraints**

\[
\begin{align*}
    y^0_L &= \langle x^1_L, x^1_B \rangle \\
    y^1_L &= \langle x^0_L, x^0_B \rangle \\
    y_B &= x_B \wedge x_B = \ldots
\end{align*}
\]

**Shared bi-cell synthesis**

\[
\exists c : y_L = c = y_B ? \quad \text{x_B candidate}
\]

\[
y_L = x_B \quad ? \checkmark
\]
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
output(y); // y_L = y_B
```

**Invariants and bi-cell constraints**

\[ y^0_L = \langle x^1_L, x^1_B \rangle \]
\[ y^1_L = \langle x^0_L, x^0_B \rangle \]
\[ y_B = x_B \land x_B = \ldots \]

**Shared bi-cell synthesis**

\[ \exists c : y_L = c = y_B \] \(\checkmark\) \(c = x_B\)
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
output(y); // y_L = y_B
```

Invariants and bi-cell constraints

\[ y^0_L = \langle x^1_L, x^1_B \rangle \]
\[ y^1_L = \langle x^0_L, x^0_B \rangle \]

\[ y_B = x_B \land x_B = \ldots \]

Shared bi-cell synthesis

\[ \langle y_L, y_B \rangle \] synthesized by

- pattern-matching
- simple equalities
  (symbolic propagation)
The bit-slice numerical domain

Motivating example

```c
u16 x; u8 *p = (u8 *)&x;
u8 y = input(0,255);
#if __BYTE_ORDER == __LITTLE_ENDIAN
    x = y | 0xff00;
#else
    x = (y << 8) | 0xff;
#endif
output(p[0]);
output(p[1]);
```

Program invariants

\[ x_L = \langle y_L, y_B \rangle + 65280 \]
\[ x_B = 256 \times \langle y_L, y_B \rangle + 255 \]

\[ x_0^L \iff x_0^B \]
\[ x_1^L \iff x_1^B \]

Bi-cell constraints

\[ x_0^L = \text{byte}(x_L, 0) \]
\[ x_0^B = \text{byte}(x_B, 1) \]
\[ x_1^L = \text{byte}(x_L, 1) \]
\[ x_1^B = \text{byte}(x_B, 0) \]
The bit-slice numerical domain

Symbolic predicates (inspired by Miné [2006b], Miné [2012])

```c
u16 x; u8 *p = (u8 *)&x;
u8 y = input(0,255);
# if __BYTE_ORDER == __LITTLE_ENDIAN
 x = y | 0xff00;
# else
 x = (y << 8) | 0xff;
# endif
output(p[0]);
output(p[1]);
```

Program invariants

```
byte(x_L, 0) = ⟨y_L, y_B⟩
byte(x_L, 1) = 255
byte(x_B, 0) = 255
byte(x_B, 1) = ⟨y_L, y_B⟩
```

```
x^0_L = x^0_B
x^1_L = x^1_B
```

Bi-cell constraints

```
x^0_L = byte(x_L, 0)  x^0_B = byte(x_B, 1)
x^1_L = byte(x_L, 1)  x^1_B = byte(x_L, 0)
```
Implementation

Extensions of prototype abstract interpreter

Compared to the previous version (6,700 lines of OCaml)

- 300 lines updated in the bi-cell memory domain (8%)
- 1,000 lines added for the bit-slice predicate domain
Implementation

**Patch analysis (bi-cells)**

- D.program
- D.interproc
- D.intraproc
- D.loops
- C.program
- C.interproc
- C.intraproc
- C.loops
- C.libraries
- C.Aggregates

**Endian portability analysis (bi-cells and bit-slices)**

- D.program
- D.interproc
- D.intraproc
- D.loops
- C.program
- C.interproc
- C.intraproc
- C.loops
- C.libraries
- C.Aggregates

- Sequence
- Reduced product
- Cartesian product
- Composition

- Universal
- C specific
- Double C

- U.linearRel
- U.intervals
- U.congruences
- U.bitSlices
## Benchmarks

<table>
<thead>
<tr>
<th>Origin</th>
<th>Name</th>
<th>LOC</th>
<th>Time</th>
<th>Revision</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source</td>
<td>GENEVE</td>
<td>218</td>
<td>1 s</td>
<td>2014-1, 2014-2</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2016, 2017</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>MLX5</td>
<td>125</td>
<td>155 ms</td>
<td>2017, 2020-1</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2020-2</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Squashfs</td>
<td>110</td>
<td>150 ms</td>
<td>2020-1, 2020-2</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>Industrial</td>
<td>Module S</td>
<td>300 K</td>
<td>9.7 h</td>
<td>2020</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Module A</td>
<td>1 M</td>
<td>20.4 h</td>
<td>2020, 2021</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Disclaimer:**

- Modules A and S are part of an early prototype, not in production yet.
- All findings have been incorporated into the development cycle.
1. Introduction
2. Patch analysis for numerical programs
3. Patch analysis for C and structure layout portability
4. Endian portability analysis for C programs
5. Conclusion
Contributions

**Double program semantics**
- concrete semantics for two versions
- joint analysis by induction on syntax
- double program construction algorithm
- *support for unbounded input streams*

**Bi-cell memory domain**
- symbolic relations between memories
- scalable patch analyses
- scalable portability analyses

**Numerical domains**
- bit-slice domain
- *Delta domain*
- near-linear cost

**Implementation and experimentation**
- prototype analyzer on Mopsa
- small slices of open source software
- large real-world avionics software
Future work

Industrialization
- endian portability for simulation
- non regression for product-lines

Semantic differencing
- characterize semantic differences
- infer a semantic distance
- evaluate the cost of a patch
- infer an “improvement” property

Portability analysis
- 32-bit versus 64-bit
- different 64-bit data models
- porting from x86 or PowerPC to ARM
- changes in OS data types
- Year 2038 problem
- different ranges of inputs (Ariane 5.01)

Hyperproperties and information flow
- 2-safety properties
- prove secrecy and noninterference
- experiment on more complex programs
Summary

Topics
- patch analysis
- structure layout portability analysis
- endian portability analysis

Contributions
- Double program semantics
- Bi-cell memory domain
- Numerical domains
- Implementation and experimentation

Future work
- Industrialization
- Portability analysis
- Semantic differencing
- Hyperproperties and information flow

Thank you for your attention
Questions?
Backup slides
References


