

# Coping With Implementation Dependencies

Marius Nita  
(joint with Dan Grossman & Craig Chambers)

WASP Group  
[wasp.cs.washington.edu](http://wasp.cs.washington.edu)

# Why?

- Imagine an option in your favorite C/C++ devel environment ->  
Check for portability issues ->  
Pick platform(s) to check against ->  
IA32/gcc4.0/noflags  
IA32/gcc4.0/-fpack-struct  
SPARC/.../...  
...
- It is easy to write implementation-dependent code without knowing it.
- ...and sometimes you have to.
- ...and then you're faced with having to port.

# Implementation Dependency

- **Implementation**  
= architecture + compiler/interpreter
- **Dependency**  
= an assumption about a particular implementation
- **How**
  - The language, by design, allows inspection of the underlying machine.
  - “Unspecified” holes in the spec.

# C Example #1

```
struct RGB { char r,g,b; };  
struct RGB image[1024*768];  
  
char *p = (char*)image;  
while (...) {  
    doRed(*p++);  
    doGreen(*p++);  
    doBlue(*p++);  
}
```

# C Example #1

```
struct RGB { char r,g,b; };  
struct RGB image[1024*768];
```

```
char *p = (char*)image;  
while (...) {  
    doRed(*p++);  
    doGreen(*p++);  
    doBlue(*p++);  
}
```

- Works fine on X86. 

r	g	b	r	g	b
---	---	---	---	---	---
- What if structs are 4-byte aligned?  
(E.g. ARM fetches on 4-byte boundaries)



# C Example #2

```
struct T { char x; double y; };  
struct U { double a, b; };  
  
struct U *u = malloc(...);  
struct T *t = (struct T*)u;  
... t->y ...
```

# C Example #2

```
struct T { char x; double y; };  
struct U { double a, b; };  
  
struct U *u = malloc(...);  
struct T *t = (struct T*)u;  
... t->y ...
```

- On 64-bit SPARC, `t->y == u->b`.
- Not on X86 (doubles are 4-byte aligned).

# Caml Example

```
let f x y = ()  
let _ = f (print_string "hi")  
         (print_string "bye")
```

# Caml Example

```
let f x y = ()  
let _ = f (print_string "hi")  
         (print_string "bye")
```

- On SPARC it prints "hibye"
- On X86 it prints "byehi"
- Same Caml implementation.  
(Order of evaluation unspecified.)

# Our Work

- Formalized a small, C-like language  
(with byte-addressing, pointers, structs, padding)
- Isolates a notion of implementation  
Oracle for:
  - type size
  - type alignment
  - struct field offset
  - alignment restrictions on fetch, etc.
- From the source code, we determine a set of implementations on which the program “works.”

# Formal Model

- “Works” = “doesn’t crash” = “memory-safe”  
(may produce strange/unintended results)
- Given a program **P**:

```
let S = extract_impls(P);  
if M ⊨ S then P doesn't crash on M.
```

- We represent **S** as a logic formula (constraint).
- **M ⊨ S** means “implementation **M** is a model that makes the formula **S true.**”

# Example

```
struct T { char x[8] };  
struct U { int a,b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

# Example

```
struct T { char x[8] };  
struct U { int a,b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

# Example

```
struct T { char x[8] };  
struct U { int a,b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

```
S = ST(trans(struct T*),trans(struct U*))
```

# Example

```
struct T { char x[8] };  
struct U { int a,b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

S = ST(trans(struct T\*), trans(struct U\*))

X ⊨ ST(trans(struct T\*), trans(struct U\*)) ??

# Example

```
struct T { char x[8] };  
struct U { int a,b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

$S = \text{ST}(\text{trans}(\text{struct T}^*), \text{trans}(\text{struct U}^*))$

$X \models \text{ST}(\text{trans}(\text{struct T}^*), \text{trans}(\text{struct U}^*))$  ??

$X.\text{trans}(\text{struct T}^*) \leq X.\text{trans}(\text{struct U}^*)$  ??

# Example

```
struct T { char x[8] };  
struct U { int a,b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

```
S = ST(trans(struct T*),trans(struct U*))  
X ⊨ ST(trans(struct T*),trans(struct U*)) ??  
X.trans(struct T*) ≤ X.trans(struct U*) ??  
ptr[1](byte[8]) ≤ ptr[4](byte[4] byte[4]) ??
```

# Example

```
struct T { char x[8] };  
struct U { int a,b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

```
S = ST(trans(struct T*),trans(struct U*))  
X ⊨ ST(trans(struct T*),trans(struct U*)) ??  
X.trans(struct T*) ≤ X.trans(struct U*) ??  
ptr[1](byte[8]) ≤ ptr[4](byte[4] byte[4]) ??
```

No: Can't cast from unaligned to 4-byte aligned.

# Example

```
struct T { char x; int y; };  
struct U { int a; int b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

```
S = ST(trans(struct T*), trans(struct U*))  
X ⊨ ST(trans(struct T*), trans(struct U*)) ??  
X.trans(struct T*) ≤ X.trans(struct U*) ??  
ptr[4](byte[1] pad[3] byte[4])  
    ≤ ptr[4](byte[4] byte[4]) ??
```

# Example

```
struct T { char x; int y; };  
struct U { int a; int b; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

```
S = ST(trans(struct T*), trans(struct U*))  
X ⊢ ST(trans(struct T*), trans(struct U*)) ??  
X.trans(struct T*) ≤ X.trans(struct U*) ??  
ptr[4](byte[1] pad[3] byte[4])  
  ≤ ptr[4](byte[4] byte[4]) ??
```

No: Can't cast from padding to data.

# Example

```
struct T { double a; };  
struct U { char x; short y; };
```

```
struct T *t = malloc(...);  
struct U *u = (struct U*)t;
```

```
S = ST(trans(struct T*), trans(struct U*))  
X ⊨ ST(trans(struct T*), trans(struct U*)) ??  
X.trans(struct T*) ≤ X.trans(struct U*) ??  
ptr[4](byte[8]) ≤ ptr[2](byte[1] pad[1] byte[2]) ??
```

YES

# Current Work

- Building a tool to find portability bugs in C code.
- Focusing on data layout discrepancies as in formalism.
- A set of interesting implementation descriptions is written down.
- Pointer casts are gathered from a C program.
- Casts checked against each implementation of interest.

# Future Work

- Strengthen our system to handle “full” portability.
  - Equivalence instead of memory safety.
- Design a C-like programming language that makes it impossible to write unportable code.
  - For some definition of “portable.”
- Automatic porting of code.
- The all-important menu.

# Thank You!

<http://www.cs.washington.edu/homes/marius>

<http://wasp.cs.washington.edu>