

# LLPE

Highly accurate partial evaluation for LLVM IR

Christopher Snowton

*University of Manchester (kinda)*

# LLPE

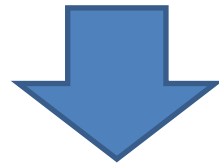
- Partial evaluator / program specialiser for LLVM IR
- Thus in effect a cross-module C/C++/FORTRAN/... PE
- Supports 99% of IR (including e.g. funky reinterpret casts)
- Specialises multithreaded programs, or those that interact with the kernel, or...

# Overview

- Quick review of PE
- LLPE features
- Practical Experience
- LLVM: a good environment for PE?
- LLPE current status

# Review: Partial Evaluation (functional)

```
f(x, y) = if x then y * 2 else y * 3
```



```
f_true(y) = y * 2
```

# Review: Partial Evaluation (imperative)

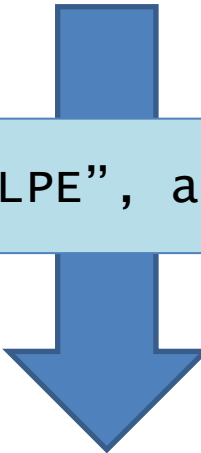
```
char get_checkbyte(const char* in, char init) {  
    char checkbyte = init;  
    for(int i = 0, ilim = strlen(in); i != ilim; ++i)  
        checkbyte ^= in[i];  
    return checkbyte;  
}
```



```
char get_checkbyte_LLPE(char init) {  
    return init ^ 'L' ^ 'L' ^ 'P' ^ 'E';  
}
```

# Review: Partial Evaluation (dealing with uncertainty)

```
char* digest(const char* in, const char* algo) {  
    if(!strcmp(algo, "MD5"))  
        return digest_md5(in);  
    else if(!strcmp(algo, "SHA1"))  
        return digest_sha1(in);  
    ...  
}
```



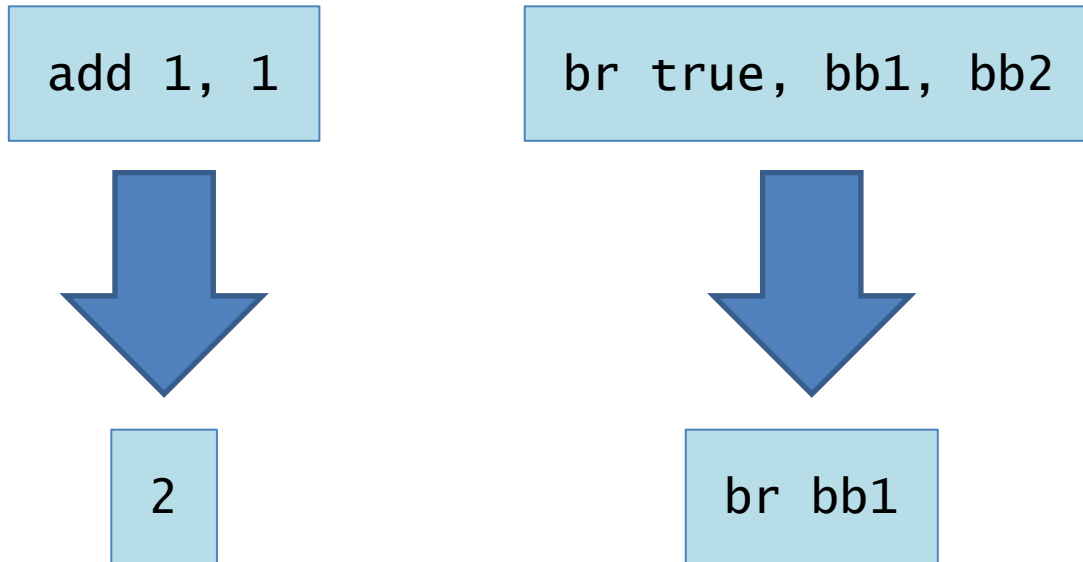
in = "LLPE", algo = ?

# Review: Partial Evaluation

- Inline functions
- Peel loop iterations
- What to do when control flow is uncertain during specialisation?
  - Specialise both paths?
  - Stop specialising?
  - Ask the user?

# LLPE: Features

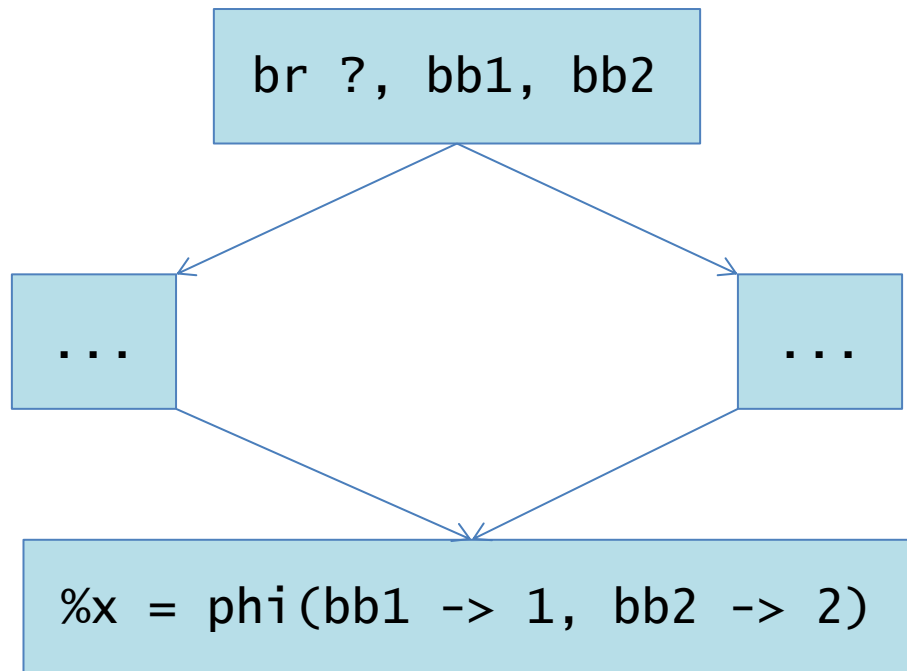
(Arithmetic and basic control)





# LLPE: Features

## (Uncertain control flow)



- `%x` gets a set value `{1, 2}`
- Can't emit this in the specialised program...
- ...but can use it to guide further specialisation

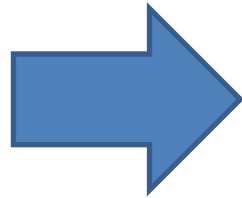
# LLPE: Features

## (Calls and loops)

- Non-recursive calls: always analyse in context (maximally context sensitive)
- Potentially-unbounded loop or recursion: analyse in context **if** we are certain to **enter** the loop/call given the **specialisation conditions**
- Otherwise find a fixed point solution.

# LLPE Features (Memory)

```
alloca i32  
call malloc(...)  
@0 = i32 0
```



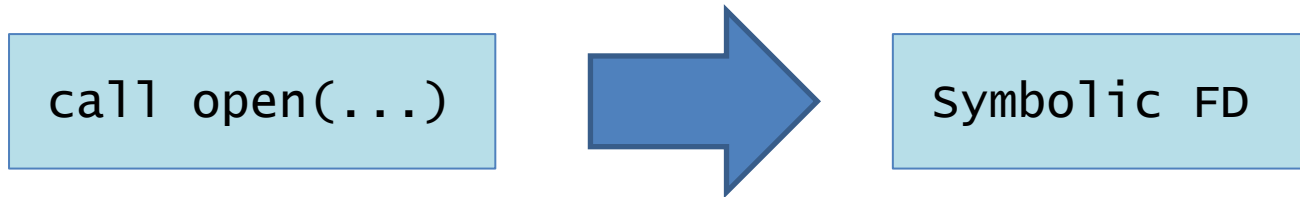
Symbolic pointer

- Symbolic pointer arithmetic, comparison
- Casts to integer types and back
- Arbitrary pointer casts (e.g. i64 -> [i8 x 8])
- Can't examine pointer bytes

# LLPE Features (Memory)

- Loads from symbolic pointers (or sets of symbolic pointers) can be resolved during specialisation
- Stores write to symbolic memory; eliminated if all reading loads are eliminated
- Symbolic memory merged at control-flow merge points

# LLPE Features (I/O)



- Results in a runtime check: file unchanged?
- Check fails -> branch to unmodified code

# LLPE Features

## (Threads and Processes)

- Volatile or memory-ordering attributes indicate potential for concurrent interference:
  - By another thread
  - By another process
  - By a signal handler
- By default, continue specialising but tag all memory as **tentative**.
- Tentative loads require a **runtime check**

# LLPE Features (limitations)

- Broad IR support
- Can't specialise across a throw -> catch
  - Exception propagation introspects on the binary
- Can't specialise across inter-thread communication (only tolerate its incidental presence)
- Can't specialise across ASM sections with unbounded side-effects (but rare)

# LLPE: Experiences

- Specialised Nginx with respect to an XSLT document
  - Effectively “pre-compiling” part of the transform
  - 30% speedup when requesting a doc using that XSLT sheet; negligible impact when requesting a different document
- Pared Nginx by “baking in” a particular config
  - Reduced binary size by 30%



# LLPE: Experiences

- Efficiency (time, space) a concern
  - Around 10,000 : 1 slowdown relative to conventional execution, but unoptimised
- User assistance minimal
  - Around 20 directives to successfully specialise Nginx
    - Annotating TLS
    - Annotating bounded loops

# LLPE: Experiences

- Still “hits the wall” when confronted with a write-through-unknown-pointer
- Value sets and vague pointers designed to minimise writes through unknown

# LLVM: A Good Place to PE

- Fully-linked image with rich semantic information
- Dodges shortcomings of C language
  - Implementation-defined behaviour
- Rich toolbox of primitive manipulations

# LLVM: A Bad Place to PE

- Uniqued Constants drove me to copy-paste the guts of the constant folder
- Inability to keep LoopInfo alive for many functions at once was a pain
- Would be nice to get the IR printout in structured form, for debugging
- Ultimately minor complaints; A+++ would develop again

# Current Status

- 95% feature complete
  - A few missing corner cases, e.g. mutually recursive functions
- Functioning version for LLVM 3.2
- Forward port to LLVM 3.6
- Code rationalisation; developer documentation in progress (ETA 1 month)

# www.11pe.org

- Forums / mailing lists
- Bug tracker
- Github repo