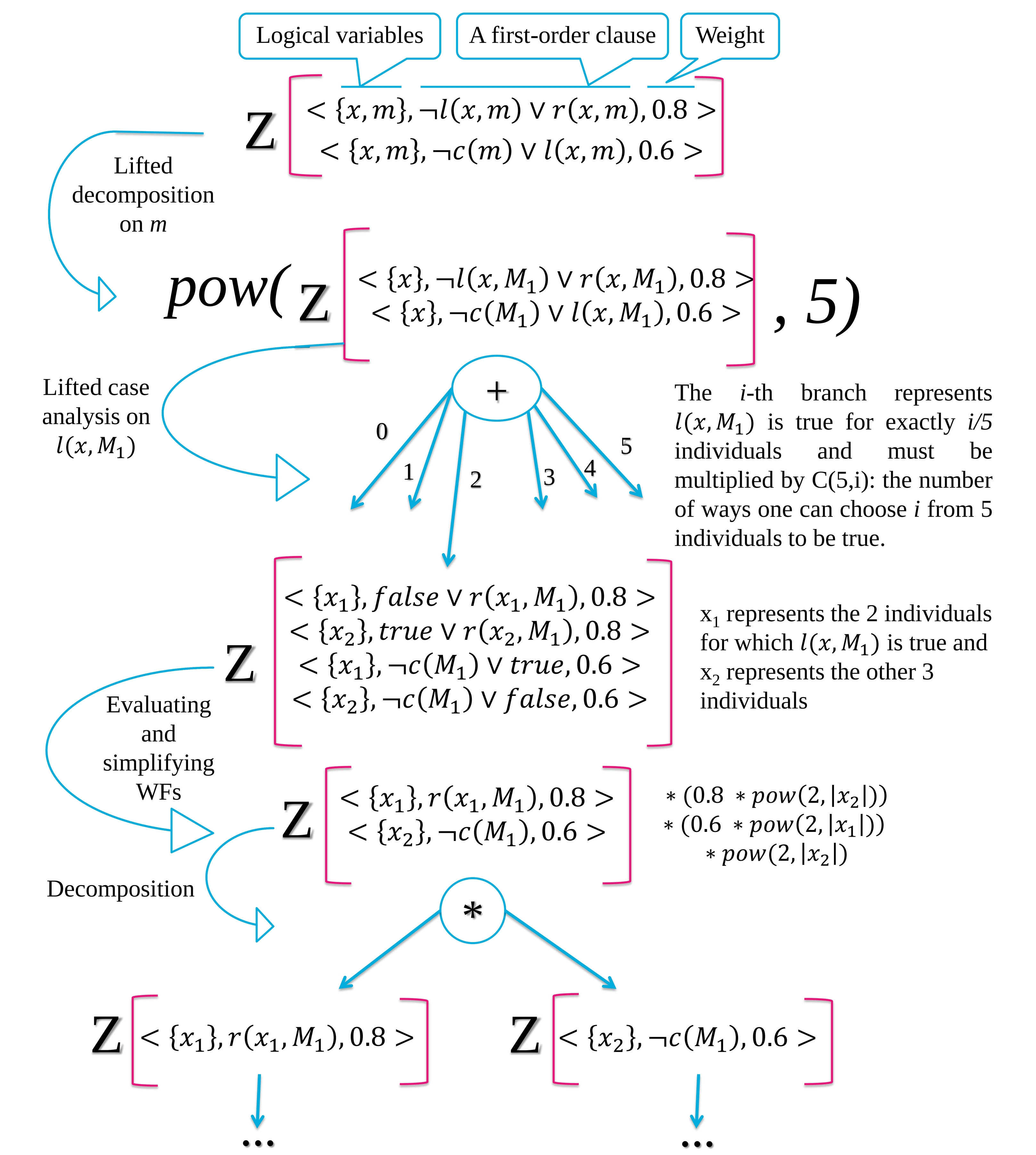
Why is Compiling Lifted Inference into a Low-Level Language so Effective?

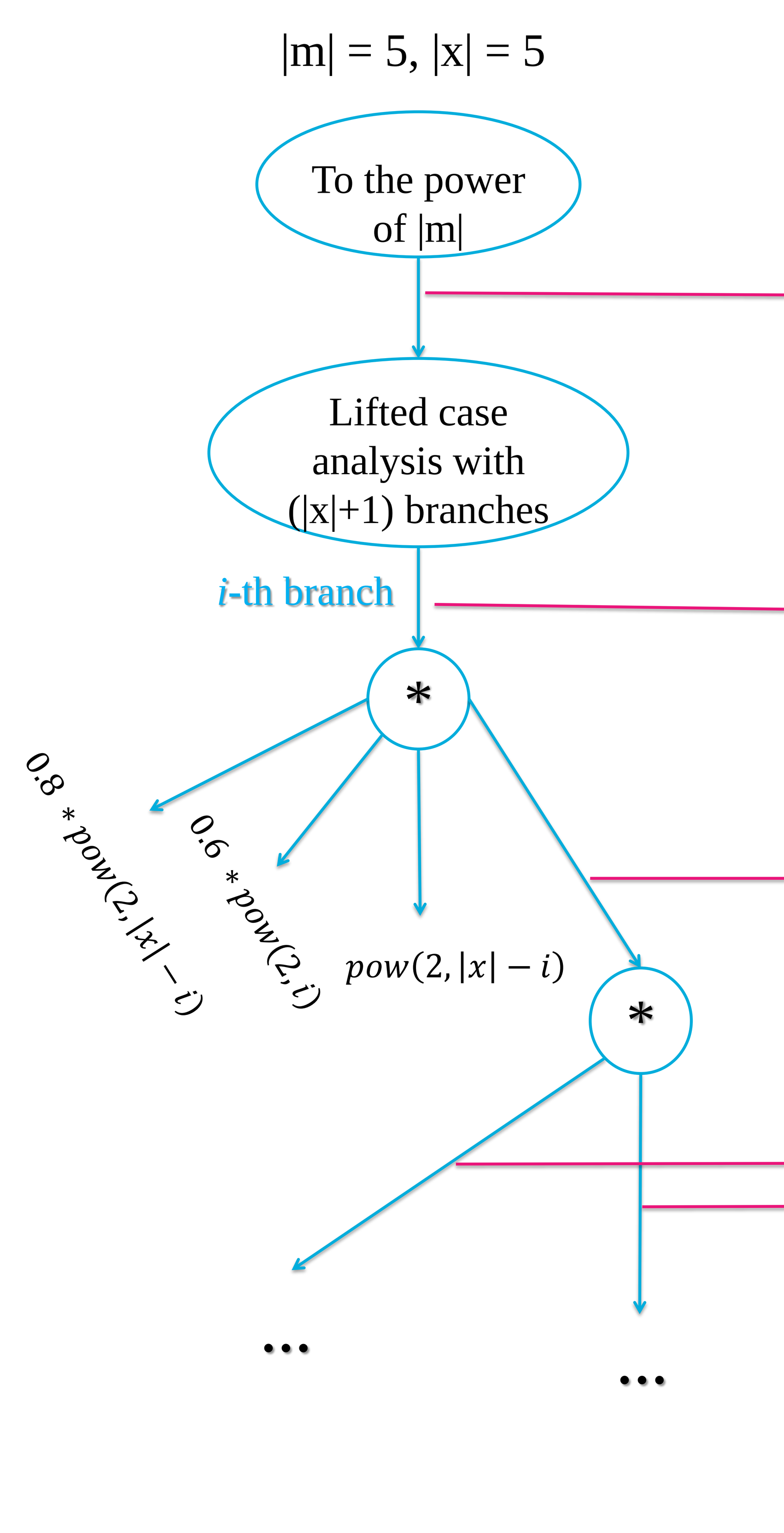
Seyed Mehran Kazemi & David Poole
http://www.cs.ubc.ca/~smkazemi https://www.cs.ubc.ca/~poole

Calculating Z by Lifted Search



Calculating Z by Knowledge Compilation

Compiling to Data Structures



Compiling to C++ Programs

```
pop_m = 5;
pop_x = 5;
```

C++ code which stores Z of the sub-graph in *v2*

```
v1 = pow(v2, pop_m);
```

```
v2 = 0;
for(int i = 0; i <= pop_x; i++){
```

C++ code which stores Z of the *i*-th sub-graph in *v3*

```
    v2 += choose(pop_x, i) * v3;
}
```

C++ code which stores Z of the rightmost sub-graph in *v4*

```
v3 = (0.8 * pow(2, pop_x - i)) *
     (0.6 * pow(2, i)) * pow(2, pop_x - i) * v4;
```

C++ code which stores Z of the leftmost sub-graph in *v5*

C++ code which stores Z of the rightmost sub-graph in *v6*

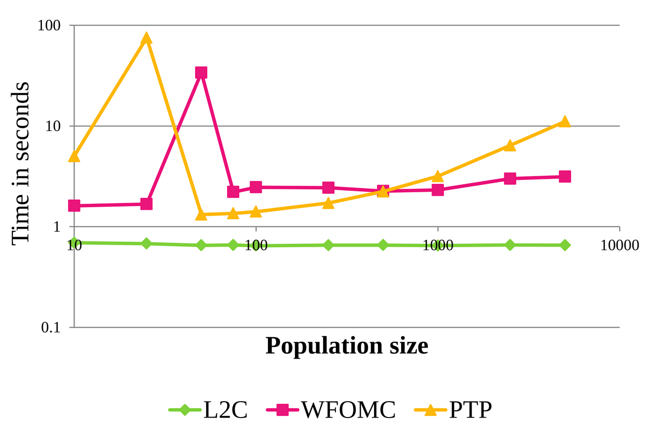
```
v4 = v5 * v6;
```

...

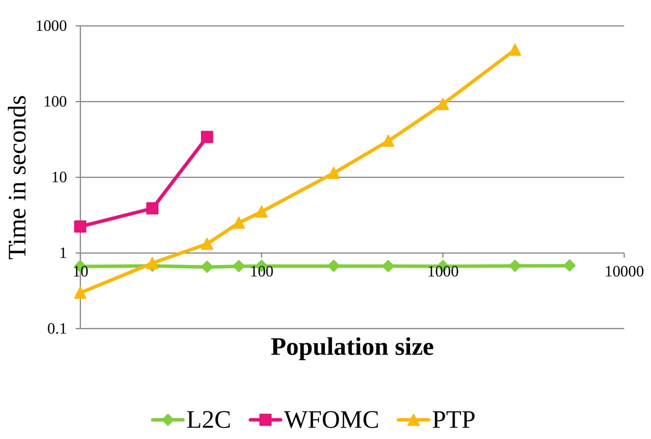
Comparing the Three Approaches (From KR-2016)

L2C (our compiler) compiles to C++ programs	WFOMC compiles to data structures	PTP uses lifted search
http://github.com/Mehran-k/L2C	https://code.google.com/archive/p/alchemy-2/	https://code.google.com/archive/p/alchemy-2/

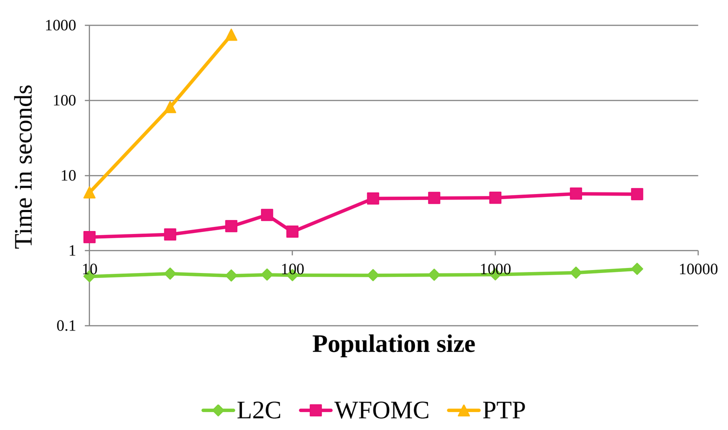
$goodProf(x) \land goodStu(y) \land advises(x, y) \Rightarrow futureProf(y)$
$coauth(x, y) \Rightarrow advises(x, y)$, varying |y|



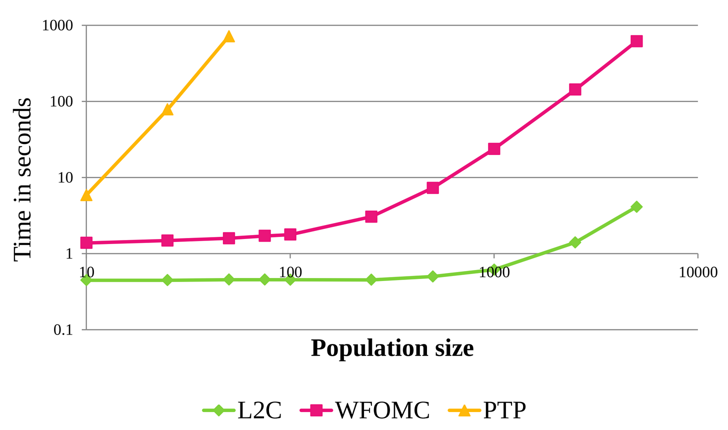
Similar to the first benchmark but varying |x| and |y| at the same time



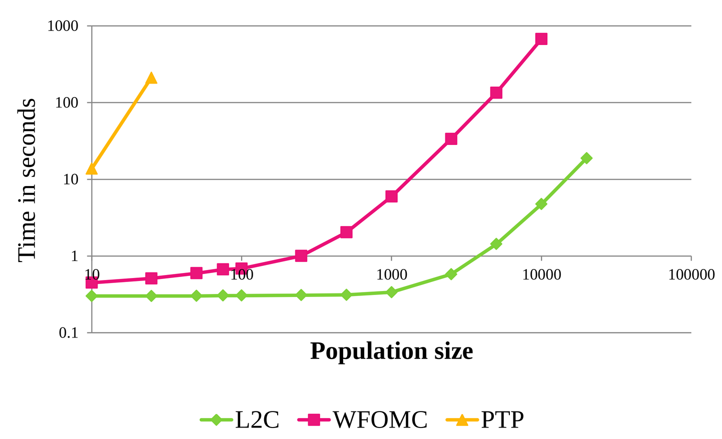
$a(x) \land b(y) \quad a(x) \land c(x) \quad b(y) \land d(y)$
$c(x) \land d(y) \quad e \land d(y)$, varying |y|



Similar to the third benchmark but varying |x| and |y| at the same time



$a(x) \land b(x) \land c(x, y) \land d(y) \land e(y) \land f$
varying |x| and |y| at the same time



Why is Compiling to Low-Level Languages more Efficient?

For a theory, lifted inference requires the same operations using lifted search (LS), compile to a data structure (DS), or to a low-level program (LP).

Question: Why are the runtimes so different if the operations are the same?

Here's how the three approaches can be viewed:

LS: Operations —Interpreter→ Result

DS: Operations —Compiler→ Low-level operations (The data structure) —Interpreter→ Result

LP: Operations —Compiler→ Low-level operations (The C++ program) —Compiler Optimizer→ More Low-level and optimized operations (Machine code) —Execute→ Result

Hypothesis: The extra compilation & optimization steps speedup the reasoning

Validation: Suppose for the LP we run the C++ programs using an interpreted instead of a compiler:

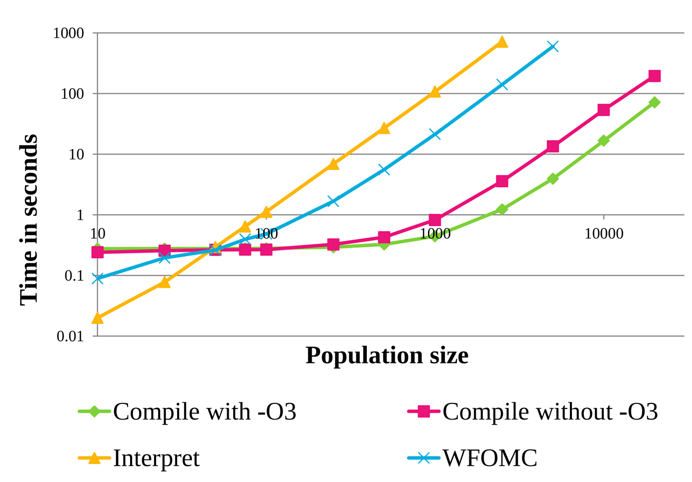
Operations —Compiler→ Low-level operations (The C++ program) —Interpreter→ Result

If the extra compilation & optimization steps are the reason behind the speedup, interpreting the C++ programs must perform similarly as compiling to data structures.

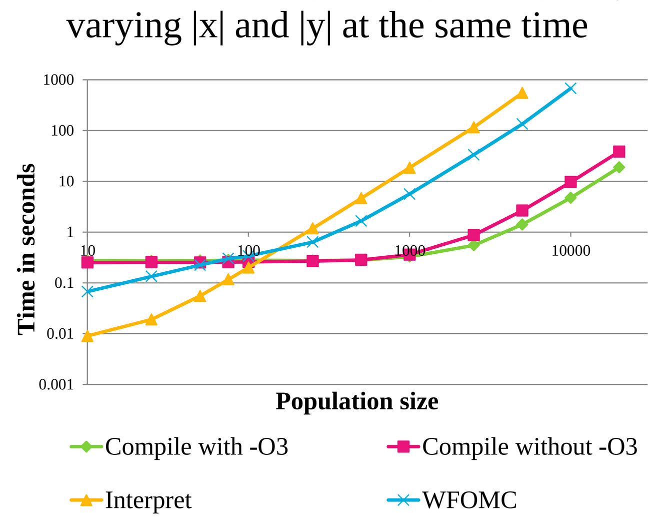
Test: See the results on the right.

- Comparing 4 approaches: 1- compile to C++, then compile and optimize the C++ programs (-O3 is the optimizer)
  2- compile to C++, then compile the C++ programs without optimization
  3- compile to C++, then interpret the C++ programs
  4- compiling to data structures and executing the data structure (WFOMC).
- Compilation into target circuits/programs takes approximately the same time, so is excluded.

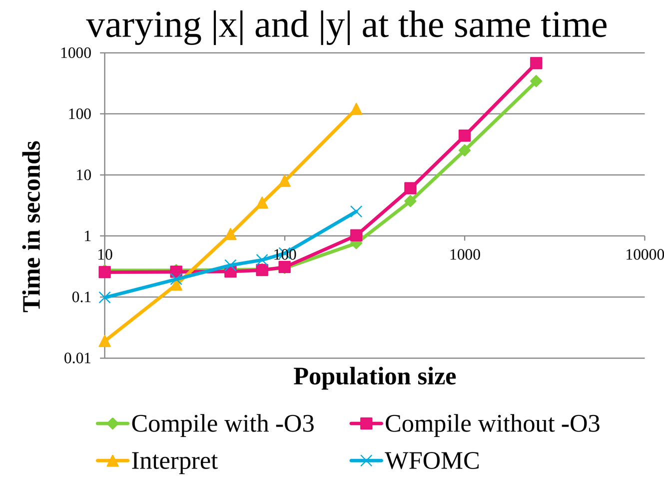
$a(x) \land b(y) \quad a(x) \land c(x) \quad b(y) \land d(y)$
$c(x) \land d(y) \quad e \land d(y)$, varying |x| and |y|



$a(x) \land b(x) \land c(x, y) \land d(y) \land e(y) \land f$
varying |x| and |y| at the same time



$a(x) \land b(x) \land c(x) \land d(x, y) \land e(y) \land f(y) \land g(y) \land h$
varying |x| and |y| at the same time



When |x|=500, neither Interpret nor WFOMC completed in 1000s.

- For the above three benchmarks:
  - Compiling the compiled operations offers an average of **175x** speedup compared to interpreting them.
  - Compiling & optimizing the compiled operations offers an average of **2.3x** speedup compared to only compiling the compiled operations.

References

1) Seyed Mehran Kazemi and David Poole. *Knowledge compilation for lifted probabilistic inference: Compiling to a low-level language* , In KR-2016.
2) Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. *Lifted probabilistic inference by first-order knowledge compilation.* In IJCAI-2011.
3) Vibhav Gogate and Pedro Domingos. *Probabilistic theorem proving*. In UAI-2011.