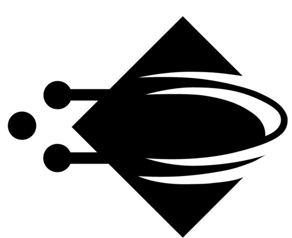


IT University of Copenhagen

Relational Reasoning for Programs using Higher-Order Store

PhD student: **Aleš Bizjak** *Supervisor*: **Lars Birkedal**

IT University of Copenhagen

Microsoft Research

When is a program transformation correct?

A program transformation should preserve the meaning of the source program. Consider an ML-like language (i.e. call by value language with effects) and a compiler optimization that replaces

```
let r = g (f 13) (f 13) in r * r
```

with

```
let x = f 13 in let r = g x x in r * r
```

This transformation is not sound in general, as the function f could, for example, increase a global counter, write to a file, rely on a random number generator or fire a missile.

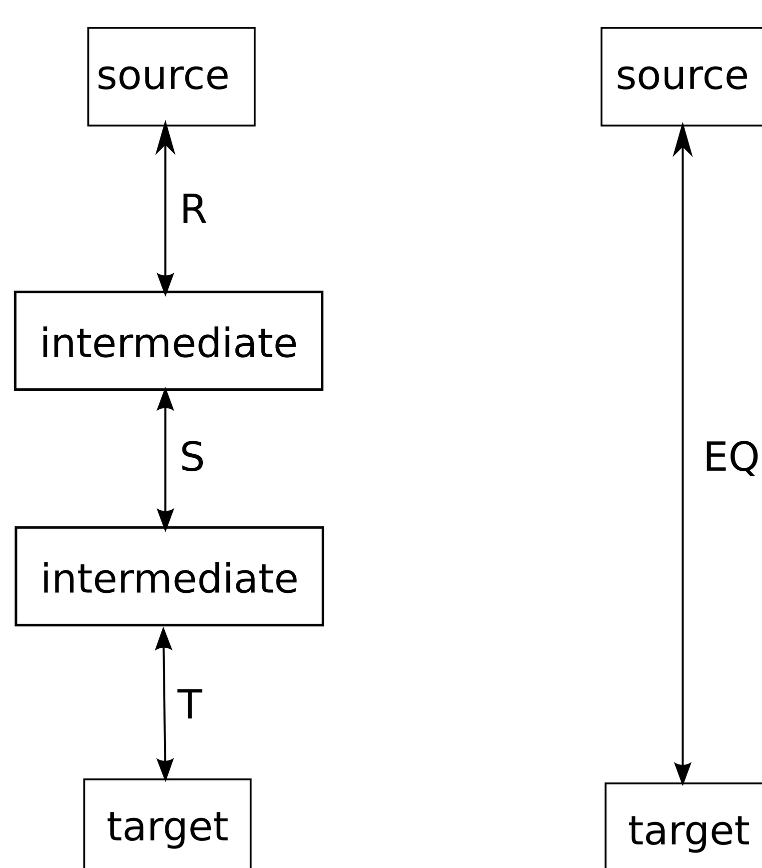
In general, a program transformation transforms an expression e to e'. This is sound if e is contextually equivalent to e', meaning that for any program $C[e]$ has the same *the observable behavior* as $C[e']$.

It is difficult to prove contextual equivalence directly from the definition, even for simple programs, and the main difficulty stems from the fact that a context can bind free variables of an expression in nontransparent ways.

When is a compiler correct?

Another desirable property of a compiler is that the program it produces is *observationally equal* to the source program. Here, we need a relation between source and target languages that reifies our intuitive notion of equivalence. In the absence of computational effects and at ground types, such a relation is relatively easy to specify (e.g. if two programs produce integers they should be equal if, when run, they produce the same value), but for modular reasoning (e.g. to enable linking) we need to extend it to higher types.

Suppose we have a multi-stage compiler that first transforms the source language to an intermediate language, (possibly) does some program transformations on the intermediate language and then generates some target language code. We want a relation between source and intermediate language, R, that reflects a notion of equivalence and that enables us to prove that the first part of the compiler generates correct code. Then we need some approximation of contextual equivalence, S, that enables us to reason about the correctness of optimizations and, finally, we need a relation, T, that enables us to reason about the final phase of the compiler.



In addition, we would also like to have a relation between the source and target languages, EQ, that captures when a target program *realizes* the source program. EQ is needed because we would like to combine code, generated by different compilers, in the final program (imagine using a C# implemented library in an F# program).

Recent and ongoing work, the foundations

Developing suitable relations for increasingly rich languages has been the focus of a lot of recent research. One popular and useful technique has been the method of *logical relations*. It leverages *types* to build a family of relations. It consists, roughly, of giving, for each *type*, a relation on some set of realizers that expresses the properties we wish expressions of that type to have. The adjective *logical* comes from the property that the family is not an arbitrary type-indexed family of relations, but, loosely speaking, admits induction on the type structure; it is a lifting of relations on ground types to higher types. Logical relations have been applied to reasoning about contextual equivalence for ML-like languages with higher-order store, recursive types and impredicative polymorphism. Specifically, accommodating higher-order store has led to the development of Kripke models over recursive worlds. Logical relations have also been applied to proving correctness of simple, single-stage compilers. [1, 4, 3]

The future

The equivalences we can prove depend very much on what properties we can express using the types; having a richer type system enables more refined reasoning. For instance, in the starting example on the left, if the type system would be able to express that the function f is *pure*, we could make the transformation.

Such properties can be expressed using effect annotations. Typing judgments are thus extended with *effect* and *regions*:

$$\underbrace{\Pi}_{\text{set of regions}} \mid \overbrace{\Gamma}^{\text{environment}} \vdash \underbrace{e}_{\text{expression}} : \overbrace{\tau}^{\text{type}} \mid \underbrace{\varepsilon}_{\text{set of effects}} .$$

The set of effects describe *what* could possibly happen by running the computation and the set of regions describes *where* it could happen.

Soundness of program transformations based on effects can be proven using logical relations. An example of this approach, where effects are only `read`, `write` and `allocate`, is a recent paper [2] showing a "parallelization theorem" for an ML-like language with concurrency primitives.

Extending this approach to a richer language with a richer set of effects, i.e. more realistic language, is a possible direction.

Another possible direction is investigating methods for reasoning about multi-stage compilers. The methods that have been developed do not provide a good way to do so. In the example, the ideal would be that the relation EQ would simply be the composition of relations R, S and T, but the composition of logical relations in not well behaved. The challenge, then, is to find definitions of well-behaved relations R, S, T and EQ, such that, at least,

$$R \circ S \circ T \subseteq EQ$$

holds.

References

[1] Nick Benton and Chung-Kil Hur. Realizability and compositional compiler correctness for a polymorphic language. Technical report, MSR-TR-2010-62, 2010.

[2] Lars Birkedal, Filip Sieczkowski, and Jacob Thamsborg. A concurrent logical relation, 2012. Submitted for publication.

[3] Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. Realisability semantics of parametric polymorphism, general references and recursive types. *Mathematical Structures in Computer Science*, 20(4):655–703, 2010.

[4] Jacob Thamsborg and Lars Birkedal. A kripke logical relation for effect-based program transformations. In *ICFP*, pages 445–456, 2011.