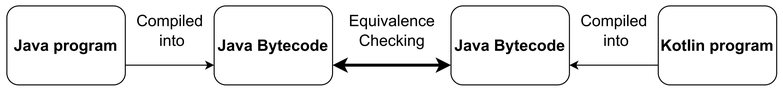
From Java to Kotlin with Contextual Equality Saturation

Alexandre Drewery (Inria)
Supervisors: Thomas Jensen (Inria) – David Pichardie (META)

1. Motivation

New programming languages appear → need for codebase migration
Millions of line of codes to migrate + the translation **must be safe**

Perform automatic translation validation: **increased speed** and **confidence**
Java to Kotlin migration: compiled Java bytecodes must be **equivalent**



2. Experimentation with existing tool

Peggy [1]: a rewriting tool for Java bytecode equivalence checking
Can Peggy prove equivalence for **atomic** Java program + **automatic** Kotlin translation with its base rewriting rules?

Program main feature	Observed difference in programs	Peggy-proved
Arithmetic operations	Permutation of registers used and optimization	Yes
Conditional structure	If-statement vs If-expression	Yes
Constant foldable if	Optimization: removed conditional dead branch	Yes
Dead code (repeated if)	If-statement vs If-expression	Yes
Basic loop for on *int*	**Different control flow and loop exit condition**	**No**
Loop while	Permutation of registers used	Yes
Increment-by-two loop for	Converted into a while, permutation of registers	Yes
Recursion	If-statement vs If-expression	Yes
Infinite while-loop	Kotlin bytecode has no return instruction	No
Calling another function	Optimization: suppressed an useless local variable	Yes
Objects, accessing field	Field access is a method in Kotlin	No
Uninitialized local/field	Different covention on field initialization	Yes
Iterating over array	Permutation of registers used and operations order	Yes

Challenge: the loop *for* structure

```
#in Java
i = i0
while (i <= final) {
    loop body
    increment i
}
```

```
#in Kotlin
i = i0
if (i <= final) {
    loop body
    while (i != final){
        increment i
        loop body} }
```

More rewriting rules → **can prove equivalence** but **longer proofs, worse performances**

Our idea: **improving the rewriting technique with a new kind of rewriting rules**

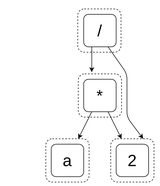
3. Background: Equality Saturation

Peggy's method: **Equality Saturation** on a term representation of programs
Initial term t + rewriting rules → set of terms equivalent to t
Iteratively apply all possible rules on growing equivalence classes of terms

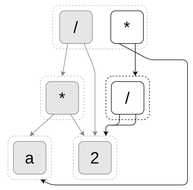
The e-graph data structure [2]

- Heart of Equality Saturation, from the SMT term management core
- Represents simultaneously a subterm relation and an equivalence relation on terms
- Supports the powerful **e-matching** operation to look for pattern in the set of terms

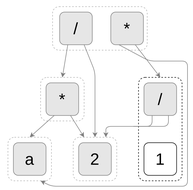
An example of equality saturation on arithmetical terms



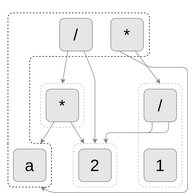
E-graph for initial term $div(mult(a, 2), 2)$
Looks like a term tree
Each node (*e-node*) is in an equivalence class of terms (*e-class)*, edges point towards e-classes



Applying rule $div(mult(A, B), C) \rightleftharpoons mult(A, div(B, C))$
This adds a new term $mult(a, div(2, 2))$ to the top e-class
New e-nodes are created, e-class are merged



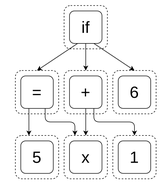
Applying rule $div(A, A) \rightleftharpoons 1$
Adds a new e-class and e-node for 1 and merges it with e-node $div(2, 2)$



Applying rule $mult(A, 1) \rightleftharpoons A$
The e-class of a and the e-class representing $mult(a, 1)$ are merged
The e-node of our initial term is in the same e-class as the e-node for a
We proved the term equivalent to a

4. Our approach: Contextual Equality Saturation

Limit of Equality Saturation : rewriting rules are **local**
Example: rule $if(equal(A, B), A, C) \rightleftharpoons if(equal(A, B), B, C)$ cannot be applied immediately on the following e-graph:



Current solution: more rules to "swap" *if* and *plus* → long proofs, bad performances

In this work we introduce **contextual rewriting rules**: a context (boolean pattern) + a rewriting rule

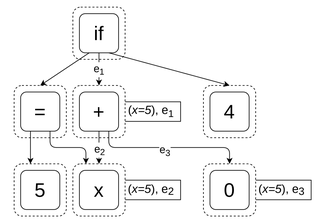
$$equal(A, B) \vdash A \rightleftharpoons B$$

"We can replace A by B under a term under which context $equal(A, B)$ is true"

5. Algorithms for Contextual Equality Saturation

Annotate each e-class with true contexts via **dataflow static analysis** on the **e-graph structure**

Applying a contextual rewriting rule
1. E-match for e-class that has context annotation + pattern
2. Add rewritten term and merge e-classes
3. Update annotations



Done: contextual equality saturation algorithms are designed
Ongoing: experimentation for comparison with state of the art
Todo: large scale translation validation

6. References

[1] R. Tate, M. Stepp, Z. Tatlock, and S. Lerner. Equality saturation: A new approach to optimization. *LMCS*, 7, mar 2011.

[2] M. Willsey, C. Nandi, Y. Wang, O. Flatt, Z. Tatlock, and P. Panchekha. egg: Fast and extensible equality saturation. *In Proc. of the ACM on Programming Languages*, 5:1–29, jan 2021.