Translating First-Order Predicate Logic to Relation Algebra, an Implementation

Anthony Brogni & Sebastiaan J.C. Joosten

Motivation

- There are some really cool Relation Algebra (RA) tools, but:
  - Some people refuse to use them
  - They can be hard to compare to First-Order Logic (FOL) tools

Our Solution

- Implement a translation between FOL and RA!
  - Standalone Python tool
    - Allows for many use cases
    - Easily Reusable
  - Thoroughly tested using Z3

Background

- Adaptation of Yoshiki Nakamura’s procedure
    - *Expressive Power and Succinctness of the Positive Calculus of Relations*
        - Published in the proceedings of RAMiCS 2020

Background

The Language for FO3

An FO3 formula ϕ is a formula over the following language, where $a \in \mathcal{A}$ is from a set of binary predicate symbols, x, y are from a set of three variables, and $D \in \mathcal{D}$ is from a set of sorts:

$$\phi, \psi \in \mathrm{FO3} = a(x, y) \mid x = y \mid \mathrm{t} \mid \mathrm{f} \mid \phi \vee \psi \mid \phi \wedge \psi \mid \exists x \in D.\ \phi \mid \forall x \in D.\ \phi \mid \neg\phi$$

- Closed FO3 formulas
- Type of an occurrence of a variable
- Homogeneous vs Heterogeneous

Background

The Language for RA

An RA formula ϕ is a formula over the following language, where a is from $\mathcal{A}$, and D_1, D_2 from $\mathcal{D}$:

$$\phi, \psi \in \mathrm{RA} = a \mid \boldsymbol{T}[D_1, D_2] \mid \mathbf{0}[D_1, D_2] \mid \mathbf{1}[D_1] \mid \phi \cup \psi \mid \phi \cap \psi \mid \phi \circ \psi \mid \phi \dagger \psi \mid \overline{\phi} \mid \phi^{-1}$$

- Well-typed RA formulas

– Occurrences of $\phi \cup \psi$ and $\phi \cap \psi$ satisfy $d(\phi) = d(\psi)$

– Occurrences of $\phi \circ \psi$ and $\phi \dagger \psi$ satisfy $d_2(\phi) = d_1(\psi)$

Background

The Interpretation Function

To describe the semantics of FO3 and RA, we use an interpretation function $\mathcal{I}$ that takes a closed formula in FO3, a formula in RA, or a domain, and produces a Boolean, a set of pairs, or a set, respectively. An interpretation $\mathcal{I}$ over a universe $\mathcal{U}$ maps each expression to a subset of the Cartesian square $\mathcal{U}^2 = \mathcal{U} \times \mathcal{U}$ such that:

$$\mathcal{I}(\mathbf{0}) = \emptyset$$

$$\mathcal{I}(\boldsymbol{T}) = \mathcal{U}^2$$

$$\mathcal{I}(\mathbf{1}) = \{(x, x) \mid x \in \mathcal{U}\}$$

$$\mathcal{I}(\overline{\phi}) = \{(x, y) \in \mathcal{U}^2 \mid (x, y) \notin \mathcal{I}(\phi)\}$$

$$\mathcal{I}(\phi^{-1}) = \{(x, y) \mid (y, x) \in \mathcal{I}(\phi)\}$$

$$\mathcal{I}(\phi \cap \psi) = \{(x, y) \mid (x, y) \in \mathcal{I}(\phi) \wedge (x, y) \in \mathcal{I}(\psi)\}$$

$$\mathcal{I}(\phi \cup \psi) = \{(x, y) \mid (x, y) \in \mathcal{I}(\phi) \vee (x, y) \in \mathcal{I}(\psi)\}$$

$$\mathcal{I}(\phi \circ \psi) = \{(x, y) \mid \exists z.\ (x, z) \in \mathcal{I}(\phi) \wedge (z, y) \in \mathcal{I}(\psi)\}$$

$$\mathcal{I}(\phi \dagger \psi) = \{(x, y) \mid \forall z \in \mathcal{U}.\ (x, z) \in \mathcal{I}(\phi) \vee (z, y) \in \mathcal{I}(\psi)\}$$

The Translation Process

Translation Steps:

1. Negation Normal Form

2. “Good” translation

3. “Nice” translation

4. Final step

The Translation Process

$$\boxed{\forall\ x,y.\ \exists\ z.\ (\neg\ (a(x,z)\ \wedge\ b(z,x))\ \wedge\ c(x,y))}$$

Translation Steps:

1. Negation Normal Form $\forall\ x,y.\ \exists\ z.\ ((\neg a(x,z)\ \vee\ \neg b(z,x))\ \wedge\ c(x,y))$
2. "Good" translation $\forall\ x,y.\ (\exists\ z.\ \neg a(x,z)\ \wedge\ c(x,y))\ \vee\ (\exists\ z.\ \neg b(z,x)\ \wedge\ c(x,y))$
3. "Nice" translation $\forall\ x,y.\ ((\exists\ z.\ \neg a(x,z))\ \wedge\ c(x,y))\ \vee\ ((\exists\ z.\ \neg b(z,x))\ \wedge\ c(x,y))$
4. Final step $\mathbf{0}\ \dagger\ ((C \cap -A \circ T)\ \cup\ (C \cap -B^{-1} \circ T))\ \dagger\ \mathbf{0}$

Repeated Arguments

$$R(x, x) \equiv \exists y.\, R(x, y) \wedge x = y$$

This translates $R(x, x)$ into RA as follows:

$$(R \cap \mathbf{1}) \circ \boldsymbol{T}$$

Types can be added to $\mathbf{1}$ and $\boldsymbol{T}$ according to the type of the occurrence of x.

Heterogeneous Relation Algebra

- Checking for well-typed RA formulas

```
class Typed_Union:
    """ This class describes the union between two typed relations arg1 and arg2 """

    def __init__(self, arg1, arg2):
        if arg1.type() != arg2.type():  # Type checking
            raise Exception(f'ERROR: Union type mismatch! Type 1 is:{arg1.type()} and Type 2 is:{arg2.type()}')
        else:
            self.argument1 = arg1
            self.argument2 = arg2

    def __str__(self) -> str:
        return f'({self.argument1}) ∪ ({self.argument2})'
```

Heterogeneous Relation Algebra (continued)

- Treat the conjunctions that depend only on a single variable separately

$$\boxed{\exists z.\phi_1(x,z) \wedge \phi_2(z,y) \text{ translates to } P_1 \circ P_2}$$

$$\Downarrow$$

$$\boxed{\exists z.\, \phi_1(x,z) \wedge \phi_2(z) \wedge \phi_3(z,y) \text{ is translated to } P_1 \circ (P_2 \cap \mathbf{1}) \circ P_3}$$

“Additional Variables”

Take the following first-order logic expression as an example:

$$\exists x \in A . \exists y \in B . \exists z \in C . \exists w \in D .\ a(x, w)$$

Its translation would be:

$$\boldsymbol{T}_{[Left \times A]} \circ a \circ \boldsymbol{T}_{[D \times right]}$$

Implementation Details

- Python 3.11
- Pattern Matching

```
def final_translation(expression, var1, var2):
    """ This method computes the final step of the translation from FO3 into COR """
    match expression:
        case Predicate(letter=l, argument1=arg1, argument2=arg2) if arg1 == var1 and arg2 == arg1:
            return Composition(Intersection(Relation(l), IdentityRelation()), UniversalRelation())
        case Predicate(letter=l, argument1=arg1, argument2=arg2) if arg1 == var2 and arg2 == arg1:
            return Composition(UniversalRelation(), Intersection(Relation(l), IdentityRelation()))
        case Predicate(letter=l, argument1=arg1, argument2=arg2) if arg1 == var1 and arg2 == var2:
            return Relation(l)
        case Predicate(letter=l, argument1=arg1, argument2=arg2) if arg1 == var2 and arg2 == var1:
            return Converse(Relation(l))
        case ff():
            return EmptyRelation()
        case tt():
            return UniversalRelation()
        case Equals(argument1=arg1, argument2=arg2) if arg1 == arg2:
            return UniversalRelation()
        case Equals(argument1=arg1, argument2=arg2) if arg1 == var1 and arg2 == var2:
            return IdentityRelation()
        case Equals(argument1=arg1, argument2=arg2) if arg1 == var2 and arg2 == var1:
            return Converse(IdentityRelation())
        case OR(argument1=arg1, argument2=arg2):
            return Union(final_translation(arg1, var1, var2),
                         final_translation(arg2, var1, var2))
        case AND(argument1=arg1, argument2=arg2):
            return Intersection(final_translation(arg1, var1, var2),
                                final_translation(arg2, var1, var2))
        case Negation(argument=arg):
            return Complement(final_translation(arg, var1, var2))
```

Validation

- Generate random formulas
- Automated testing
- Z3-solver

```
s = z3.Solver()
s.add(z3.Not(asZ3(fo3_expression) == asZ3(final_result)))
s.set("timeout", 1000)  # If this returns an error, update the z3 module
z3result = s.check()
if z3result == z3.sat:
    print("\nZ3 found a bug! (this is bad!)")
    print(s.model())
    print("\nZ3 lhs: ", asZ3(fo3_expression))
    print("\nZ3 rhs: ", asZ3(final_result))
    print("\nZ3 constraint: ", z3.Not(asZ3(fo3_expression) == asZ3(final_result)))
    return -1
elif z3result == z3.unsat:
    print("\nZ3 proved that the round-trip returned something equivalent (this is good!)")
    return 1
else:
    print("\nZ3 timed out and returned ", z3result)
    return 0
```

Conclusions + Future work

- We can translate all of FOL with up to 3 variables (and a bit more)
  - Untyped translation
  - Typed translation
  - Readable implementation
- Future work:
  - Even nicer looking formulas
  - More than 3 variables
  - Non-binary predicates

References

Anthony Brogni and Sebastiaan J. C. Joosten. Translating first-order predicate logic to relation algebra. http://doi.org/10.5281/zenodo.7566106, 2023.

Yoshiki Nakamura. Expressive power and succinctness of the positive calculus of relations. In *International Conference on Relational and Algebraic Methods in Computer Science*, pages 204–220. Springer, 2020.