Labeled Union-Find for Constraint Factorization

Matthieu Lemerre
Université Paris-Saclay, CEA, List
Palaiseau, France
matthieu.lemerre@cea.fr

Dorian Lesbre
Université Paris-Saclay, CEA, List
Palaiseau, France
dorian.lesbre@cea.fr

Abstract

In this work-in-progress paper, we present labeled union-find, an extension of the union-find data structure where edges are annotated with labels that form a group algebraic structure. This structure allows to very efficiently represent the transitive closure of many useful binary relations, such as two-variables per equality (TVPE) constraints of the form $y = a * x + b$. We characterize the properties of labeled union-find when used to represent binary relations between variables. More specifically, we study the use of this domain in a static analysis; either to represent binary relations, or as a reduced product with non-relational abstract domains with constraint propagation; as well as the design of efficient algorithms for the join of labeled union-find structures. We believe that this structure could be used as a low-cost relational domain or decision procedure, and that it could make other relational domains more efficient by removing the need to track some variables.

CCS Concepts: • **Software and its engineering** → **Formal software verification**; • **Theory of computation** → **Abstraction**; **Equational logic and rewriting**.

Keywords: relational abstract domain, labeled union-find

1 Introduction

A natural way to represent the binary relations between any pairs of program variables is using a graph whose nodes are variables and edges are labeled using an abstraction of a relation (e.g., Miné [2002]; Pratt [1977]).

Outside specific difficulties pertaining to the precision of operator composition [Miné 2004, §5.2.3], the main problem with these weakly-relational domains is their supra-quadratic cost. Even though they are much less expensive than more expressive abstractions like polyhedra [Cousot and Halbwachs 1978], they still require a $O(n^2)$ space cost to represent all the edges between n variables. Furthermore, the transitive-closure algorithm used to compute the most precise relation between any two variables is $O(n^3)$. These supra-quadratic costs prevent performing a direct analysis of large programs. Therefore, a long line of research has been devoted to mitigating this cost, improving efficiency by forgetting the relations between some variables [Blanchet et al. 2003; Gange et al. 2021; Logozzo and Fähndrich 2008; Singh et al. 2015], sometimes at the cost of precision.

The core reason for this cost is that there may be several paths between any two variables carrying relations of different precision. Therefore, we have to store the edges on every path, and finding the most precise path requires a costly transitive closure computation. The problem would be vastly simplified if we were **in a situation where all the paths between any two variables would correspond to the same abstract relation**. This is the problem that we are currently studying, with many promising results.

2 Union-Find with a Group of Edge-Labels

Let us consider (possibly infinite) directed graphs whose edges are labeled. Formally, we have a set of *nodes* $\mathbb{N}$, a set of *edge labels* $\mathbb{L}$, and an *edge predicate* $n_1 \xrightarrow{\ell} n_2 \in \mathbb{L} \to \mathcal{P}(\mathbb{N} \times \mathbb{N})$. Now, we suppose that labels are equipped with an infix *composition operator* $\fatsemi \in \mathbb{L} \times \mathbb{L} \to \mathbb{L}$. This allows defining the notion of *label of a path*, which is the composition of labels on the edges traversed by the path. Formally, we define a path predicate $\xrightarrow{\ell}\!\!\!\!\!\rightarrow \in \mathbb{L} \to \mathcal{P}(\mathbb{N} \times \mathbb{N})$ as the transitive closure of the edge predicate, i.e., as follows:

$$\frac{n_1 \xrightarrow{\ell} n_2}{n_1 \xrightarrow{\ell}\!\!\!\!\!\rightarrow n_2}\ \textsc{Edge} \qquad \frac{n_1 \xrightarrow{\ell_1}\!\!\!\!\!\rightarrow n_2 \qquad n_2 \xrightarrow{\ell_2}\!\!\!\!\!\rightarrow n_3}{n_1 \xrightarrow{\ell_1 \fatsemi \ell_2}\!\!\!\!\!\rightarrow n_3}\ \textsc{Trans}$$

Our main requirement is that the label on every path between two nodes is unique:

$$n_1 \xrightarrow{\ell}\!\!\!\!\!\rightarrow n_2 \wedge n_1 \xrightarrow{\ell'}\!\!\!\!\!\rightarrow n_2 \implies \ell = \ell' \qquad (\textsc{HUniqueLabel})$$

These definitions imply, in many cases, that the label composition is associative; that the labels of cycles are a neutral element for composition; and that the elements appearing in cycles are invertible. Thus, the natural settings deriving from HUniqueLabel is that $\langle \mathbb{L}, \fatsemi \rangle$ is a monoid with a neutral element `id`, where some elements are invertible (using a function `inv`). In this paper, we study the case where all elements are invertible, i.e. when $\langle \mathbb{L}, \fatsemi, \texttt{id}, \texttt{inv} \rangle$ is a group.

Since the relation between any two nodes is the same on every path, any relation can be retrieved by storing a spanning forest of the original graph, which minimizes the space required. If we also try to minimize the height of the tree and intertwine edge-additions with queries, our problem becomes very similar to the dynamic connectivity problem on graphs. That problem is solved using the efficient union-find data structure [Tarjan and van Leeuwen 1984] representing trees as nodes pointing to their parent. We just add a label to the edge from a node to its parent, i.e., it is a *labeled* union-find data structure $U \in \mathbb{U} \triangleq \mathbb{N} \rightharpoonup \mathbb{L} \times \mathbb{N}$ (mapping non-root nodes to their parent through a label). It only requires $O(n)$ space to store all the $O(n^2)$ edges between n nodes, and can

find label between any pair of nodes in amortized-constant time.

There are several interesting applications of the labeled union-find structure. One is the efficient maintenance of a set of binary relations, which we describe in Section 3. Another is *map factorization*. Suppose that a *group action* $\mathcal{A} \in \mathbb{L} \to (\mathbb{V} \to \mathbb{V})$ (i.e., group homomorphism) exists that maps labels to functions from values to values; and that we have a partial valuation $v \in \mathbb{N} \rightharpoonup \mathbb{V}$ from the roots nodes to values. This partial valuation can be extended as

$$\begin{aligned} v^*[n] = v[n] \qquad & \text{if } n \in \mathrm{dom}(v) \\ \mid \mathcal{A}(\ell)(v[n_2]) \quad & \text{if } n_2 \in \mathrm{dom}(v) \wedge n \xrightarrow{\ell} n_2 \end{aligned}$$

Thus, we can represent a valuation using only the values of the root nodes; which is interesting if we have to represent different valuations, or if we need to update the values attached to variables, as explained in Section 5.

3 Abstract Group of Relations

The labeled union-find data structure is well-suited to represent binary relations between variables. In this setting, we have *variables* $x \in \mathbb{X}$ as nodes, and *abstract (binary) relations* $\mathcal{R}^\sharp \in \mathbb{R}^\sharp$ as labels. Abstract relations are concretized to concrete relations on *values* $v \in \mathbb{V}$ using the concretization function $\gamma_{\mathbb{R}^\sharp} \in \mathbb{R}^\sharp \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$. We require all operations (identity, inversion, and composition) on abstract relations to be sound over approximations of relational identity, inversion and composition:

$$\begin{aligned} \gamma_{\mathbb{R}^\sharp}(\mathtt{id}) &\supseteq \{(v, v) \mid v \in \mathbb{V}\} \\ \gamma_{\mathbb{R}^\sharp}(\mathtt{inv}(\mathcal{R}^\sharp)) &\supseteq \{(v_2, v_1) \mid (v_1, v_2) \in \gamma_{\mathbb{R}^\sharp}(\mathcal{R}^\sharp)\} \\ \gamma_{\mathbb{R}^\sharp}(\mathcal{R}_1^\sharp \fatsemi \mathcal{R}_2^\sharp) &\supseteq \left\{ (v_1, v_3) \;\middle|\; \begin{array}{r} \exists v_2,\ (v_1, v_2) \in \gamma_{\mathbb{R}^\sharp}(\mathcal{R}_1^\sharp) \\ \wedge\ (v_2, v_3) \in \gamma_{\mathbb{R}^\sharp}(\mathcal{R}_2^\sharp) \end{array} \right\} \end{aligned}$$

Then, a union-find data structure labeled with abstract relations is a relational abstract domain representing possible valuations over the different variables:

$$\begin{aligned} \gamma &\in (\mathbb{X} \rightharpoonup \mathbb{R}^\sharp \times \mathbb{X}) \to \mathcal{P}(\mathbb{X} \to \mathbb{V}) \\ \gamma(U) &\triangleq \{v \mid \forall x \mapsto (\mathcal{R}^\sharp, y) \in U, (v[x], v[y]) \in \gamma_{\mathbb{R}^\sharp}(\mathcal{R}^\sharp)\} \end{aligned}$$

Alternatively, labeled union-find can be seen as a fast and incremental procedure to decide if there is a relation between two variables (and finding the relation). It only requires the set of binary relations to have an algebraic group structure. This view applies when there is no need to join different sets of relations.

4 Properties of Abstract Relations

The fact that abstract relations are a group, and that all operations are sound, implies many interesting theorems:

- `inv` is exact [Ranzato 2013] (i.e., γ-complete);
- $\gamma_{\mathbb{R}^\sharp}(\mathtt{id})$ is an equivalence relation;
- Every abstract relation $\mathcal{R}^\sharp$ describes an injective partial function between the equivalence classes induced by the $\gamma_{\mathbb{R}^\sharp}(\mathtt{id})$ relation.

This both limits the possible candidates for suitable abstract relations (e.g., inequality constraints are not expressible), but also makes it easy to find them.

The canonical setting for these abstract relations is given by a theorem that establishes the equivalence of $\gamma_{\mathbb{R}^\sharp}$ being a group homomorphism, $\fatsemi$ to be exact, and the concretization of $\mathcal{R}^\sharp$ to be bijective, total or surjective functions between equivalence classes. In this case, abstract relations are just a compact representation of a bijective function (on equivalence classes, which is generally the identity). We can thus easily find a lot of interesting and common relations that fit this category.

Example 4.1 (Two-values per equality domain)**.** Take $\mathbb{V} = \mathbb{Q}$, the set of rationals, and use the relations $\mathbb{R}^\sharp \triangleq \mathbb{Q}_{\neq 0} \times \mathbb{Q}$ concretized as $\gamma_{\mathbb{R}^\sharp}(a, b) \triangleq \{(x, y) \in \mathbb{Q} \times \mathbb{Q} \mid y = ax + b\}$. We name this domain TVPE by similarity with the two-variables per inequality (TVPI) domain [Simon and King 2010];

Example 4.2 (`xor` and rotation)**.** Let $\mathbb{V} = \mathbb{BV}$, the set of bitvectors, with the relations $\mathbb{R}^\sharp \triangleq \mathbb{N} \times \mathbb{BV}$ concretized as $\gamma_{\mathbb{R}^\sharp}(n, c) = \{(x, y) \in \mathbb{BV} \times \mathbb{BV} \mid y = (x \mathbin{\mathtt{xor}} c) \mathbin{\mathtt{rot}} n\}$. These do form a group, as they are a combination of two bijective functions on bitvectors. While such direct rotations are uncommon in bitvector manipulations, shifts are very common; and a shifting a bitvector whose erased bits are known can be transformed as a sequence of (xor with constant, rotation). An extension to this domain would be considering arbitrary permutations instead of just rotations;

Example 4.3 (Modular arithmetic)**.** In $\mathbb{Z}/2^n\mathbb{Z}$, addition with a constant, or multiplication with an odd value, are invertible operations. Thus, they can be seen as a TVPE abstract domain for modular arithmetic. Multiplication with a power of two is not an invertible operation, but it can be encoded as a xor + rotation if the erased bits are known (e.g. if there are no overflows);

Example 4.4 (Invertible matrices multiplications)**.** If $\mathbb{V}$ is a vector of values in a field, then we can relate different such vector values with using matrix multiplication, provided the matrices are invertible.

Not all abstract relations have an exact composition operator. An interesting example is the TVPE abstract domain on integers (i.e. when $\mathbb{V} = \mathbb{Z}$). In this example, $(1/2, 0) \fatsemi (2, 0) = \mathtt{id}$, while the concrete composition would have returned $\{(x, x) \mid x \in 2 * \mathbb{Z}\}$, i.e. the abstract composition forgot that the value is even. This allows the TVPE abstract domain to be used on integer values, but it means that path compression in this case may lead to losses in precision; we have investigated solutions to work around this problem.

A last property is that if $\fatsemi$ is monotonic (which is the case when it computes the best approximation), then the lattice of

abstract relations is flat. This means that the intersection of two relations cannot be represented by a labeled union-find; for instance the intersection of the relations $y = 3x + 2$ and $y = 2x + 7$ must be represented in another domain (e.g. the constant or interval domain that would contain the solution of these equations).

5 Reduced Product with Non-Relational and Constant Propagation Domains

There are several interesting interactions between a labeled union-find $\mathbb{U}$ representing a set of abstract relations, and a non-relational abstract domain $\mathbb{X} \to \mathbb{V}^\sharp$. The first one is to view the labeled union-find domain as a set of constraints, and use them to perform constant propagation until arc-consistency (generalized to the abstract) is obtained. More precisely, we consider the operation `refine` that reduces the value abstraction attached to two variables based on the relation between these variables, i.e. is such that

$$\texttt{refine} \in \mathbb{R}^\sharp \times \mathbb{V}^\sharp \times \mathbb{V}^\sharp \to \mathbb{V}^\sharp \times \mathbb{V}^\sharp$$

$$\texttt{refine}(\mathcal{R}^\sharp, v_1^\sharp, v_2^\sharp) = (v_1^{\sharp\prime}, v_2^{\sharp\prime}) \Rightarrow$$
$$\gamma_{\mathbb{V}^\sharp}(v_1^{\sharp\prime}) \supseteq \left\{ v_1 \in \gamma_{\mathbb{V}^\sharp}(v_1^\sharp) \;\middle|\; \exists v_2 \in \gamma_{\mathbb{V}^\sharp}(v_2^\sharp), (v_1, v_2) \in \gamma_{\mathbb{R}^\sharp}(\mathcal{R}^\sharp) \right\}$$

(and similarly for $v_2^{\sharp\prime}$). Then, we propagate constraints using `refine` between any pair of variables using consistency (i.e., a fixpoint) is reached. An algorithm such as AC-3 [Mackworth 1977] provides a suitable schedule. Note that this reduction is not specific to the relations representable by a labeled union-find; for instance, reduced product of non-relational abstraction with symbolic expressions [Lesbre and Lemerre 2024] performs the same kind of propagation.

However, in the case of relations represented by a labeled union-find, this propagation can often be done more efficiently. Indeed, we can use the labeled union-find to perform a map factorization, replacing a map $m \in \mathbb{X} \to \mathbb{V}$ with a partial map $\hat{m} \in \mathbb{X} \rightharpoonup \mathbb{V}$ containing only the roots of the labeled union-find trees. This is in particular the case when $\wr$ and $\texttt{apply}^\sharp \in \mathbb{R}^\sharp \times \mathbb{V}^\sharp \to \mathbb{V}^\sharp$ are complete, where $\texttt{apply}^\sharp$ is the abstract counterpart of the application of a binary relation $\mathcal{R}$ to a set $\mathcal{S}$, defined as $\mathcal{R}(\mathcal{S}) \triangleq \{y \mid \exists x \in \mathcal{S}, (x, y) \in \mathcal{R}\}$. Applying a relation to a set generalizes the notion of applying a function to a value. Formally, $\texttt{apply}^\sharp$ must be a sound over-approximation of the application of a relation to a set:

$$\gamma_{\mathbb{V}^\sharp}(\texttt{apply}^\sharp(\mathcal{R}^\sharp, v^\sharp)) \supseteq \gamma_{\mathbb{R}^\sharp}(\mathcal{R}^\sharp)(\gamma_{\mathbb{V}^\sharp}(v^\sharp))$$

There are interesting examples of abstract relations and abstract values such that the $\texttt{apply}^\sharp$ operation is complete. For instance, the TVPE relations are sound and complete for the interval abstraction: if $y = 3x + 7$, then we have $x \in [0, 5]$ if, and only if, $y \in [7, 22]$. It is also complete for the congruence abstraction [Granger 1989]. Applying xor or rotation relations is also complete wrt. the bitwise (or tristate) abstraction of bitvectors [Miné 2012; Vishwanathan et al. 2022].

There are two benefits of this map factorization: storage is cheaper, but also constraint propagation is faster, as we instantly improve the value of all the variables in the same union-find tree, instead of trying every combination of two variables in the tree as would be done by standard AC-3.

6 Factorizing Other Relational Domains

Labeled union-find could be used not only to factorize non-relational constraints, but also relational ones. In particular, the TVPE domain can represent numerical constraints of the form $y = a * x + b$ that appear in many other numerical domains or decision procedures for the satisfiability of systems of rational or integer constraints:

- Inequality constraints [Dill 1989; Pratt 1977], as found in the DBM abstract domain [Miné 2001];
- Unit TVPI constraints [Harvey and Stuckey 1995] as found in the Octagon domain [Miné 2006];
- TVPI constraints [Shostak 1981] as found in the TVPI domain [Simon and King 2010]
- Conjunction of linear equalities, decided by Gauss-Jordan elimination procedure and used in Karr's domain [Karr 1976]
- Conjunction of linear inequalities [Dantzig 1990; Shostak 1981][Kroening and Strichman 2008, §5.4] as in the polyhedral domain [Cousot and Halbwachs 1978]

An interesting direction is to represent simple two-variables per equality constraints using the labeled union-find structure, and replace variables by their root in this structure (we call this process constraint factorization). This would allow reducing the number of variables that need to be related, which is important as the complexity of relational abstract domains depends on this number of variables. As TVPE constraints are commonplace in many programs, we hope that this method could help improve the efficiency of other abstract domains.

7 Abstract Operations: Join, Forget

Using a union-find structure to represent relations is both time and space efficient, however some work is required to get it to work well in an abstract interpretation setting. Firstly, killing/forgetting the relations about a variable is hard to implement in a union-find structure, as one cannot easily remove the representative from a class. A solution to this problem is to implement the domain as an SSA domain [Lesbre and Lemerre 2024] working on the SSA representation of the program, where variables are never killed.

An even harder difficulty is that union-find is generally implemented as imperative data structure which is expensive to copy. In abstract interpretation, we need copies at control-flow splits to interpret each branch with a different set of relations. Furthermore, we need a join operation that reconciles copies at control-flow joins by keeping only the relations present in all copies.

One solution to this problem is to only store flow-insensitive relations in the imperative union-find structure. Effectively, this avoids having to do copies and joins, since the information stored is always valid anywhere in the program. For instance, the relation between SSA terms $3e + 1 \xrightarrow{(3,1)} e$ holds at any program point, and can thus be stored in an imperative union-find domain. Note that these flow-insensitive relations can still be reduced (Section 5) with a flow-sensitive non-relational domain.

Another solution uses persistent maps to store the pointer from nodes to their parents and labels. These can be copied in constant time, but add a logarithmic factor to the find and union operations. We also need to define a join on this data structure. In theory, the join can be implemented as performing a full reduction (computing the transitive relation between any nodes), then performing intersection between these relations, then compacting the resulting graph as a union-find. In practice, we need to intertwine the reduction in the join operation to make it efficient, which is even harder when we also want to benefit from the fact that copies derive from a common ancestor [Blanchet et al. 2003], and we also need to take into account the reduced product with other domains in the reduction. We found a suitable solution using Patricia-tree based maps [Okasaki and Gill 1998], known for their fast merge operation, that we still need to evaluate.

We also want to consider optimizations such as the layering of both solutions: using both a fast imperative union find to store flow-insensitive information, and a slower persistent one for flow-sensitive relations.

8 Conclusion

We have discovered an interesting family of low-cost relational abstractions, based on abstract relations that follow the laws of group theory. We believe that there are many interesting applications of these abstractions, such as the TVPE abstract domain, and in particular that they could be used to optimize constraints used in other relational abstractions by reducing the number of variables that need to be tracked. We are working toward an efficient implementation and evaluation of this family of abstraction.

Acknowledgments

This research was supported in part by the Agence Nationale de la Recherche (ANR) grant agreement ANR-22-CE39-0014-03 (EMASS project).

References

B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. 2003. A Static Analyzer for Large Safety-Critical Software. In *Programming Language Design and Implementation (PLDI)*.

Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. *Principles of Programming Languages (POPL)*. https://doi.org/10.1145/512760.512770

George B. Dantzig. 1990. *Origins of the simplex method.* Association for Computing Machinery, New York, NY, USA, 141–151.

David L. Dill. 1989. Timing Assumptions and Verification of Finite-State Concurrent Systems. In *Proceedings of Automatic Verification Methods for Finite State Systems (Lecture Notes in Computer Science)*. 197–212. https://doi.org/10.1007/3-540-52148-8_17

Graeme Gange, Zequn Ma, Jorge A. Navas, Peter Schachte, Harald Søndergaard, and Peter J. Stuckey. 2021. A Fresh Look at Zones and Octagons. *TOPLAS* (2021). https://doi.org/10.1145/3457885

Philippe Granger. 1989. Static analysis of arithmetical congruences. *International Journal of Computer Mathematics* 30, 3-4 (1989), 165–190. https://doi.org/10.1080/00207168908803778

Warwick Harvey and Peter Stuckey. 1995. *A unit two variable per inequality integer constraint solver for constraint logic programming.* Technical Report. University of Melbourne.

Michael Karr. 1976. Affine Relationships Among Variables of a Program. *Acta Informatica* 6 (1976), 133–151. https://doi.org/10.1007/BF00268497

Daniel Kroening and Ofer Strichman. 2008. *Decision Procedures: An Algorithmic Point of View.* Springer.

Dorian Lesbre and Matthieu Lemerre. 2024. Compiling with Abstract Interpretation. *PACMPL* 8, PLDI, Article 162 (6 2024), 25 pages. https://doi.org/10.1145/3656392

Francesco Logozzo and Manuel Fähndrich. 2008. Pentagons: a weakly relational abstract domain for the efficient validation of array accesses. In *ACM Symposium on Applied computing*.

Alan K. Mackworth. 1977. Consistency in networks of relations. *Artificial Intelligence* 8, 1 (1977). https://doi.org/10.1016/0004-3702(77)90007-8

Antoine Miné. 2001. A New Numerical Abstract Domain Based on Difference-Bound Matrices. In *Programs as Data Objects PADO (Lecture Notes in Computer Science)*. https://doi.org/10.1007/3-540-44978-7_10

Antoine Miné. 2004. *Weakly relational numerical abstract domains.* Ph. D. Dissertation. École Polytechnique. http://www.di.ens.fr/~mine/these/these-color.pdf.

Antoine Miné. 2006. The octagon abstract domain. *Higher-order and symbolic computation* 19, 1 (2006), 31–100.

Antoine Miné. 2012. Abstract domains for bit-level machine integer and floating-point operations. In *WING'12 - 4th International Workshop on invariant Generation.* Manchester, United Kingdom, 16.

Antoine Miné. 2002. A Few Graph-Based Relational Numerical Abstract Domains. In *Static Analysis Symposiun SAS*.

Chris Okasaki and Andrew Gill. 1998. Fast Mergeable Integer Maps. In *Workshop on ML*. 77–86.

Vaughan Pratt. 1977. *Two easy theories whose combination is hard.* Technical Report.

Francesco Ranzato. 2013. Complete Abstractions Everywhere. In *VMCAI (Lecture Notes in Computer Science, Vol. 7737)*, Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.). Springer, 15–26. https://doi.org/10.1007/978-3-642-35873-9_3

Robert E. Shostak. 1981. Deciding Linear Inequalities by Computing Loop Residues. *J. ACM* 28 (1981), 769–779.

Axel Simon and Andy King. 2010. The two variable per inequality abstract domain. *High. Order Symb. Comput.* 23, 1 (2010), 87–143. https://doi.org/10.1007/s10990-010-9062-8

Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2015. Making numerical program analysis fast. In *Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/2737924.2738000

Robert Endre Tarjan and Jan van Leeuwen. 1984. Worst-case Analysis of Set Union Algorithms. *J. ACM* 31, 2 (1984), 245–281. https://doi.org/10.1145/62.2160

Harishankar Vishwanathan, Matan Shachnai, Srinivas Narayana, and Santosh Nagarakatte. 2022. Sound, Precise, and Fast Abstract Interpretation with Tristate Numbers. In *Code Generation and Optimization (CGO)*. https://doi.org/10.1109/CGO53902.2022.9741267