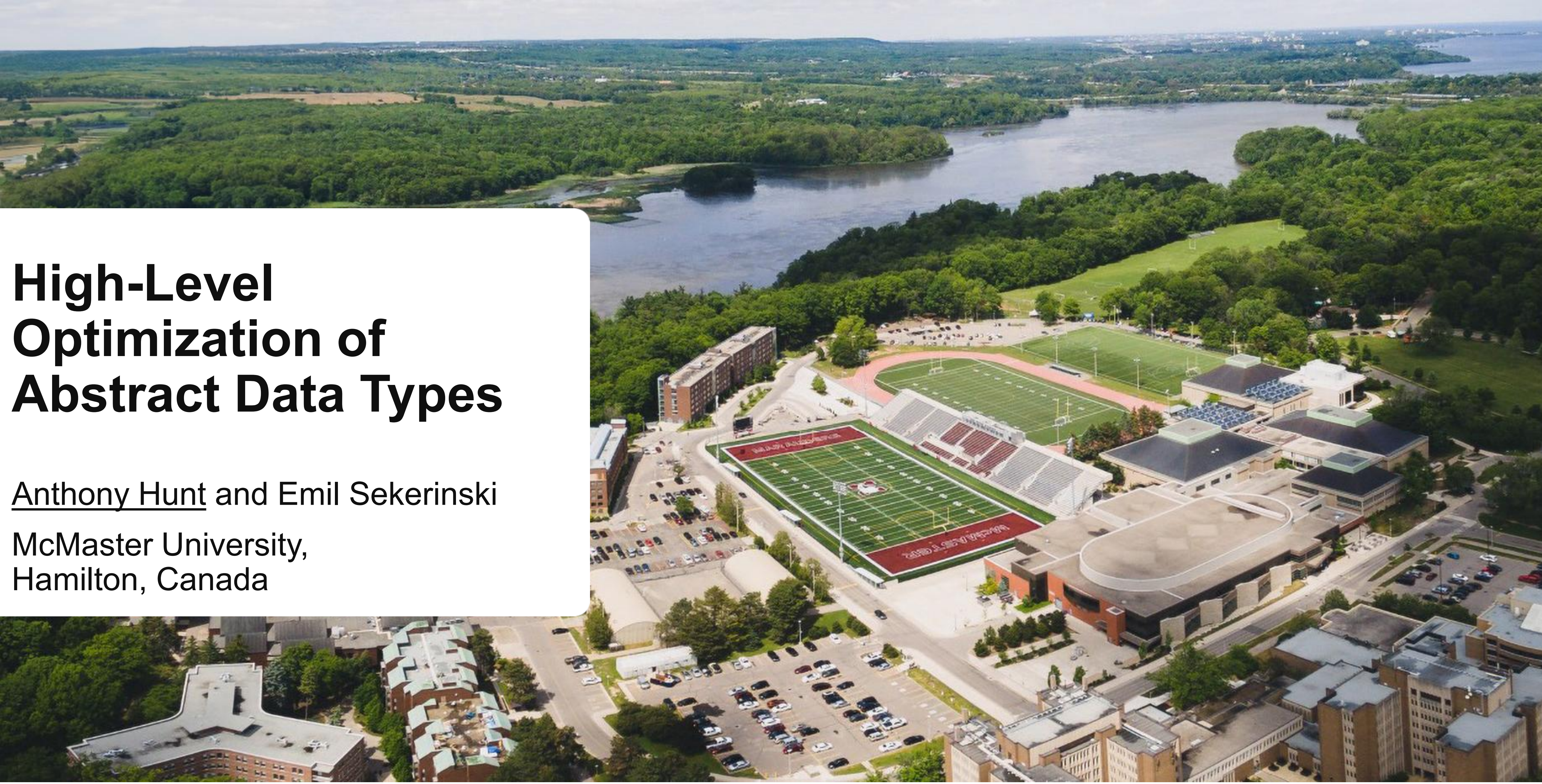


High-Level Optimization of Abstract Data Types

Anthony Hunt and Emil Sekerinski

McMaster University,
Hamilton, Canada

Programming With Abstract Data Types

- System specifications use sets and relations
  - Expressive and provable
- Semantics from mathematics

Syntax	Label	Syntax	Label
$set(T)$	Unordered, unique collection	$S \cup T$	Union
$S \leftrightarrow T$	Relation, $set(S \times T)$	$S \cap T$	Intersection
$S \rightarrow \mathbb{N}$	Bag/Multiset	$S \setminus T$	Difference
$\mathbb{N} \rightarrow S$	Sequence	$S \times T$	Cartesian Product
$\{x, y, \dots\}$	Set Enumeration	$dom(R)$	Domain
$\{x \cdot P \mid E\}$	Set Comprehension	$R[S]$	Image

Jean-Raymond Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.
David Gries and Fred Schneider. *A Logical Approach to Discrete Math*. Springer New York, 1993.

Example - Visitor Information System

- Academics attend workshops throughout CASCON 2025
- Every workshop must be held in one room
- Visitors may attend at most one workshop at a time

$$Visitor = \{J.\ Nelson, Mark, Ehsan\}$$
$$Room = \{1,2,3,4\}$$
$$Workshop = \{CDP, SENGEC, COGAI\}$$

Total Injective Function

$$location : Workshop \rightarrowtail Room$$
$$attends : Visitor \nrightarrow Workshop$$

Partial Function

Then, the number of meals to prepare for a specific $room$ would be:

$$card((location^{-1} \circ attends^{-1})[\{room\}])$$

Relational Image

Example - Warehouse Inventory System

- Furniture material catalogue
- Warehouse inventory
- Product recipes

$Material = \{2 \times 4\ Plank, Hex\ Bolt, 3\ Inch\ Screw, \dots\}$
$Price = \mathbb{N}_0$
$Product = \{Cabinet, Desk, Bookshelf, \dots\}$

$catalogue: Material \rightarrow Price$
$inventory: bag[Material]$
$recipes: Product \rightarrow bag[Material]$

Equivalent to: $Material \nrightarrow \mathbb{N}$

Restocking price, with a target inventory $inventory_t$:

$$\sum p \mapsto n_m \cdot$$
$$p \mapsto n_m \in catalogue^{-1} \circ (inventory_t - inventory) \mid p \times n_m$$

Bag Difference

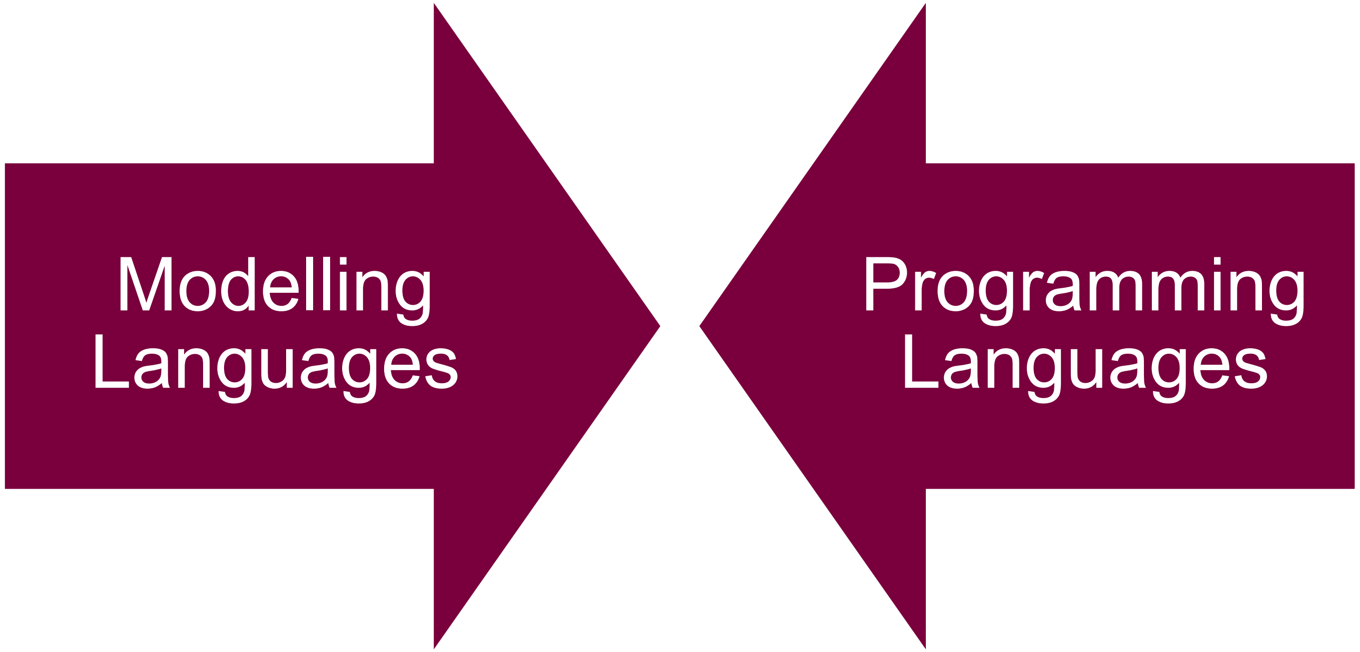
Abstract Data Types

In Programming and Modelling Languages

		Sets				Relations									Efficiently Executable	Implementation Free
		∪	∩	×	$\{x \cdot P \mid E\}$	dom	◁	∪	∩	⩤	∘	$R[S]$	R^{-1}	Multiplicity		
Programming Languages	Python	✓	✓	✓	✓	✓	✗	✗	*	✓	✗	*	✗	✗	*	✗
	Haskell	✓	✓	✓	*	✓	*	✗	✓	✓	*	*	✓	*	✓	✗
	Rust	✓	✓	*	*	✓	✗	✗	✗	*	✗	✓	✗	✗	✓	✗
	C	*	*	*	✗	*	*	*	*	*	*	*	*	✗	✓	✗
Modelling Languages	SetL	✓	✓	*	✓	✓	✓	✓	✓	*	*	✓	*	*	*	*
	UML	✓	✓	✓	✗	✓	✗	✗	✗	✓	✗	✓	✗	✓	*	✓
	Event-B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	*	✓
	Alloy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓

Goals

- Syntax close to discrete mathematics
- Simple, small, and usable
- High-level set-theory optimization
- Efficient in memory and time



Related Work

- ProB animation engine for Event-B [1]
- GHC rewrite rules [2]
- Rewrite systems for set-theory focused optimization [3,4,5]
- SETL data type *lowering* from abstract types to suitable concrete structures [6,7]

[1] Michael Leuschel. *Programming in B: Sets and Logic all the Way Down.* In: LPOP 2022.
[2] Simon Peyton Jones, Andrew Tolmach, and Tony Hoare. *Playing by the rules: rewriting as a practical optimisation technique in GHC*. In: 2001 Haskell Workshop. ACM SIGPLAN. Sept. 2001.
[3] Maximiliano Cristia and Gianfranco Rossi. *{log}: Programming and Automated Proof in Set Theory.* In: LPOP 2022.
[4] Elco Visser, Zine-el-Abidine Benaissa, and Andrew Tolmach. Building program optimizers with rewriting strategies. In: ICFP 1998.
[5] Douglas R. Smith and Stephen J. Westfold. *Transformations for Generating Type Refinements*. In: FM 2019.
[6] Edmond Schonberg, Jacob T. Schwartz, and Micha Sharir. 1979. *Automatic data structure selection in SETL*. In: POPL 1979.
[7] Stefan M. Freudenberger, Jacob T. Schwartz, and Micha Sharir. *Experience with the SETL Optimizer.* Association for Computing Machinery, 1983

Optimizing Abstract Data Type Operations (in Python)

$$(S \cup T) \cap V \rightsquigarrow (S \cap V) \cup ((T \setminus S) \cap V)$$

```python
# Python translation:
ret = set()
for x in S:
    ret.add(x)
# Step 1: len(ret) == len(S)
for x in T:
    ret.add(x)
# Step 2: len(ret) <= len(S) + len(T)
for x in ret:
    if x not in V:
        ret.remove(x)
# Step 3: len(ret) <= len(V)
```

```python
# New Python translation:
ret = set()
for x in S:
    if x in V:
        ret.add(x)
# Step 1: len(ret) <= min(len(S), len(V))
for x in T:
    if x not in S and x in V:
        ret.add(x)
# Step 2: len(ret) <= min(len(S) + len(T), len(V))
```

What if $len(S) + len(T) > len(V)$?

ret never grows larger than needed

Concrete Data Types of the Visitor Information System

$$location: Workshop \rightarrowtail Room$$
$$attends: Visitor \nrightarrow Workshop$$

$$location: set[tuple[Workshop, Room]]$$
$$attends: set[tuple[Visitor, Workshop]]$$

$$location: BiMap[Workshop, Room]$$
$$attends: BiMap[Visitor, Workshop]$$

$$location: HashMap[Workshop, Room]$$
$$attends: HashMap[Visitor, Workshop]$$

Naïve Implementation of the Visitor Information System

Concrete Types

$$location: set[tuple[Workshop, Room]]$$
$$attends: set[tuple[Visitor, Workshop]]$$

Imperative Interpretation

$$card((location^{-1} \circ attends^{-1})[\{room\}])$$

$$\rightsquigarrow$$

$$\begin{aligned} l &:= inverse(location) \\ a &:= inverse(attends) \\ c &:= compose(l, a) \\ &\ldots \end{aligned}$$

Library

$\mathbf{proc}\ inverse(r)$:
  $r' := \{\}$
  $\mathbf{for}\ k, v \in r\ \mathbf{do}$
    $r' := r' \cup \{(v, k)\}$
  $\mathbf{return}\ r'$

Runtime: $O(n)$
Memory: up to $O(n)$

$\mathbf{proc}\ compose(r_1, r_2)$:
  $r' := \{\}$
  $\mathbf{for}\ k, v \in r_1\ \mathbf{do}$
    $\mathbf{for}\ v', t \in r_2\ \mathbf{do}$
      $\mathbf{if}\ v = v'\ \mathbf{then}$
        $r' := r' \cup \{(k, t)\}$
  $\mathbf{return}\ r'$

Runtime: $O(n^2)$
Memory: $O(n)$

Designing Optimizations

- Set theory semantics for optimizations
- Strong type system with refinements
- Rewrite system:
  - Simplify operations
  - Select concrete data representations from refined types
  - Generate efficient code for every concrete representation and operation sequence

Term Rewriting for Hash-based Sets and Relations

1. Comprehension Construction
2. Generator Selection
3. Loop Lowering
4. Relational Subtyping Loop Simplification
5. Loop Code Generation
6. Replace and Simplify

Phase 1: Set Comprehension Construction

Set-typed variables and literals are decomposed into set comprehensions

Rewrite Rules	
Predicate Operations	$S \cup T \rightsquigarrow \{x \cdot x \in S \vee x \in T \mid x\}$
	$S \cap T \rightsquigarrow \{x \cdot x \in S \wedge x \in T \mid x\}$
	$S \setminus T \rightsquigarrow \{x \cdot x \in S \wedge x \notin T \mid x\}$
Membership Collapse	$x \in \{E \mid P\} \rightsquigarrow \exists y \cdot P \wedge x = E$
Image	$R[S] \rightsquigarrow \{y \cdot \exists x \cdot x \mapsto y \in R \wedge x \in S \mid y\}$
Composition	$x \mapsto z \in R \circ Q \rightsquigarrow x, z \cdot \exists y, y' \cdot x \mapsto y \in R \wedge y' \mapsto z \in Q \wedge y = y'$
Inverse	$x \mapsto y \in R^{-1} \rightsquigarrow y \mapsto x \in R$
Cardinality	$card(S) \rightsquigarrow \sum x \cdot x \in S \mid 1$

Post-condition:

- All set-like terms must be comprehensions

$Workshop \rightarrowtail Room$ $\quad$ $Visitor \nrightarrow Workshop$

$$numMeals = card((location^{-1} \circ attends^{-1})[\{room\}])$$

Image

$$numMeals = card(\{\boldsymbol{v} \cdot \exists \boldsymbol{r} \cdot \boldsymbol{r} \mapsto \boldsymbol{v} \in (location^{-1} \circ attends^{-1}) \wedge \boldsymbol{r} \in \{\boldsymbol{room}\} \mid \boldsymbol{v}\})$$

$Workshop \rightarrowtail Room$

Composition, Inverse

$$numMeals = card(\{v, \boldsymbol{r} \cdot \exists \boldsymbol{p}, \boldsymbol{p'} \cdot \boldsymbol{v} \mapsto \boldsymbol{p'} \in \boldsymbol{attends} \wedge \boldsymbol{p} \mapsto \boldsymbol{r} \in \boldsymbol{location} \wedge \boldsymbol{p} = \boldsymbol{p'} \wedge r \in \{room\} \mid v\})$$

$Workshop \rightarrowtail Room$

Cardinality, Membership

$$numMeals = \sum v, r \cdot \exists p, p' \cdot v \mapsto p' \in attends \wedge p \mapsto r \in location \wedge p = p' \wedge r = room \mid \mathbf{1}$$

Phase 2: Generator Selection

Selecting element generators for use as iterables in generated for-loops

Rewrite Rules	
Generator Selection	$\bigwedge P_i \rightsquigarrow P_g \wedge \bigwedge_{P_i \neq P_g} P\_i$ • P_g is of the form $x \in S$ • x is the quantifier's bound variable • S is a set-like term

Post-condition:

- All set-like terms must be comprehensions
- Quantifier predicates are in DNF
  - Guaranteed top-level-∨ operation
- One bound variable per generator
- All predicate ∨-clauses have an assigned generator

- In the case of multiple candidate generators, selection is based on heuristics

$$numMeals = \sum v, r \cdot \exists p, p' \cdot v \mapsto p' \in attends \\ \wedge p \mapsto r \in location \wedge p = p' \wedge r = room \mid 1$$

Generator Selection

$$numMeals = \sum \boldsymbol{v}, \boldsymbol{r} \cdot \exists p, p' \cdot \boldsymbol{v} \mapsto \boldsymbol{p'} \in \boldsymbol{attends} \\ \wedge \boldsymbol{p} \mapsto \boldsymbol{r} \in \boldsymbol{location} \wedge p = p' \wedge r = room \mid 1$$

Phase 3: Loop Lowering

Start lowering expressions into imperative-like loops

Rewrite Rules			
Quantifier Generation	$\bigoplus E \mid P$	$\rightsquigarrow$	$a := identity(\bigoplus)$ $\textbf{loop } P \textbf{ do}$ $a := a \bigoplus E$
Chained Generators	$\textbf{loop } G_1 \land G_2 \textbf{ do}$ $a := a \bigoplus E$	$\rightsquigarrow$	$\textbf{loop } G_1 \textbf{ do}$ $\textbf{loop } G_2 \textbf{ do}$ $a := a \bigoplus E$
Disjunct Generators	$\textbf{loop } G_1 \lor G_2 \textbf{ do}$ $a := a \bigoplus E$	$\rightsquigarrow$	$\textbf{loop } G_1 \textbf{ do}$ $a := a \bigoplus E$ $\textbf{loop } G_2 \land \neg G_1 \textbf{ do}$ $a := a \bigoplus E$

Post-condition:

- No quantifiers exist within the AST
- No $\lor$-operators exist within a $\boldsymbol{loop}$'s predicate
- All predicate $\lor$-clauses have an assigned generator

- $\bigoplus$ is any of $\{\ldots\}$, Σ, Π, $\cup$, $\cap$

$$\begin{aligned}&numMeals = \\ &\quad \Sigma \boldsymbol{v}, \boldsymbol{r} \cdot \exists p', p \cdot \boldsymbol{v} \mapsto \boldsymbol{p'} \in \boldsymbol{attends} \\ &\qquad \land \boldsymbol{p} \mapsto \boldsymbol{r} \in \boldsymbol{location} \land r = room \land p = p' \mid 1)\end{aligned}$$

Quantifier Generation

$$\begin{aligned}&numMeals := 0 \\ &\textbf{loop } \boldsymbol{v} \mapsto \boldsymbol{p'} \in \boldsymbol{attends} \\ &\qquad \land \boldsymbol{p} \mapsto \boldsymbol{r} \in \boldsymbol{location} \land p = p' \land r = room \textbf{ do} \\ &\qquad\quad numMeals := numMeals + 1\end{aligned}$$

Chained Generators

$$\begin{aligned}&numMeals := 0 \\ &\textbf{loop } \boldsymbol{v} \mapsto \boldsymbol{p'} \in \boldsymbol{attends} \textbf{ do} \\ &\quad \textbf{loop } \boldsymbol{p} \mapsto \boldsymbol{r} \in \boldsymbol{location} \land r = room \land p = p' \textbf{ do} \\ &\qquad numMeals := numMeals + 1\end{aligned}$$

Phase 4: Relation Subtyping Loop Simplification

Eliminate unnecessary loops

Rewrite Rules			
Restricted Generator	**loop** $x \mapsto y \in R$ $\land x = x' \land y = y'$ **do** body • R is total • R is one-to-one	$\rightsquigarrow$	**if** $R(x') = y'$ **then** body

One-to-one Total Function

$numMeals := 0$
loop $v \mapsto p' \in attends$ **do**
  loop $p \mapsto r \in location \land r = room \land p = p'$ **do**
    $numMeals := numMeals + 1$

$\rightsquigarrow$

$numMeals := 0$
loop $v \mapsto p' \in attends$ **do**
  if $location(p') = room$ **then**
    $numMeals := numMeals + 1$

Post-condition:

- No quantifiers exist within the AST
- No $\lor$-operators exist within a ***loop***'s predicate
- All predicate $\lor$-clauses have an assigned generator

Phase 5: Loop Code Generation

Lower into imperative ***loop*** structures

Rewrite Rules	
Conjunct Conditional	$\textbf{loop } P_g \wedge \bigwedge P_i \textbf{ do } body \rightsquigarrow \textbf{if } \bigwedge_{free(P_i)} P_i \textbf{ then for } P_g \textbf{ do if } \bigwedge_{\neg free(P_i)} P_i \textbf{ then } body$

$$
\begin{array}{l}
numMeals := 0 \\
\textbf{loop } v \mapsto p' \in attends \textbf{ do} \\
\quad \textbf{if } location(p') = room \textbf{ then} \\
\quad\quad numMeals := numMeals + 1
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{l}
numMeals := 0 \\
\textbf{for } v, p' \in attends \textbf{ do} \\
\quad \textbf{if } location(p') = room \textbf{ then} \\
\quad\quad numMeals := numMeals + 1
\end{array}
$$

Post-condition:

- Code is imperatively executable

Visitor Information System - Translation

One-to-One Total Function

$location: Workshop \rightarrowtail Room$
$attends: Visitor \nrightarrow Workshop$

$$numMeals = card((location^{-1} \circ attends^{-1})[\{room\}])$$

$$\Downarrow$$

$$numMeals = \sum v, r \cdot \exists p, p' \cdot v \mapsto p' \in attends \land p \mapsto r \in location \land \boldsymbol{p} = \boldsymbol{p'} \land \boldsymbol{r} = \boldsymbol{room} \mid 1$$

$$\Downarrow$$

$$numMeals = \sum v \cdot \exists p' \cdot v \mapsto p' \in attends \land location(p') = room \mid 1$$

$numMeals := 0$
for $v, p' \in attends$ **do**
  if $location(p') = room$ **then**
    $numMeals := numMeals + 1$

Warehouse Inventory System – Translation

Many-to-One Total Function

Many-to-One Function

$catalogue: Material \to Price$
$inventory: bag[Material]$
$recipes: Product \to bag[Material]$

Restocking price, with a target inventory $inventory_t$:

$$price = \sum p \mapsto n_m \cdot p \mapsto n_m \in catalogue^{-1} \circ (inventory_t - inventory) \mid p \times n_m$$

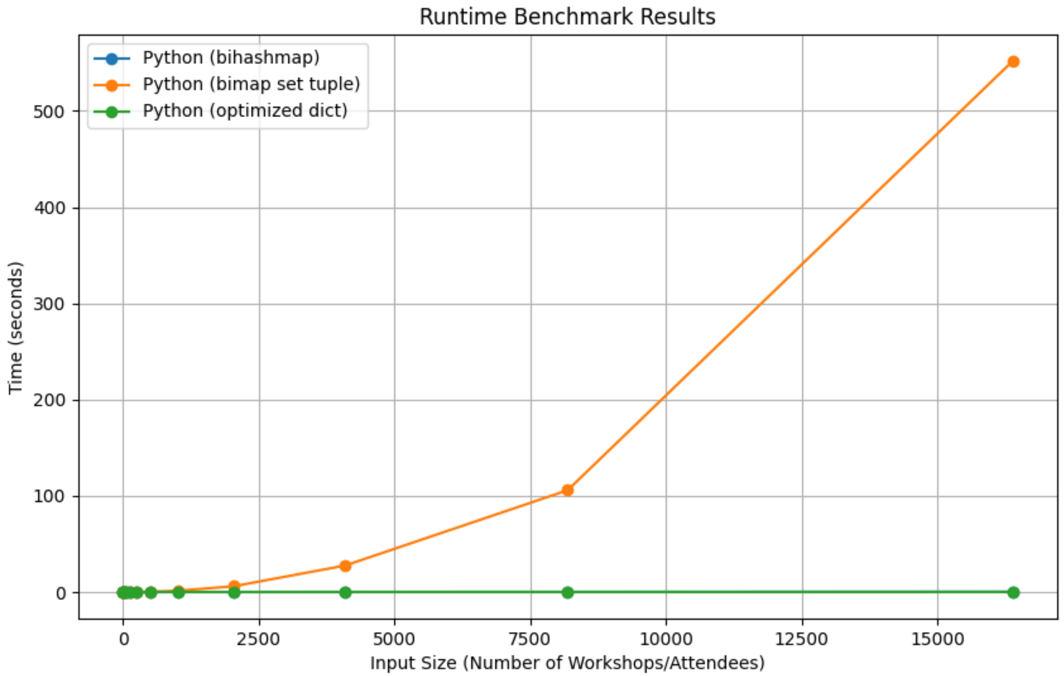
$$price = \sum \boldsymbol{m} \mapsto \boldsymbol{n_m} \cdot \exists n, m, p \cdot \boldsymbol{m} \mapsto \boldsymbol{n_m} \in \boldsymbol{inventory_t} \wedge n = n_m - inventory(m) \wedge n \geq 0 \wedge p = catalogue(m) \mid p \times n_m$$

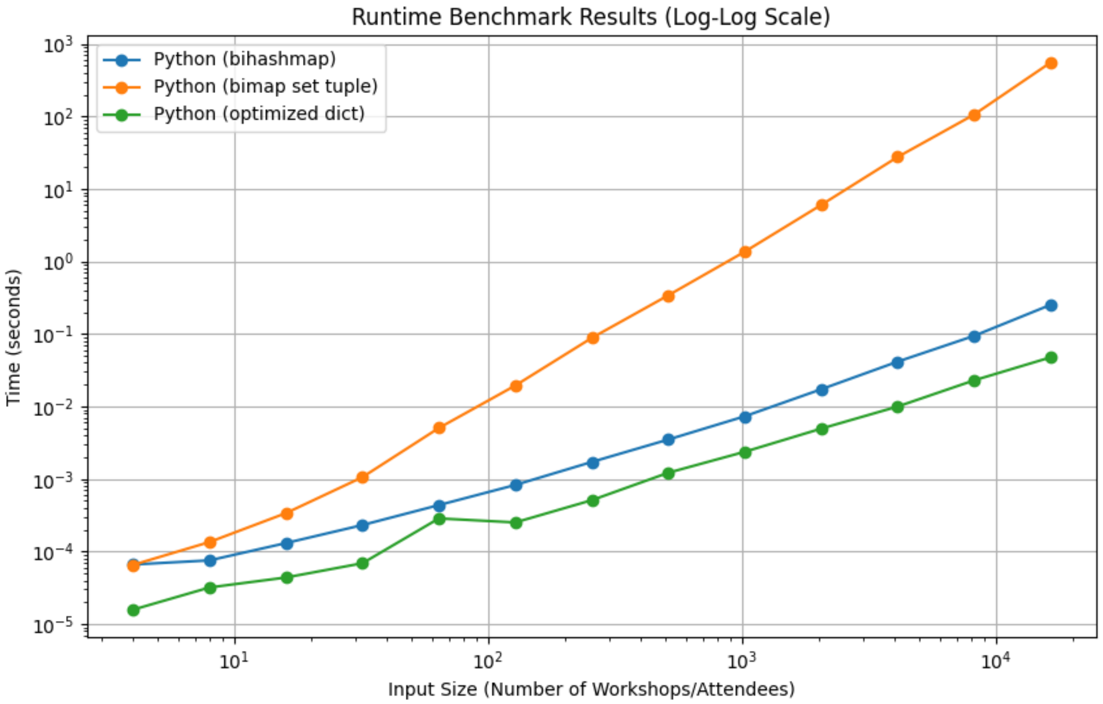
$price := 0$
$\textbf{for}\ m, n_m \in inventory_t\ \textbf{do}$
$\quad n := n_m - inventory(m)$
$\quad \textbf{if}\ n \geq 0\ \textbf{then}$
$\quad\quad price := price + catalogue(m) \times n$

Preliminary Results

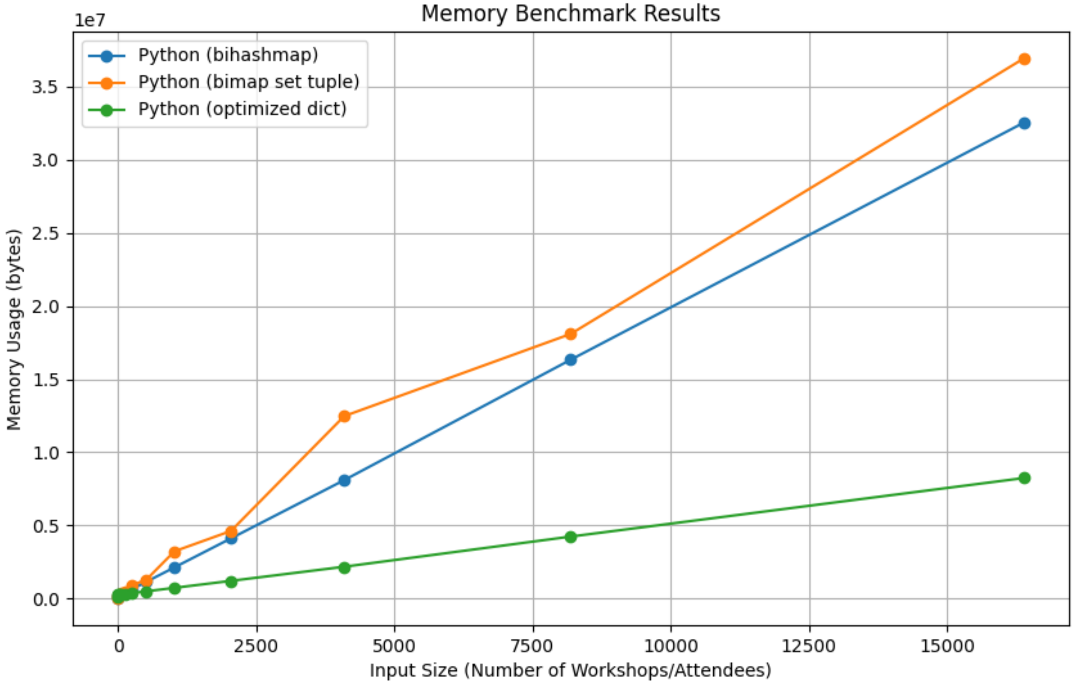
- Intermediate sets never grow larger than $max(starting\ sets, resulting\ set)$
- Unions, Relation Overriding, and Cartesian Products are limited in growth
- Other operators only decrease the size of a resulting set
- What about running time?…

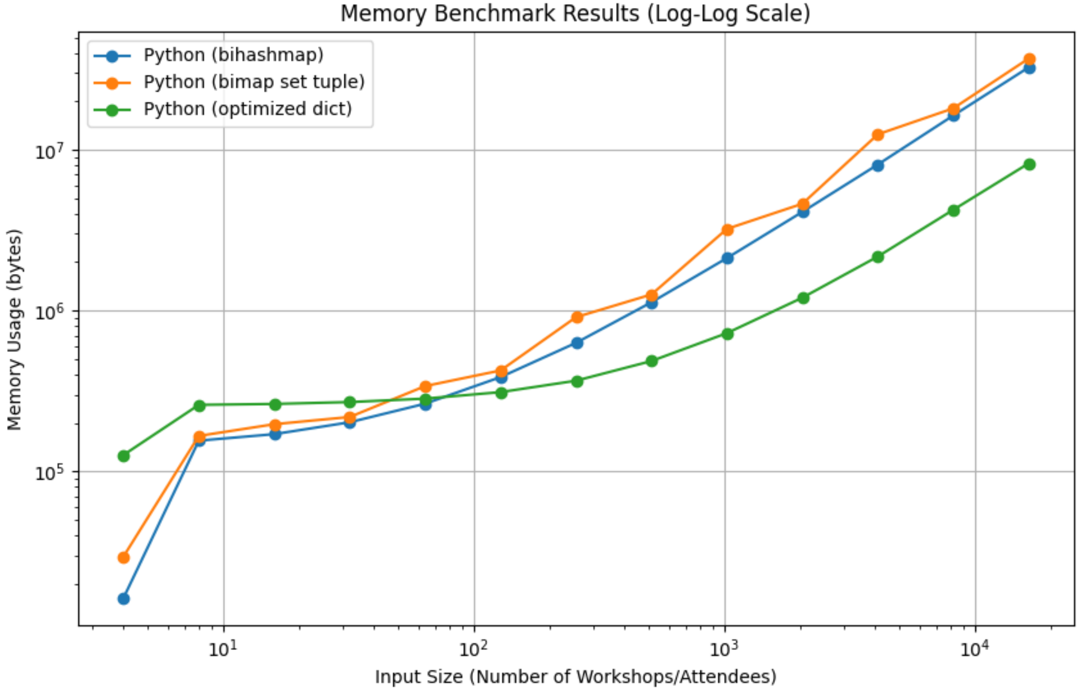
Visitor Information System Benchmark – Runtime





Visitor Information System Benchmark – Memory Consumption





Examples Running Time

	Core Expression	Naïve Running Time	Optimized Running Time
Visitor Information System	$(location^{-1} \circ attends^{-1})[\{room\}]$	$O(\|location\|\|attends\|)$	$O(\|attends\|)$
Warehouse Inventory System	$\sum p \mapsto n_m \cdot p \mapsto n_m \in (inventory_t - inventory) \circ catalogue^{-1} \mid p \times n_m$	$O(\|inventory_t\|\|catalogue\|)$	$O(\|inventory_t\|)$

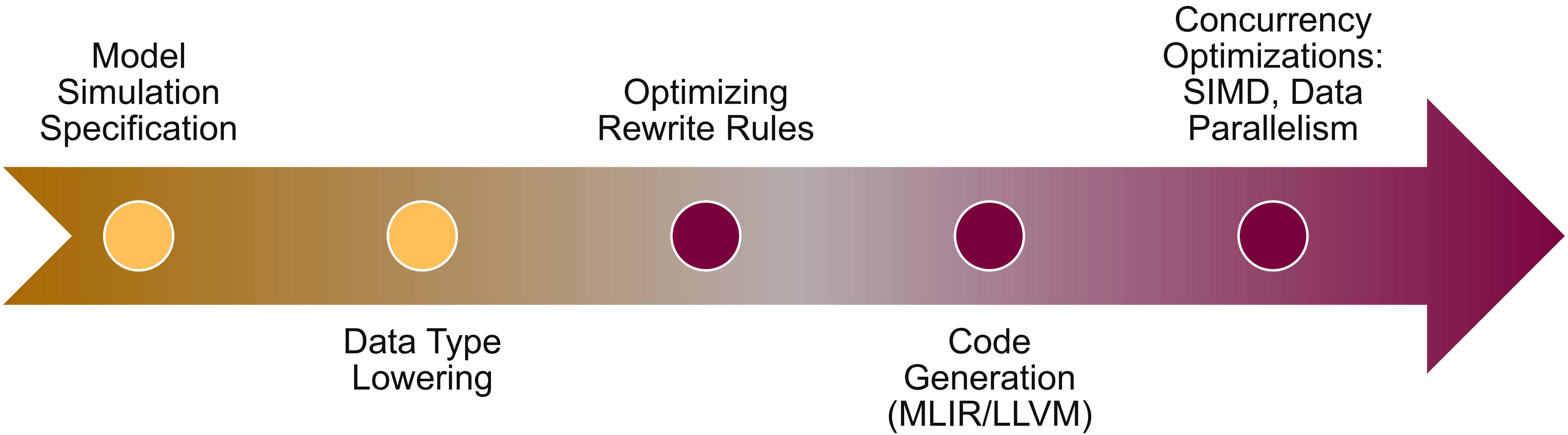
Operation Running Time With (Bi-directional) HashMaps

Operation	Expected Running Time
$S \cup T$	$O(\lvert S\rvert + \lvert T\rvert)$
$S \cap T$	$O(min(\lvert S\rvert, \lvert T\rvert))$
$R[S]$	$O(min(\lvert S\rvert, \lvert R\rvert))$
$\mathrm{R} \circ Q$	$O(\lvert R\rvert\lvert Q\rvert)$
$(S \cup T) \cap U$	$O(min(\lvert S\rvert + \lvert T\rvert, \lvert U\rvert))$
$S \cap R[T]$	$O(min(\lvert S\rvert, \lvert R\rvert, \lvert T\rvert))$
$(\mathrm{R} \circ Q)[S]$	$O(min(\lvert S\rvert, \lvert R\rvert) + min(\lvert R[S]\rvert, \lvert Q\rvert))$

- Simplify large groups of ∧-predicates
- Eagerly apply conditions on sequential ∨-predicates

Current Compiler State and Roadmap

Development of a High-Level, Efficient, Set-Based Language



References

- Michael Leuschel. *ProB2 Jupyter Notebooks GitLab.* 2020. https://gitlab.cs.uni-duesseldorf.de/general/stups/prob2-jupyter-notebooks/-/blob/9d830112d9713d0cbc12d9d285f44ee61c827ccf/puzzles/Game_of_Life.ipynb
- Jean-Raymond Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.
- David Gries and Fred Schneider. *A Logical Approach to Discrete Math*. Springer New York, 1993.
- Robert Dewar. *The SetL Programming Language.* Bell Laboratories, 1979.
- Simon Peyton Jones, Andrew Tolmach, and Tony Hoare. *Playing by the rules: rewriting as a practical optimisation technique in GHC*. In: 2001 Haskell Workshop. ACM SIGPLAN. Sept. 2001.
- Michael Leuschel. *Programming in B: Sets and Logic all the Way Down.* In: LPOP 2022.
- Maximiliano Cristia and Gianfranco Rossi. *{log}: Programming and Automated Proof in Set Theory.* In: LPOP 2022.
- Douglas R. Smith and Stephen J. Westfold. *Transformations for Generating Type Refinements*. In: Formal Methods. FM 2019 International Workshops. Springer International Publishing, 2020, pp. 371-387
- Edmond Schonberg, Jacob T. Schwartz, and Micha Sharir. 1979. *Automatic data structure selection in SETL*. In Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '79). Association for Computing Machinery, New York, NY, USA, 197–210. https://doi.org/10.1145/567752.567771
- Stefan M. Freudenberger, Jacob T. Schwartz, and Micha Sharir. 1983. *Experience with the SETL Optimizer.* ACM Trans. Program. Lang. Syst. 5, 1 (Jan. 1983), 26–45. https://doi.org/10.1145/357195.357197