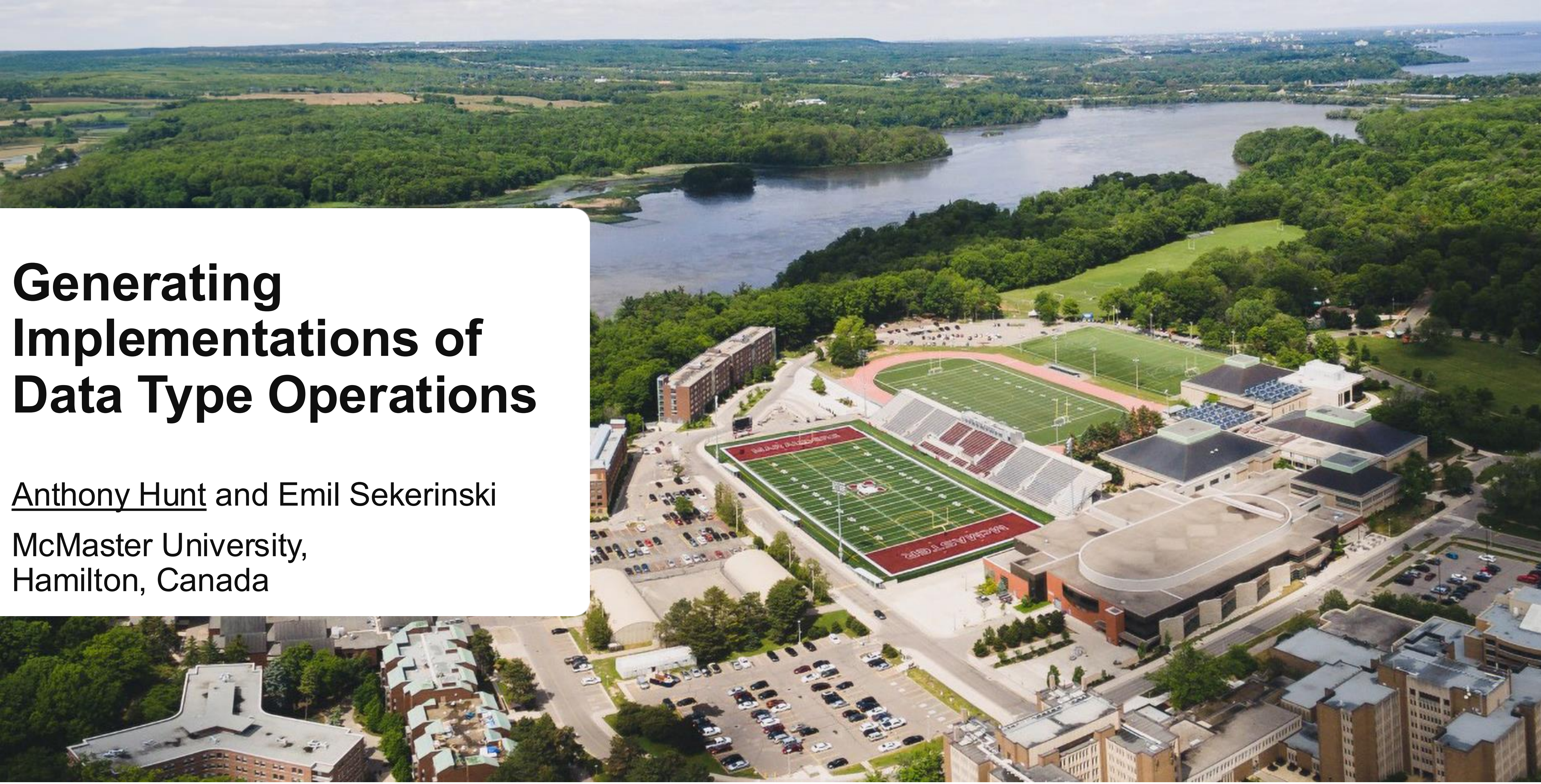


Generating Implementations of Data Type Operations

Anthony Hunt and Emil Sekerinski

McMaster University,
Hamilton, Canada

Abstract Data Types

Operations [Abrial 10][Gries, Schneider 93]

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
$\emptyset$	Empty Set	$S \times T$	Cartesian Product
$\{x, y, \ldots\}$	Set Enumeration	$dom(R)$	Domain
$\{x \cdot P \mid E\}$	Set Comprehension	$ran(R)$	Range
$x \mapsto y$	Pair	$R[S]$	Image
$S \mathrel{\rightarrow\!\!\!\!\!+} T$	Partial Function	$R \square Q$	Overriding
$S \rightarrowtail T$	Total Injective Function	$R \circ Q$	Composition
$S \subseteq T$	Subset	$S \lhd R$	Domain Restriction
$S - T$	Bag Difference	R^{-1}	Inverse

Visitor Information System

Motivating Examples

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

$$Visitor = \{Shufang, Mark, Grigory\}$$
$$Room = \{D346, D273, D305, D160, D152\}$$
$$Workshop = \{SYNT, MOSCA, HCVS\}$$

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

Warehouse Inventory System

Motivating Examples

- Furniture material catalogue
- Warehouse inventory
- Product recipes

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

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

Restocking price, with a target inventory $inventory_t$:

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

Conway's Game of Life

Motivating Examples [Leuschel 20]

- Next state depends on previous position
- Cells with 2 or 3 neighbours remain living
- Dead cells with 3 living neighbours are revived

$$Boundary \in \mathbb{N}$$
$$Living \in \mathbb{P}(Boundary \times Boundary)$$

$$\begin{aligned} neigh = \{x, y, x_n, y_n \cdot \; & x \in Boundary \land y \in Boundary \\ & \land x_n \in (x-1)..(x+1) \land y_n \in (y-1)..(y+1) \\ & \land x \neq x_n \land y \neq y_n \\ & \mid (x \mapsto y) \mapsto (x_n, \mapsto y_n)\} \end{aligned}$$

$$\begin{aligned} nextStep = \; & \\ & \{c \cdot c \in Living \;\land\; card(neigh[\{c\}] \cap Living) = \; 2 \mid c\} \\ \cup \; & \{c \cdot c \in neigh\;[Living] \\ & \quad \land \; card(neigh\;[\{c\}] \; \cap Living) = 3 \mid c\} \end{aligned}$$

Credit (Image): *Conway's Game of Life.* Wikipedia, *2025. https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life*

Abstract Data Types

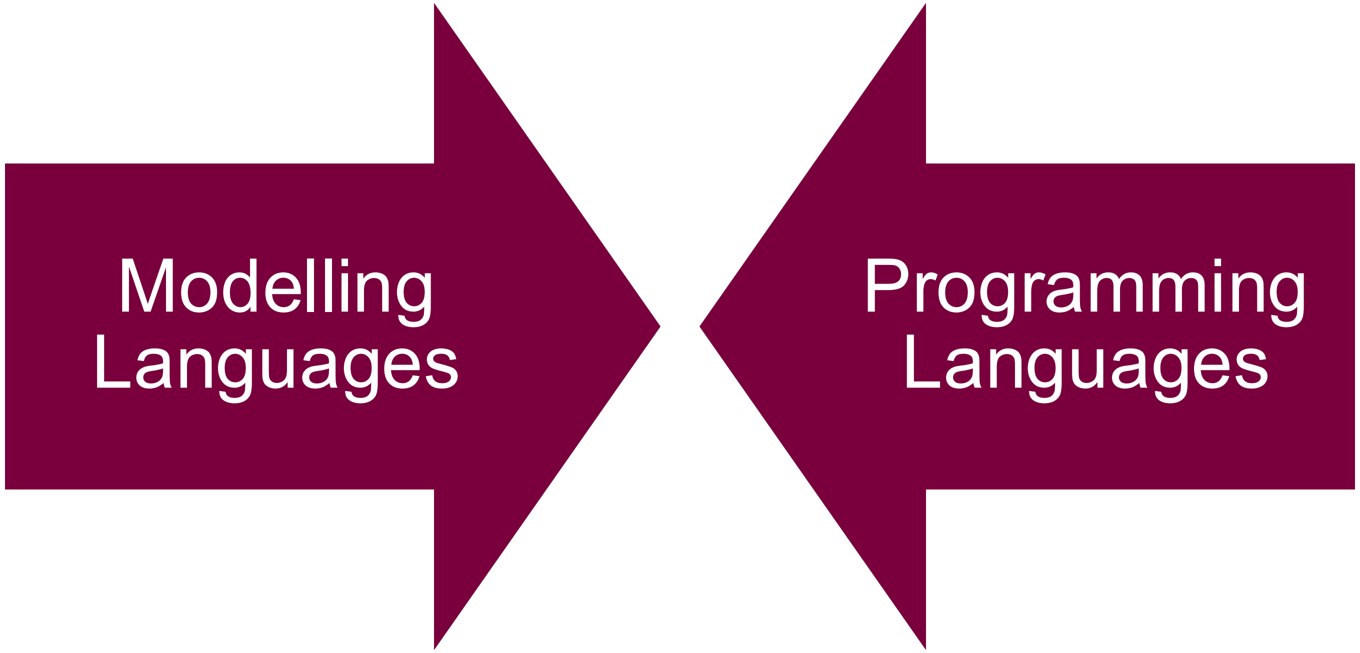
In Programming and Modelling Languages

		Sets						Relations											Efficiently Executable
		$\cup$	$\cap$	$\setminus$	$\times$	$\subseteq$	$\{x \cdot P \mid E\}$	dom ran	$\cup$	$\cap$	$\setminus$	$\subseteq$	$R[S]$	[illegible]	$\circ$	$\lhd$	R^{-1}	Multiplicity	
Programming Languages	Python	✓	✓	✓	✓	✓	✓	✓	✗	*	*	*	*	✓	✗	✗	✗	✗	*
	Haskell	✓	✓	✓	✓	✓	*	✓	✗	✓	✓	✓	*	✓	*	*	✓	*	✓
	Rust	✓	✓	✓	*	✓	*	✓	✗	✗	✗	✗	✓	*	✗	✗	✗	✗	✓
	C	*	*	*	*	*	✗	*	*	*	*	*	*	*	*	*	*	✗	✓
	APL	✓	✓	✓	✓	*	✗	✓	*	*	*	*	*	*	✓	*	✓	✗	✓
Modelling Languages	SetL	✓	✓	✓	*	✓	✓	✓	✓	✓	✓	✓	✓	*	*	✓	*	*	*
	UML	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	*
	Event-B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	*
	Alloy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗

Goals

Of our set-based language

- Syntax close to discrete mathematics
- Simple, small, and usable
- Highly performant execution
- High-level set-theory optimization



Abstract Data Types in Python

Implementation of Set Operations in Modern Languages

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

```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))
```

Related Work

- ProB animation engine for Event-B [Leuschel 22]
- GHC rewrite rules [Jones, Tolmach, Hoare 01]
- Rewrite systems for set-theory focused optimization [Cristia, Rossi 22][Vissier, Benaissa, Tolmach 98][Smith, Westfold 20]
- SETL data type *lowering* from abstract types to suitable concrete structures [Schonberg, Schwartz, Sharir 79][Freudenberger, Schwartz, Sharir 83]

Our Term Rewriting System

Overview: 7 Phases

- Set Comprehension Construction
- Predicate Disjunctive Normal Form
- Nesting
- Generator Selection
- Code Generation 1
- Equality Elimination
- Code Generation 2

Phase 1: Set Comprehension Construction

Set-typed variables and literals are decomposed into set comprehensions

Rewrite Rules	
Predicate Operations	$S \cup T \rightsquigarrow \{x \cdot x \in S \lor x \in T \mid x\}$
	$S \cap T \rightsquigarrow \{x \cdot x \in S \land x \in T \mid x\}$
	$S \setminus T \rightsquigarrow \{x \cdot x \in S \land x \notin T \mid x\}$
Membership Collapse	$x \in \{E \mid P\} \rightsquigarrow \exists y \cdot P \land x = E$
Image	$R[S] \rightsquigarrow \{y \cdot \exists x \cdot x \mapsto y \in R \land x \in S \mid y\}$
Composition	$x \mapsto z \in R \circ Q \rightsquigarrow \exists y, y' \cdot x \mapsto y \in R \land y' \mapsto z \in Q \land y = y'$
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

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

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

$$\rightsquigarrow$$

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

Phase 2: Predicate Disjunctive Normal Form (DNF)

Predicate structure: top-level ∨-operations with nested ∧-operations

Rewrite Rules	
Flatten Nested ∧	$x_1 \wedge \cdots \wedge (x_i \wedge x_{i+1}) \wedge \cdots \rightsquigarrow x_1 \wedge \cdots \wedge x_i \wedge x_{i+1} \wedge \ldots$
Flatten Nested ∨	$x_1 \vee \cdots \vee (x_i \vee x_{i+1}) \vee \cdots \rightsquigarrow x_1 \vee \cdots \vee x_i \vee x_{i+1} \vee \ldots$
Distribute ∧ over ∨	$x \wedge (y \vee z) \rightsquigarrow (x \wedge y) \vee (x \wedge z)$
Double Negation	$\neg\neg x \rightsquigarrow x$
De Morgan	$\neg(x \wedge y) \rightsquigarrow \neg x \vee \neg y$
	$\neg(x \vee y) \rightsquigarrow \neg x \wedge \neg y$

Post-condition:

- All set-like terms must be comprehensions
- Quantifier predicates are in DNF

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

Phase 3: Nesting

Restructuring quantifications to better suit code generation

Rewrite Rules	
Nesting	$\{x, y \cdot P \land Q \mid E\} \rightsquigarrow \{x \cdot P \mid \{ y \cdot Q \mid E \}\}$ Where y does not occur in P

- Nesting selection is currently arbitrary; in the future we plan to use better selection heuristics

Post-condition:
- All set-like terms must be comprehensions
- Quantifier predicates are in DNF
- One bound variable per quantifier

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

$$\Downarrow$$

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

Phase 4: Generator Selection

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

Rewrite Rules	
Generator Selection	$\bigwedge P_i \rightsquigarrow P_g \land \bigwedge_{P_i \neq P_g} P\_i$ Where: • P_g is of the form $x \in S$ • x is the quantifier's bound variable • S is 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 quantifier
- All predicate ∨-clauses have an assigned generator

- In the case of multiple candidate generators, selection is arbitrary
- Generator Selection is only valid inside a quantifier's predicate

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

$$\Downarrow$$

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

Phase 5: Code Generation 1

Start lowering expressions into imperative-like loops

Rewrite Rules			
Quantifier Generation	$\bigoplus E \mid P$	$\rightsquigarrow$	$a := identity(\bigoplus)$ $\textbf{loop}\ P\ \textbf{do}$ $\quad a := a \bigoplus E$
Disjunct Conditional	$\textbf{loop}\ \bigvee P_i\ \textbf{do}$ $\quad body$	$\rightsquigarrow$	$\textbf{loop}\ P_0\ \textbf{do}$ $\quad body$ $\textbf{loop}\ P_1 \land \neg P_0\ \textbf{do}$ $\quad body$ $\ldots$
Nested Quantifier	$a := a \bigoplus (\bigoplus E \mid P)$	$\rightsquigarrow$	$\textbf{loop}\ P\ \textbf{do}$ $\quad a := a \bigoplus E$

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

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

$$\rightsquigarrow$$

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

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 6: Equality Elimination

Eliminate all undefined variables (with implicit ∃ quantifiers)

Rewrite Rules			
Equality Elimination	$\textbf{loop} \bigwedge P_i \textbf{ do}$ $\quad body$	$\rightsquigarrow$	$\textbf{loop} \bigwedge_{P_i \neq P_{eq}} P_i[x := E] \textbf{ do}$ $\quad body[x := E]$ Where: • P_{eq} is of the form $x = E$ • x is undefined and unbound in the current scope

Post-condition:

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

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

Phase 7: Code Generation 2

Eliminate all intermediate $\boldsymbol{loop}$ structures

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

$$
\begin{array}{l}
numMeals := 0 \\
\textbf{loop } \boldsymbol{p \mapsto r \in location} \wedge r = room \textbf{ do} \\
\quad \textbf{loop } \boldsymbol{v \mapsto p' \in attends} \wedge p = p' \textbf{do} \\
\quad\quad numMeals := numMeals + 1
\end{array}
$$

$$\rightsquigarrow$$

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

Post-condition:

- Code is imperatively executable

Visitor Information System - Translation

Motivating Examples

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

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

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

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

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

Warehouse Inventory System – Translation

Motivating Examples

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

Restocking price, with a target inventory $inventory_t$:

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

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

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

Conway's Game of Life – Translation

Motivating Examples

$$
\begin{aligned}
neigh = \{x, y, x_n, y_n \cdot\ & x \in Boundary \land y \in Boundary \\
& \land x_n \in (x-1)..(x+1) \\
& \land y_n \in (y-1)..(y+1) \\
& \land x \neq x_n \land y \neq y_n \\
& \mid (x, y) \mapsto (x_n, y_n)\}
\end{aligned}
$$

$$
\begin{aligned}
& nextStep = \\
& \{\, c \cdot c \in Living \land\ card(neigh[\{c\}] \cap Living) = 2 \mid c\} \\
& \ \cup\ \{\, c \cdot c \in neigh\,[Living] \\
& \qquad \land\ card(neigh\,[\{c\}]\ \cap Living) = 3 \mid c\}
\end{aligned}
$$

$$
\begin{aligned}
& nextStep = \\
& \quad \{c \cdot c \in Boundary \times Boundary \\
& \ \land \exists n \cdot n = card(neigh[\{c\}] \cap Living) \\
& \ \land ((c \in Living \land n = 2) \lor n = 3) \mid c\}
\end{aligned}
$$

$$
\begin{aligned}
& nextStep\ := \emptyset \\
& \textbf{for } i \in Boundary \textbf{ do} \\
& \quad \textbf{for } j \in Boundary \textbf{ do} \\
& \quad\quad n := 0 \\
& \quad\quad \textbf{for } n' \in neigh[\{i \mapsto j\}] \textbf{ do} \\
& \quad\quad\quad \textbf{if } n' \in Living \textbf{ then} \\
& \quad\quad\quad\quad n := n + 1 \\
& \quad\quad \textbf{if } n = 3 \lor (n = 2 \land i \mapsto j \in Living) \textbf{ then} \\
& \quad\quad\quad nextStep := nextStep \cup \{i \mapsto j\}
\end{aligned}
$$

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?…

Preliminary Results

Operation Running Time

Operation	Expected Running Time
$S \cup T$	$O(\lvert S\rvert + \lvert T\rvert)$
$S \cap T$	$O(min(\lvert S\rvert, \lvert T\rvert))$
$S \times T$	$O(\lvert S\rvert\lvert T\rvert)$
$R[S]$	$O(min(\lvert S\rvert, \lvert R\rvert))$
$S \lhd$R	$O(\lvert R\rvert)$
R⍰Q	$O(max(\lvert Q\rvert, \lvert R\rvert))$
R $\circ$ Q	$O(\lvert R\rvert\lvert Q\rvert)$

Preliminary Results

Examples Running Time

	Core Expression	Naïve Running Time	Optimized Running Time
Visitor Information System	$(location^{-1} \circ attends^{-1})[\{room\}]$	$O(\lvert location\rvert \lvert attends\rvert)$	$O(\lvert attends\rvert)$
Warehouse Inventory System	$\sum p \mapsto n_m \cdot p \mapsto n_m$ $\in (inventory_t - inventory) \circ catalogue^{-1}$ $\mid p \times n_m$	$O(\lvert inventory_t\rvert \lvert catalogue\rvert)$	$O(\lvert inventory_t\rvert)$
Conway's Game of Life	$\{c \cdot c \in Living$ $\wedge\ card(neigh[\{c\}] \cap Living) = 2 \mid c\}$ $\cup \{c \cdot c \in neigh\,[Living]$ $\wedge\ card(neigh\,[\{c\}] \cap Living) = 3 \mid c\}$	$O(2\lvert Boundary\rvert^2)$	$O(\lvert Boundary\rvert^2)$

$Living \in \mathbb{P}(Boundary \times Boundary)$

Preliminary Results

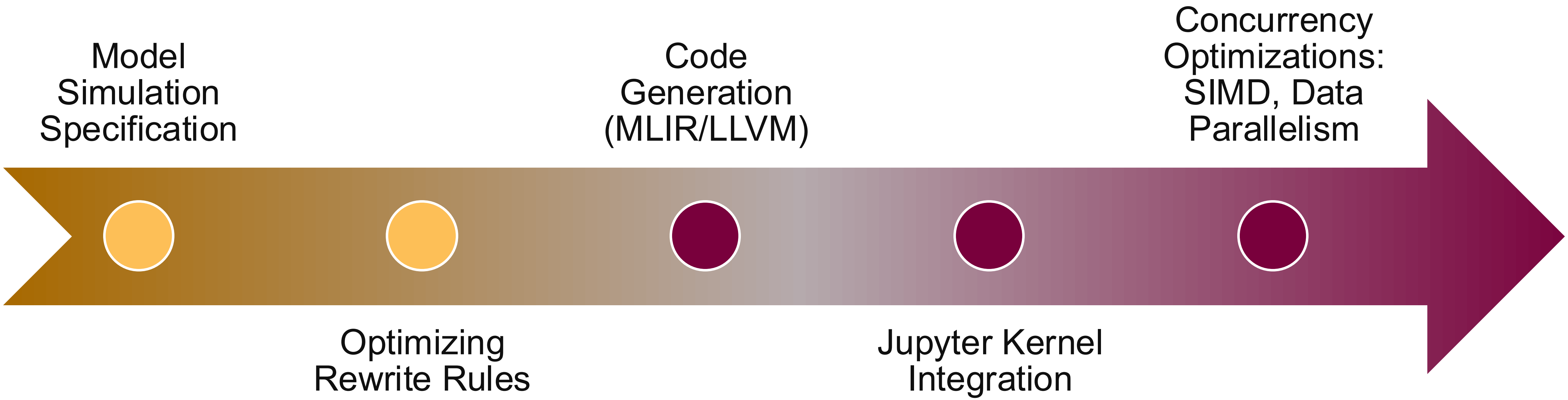
Complex Operation Running Time

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

- 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