Relations as Executable Specifications

Taming Partiality and Non-determinism Using Invariants

Nuno Macedo Hugo Pacheco Alcino Cunha

HASLab — High Assurance Software Laboratory
INESC TEC & Universidade do Minho, Braga, Portugal

RAMiCS 13
September 20, 2012, Cambridge

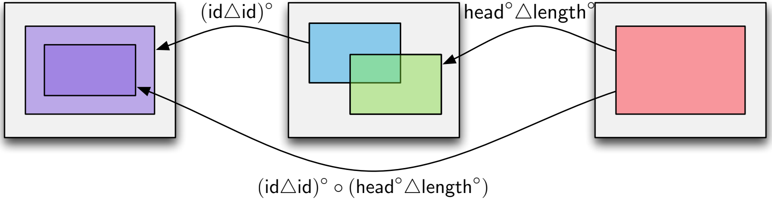
Introduction

- *Relational calculus* provides a more natural way to specify programs;
- Many programs are *partial* and *non-deterministic*;
- A *point-free* (PF) version has been used in a variety of computer science areas;
- Such specifications are not amenable for execution.

Motivating Example

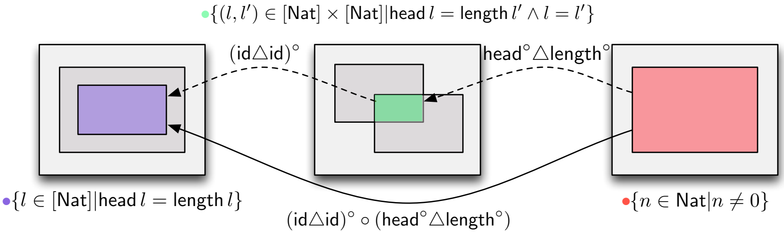
$$(\mathsf{id} \vartriangle \mathsf{id})^\circ \circ (\mathsf{head}^\circ \vartriangle \mathsf{length}^\circ) : \mathsf{Nat} \rightarrow [\mathsf{Nat}]$$

- Calculates a list with length n and the same n at its head;
- Not *total* nor *functional*;
- Very inefficient given a naive semantics;
- head° could be generating all lists by increasing length and never reach n.



Motivating Example

- We can *predict* the behavior of partial expressions by calculating the exact *domain* and *range*;
- They can also be used to *narrow* non-deterministic executions.



Taming Partiality and Non-determinism

- We propose a PF relational framework where data-types are enhanced with *invariants*;
- The simplicity of the PF calculus allows us to develop practical type-inference and type-checking algorithms;
- Those invariants can then used to run the specifications more efficiently;
- *Bidirectional transformations* are our application scenario.

PF Relational Calculus

Identity	$id : A \rightarrow A$
Top	$\top : A \rightarrow B$
Bottom	$\bot : A \rightarrow B$
Converse	$\cdot^{\circ} : (A \rightarrow B) \rightarrow (B \rightarrow A)$
Composition	$\cdot \circ \cdot : (B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)$
Intersection	$\cdot \cap \cdot : (A \rightarrow B) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow B)$
Union	$\cdot \cup \cdot : (A \rightarrow B) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow B)$
Split	$\cdot \vartriangle \cdot : (A \rightarrow B) \rightarrow (A \rightarrow C) \rightarrow (A \rightarrow B \times C)$
Projections	$\pi_1 : A \times B \rightarrow A$ and $\pi_2 : A \times B \rightarrow B$
Either	$\cdot \triangledown \cdot : (B \rightarrow A) \rightarrow (C \rightarrow A) \rightarrow (B + C \rightarrow A)$
Injections	$i_1 : A \rightarrow (A + B)$ and $i_2 : B \rightarrow (A + B)$
Constants	$! : A \rightarrow 1$ and $\underline{\cdot} : B \rightarrow (A \rightarrow B)$
Conditional	$\cdot ? : (A \rightarrow A) \rightarrow (A \rightarrow A + A)$

- The simple equational rules are harnessed in a strategic rewrite system that simplifies expressions.

Membership Semantics

$$
\begin{array}{ll}
a' \; [\![\mathrm{id}]\!] \; a & = a \equiv a' \\
a \; [\![R^\circ]\!] \; b & = b \; [\![R]\!] \; a \\
b \; [\![S \circ R]\!] \; a & = \exists \; c. \; b \; [\![S]\!] \; c \wedge c \; [\![R]\!] \; a \\
b \; [\![R \cap S]\!] \; a & = b \; [\![R]\!] \; a \wedge b \; [\![S]\!] \; a \\
b \; [\![R \cup S]\!] \; a & = b \; [\![R]\!] \; a \vee b \; [\![S]\!] \; a \\
(b, c) \; [\![R \vartriangle S]\!] \; a & = b \; [\![R]\!] \; a \wedge c \; [\![S]\!] \; a \\
a' \; [\![\pi_1]\!] \; (a, b) & = a \equiv a' \\
b' \; [\![\pi_2]\!] \; (a, b) & = b \equiv b' \\
a \; [\![R \triangledown S]\!] \; (\mathrm{Left} \; b) & = a \; [\![R]\!] \; b \\
a \; [\![R \triangledown S]\!] \; (\mathrm{Right} \; c) & = a \; [\![S]\!] \; c \\
(\mathrm{Left} \; a') \quad [\![i_1]\!] \; a & = a \equiv b \\
(\mathrm{Left} \; a') \quad [\![i_2]\!] \; b & = \mathrm{False} \\
(\mathrm{Right} \; b') \; [\![i_1]\!] \; a & = \mathrm{False} \\
(\mathrm{Right} \; b') \; [\![i_2]\!] \; b & = a \equiv b \\
b \; [\![\top]\!] \; a & = \mathrm{True} \\
b \; [\![\bot]\!] \; a & = \mathrm{False} \\
1 \; [\![!]\!] \; a & = \mathrm{True} \\
b' \; [\![\underline{b}]\!] \; a & = b \equiv b'
\end{array}
$$

Execution Semantics

$$
\begin{array}{lll}
[\![\mathrm{id}]\!] & a & = \{a\} \\
[\![R^\circ]\!] & b & = \{a \mid a \leftarrow A, a\, [\![R^\circ]\!]\, b\} \\
[\![S \circ R]\!] & a & = \{b \mid c \leftarrow [\![R]\!]\, a, b \leftarrow [\![S]\!]\, c\} \\
[\![R \cap S]\!] & a & = \{b \mid b \leftarrow [\![R]\!]\, a, b\, [\![S]\!]\, a\} \\
[\![R \cup S]\!] & a & = [\![R]\!]\, a \,\cup\, [\![S]\!]\, a \\
[\![R \vartriangle S]\!] & a & = \{(b, c) \mid b \leftarrow [\![R]\!]\, a, c \leftarrow [\![S]\!]\, a\} \\
[\![\pi_1]\!] & (a, b) & = \{a\} \\
[\![\pi_2]\!] & (a, b) & = \{b\} \\
[\![R \triangledown S]\!] & (\mathrm{Left}\ b) & = [\![R]\!]\, b \\
[\![R \triangledown S]\!] & (\mathrm{Right}\ c) & = [\![S]\!]\, c \\
[\![i_1]\!] & a & = \{\mathrm{Left}\ a\} \\
[\![i_2]\!] & b & = \{\mathrm{Right}\ b\} \\
[\![\top]\!] & a & = B \\
[\![\bot]\!] & a & = \{\} \\
[\![!]\!] & a & = \{1\} \\
[\![\underline{b}]\!] & a & = \{b\}
\end{array}
$$

Predicates as Coreflexives

- *Domain* δR and *range* ρR are predicates that can be defined as coreflexives;
- *Coreflexives* $\Phi : A \to A$ are relations smaller than the identity;
- If a satisfies the predicate Φ then $a \llbracket \Phi \rrbracket a$;
- For pairs $A \times B$ related by $R : A \to B$, the invariant is lifted as $[R] : A \times B \to A \times B$;
- Invariants on coproducts are simply the coproduct $\Phi + \Psi$;
- $R : \Phi \to \Psi$ denotes $\delta R = \Phi$ and $\rho R = \Psi$.

Inferring Checkable Invariants

- Domain and range can be directly computed as $\delta R = R^\circ \circ R \cap \mathsf{id}$ and $\rho R = R \circ R^\circ \cap \mathsf{id}$;
- May result in inefficient membership tests (due to composition);
- By expanding the definition, we define an equivalent definition with most compositions removed;
- Others will fall in the special case $b\ (f^\circ \circ R \circ g)\ a \equiv (f\ b)\ R\ (g\ a)$;
- The rewriting system further simplifies the formula and issues a warning if problematic expressions remain.

Example

$$(id \triangle id)^\circ \circ (head^\circ \triangle length^\circ) : Nat \rightarrow [Nat]$$

$$
\begin{aligned}
&\rho((id \triangle id)^\circ \circ (head^\circ \triangle length^\circ)) \\
&\quad = \{\textit{Range definition}\} \\
&\rho((id \triangle id)^\circ \circ \rho(head^\circ \triangle length^\circ)) \\
&\quad = \{\textit{Range definition}\} \\
&\rho((id \triangle id)^\circ \circ [length^\circ \circ head]) \\
&\quad = \{\textit{Range definition}\} \\
&\delta([length^\circ \circ head] \circ (id \triangle id)) \\
&\quad = \{\textit{Domain definition}, \textit{Simplifications} : \textit{PF Laws}\} \\
&(head^\circ \circ length) \cap id
\end{aligned}
$$

$$(id \triangle id)^\circ \circ (length^\circ \triangle head^\circ) : in_N \circ (\bot + id) \circ out_N \rightarrow (head^\circ \circ length) \cap id$$

Optimizing Non-deterministic Executions

- After determining the domain and range, they can be propagated down to *primitives*,
- This reduces the generation of irrelevant intermediate values;

$$
\begin{aligned}
&[\![\mathrm{id} : \Phi \rightarrow \Psi]\!]\ a = [\![\Psi]\!]\ a \\
&[\![\pi_1 : [U] \rightarrow \Psi]\!]\ (a, b) = [\![\Psi]\!]\ a \\
&[\![R \circ S : \Phi \rightarrow \Psi]\!]\ a = \{\, b \mid \ c \leftarrow [\![S : \Phi \rightarrow \delta R]\!]\ a, \\
&\qquad\qquad\qquad\qquad\qquad\ \ b \leftarrow [\![R : \rho S \rightarrow \Psi]\!]\ c \,\} \\
&[\![R \cap S : \Phi \rightarrow \Psi]\!]\ a = \{\, b \mid b \leftarrow [\![R : \Phi \cap \delta S \rightarrow \rho(\Psi \circ S \circ \underline{a})]\!]\ a \,\}
\end{aligned}
$$

Recursive Relations with Invariants

- We support the well-know recursion patterns of *catamorphisms* (folds) and *anamorphisms* (unfolds);
- Execution is not problematic, as it is performed by unfolding their definitions;
- However, there is no known normal form for invariants over recursive types;
- The rewrite system tries to simplify the generic domain/range expression.

Recursive Relations: Example

$$\mathsf{unzip} : [A \times B] \to [A] \times [B]$$

$$
\begin{aligned}
& \rho\,\mathsf{unzip} \\
& \quad = \{\textit{Range definition}\} \\
& (\mathsf{unzip} \circ \mathsf{unzip}^\circ) \;\cap\; \mathsf{id} \\
& \quad = \{\textit{Simplifications} : \mathsf{unzip}\ \textit{is functional}, \textit{Liftify} : \textit{range of}\ \mathsf{unzip}\ \textit{is a product}\} \\
& [\pi_2 \circ \mathsf{unzip} \circ \mathsf{unzip}^\circ \circ \pi_1^\circ] \\
& \quad = \{\textit{Catamorphism fusion} : \pi_1 \circ g = \mathsf{nil} \triangledown (\mathsf{cons} \circ (\pi_1 \times \mathsf{id})) \circ F\ \pi_1\} \\
& [(\!|\mathsf{nil} \triangledown (\mathsf{cons} \circ (\pi_1 \times \mathsf{id}))|\!) \circ (\!|\mathsf{nil} \triangledown (\mathsf{cons} \circ (\pi_2 \times \mathsf{id}))|\!)^\circ] \\
& \quad = \{\textit{Definitions} : \mathsf{map}\} \\
& [(\mathsf{map}\ \pi_1) \circ (\mathsf{map}\ \pi_2)^\circ] \\
& \quad = \{\textit{Simplifications} : \mathsf{map}\ \textit{converse}, \mathsf{map}\ \textit{fusion}\ (\textit{see below})\} \\
& [\mathsf{map}\ (\pi_1 \circ \pi_2^\circ)] \\
& \quad = \{\textit{Simplifications} : \textit{PF Laws}\} \\
& [\mathsf{map}\ \top]
\end{aligned}
$$

$$\mathsf{unzip} : \mathsf{id} \to [\mathsf{map}\ \top]$$

Bidirectional Transformations

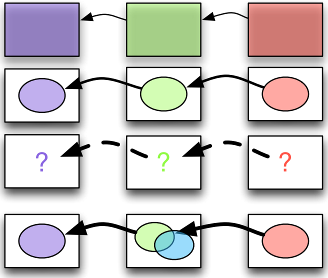
- *Bidirectional transformations* are transformations between data-structures that may be applied in both directions;
- Defining the transformations individually is expensive and error-prone;
- BXs should be inferred a single specification;
- We propose a solution using the relational calculus.

Bidirectional Transformations

- *Lenses* are one of the most famous BX frameworks;
- A lens $S \trianglerighteq V$ between sources S and more abstract views V consists of transformations $\mathsf{Get} : S \to V$ and $\mathsf{Put} : V \times S \to S$;
- It is said to be *well-behaved* if:
  - $\mathsf{Get} \circ \mathsf{Put} \subseteq \pi_1$ (acceptability);
  - $\mathsf{Put} \circ (\mathsf{Get} \vartriangle \mathsf{id}) \subseteq \mathsf{id}$ (stability);
- Usually there are multiple valid Puts, but existing frameworks are deterministic.

Bidirectional Transformations

- In principle, it is possible to lift any functional expression to a well-behaved lens;
- Existing frameworks either:
  - Have maximum updatability but disregard some operators;
  - Refine the type-system to allow operators with smaller updatability;
  - Support any operator but disregard updatability;
- We refine the type-system and guarantee maximum updatability.



Generic Non-deterministic Lenses

- Using relational calculus we can define a generic Put that is the largest relation that satisfies the properties;
- A transformation $\mathsf{get} : A \to B$ can be lifted to a well-behaved non-deterministic lens $\lfloor \mathsf{get} \rfloor : \delta \mathsf{get} \trianglerighteq \rho \mathsf{get}$, with

$$\mathsf{Put} = (\pi_2 \bigtriangledown (\mathsf{get}^\circ \circ \pi_1)) \circ [\mathsf{get}^\circ]\,?$$

- Emerges naturally from the lens laws:
  - $[\mathsf{get}^\circ]\,?\,(v, s)$ tests if v was changed, returning either the original s (acceptability), or any source s' such that $s' = \mathsf{get}\ v$ (stability);
  - Maximum updatability: $\delta \mathsf{Put} = \rho \mathsf{get} \times \delta \mathsf{get}$.

Generic Non-deterministic Lenses

- Type-checking over δget and ρget directly could be undecidable and due to the central role of the converse, Put can not be directly executed;
- Both these issues can be addressed by the optimizations already presented;
- Forward transformations are functional so problematic cases are very limited:
  - Regarding type-checking, only particular ranges of splits are possibly undecidable;
  - The backward transformation can be efficiently executed;
- Recursive expressions are also supported as they preserve the functionality of their algebras.

Generic Non-deterministic Lenses: Example

$$\lfloor \pi_1 \vartriangle \mathsf{id} \rfloor : \mathsf{id} \trianglerighteq [\pi_1^\circ]$$

- The range is $[\pi_1^\circ] : A \times (A \times B) \to A \times (A \times B)$, so Put only takes views $(a, (b, c))$ where $a \equiv b$;
- When the view is updated, $(\pi_1 \vartriangle \mathsf{id})^\circ$ will run, and π_1° could generate all pairs until reaching (a, c);
- With our optimization, the output of π_1 is restricted to have c in the second element.

Conclusions

- We have presented mechanisms for the efficient execution PF relational expressions over data-types with invariants;
- Regarding BX, we identify an open problem in the composition of lenses;
- By modeling lenses in this framework we were able to implement an expressive PF BX language with maximum updatability;

- Researching possible normal forms for invariants over recursive types;
- Exploring mechanisms for a better control of the non-determinism through user-defined quality measures.