Confluence of a Computational Lambda Calculus for Higher-Order Relational Queries

Claudio Sacerdoti Coen   Riccardo Treglia

IWC 23 - Obergurgl, 23/08/2023

Università di Bologna

Contents

Introduction

Our Starting Point

[W. Ricciotti, J. Cheney - Strongly Normalizing Higher-Order Relational Queries]

The Nested Relational Calculus (NRC) provides a principled foundation for integrating database queries into PL.

It is easy to implement a terminating rewriting algorithm for normalizing NRC queries to flat relational queries, which can be translated to idiomatic SQL queries.

Our ongoing work

A monadic calculus mirroring NRC

Define a reduction theory and prove it confluent

Non-idempotent intersection type assignment system

Computational Monads

A *monad* over a category of domains $\mathcal{D}$ is a triple $(T, [\cdot], \star)$

Objects

D is the type of a value;

TD is the type of computations (possibly with **effects**) over D.

Computational Monads

A *monad* over a category of domains $\mathcal{D}$ is a triple $(T, [\cdot], \star)$

Objects

D is the type of a value;

TD is the type of computations (possibly with **effects**) over D.

Operators

$[\cdot] : D \to TD$ (Haskell: **return**);

$\star : TD \to (D \to TD) \to TD$ (Haskell: $>>=$).

The monadic approach

The computational λ-calculus, was introduced as a metalanguage to describe **computational effects** in programming languages.

The monadic approach

The computational λ-calculus, was introduced as a metalanguage to describe **computational effects** in programming languages.

At a semantic level, it relies on the categorical notion of monad.

$$f : A \to \mathbf{T}B \text{ where } \mathbf{T} \text{ is a monad}$$

The monadic approach

The computational λ-calculus, was introduced as a metalanguage to describe **computational effects** in programming languages.

In my previous works (see e.g. IWC'20, IWC'21), the computational core $\lambda_{©}$ was presented.

Computational core

$\approx$

Plotkin's call-by-value λ-calculus + monad operators

The monadic approach

The computational λ-calculus, was introduced as a metalanguage to describe **computational effects** in programming languages.

In this works, the computational core $\lambda_{©}$ is extended with specific operations to handle with tables, such as:

- Join tables
- Say: 'this is a table' ... aka reflection
- The inverse of reflection: reification

$$\lambda_{\mathsf{SQL}}$$
$$\approx$$
computational core + (list) monad operators + reify/reflect tables

SQL λ-calculus

Definition (Term syntax)

$$\begin{array}{rrcl} \mathit{Val}: & V, W & ::= & x \mid \lambda x.M \\ \mathit{Com}: & M, N & ::= & [V] \mid M \star V \end{array}$$

SQL λ-calculus

Definition (Term syntax)

$$
\begin{array}{rrcl}
Val: & V, W & ::= & x \mid \lambda x.M \mid \langle\!\langle M \rangle\!\rangle \\
Com: & M, N & ::= & [V] \mid M \star V \mid M \uplus M \mid \emptyset \mid\, !V
\end{array}
$$

Definition (Reduction)
The relation $\rightarrow_{\lambda_{\mathsf{SQL}}}$ is the union of the following binary relations over *Com*:

$$
\begin{array}{rrll}
\beta_c) & [V] \star \lambda x.M & \mapsto_{\beta_c} & M\{V/x\} \\
\sigma\,) & (L \star \lambda x.M) \star \lambda y.N & \mapsto_{\sigma} & L \star \lambda x.(M \star \lambda y.N) \qquad \text{for } x \notin \mathsf{fv}(N) \\
\uplus_l) & (M \uplus N) \star \lambda x.P & \mapsto_{\uplus_l} & (M \star \lambda x.P) \uplus (N \star \lambda x.P) \\
\uplus_r) & M \star \lambda x.(N \uplus P) & \mapsto_{\uplus_r} & (M \star \lambda x.N) \uplus (M \star \lambda x.P) \\
\emptyset_1) & \emptyset \star \lambda x.M & \mapsto_{\emptyset_1} & \emptyset \\
\emptyset_2) & M \star \lambda x.\emptyset & \mapsto_{\emptyset_2} & \emptyset \\
!) & !\langle\!\langle M \rangle\!\rangle & \mapsto_{!} & M
\end{array}
$$

Definition (Reduction)
The relation $\to_{\lambda_{\mathsf{SQL}}}$ is the union of the following binary relations over *Com*:

$$
\begin{array}{rrcll}
\beta_c) & [V] \star \lambda x.M & \mapsto_{\beta_c} & M\{V/x\} & \\
\sigma\,) & (L \star \lambda x.M) \star \lambda y.N & \mapsto_{\sigma} & L \star \lambda x.(M \star \lambda y.N) & \text{for } x \notin \mathsf{fv}(N) \\
\uplus_l) & (M \uplus N) \star \lambda x.P & \mapsto_{\uplus_l} & (M \star \lambda x.P) \uplus (N \star \lambda x.P) & \\
\uplus_r) & M \star \lambda x.(N \uplus P) & \mapsto_{\uplus_r} & (M \star \lambda x.N) \uplus (M \star \lambda x.P) & \\
\emptyset_1) & \emptyset \star \lambda x.M & \mapsto_{\emptyset_1} & \emptyset & \\
\emptyset_2) & M \star \lambda x.\emptyset & \mapsto_{\emptyset_2} & \emptyset & \\
!) & !\langle\!\langle M \rangle\!\rangle & \mapsto_{!} & M &
\end{array}
$$

The *reduction* $\to_{\lambda_{\mathsf{SQL}}}$ is the contextual closure of λ_{SQL} under *computational contexts*, where such contexts are mutually defined with value contexts as follows:

$$\mathrm{V} ::= \langle \cdot_{Val} \rangle \mid \lambda x.\mathrm{C} \mid \langle\!\langle \mathrm{C} \rangle\!\rangle$$

$$\mathrm{C} ::= \langle \cdot_{Com} \rangle \mid [\mathrm{V}] \mid \mathrm{C} \star V \mid M \star \mathrm{V} \mid \mathrm{C} \uplus M \mid M \uplus \mathrm{C} \mid !\mathrm{V}$$

We equip the calculus with an equational theory for multisets, taken from [Ricciotti and Cheney, 22].

Definition (Equational theory)
Be E an equational theory defined, as follows, plus associativity:

$$\mathit{Comm}) \quad M \uplus N \;=\; N \uplus M \qquad\qquad \mathit{Empty}) \quad \emptyset \uplus \emptyset \;=\; \emptyset$$

Note: $\emptyset \uplus M \neq M$

Modularizing Confluence - Getting rid of the equational theory

Definition
Given a reduction relation $\rightarrow$ and an equational theory $=_E$, we say that $\rightarrow$ commutes over $=_E$ if for all M, N, L such that $M =_E N \rightarrow L$, there exists P such that $M \rightarrow P =_E L$.

Lemma (Hindley-Rosen)
Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be relations on the set A. If $\mathcal{R}_1$ and $\mathcal{R}_2$ are confluent and commute with each other, then $\mathcal{R}_1 \cup \mathcal{R}_2$ is confluent.

We will exploit that to focus just on the reduction relation while proving confluence.

Hence, by since $=_E$ commutes with $\rightarrow$, one needs just the confluence of $\rightarrow$ to assert the confluence of $\rightarrow$ modulo E.

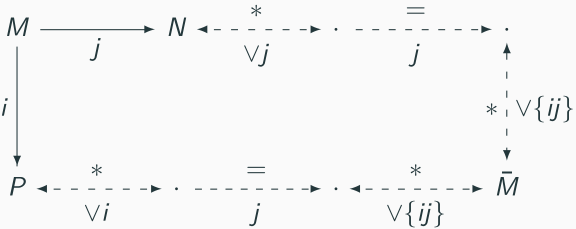
Decreasing diagram

Definition (Decreasing, van Oostrom)

An rewriting relation $\mathcal{R}$ is *locally decreasing* if there exist a presentation $(R, \{\to_i\}_{i \in I})$ of $\mathcal{R}$ and a well-founded strict order $>$ on I such that:

$$\xleftarrow[i]{} \cdot \xrightarrow[j]{} \;\subseteq\; \xleftrightarrow[\vee i]{*} \cdot \xrightarrow[j]{=} \cdot \xleftrightarrow[\vee\{ij\}]{*} \cdot \xleftarrow[i]{=} \cdot \xleftrightarrow[\vee j]{*} ,$$

where $\vee \bar{I} = \{i \in I \mid \exists k \in \bar{I}.\ k > i\}$.



Decreasing diagram

Definition (Decreasing, van Oostrom)

An rewriting relation $\mathcal{R}$ is *locally decreasing* if there exist a presentation $(R, \{\to_i\}_{i \in I})$ of $\mathcal{R}$ and a well-founded strict order $>$ on I such that:

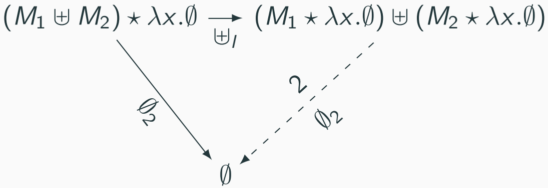
$$\underset{i}{\leftarrow} \cdot \underset{j}{\rightarrow} \;\subseteq\; \underset{\vee i}{\overset{*}{\longleftrightarrow}} \cdot \underset{j}{\overset{=}{\longrightarrow}} \cdot \underset{\vee\{ij\}}{\overset{*}{\longleftrightarrow}} \cdot \underset{i}{\overset{=}{\longleftarrow}} \cdot \underset{\vee j}{\overset{*}{\longleftrightarrow}} \,,$$

where $\vee \bar{I} = \{ i \in I \mid \exists k \in \bar{I}.\ k > i \}$.

Theorem (van Oostrom)

Every locally decreasing rewriting relation $\mathcal{R}$ is confluent.

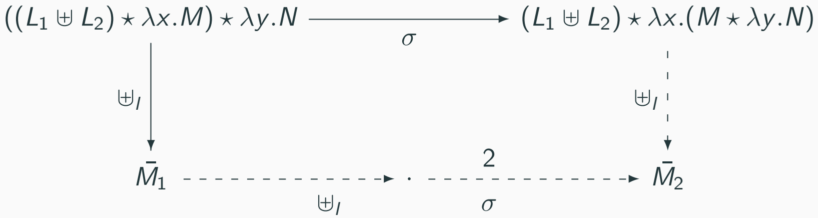
Considers diagrams involving rules of $\uplus_l$ or $\uplus_r$ vs. $\emptyset_1$ and $\emptyset_2$, it is easy to perceive how these rules should be ordered as labels of a potential labellings. Consider, for instance, the following diagram:



In fact, the rules concerning the empty table, $\emptyset_1$ and $\emptyset_2$, can be bottom elements of the order over labels we are searching for.

Which order? $\uplus_I$ vs. σ

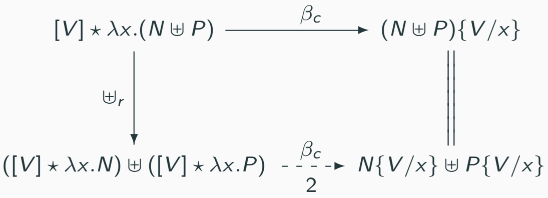
When it comes to comparing $\uplus_I$ vs. σ, the situation is a bit trickier because $\uplus_I$ only quasi-commutes over σ. The following diagrams shows that $\uplus_I$ must be made greater than σ.



where $\bar{M}_1 = ((L_1 \star \lambda x.M) \uplus (L_2 \star \lambda x.M)) \star \lambda y.N)$,
$\bar{M}_2 = (L_1 \star \lambda x.(M \star \lambda y.N)) \uplus (L_2 \star \lambda x.(M \star \lambda y.N))$.

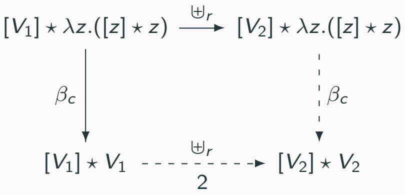
Which order?: β_c vs. $\uplus_r$

The case for β_c vs $\uplus_r$ shows the need for a non-trivial approach, since depending in which context the rules are applied, we need either $\beta_c > \uplus_r$ or $\beta_c < \uplus_r$.



... but ...

$V_1 = \lambda x.(M \star \lambda y.(N_1 \uplus N_2))$
$V_2 = \lambda x.((M \star \lambda y.N_1) \uplus (M \star \lambda y.N_2))$



$(\sigma$ vs. $\uplus_r)$

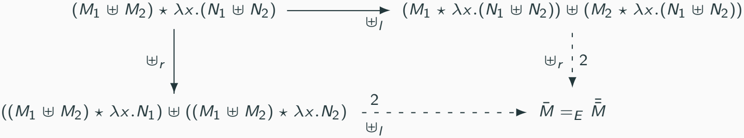
$$
\begin{array}{ccc}
(M \star \lambda x.(N_1 \uplus N_2)) \star \lambda y.L & \xrightarrow[\sigma]{} & M \star \lambda x.((N_1 \uplus N_2) \star \lambda y.L) \\
\downarrow \uplus_r & & \downarrow \uplus_l \\
((M \star \lambda x.N_1) \uplus (M \star \lambda x.N_2)) \star \lambda y.L & & M \star \lambda x.((N_1 \star \lambda y.L) \uplus (N_2 \star \lambda y.L)) \\
\downarrow \uplus_l & & \downarrow \uplus_r \\
((M \star \lambda x.N_1) \star \lambda y.L) \uplus ((M \star \lambda x.N_2) \star \lambda y.L) & \xrightarrow[\sigma]{2} & (M \star \lambda x.(N_1 \star \lambda y.L)) \uplus (M \star \lambda x.(N_2 \star \lambda y.L))
\end{array}
$$

Multi-reduction

The confluence proof we are going to sketch avoids the issue with β_c vs. $\uplus_r$ reported above by considering *multiple reductions*.

A **parallel rewrite step** is a sequence of reductions at a set P of **parallel** positions, ensuring that the result does not depend upon a particular sequentialization of P.
Given a reduction step γ we define its parallel version as $\mathbf{Par}\gamma$.

The case for $\uplus_l$ vs. $\uplus_r$ can seem innocent, for example:



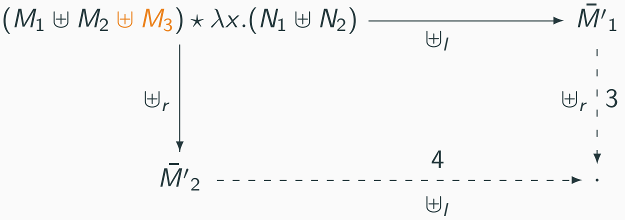
where

$\bar{M} \equiv (M_1 \star \lambda x.N_1) \uplus (M_2 \star \lambda x.N_1) \uplus (M_1 \star \lambda x.N_2) \uplus (M_2 \star \lambda x.N_2)$

and

$\bar{\bar{M}} \equiv (M_1 \star \lambda x.N_1) \uplus (M_1 \star \lambda x.N_2) \uplus (M_2 \star \lambda x.N_1) \uplus (M_2 \star \lambda x.N_2)$

Generalized version of $\uplus_l$ and $\uplus_r$



$$\bar{M}'_1 \equiv (M_1 \star \lambda x.(N_1 \uplus N_2)) \uplus (M_2 \star \lambda x.(N_1 \uplus N_2)) \uplus (M_3 \star \lambda x.(N_1 \uplus N_2))$$

and

$$\bar{M}'_2 \equiv ((M_1 \uplus M_2 \uplus M_3) \star \lambda x.N_1) \uplus ((M_1 \uplus M_2 \uplus M_3) \star \lambda x.N_2)$$

Definition (Generalized union step)

Let us define as *generalized* $\uplus_l$ and $\uplus_r$ steps as follows

$$\textbf{Gen}\uplus_l)\quad (\ldots(M_1 \uplus M_2) \uplus \ldots \uplus M_n) \star \lambda x.N \mapsto_{\textbf{Gen}\uplus_l} (M \star \lambda x.N) \uplus (M_2 \star \lambda x.N) \uplus \ldots \uplus (M_n \star \lambda x.N)$$

$$\textbf{Gen}\uplus_r)\quad M \star \lambda x.(\ldots(N_1 \uplus N_2) \uplus \ldots \uplus N_n) \mapsto_{\textbf{Gen}\uplus_r} (M \star \lambda x.N_1) \uplus (M \star \lambda x.N_2) \uplus \ldots \uplus (M \star \lambda x.N_n)$$

Route to Confluence

We are now ready to state our main result:

Theorem (Confluence)
λ_{SQL} *is confluent.*

1. All reduction rules strongly commute with !.

2. Under the following order for parallel rewriting steps, all remaining rules are decreasing:
$\mathbf{Par}\beta_c > \mathbf{Par}\sigma > \mathbf{ParGen}\uplus_r > \mathbf{ParGen}\uplus_l > \emptyset_1 > \emptyset_2$

The diagrams for the cases $\mathbf{Par}\uplus_l$ vs $\mathbf{Par}\uplus_r$ and $\mathbf{Par}\uplus_r$ vs $\emptyset_1$ only hold up to E.
E.g., $\emptyset \; {}_{\emptyset_1}\!\!\leftarrow \emptyset \star \lambda x.M \uplus N \rightarrow_{\uplus_r} \rightarrow^2_{\emptyset_1} \emptyset \uplus \emptyset$.

3. Confluence is obtained combining the previous points.

Consequences

By confluence, λ_{SQL} normal forms (if exist) are unique.

Moreover, it is possible to characterize normal forms and provide a translation from λ_{SQL} to NRC.

Since λ_{SQL} normal forms (up to E) are translated in NRC normal forms, they are queries, as expected.

Conclusion

Considerations:

Lambda SQL is not just a computational calculus, but has also a co-computational flavour: it is a case study to understand how merge computational effects with co-computational one, also at a semantic level.

The union operator behaves like a delimited control operator that duplicate resources:
this has led some intricacies that made difficult to find a proper label.

Future work:

Unified way as done in [Felgenhauer and van Oostrom, 13].

Merging this method with [FGdLT22].