Relational Reasoning
(Relationel ræsonnement)

Mathias Pedersen, 201808137

Bachelor Report (15 ECTS) in Computer Science
Advisor: Amin Timany
Department of Computer Science, Aarhus University
September 2021

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

▶in English...◀

Mathias Pedersen,
Aarhus, September 2021.

Contents

Chapter 1

Introduction

►motivate and explain the problem to be addressed◄

►example of a citation: [1]◄ ►get your bibtex entries from `https://dblp.org/`◄

Chapter 2

Definition of Language

►create draft◄

Syntax

$$
\begin{aligned}
e ::=\; & () \mid && \text{(unit value)}\\
& x \mid && \text{(variables)}\\
& \overline{n} \mid e+e \mid e-e \mid e \leq e \mid e < e \mid e = e \mid && \text{(integers)}\\
& \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid && \text{(booleans)}\\
& (e, e) \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \mid && \text{(products)}\\
& \mathsf{inj}_1\ e \mid \mathsf{inj}_2\ e \mid \mathsf{match}\ e\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e \mid \mathsf{inj}_2\ x \Rightarrow e\ \mathsf{end} \mid && \text{(sums)}\\
& \mathsf{rec}\ f(x) := e \mid e\ e && \text{(recursive functions)}\\
& \Lambda\, e \mid e\ \_ && \text{(polymorphism)}
\end{aligned}
$$

$$
v ::= () \mid \overline{n} \mid \mathsf{true} \mid \mathsf{false} \mid (v, v) \mid \mathsf{inj}_1\ v \mid \mathsf{inj}_2\ v \mid \mathsf{rec}\ f(x) := e \mid \Lambda\, e \qquad \text{(values)}
$$

$$
\tau ::= \mathsf{Unit} \mid \mathbb{Z} \mid \mathbb{B} \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \tau \mid \forall X.\, \tau \qquad \text{(types)}
$$

$$
\begin{aligned}
K ::=\; & [] \mid K+e \mid v+K \mid K-e \mid v-K \mid K \leq e \mid v \leq K \mid K < e \mid v < K \mid && \text{(evaluation context)}\\
& K = e \mid v = K \mid \mathsf{if}\ K\ \mathsf{then}\ e\ \mathsf{else}\ e \mid (K, e) \mid (v, K) \mid \mathsf{fst}\ K \mid \mathsf{snd}\ K \mid &&\\
& \mathsf{inj}_1\ K \mid \mathsf{inj}_2\ K \mid \mathsf{match}\ K\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e \mid \mathsf{inj}_2\ x \Rightarrow e\ \mathsf{end} \mid K\ e \mid v\ K \mid K\ \_ &&
\end{aligned}
$$

Typing rules

$$\frac{(x:\tau)\in\Gamma}{\Xi \mid \Gamma \vdash x:\tau}\ \textsc{T-Var} \qquad \frac{}{\Xi \mid \Gamma \vdash ():\mathsf{Unit}}\ \textsc{T-Unit} \qquad \frac{}{\Xi \mid \Gamma \vdash \overline{n}:\mathbb{Z}}\ \textsc{T-Int}$$

$$\frac{\Xi \mid \Gamma \vdash e_1:\mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2:\mathbb{Z}}{\Xi \mid \Gamma \vdash e_1+e_2:\mathbb{Z}}\ \textsc{T-Add} \qquad \frac{\Xi \mid \Gamma \vdash e_1:\mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2:\mathbb{Z}}{\Xi \mid \Gamma \vdash e_1-e_2:\mathbb{Z}}\ \textsc{T-Sub}$$

$$\frac{\Xi \mid \Gamma \vdash e_1:\mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2:\mathbb{Z}}{\Xi \mid \Gamma \vdash e_1\leq e_2:\mathbb{B}}\ \textsc{T-Le} \qquad \frac{\Xi \mid \Gamma \vdash e_1:\mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2:\mathbb{Z}}{\Xi \mid \Gamma \vdash e_1< e_2:\mathbb{B}}\ \textsc{T-Lt}$$

$$\frac{\Xi \mid \Gamma \vdash e_1:\mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2:\mathbb{Z}}{\Xi \mid \Gamma \vdash e_1= e_2:\mathbb{B}}\ \textsc{T-Eq} \qquad \frac{}{\Xi \mid \Gamma \vdash \mathsf{true}:\mathbb{B}}\ \textsc{T-True} \qquad \frac{}{\Xi \mid \Gamma \vdash \mathsf{false}:\mathbb{B}}\ \textsc{T-False}$$

$$\frac{\Xi \mid \Gamma \vdash e_1:\mathbb{B} \qquad \Xi \mid \Gamma \vdash e_2:\tau \qquad \Xi \mid \Gamma \vdash e_3:\tau}{\Xi \mid \Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3:\tau}\ \textsc{T-If}$$

$$\frac{\Xi \mid \Gamma \vdash e_1:\tau_1 \qquad \Xi \mid \Gamma \vdash e_2:\tau_2}{\Xi \mid \Gamma \vdash (e_1,e_2):\tau_1\times\tau_1}\ \textsc{T-Pair} \qquad \frac{\Xi \mid \Gamma \vdash e:\tau_1\times\tau_2}{\Xi \mid \Gamma \vdash \mathsf{fst}\ e:\tau_1}\ \textsc{T-Fst} \qquad \frac{\Xi \mid \Gamma \vdash e:\tau_1\times\tau_2}{\Xi \mid \Gamma \vdash \mathsf{snd}\ e:\tau_2}\ \textsc{T-Snd}$$

$$\frac{\Xi \mid \Gamma \vdash e:\tau_1}{\Xi \mid \Gamma \vdash \mathsf{inj}_1\ e:\tau_1+\tau_2}\ \textsc{T-Inj1} \qquad \frac{\Xi \mid \Gamma \vdash e:\tau_2}{\Xi \mid \Gamma \vdash \mathsf{inj}_2\ e:\tau_1+\tau_2}\ \textsc{T-Inj2}$$

$$\frac{\Xi \mid \Gamma \vdash e_1:\tau_1+\tau_2 \qquad \Xi \mid \Gamma, x:\tau_1 \vdash e_2:\tau \qquad \Xi \mid \Gamma, x:\tau_2 \vdash e_3:\tau}{\Xi \mid \Gamma \vdash \mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end}:\tau}\ \textsc{T-Match}$$

$$\frac{\Xi \mid \Gamma, f:\tau_1\to\tau_2, x:\tau_1 \vdash e:\tau_2}{\Xi \mid \Gamma \vdash \mathsf{rec}\ f(x) := e:\tau_1\to\tau_2}\ \textsc{T-Rec} \qquad \frac{\Xi \mid \Gamma \vdash e_1:\tau_1\to\tau_2 \qquad \Xi \mid \Gamma \vdash e_2:\tau_1}{\Xi \mid \Gamma \vdash e_1\ e_2:\tau_2}\ \textsc{T-App}$$

$$\frac{\Xi, X \mid \Gamma \vdash e:\tau}{\Xi \mid \Gamma \vdash \Lambda\, e:\forall X.\ \tau}\ \textsc{T-TLam} \qquad \frac{\Xi \mid \Gamma \vdash e:\forall X.\ \tau}{\Xi \mid \Gamma \vdash e\ \_:\tau[\tau'/X]}\ \textsc{T-TApp}$$

Dynamics

$$\frac{e \to_h e'}{K[e] \to K[e']}\ \textsc{Head-Step-Step}$$

$$\textsc{E-Add}\quad \overline{n_1} + \overline{n_2} \to_h \overline{n_1 + n_2}$$

$$\textsc{E-Sub}\quad \overline{n_1} - \overline{n_2} \to_h \overline{n_1 - n_2}$$

$$\frac{n_1 = n_2}{\overline{n_1} = \overline{n_2} \to_h \mathsf{true}}\ \textsc{E-Eq}$$

$$\frac{n_1 \neq n_2}{\overline{n_1} = \overline{n_2} \to_h \mathsf{false}}\ \textsc{E-Not-Eq}$$

$$\frac{n_1 \leq n_2}{\overline{n_1} \leq \overline{n_2} \to_h \mathsf{true}}\ \textsc{E-Le}$$

$$\frac{n_1 \not\leq n_2}{\overline{n_1} \leq \overline{n_2} \to_h \mathsf{false}}\ \textsc{E-Not-Le}$$

$$\frac{n_1 < n_2}{\overline{n_1} < \overline{n_2} \to_h \mathsf{true}}\ \textsc{E-Lt}$$

$$\frac{n_1 \not< n_2}{\overline{n_1} < \overline{n_2} \to_h \mathsf{false}}\ \textsc{E-Not-Lt}$$

$$\textsc{E-If-True}\quad \mathsf{if\ true\ then}\ e_2\ \mathsf{else}\ e_3 \to_h e_2$$

$$\textsc{E-If-False}\quad \mathsf{if\ false\ then}\ e_2\ \mathsf{else}\ e_3 \to_h e_3$$

$$\textsc{E-Fst}\quad \mathsf{fst}\ (v_1, v_2) \to_h v_1$$

$$\textsc{E-Snd}\quad \mathsf{snd}\ (v_1, v_2) \to_h v_2$$

$$\textsc{E-Match-Inj1}\quad \mathsf{match}\ (\mathsf{inj}_1\ v)\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end} \to_h e_2[v/x]$$

$$\textsc{E-Match-Inj2}\quad \mathsf{match}\ (\mathsf{inj}_2\ v)\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end} \to_h e_3[v/x]$$

$$\textsc{E-Rec-App}\quad (\mathsf{rec}\ f(x) := e) v \to_h e[\mathsf{rec}\ f(x) := e/f][v/x]$$

$$\textsc{E-TApp-TLam}\quad (\Lambda\, e)\ \_ \to_h e$$

Lemma 1 (Evaluation under Context)**.** $K[e] \to^* e' \implies \exists e''.(e \to^* e'') \wedge ((Val(e'') \wedge K[e''] \to^* e') \vee (\neg Val(e'') \wedge K[e''] = e'))$

Proof. So assuming $K[e] \to^* e'$, we must show

$$\exists e''.(e \to^* e'') \wedge \big((Val(e'') \wedge K[e''] \to^* e') \vee (\neg Val(e'') \wedge K[e''] = e')\big) \tag{2.1}$$

We proceed by induction on the number of steps in the evaluation $K[e] \to^* e'$. Let n denote the number of steps taken, so that $K[e] \to^n e'$.

- Base Case $n = 0$. In this case, we have that $K[e] \to^0 e'$, which means that $K[e] = e'$. Now use e for e' in 2.1. We must show

  $$(e \to^* e) \wedge \big((Val(e) \wedge K[e] \to^* e') \vee (\neg Val(e) \wedge K[e] = e')\big)$$

  Trivially, $e \to^* e$. For the second part, we proceed by case distinction on $Val(e)$.

  - $Val(e)$. We have that $K[e] \to^0 e'$, so $K[e] \to^* e'$. Thus, we have $Val(e) \wedge K[e] \to^* e'$, which matches the left part of the "$\vee$".

- $\neg Val(e)$. We know that $K[e] = e'$, so we have $\neg Val(e) \wedge K[e] = e'$, which matches the right part of the "$\vee$".

- Inductive Step $n = m+1$. Now we have $K[e] \rightarrow^{m+1} e'$. By the Induction Hypothesis, we have:

$$\forall F, f, f'. F[f] \rightarrow^{m} f' \implies \exists f''.(f \rightarrow^{*} f'') \wedge \\ \big((Val(f'') \wedge F[f''] \rightarrow^{*} f') \vee (\neg Val(f'') \wedge K[f''] = f')\big) \quad (2.2)$$

Split the evaluation, $K[e] \rightarrow^{m+1} e'$, up, so that $K[e] \rightarrow g \wedge g \rightarrow^{m} e'$. Looking at our dynamics, we must have that $K[e] = H[h]$, and $g = H[h']$, for some evaluation context H, and expressions h, h', and $h \rightarrow_{h} h'$. There are now three possible cases:

  - $K = H$ and $e = h$.
    Then $g = H[h'] = K[h']$, and $e \rightarrow_{h} h'$. Furthermore, $K[h'] \rightarrow^{m} e'$. Instantiate I.H. with this to get

$$\exists f''.(h' \rightarrow^{*} f'') \wedge \\ \big((Val(f'') \wedge K[f''] \rightarrow^{*} e') \vee (\neg Val(f'') \wedge K[f''] = e')\big) \quad (2.3)$$

    Call this quantified expression for f'', and use it for e'' in 2.1. We must then show

$$(e \rightarrow^{*} f'') \wedge \big((Val(f'') \wedge K[f''] \rightarrow^{*} e') \vee (\neg Val(f'') \wedge K[f''] = e')\big)$$

    We know that $e \rightarrow h'$, as $e \rightarrow_{h} h'$, and by 2.3, we know that $h' \rightarrow^{*} f''$, so that $e \rightarrow^{*} f''$. The second part follows directly from 2.3.

  - $K[E[]] = H$ and $e = E[h]$.
    Then $g = H[h'] = K[E[]][h'] = K[E[h']]$, thus $K[E[h']] \rightarrow^{m} e'$. Instantiate I.H. with this to get

$$\exists f''.(E[h'] \rightarrow^{*} f'') \wedge \\ \big((Val(f'') \wedge K[f''] \rightarrow^{*} e') \vee (\neg Val(f'') \wedge K[f''] = e')\big) \quad (2.4)$$

    Call this quantified expression for f'', and use it for e'' in 2.1. We must then show

$$(e \rightarrow^{*} f'') \wedge \big((Val(f'') \wedge K[f''] \rightarrow^{*} e') \vee (\neg Val(f'') \wedge K[f''] = e')\big)$$

    We know that $E[h] \rightarrow E[h']$, as $h \rightarrow_{h} h'$, and since $e = E[h]$, then $e \rightarrow E[h']$. By 2.4, we know that $E[h'] \rightarrow^{*} f''$, so that $e \rightarrow^{*} f''$. The second part follows directly from 2.4.

- $K = H[E[]]$ and $E[e] = h$. Here we have $h = E[e]$, so $E[e] \rightarrow_h h'$. Note that E is not the empty evaluation context, as otherwise, we would be in case 1. So by **►create lemma and ref here◄**, we know that $Val(e)$. Now, pick e for e'' in 2.1. We must show

  $$(e \rightarrow^* e) \wedge \big((Val(e) \wedge K[e] \rightarrow^* e') \vee (\neg Val(e) \wedge K[e] = e')\big)$$

  Trivially, $e \rightarrow^* e$. We also have that $Val(e)$, and since $K[e] \rightarrow^{m+1} e'$, then $K[e] \rightarrow^* e'$.

$\square$

Chapter 3

Contextual Equivalence

►draft◄

►fix unfinished text passage - start◄

We define CE so that two programs are CE iff under all Contexts, one program terminates iff the other does. One may ponder whether this definition is really strong enough. If both terminate, we really want the result of the two programs to have values that are behaviourally the same, otherwise, we can certainly put them in a context, where they have different behaviour. For integer values, for example, this means that the two values must be the same number, and for product types, this means that the first value in each pair should be behaviourally the same, and likewise with the second value in each pair. To give an intuition as to why this definition is strong enough, consider two CE programs e1 and e2, where $\cdot \mid \cdot \vdash e_1 : \tau$ and $\cdot \mid \cdot \vdash e_2 : \tau$. Note first that if both e1 and e2 don't terminate, then they will be behaviourally the same. If we plug them into any context, and the context makes us evaluate the expressions, both will not terminate. And if the context doesn't make us evaluate the expressions, then they have no influence in the run of the programs, so they will behave the same.

Now consider what happens if e1 terminates with some value v1. Can we then guarantee that e2 also terminates with some value v2, and v1 and v2 behave the same? Since e1 and e2 are CE, then we can put them into any context, C, and one will terminate iff the other one does. Let's assume that tau = Z. Then consider when C has the form $\mathsf{if}\ [] = v_1\ \mathsf{then}\ ()\ \mathsf{else}\ \omega$. Here $C[e_1] \downarrow ()$. But what about $C[e_2]$? If v2 != v1, then our evaluation rules tell us that we will take the else branch, and hence not terminate. However, since e1 and e2 are CE, and C[e1] terminates then we know that C[e2] must also terminate. Hence it is not the case that v2 != v1, and thus v2 = v1. So if our two programs of type integer are CE, and they both don't run forever, then they must both evaluate to the same value.

Now if tau was bool instead, the context $\mathsf{if}\ []\ \mathsf{then}\ ()\ \mathsf{else}\ \omega$ would suffice in showing that v2 = v1.

►fix unfinished text passage - end◄

Chapter 4

Logical Relations for Contextual Equivalence

►draft◄

Chapter 5

Examples of Application of Contextual Equivalence

►draft◄

Chapter 6

Comparison to Other Work and Ideas for Future Work

▸draft◂

Chapter 7

Conclusion

►conclude on the problem statement from the introduction◄

Acknowledgments

►...◄

Bibliography

[1] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In Radhia Cousot, editor, *Static Analysis, 10th International Symposium, SAS 2003, San Diego, CA, USA, June 11-13, 2003, Proceedings*, volume 2694 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2003.

Appendix A

The Technical Details

►...◄