Redex $\rightarrow$ Coq: towards a theory of decidability of Redex's reduction semantics

Mallku Soldevila[1], Rodrigo Ribeiro[2], Beta Ziliani[3]

April 18, 2024 - Dependable Systems and Software group @ Saarland University - Saarbrücken, Germany

[1]FAMAF, UNC (LIIS Group) & CONICET (Argentina)
[2]DECOM, UFOP (Brazil)
[3]FAMAF, UNC (LIIS Group) & Manas.Tech (Argentina)

Previously...

$$\frac{\delta(\text{rawget}, \mathit{objr}, v_1, \theta_1) \neq \textbf{nil} \qquad \theta_2 = \delta(\text{rawset}, \mathit{objr}, v_1, v_2, \theta_1)}{\theta_1 \;:\; \mathit{objr}\,[v_1] = v_2 \;\rightarrow^{s\_\theta}\; \theta_2 \;:\; ;}$$

$$\frac{\delta(\text{rawget}, \mathit{objr}, v_1, \theta) = \textbf{nil}}{\theta \;:\; \mathit{objr}\,[v_1] = v_2 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\, \mathit{objr}\,[v_1] = v_2 \,|\!)_{\textsc{NewIndex}}}$$

$$\frac{\delta(\text{type}, v_1) \neq \text{"table"}}{\theta \;:\; v_1\,[v_2] = v_3 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\, v_1\,[v_2] = v_3 \,|\!)_{\textsc{NewIndex}}}$$

Figure 17. Field update.

Previously...

$$\frac{\begin{array}{c}\delta(\text{rawget}, objr, \mathsf{v}_1, \theta_1) \neq \textbf{nil} \\ \theta_2 = \delta(\text{rawset}, objr, \mathsf{v}_1, \mathsf{v}_2, \theta_1)\end{array}}{\theta_1 \;:\; objr\;[\mathsf{v}_1] = \mathsf{v}_2 \;\rightarrow^{s\_\theta}\; \theta_2 \;:\; ;}$$

$$\frac{\delta(\text{rawget}, objr, \mathsf{v}_1, \theta) = \textbf{nil}}{\theta \;:\; objr\;[\mathsf{v}_1] = \mathsf{v}_2 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\; objr\;[\mathsf{v}_1] = \mathsf{v}_2 \;|\!)_{\text{NewIndex}}}$$

$$\frac{\delta(\text{type}, \mathsf{v}_1) \neq \text{"table"}}{\theta \;:\; \mathsf{v}_1\;[\mathsf{v}_2] = \mathsf{v}_3 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\; \mathsf{v}_1\;[\mathsf{v}_2] = \mathsf{v}_3 \;|\!)_{\text{NewIndex}}}$$

Figure 17. Field update.

$$\frac{(\sigma', \theta') = \text{gc}(s, \sigma, \theta) \qquad \sigma' \neq \sigma \vee \theta' \neq \theta}{\sigma : \theta : s \overset{\text{GC}}{\mapsto} \sigma' : \theta' : s}$$

Previously...

$$\frac{\delta(\mathrm{rawget}, objr, v_1, \theta_1) \neq \mathbf{nil} \qquad \theta_2 = \delta(\mathrm{rawset}, objr, v_1, v_2, \theta_1)}{\theta_1 \;:\; objr\,[v_1] = v_2 \;\rightarrow^{s\_\theta}\; \theta_2 \;:\; ;}$$

$$\frac{\delta(\mathrm{rawget}, objr, v_1, \theta) = \mathbf{nil}}{\theta \;:\; objr\,[v_1] = v_2 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\, objr\,[v_1] = v_2 \,|\!)_{\text{NewIndex}}}$$

$$\frac{\delta(\mathrm{type}, v_1) \neq \text{"table"}}{\theta \;:\; v_1\,[v_2] = v_3 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\, v_1\,[v_2] = v_3 \,|\!)_{\text{NewIndex}}}$$

Figure 17. Field update.

$$\frac{(\sigma', \theta') = gc(s, \sigma, \theta) \qquad \sigma' \neq \sigma \lor \theta' \neq \theta}{\sigma : \theta : s \overset{\text{GC}}{\mapsto} \sigma' : \theta' : s}$$

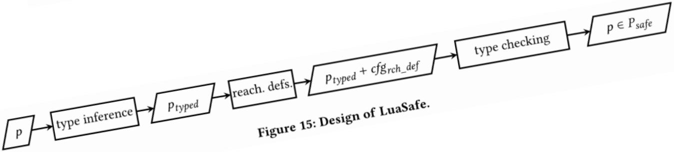


Figure 15: Design of LuaSafe.

Previously...

$$\frac{\delta(\text{rawget}, objr, v_1, \theta_1) \neq \textbf{nil} \qquad \theta_2 = \delta(\text{rawset}, objr, v_1, v_2, \theta_1)}{\theta_1 \;:\; objr\,[v_1] = v_2 \;\rightarrow^{s\_\theta}\; \theta_2 \;:\; ;}$$

$$\frac{\delta(\text{rawget}, objr, v_1, \theta) = \textbf{nil}}{\theta \;:\; objr\,[v_1] = v_2 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\, objr\,[v_1] = v_2 \,|\!)_{\text{NewIndex}}}$$

$$\frac{\delta(\text{type}, v_1) \neq \text{"table"}}{\theta \;:\; v_1\,[v_2] = v_3 \;\rightarrow^{s\_\theta}\; \theta \;:\; (\!|\, v_1\,[v_2] = v_3 \,|\!)_{\text{NewIndex}}}$$

Figure 17. Field update.

$$\frac{(\sigma', \theta') = \text{gc}(s, \sigma, \theta) \qquad \sigma' \neq \sigma \lor \theta' \neq \theta}{\sigma : \theta : s \overset{\text{GC}}{\mapsto} \sigma' : \theta' : s}$$

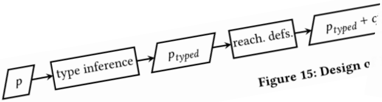
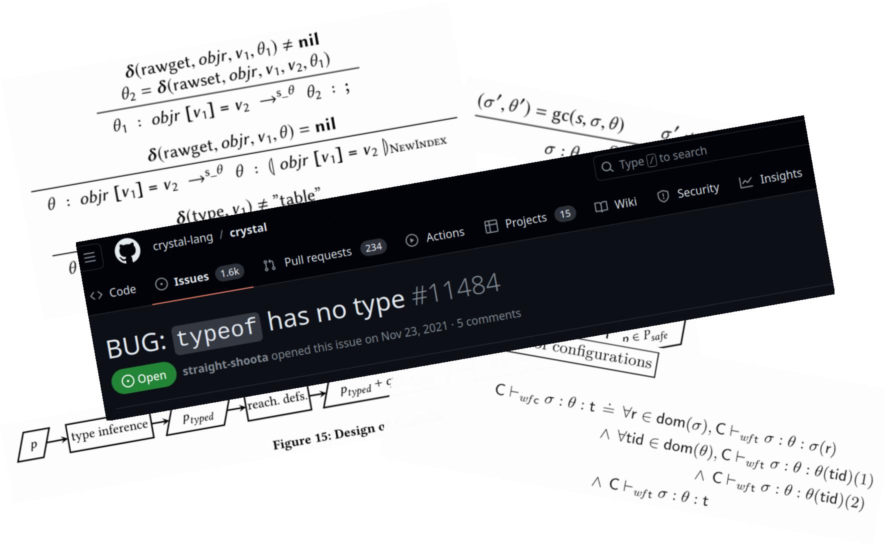


Figure 15: Design o

$\boxed{\text{Well-formedness of configurations}}$ $\boxed{p \in P_{safe}}$

$$C \vdash_{wfc} \sigma : \theta : t \doteq \forall r \in \text{dom}(\sigma), C \vdash_{wft} \sigma : \theta : \sigma(r)$$
$$\land\ \forall tid \in \text{dom}(\theta), C \vdash_{wft} \sigma : \theta : \theta(tid)(1)$$
$$\land\ C \vdash_{wft} \sigma : \theta : \theta(tid)(2)$$
$$\land\ C \vdash_{wft} \sigma : \theta : t$$

Previously...



crystal-lang / crystal
Code
Issues 1.6k
Pull requests 234
Actions
Projects 15
Wiki
Security
Insights
Type / to search
BUG: typeof has no type #11484
Open
straight-shoota opened this issue on Nov 23, 2021 · 5 comments
type inference
reach. defs.
Figure 15: Design o
configurations

We need an easy to use tool for rapid prototyping and testing...

Redex

- DSL built on top of Racket.
- Useful to easily mechanize reduction semantics with evaluation contexts, and formal systems, and test them:
  - Random-testing of properties
  - Stepper
  - Facilities to implement test suites

Redex

Example: λ-calculus with call-by-value normal-order reduction.

```
(define-language lambda
  [e ::= x (e e) v]

  [v ::= (λ x e)]

  [x ::= variable-not-otherwise-mentioned]

  [E ::= hole (E e) (v E)])
```

Figure: Grammar of λ-terms and evaluation contexts.

Example: λ-calculus with call-by-value normal-order reduction.

```
(define-metafunction lambda
  fv : e -> (x ...)

  [(fv x) (x)]

  [(fv (e_1 e_2)) (x_1 ... x_2 ...)

   (where (x_1 ...) (fv e_1))
   (where (x_2 ...) (fv e_2))]

  [(fv (λ x_1 e)) (x_2 ... x_3 ...)

   (where (x_2 ... x_1 x_3 ...) (fv e))]

  ;{x not in (fv e)}
  [(fv (λ x e)) (fv e)])
```

Figure: Free occurrences of variables in λ-terms.

Redex

Example: λ-calculus with call-by-value normal-order reduction.

```
(define reduction
  (reduction-relation
   lambda
   #:domain e

   [--> (in-hole E ((λ x e) v))
        (in-hole E (substitute e x v))
        beta_contraction]))
```

Figure: β-contraction.

Redex

Example: λ-calculus with call-by-value normal-order reduction.

```
(define-judgment-form lambda
  #:mode (normal-form I)
  #:contract (normal-form e)

  [---------------
   (normal-form x)]

  [(normal-form e)
   -------------------
   (normal-form (x e))]

  [(normal-form e_1) (normal-form e_2) (normal-form e_3)
   -----------------------------------------------------
   (normal-form ((e_1 e_2) e_3))]

  [(normal-form e)
   --------------------
   (normal-form (λ x e))]
  )
```

Figure: Formal system capturing the notion of "normal form".

Example: λ-calculus with call-by-value normal-order reduction.

```
> (judgment-holds (normal-form y))
#t
> (judgment-holds (normal-form (y z)))
#t
> (judgment-holds (normal-form ((λ x x) z)))
#f
```

Figure: Using a decision procedure extracted from the previous formal system.

Redex

Example: λ-calculus with call-by-value normal-order reduction.

```
> (generate-term lambda #:satisfying (normal-form e) 1)
'(normal-form (s ((M Z) T)))
> (generate-term lambda #:satisfying (normal-form e) 1)
'(normal-form r)
> (generate-term lambda #:satisfying (normal-form e) 1)
'(normal-form q)
> (generate-term lambda #:satisfying (normal-form e) 1)
'(normal-form ((((q P) f) ((K h) uE)) ((z A) PR)))
> (generate-term lambda #:satisfying (normal-form e) 1)
'(normal-form ((((F x) W) ((mH S) q)) ((E v) l)))
```

Figure: Using a generator extracted from the previous formal system.

Redex

What cannot be done within Redex:

- Formal verification.
- Obtain static guarantees of correctness of our definitions (beyond checks of syntax).

Problem:

- There are no facilities to export a Redex model into a proof assistant.
- There only exists formal semantics for a subset of Redex's features.[4]

[4] Casey Klein, Jay McCarthy, Steven Jaconette, and Robert Bruce F. A semantics for context- sensitive reduction semantics. In APLAS'11, 2011.

Redex

λ_{JS} experience:

https://blog.brownplt.org/2012/06/04/lambdajs-coq.html

Redex can also generate random tests to exercise your semantics. Random testing caught several more bugs in λ_{JS}.

Coq: A Machine-Checked Proof

Testing is not enough. We shipped λ_{JS} with a bug that breaks the soundness theorem above. We didn't discover it for a year. David van Horn and Ian Zerny both reported it to us independently. We'd missed a case in the semantics, which caused certain terms to get "stuck". It turned out to be a simple fix, but we were left wondering if anything else was left lurking.

To gain further assurance, we mechanized λ_{JS} with the Coq proof assistant. The soundness theorem now has a machine-checked proof of correctness. You still need to read the Coq definition of λ_{JS} and ensure it matches your intuitions. But once that's done, you can be confident that the proofs are valid.

Doing this proof was surprisingly easy, once we'd read Software Foundations and Certified Programming with Dependent Types. We'd like to thank Benjamin

Proposal

A Semantics for Context-Sensitive Reduction Semantics

Casey Klein[1], Jay McCarthy[2], Steven Jaconette[1], and Robert Bruce Findler[1]

[1] Northwestern University
[2] Brigham Young University

Abstract. This paper explores the semantics of the meta-notation used in the style of operational semantics introduced by Felleisen and Hieb. Specifically, it defines a formal system that gives precise meanings to the notions of contexts, decomposition, and plugging (recomposition) left implicit in most expositions. This semantics is not naturally algorithmic, so the paper also provides an algorithm and proves a correspondence with the declarative definition.
The motivation for this investigation is PLT Redex, a domain-specific programming language designed to support Felleisen-Hieb-style semantics. This style of semantics is the de-facto standard in operational semantics and, as such, is widely used. Accordingly, our goal is that Redex programs should, as much as possible, look and behave like those semantics. Since Redex's first public release more than seven years ago, its precise interpretation of contexts has changed several times, as we repeatedly encountered reduction systems that did not behave according to their authors' intent. This paper describes the culimation of that experience. To the best of our knowledge, the semantics given here accommodates even the most complex uses of contexts available.

$+$

New Redex features

Decision procedures

$\longrightarrow$

Redex $\longrightarrow$ Coq

- Proof assistant based on dependent-type theory and Curry-Howard correspondence.

- Many years of development, huge community, several industrial-strength applications.

- Our team has experience mechanizing in, and implementing features for, Coq (better unification algorithms, Mtac, Mtac2, mechanization of dynamic modal logics).

Coq fundamentals

Correspondence between simply-typed λ-calculus and propositional intuitionistic logic (fragment).

$$\frac{\Gamma, x : A \vdash_{\mathcal{T}} t : B}{\Gamma \vdash_{\mathcal{T}} \lambda x : A.\, t : A \to B} \qquad \frac{\Gamma, A \vdash_{\mathcal{L}} B}{\Gamma \vdash_{\mathcal{L}} A \Rightarrow B}$$

$$\frac{\Gamma \vdash_{\mathcal{T}} t : A \quad \Gamma \vdash_{\mathcal{T}} u : B}{\Gamma \vdash_{\mathcal{T}} (t, u) : A \times B} \qquad \frac{\Gamma \vdash_{\mathcal{L}} A \quad \Gamma \vdash_{\mathcal{L}} B}{\Gamma \vdash_{\mathcal{L}} A \wedge B}$$

- With dependent types we can extend the previous correspondence to quantifiers. We obtain:

$$\Gamma \vdash_{\mathcal{L}} A \leftrightarrow \hat{\Gamma} \vdash_{\mathcal{T}} t : T(A)$$

where $\hat{\Gamma}$ has hypotheses $x : T(B)$, for B in Γ.

Coq fundamentals

- To preserve the correspondence we need to:
  - Avoid any form of non-termination (general recursion, some forms of inductive definitions).
  - Enforce total functions.
- The correspondence implies only intuitionistic logic :')
  - Only constructive proofs (in their most extreme form).
  - No excluded middle.
  - No proofs by contraposition (using contrapositive).

Redex $\rightarrow$ Coq

A Semantics for Context-Sensitive Reduction Semantics, Klein et. al:

- Subset of Redex language for patterns (P) and terms (T).
- Specification of matching and decomposition:

  Matching: $\mathsf{g} \vdash \mathsf{t} : \mathsf{p} \mid \mathsf{b}$

  Decomposition: $\mathsf{g} \vdash \mathsf{t}_1 = \mathsf{p}_1 [\![\mathsf{t}_2]\!] : \mathsf{p}_2 \mid \mathsf{b}$

- Matching and decomposition as an algorithm:

$$M : G \times T \times P \rightarrow \mathcal{P}(B)$$

A Semantics for Context-Sensitive Reduction Semantics, Klein et. al:

- Proofs of soundness and completeness of M with respect to its specification:

$$\mathrm{b} \in \mathrm{M}(\mathrm{g}, \mathrm{t}, \mathrm{p}) \iff \mathrm{g} \vdash \mathrm{t} : \mathrm{p} \mid \mathrm{b}$$

Redex $\rightarrow$ Coq

Challenges:

- Matching algorithm is not in a primitive recursive fashion.
- Redex patterns absent in the model.
- Reproduce soundness and completeness proofs.
- No previous work on decision procedures or specific tactics to prove properties over languages expressed through Redex patterns.
    - Every proof is an algorithm!

Redex $\rightarrow$ Coq

Matching algorithm is not in a primitive recursive fashion.

- We generalize matching and decomposition:

  Matching: $\mathsf{g} \vdash \mathsf{t} : \mathsf{p}_{\mathsf{g'}} \mid \mathsf{b}$

  Decomposition: $\mathsf{g} \vdash \mathsf{t}_1 = \mathsf{p}_1 [\![\mathsf{t}_2]\!] : \mathsf{p}_{2\mathsf{g'}} \mid \mathsf{b}$

  Algorithm: $M : G \times T \times P \times G \rightarrow \mathcal{P}(B)$

- It can be shown that, if the grammars are *non-left recursive*, during matching we can *discard* already used productions.

- Nice: Redex only supports non-left recursive grammars.

Redex $\rightarrow$ Coq

Matching algorithm is not in a primitive recursive fashion.

- *Well-founded* relation over the tuples from $\text{dom}(M)$:
  - Input consumption:

$$<_T \subseteq T$$

  - Pattern and/or production consumption:

$$<_{P \times G} \subseteq P \times G$$

  - Lexicographic order on pairs:

$$(\text{t}, (\text{p}, \text{g})) <^{\text{G'}}_{T \times P \times G} (\text{t'}, (\text{p'}, \text{g'}))$$

$$\Longleftrightarrow$$

$$\text{t} <_T \text{t'} \vee (\text{t'} = \text{t} \wedge (\text{p}, \text{g}) <_{P \times G} (\text{p'}, \text{g'}))$$

Matching algorithm is not in a primitive recursive fashion.

- *Well-founded* relation over the tuples from $\mathrm{dom}(M)$:
  - If $<_T$ and $<_{P\times G}$ are well-founded, then $<^{\mathrm{G'}}_{T\times P\times G}$ is well-founded.

Redex $\to$ Coq

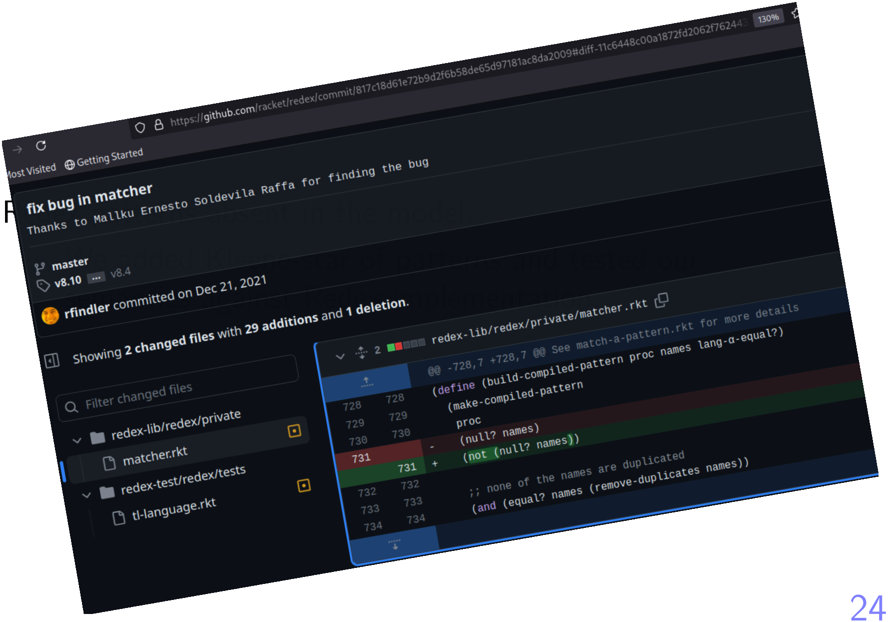
Matching algorithm is not in a primitive recursive fashion.

- We capture M as the least fixed point of a generator function, that performs primitive recursion over proofs of accessibility of tuples from $T \times P \times G$, with respect to $<^{\mathbf{G'}}_{T \times P \times G}$.

Redex $\rightarrow$ Coq

Redex patterns absent in the model.

- We added Kleene-star of patterns and tested our semantics against Redex implementation.


https://github.com/racket/redex/commit/817c18d61e72b9d2f6b58de65d97181ac8da2009#diff-11c6448c00a1872fd2062f76244
130%
Most Visited
Getting Started
fix bug in matcher
Thanks to Mallku Ernesto Soldevila Raffa for finding the bug
master
v8.10
v8.4
rfindler committed on Dec 21, 2021
Showing 2 changed files with 29 additions and 1 deletion.
Filter changed files
redex-lib/redex/private
matcher.rkt
redex-test/redex/tests
tl-language.rkt
redex-lib/redex/private/matcher.rkt
@@ -728,7 +728,7 @@ See match-a-pattern.rkt for more details
(define (build-compiled-pattern proc names lang-α-equal?)
(make-compiled-pattern
proc
- (null? names)
+ (not (null? names))
;; none of the names are duplicated
(and (equal? names (remove-duplicates names))

Reproduce soundness and completeness proofs.

- $\mathrm{b} \in \mathrm{M}(\mathrm{g}, \mathrm{t}, \mathrm{p}, \mathrm{g'}) \iff \mathrm{g} \vdash \mathrm{t} : \mathrm{p}_{g'} \mid \mathrm{b}$

- Soundness of our manipulation of grammars:
  $\mathrm{g} \vdash \mathrm{t} : \mathrm{p}_{g'} \mid \mathrm{b} \iff \mathrm{g} \vdash \mathrm{t} : \mathrm{p}_{g' \setminus n \rightarrow p} \mid \mathrm{b}$

- Completeness of our formal system:
  $\mathrm{g} \vdash \mathrm{t} : \mathrm{p} \mid \mathrm{b} \Rightarrow \mathrm{g} \vdash \mathrm{t} : \mathrm{p}_{g} \mid \mathrm{b}$

- Soundness of our formal system:
  If $\mathrm{g'} \subseteq \mathrm{g}$, $\mathrm{g} \vdash \mathrm{t} : \mathrm{p}_{g'} \mid \mathrm{b} \Rightarrow \mathrm{g} \vdash \mathrm{t} : \mathrm{p} \mid \mathrm{b}$

Redex $\longrightarrow$ Coq

No previous work on decision procedures or specific tactics to prove properties over languages expressed through Redex patterns.

- In progress: building the foundations for the future development of tactics.
  - Finite subset types of terms and patterns bounded in size (e.g., decidability of properties quantified over terms and/or patterns).
  - Poset of patterns (ordered by language inclusion) $\rightarrow$ lattice over which we can perform equational reasoning about language intersection.

Redex → Coq

In progress: transpiler from Redex to Coq.

- It is able to translate every Redex feature covered in our first iteration.
- Builds proofs for standard decidability properties (decidability of definitional equality of atomic elements of patterns and terms).

Future work:

- Add missing Redex features (more patterns, formal systems, meta-functions).
- Further develop our theory about decidable portions of reduction semantics Redex style.
- Improve efficiency of extracted interpreters: refocusing!

https://github.com/Mallku2/redex2coq

Thanks!, Questions?