# redex2coq: towards a theory of decidability of Redex's reduction semantics

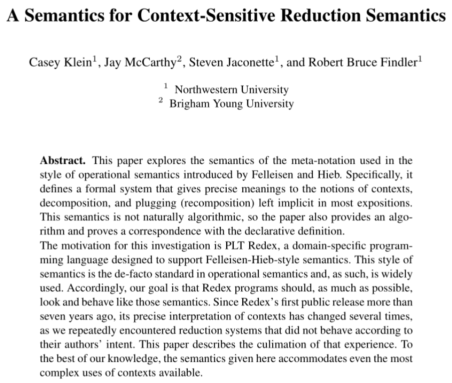
Mallku Soldevila[1], Rodrigo Ribeiro[2], Beta Ziliani[3]

September, 10 - **15th International Conference on Interactive Theorem Proving (ITP 2024)** - Tbilisi, Georgia

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

## Spoiler...

# Spoiler...



**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.

$+$

Absent Redex features

Decision procedures

`Deep embedding`

## Summary

## Summary

- Redex.
    - Problem.
        - Proposal.
- redex**2**coq : Redex $\rightarrow$ Coq
    - RedexK.
    - Challenges for the mechanization in Coq.
    - Ongoing development.
    - Future work.
- Conclusion.

## Redex

## Redex

- DSL built on top of Racket.
- Fast mechanization of:
    - Reduction semantics with evaluation contexts.
    - Formal systems (to capture arbitrary relations over terms).

## Redex

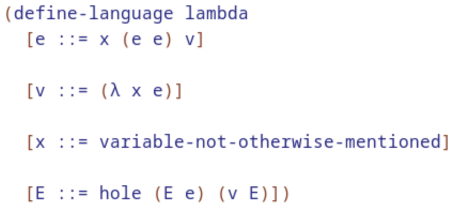
- Tools for testing/prototyping:
  - Random-testing of properties.
  - Stepper
  - Facilities to implement test suites

## Redex

- Already used in several formalization efforts for real programming languages:
  - JavaScript
  - Python
  - Scheme
  - Lua

## Redex

Example: $\lambda$-calculus with call-by-value normal-order reduction.

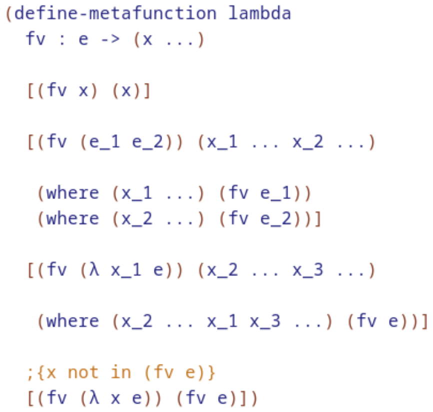


```
(define-language lambda
  [e ::= x (e e) v]

  [v ::= (λ x e)]

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

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

Figure: Grammar of $\lambda$-terms and evaluation contexts.

## Redex

Example: $\lambda$-calculus with call-by-value normal-order reduction.

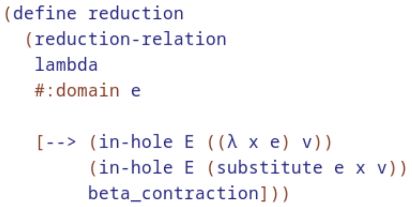


```
(define-metafunction lambda
  fv : e -> (x ...)

  [(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 $\lambda$-terms.

## Redex

Example: $\lambda$-calculus with call-by-value normal-order reduction.

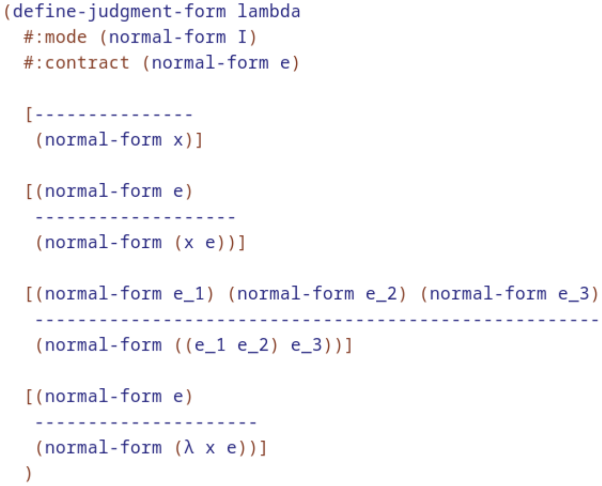


```
(define reduction
  (reduction-relation
   lambda
   #:domain e

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

Figure: $\beta$-contraction and its compatible closure.

## Redex

Example: $\lambda$-calculus with call-by-value normal-order reduction.

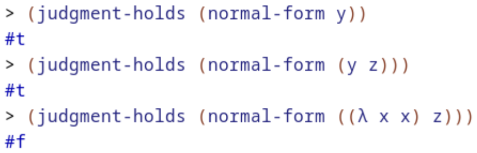


```
(define-judgment-form lambda
  #:mode (normal-form I)
  #:contract (normal-form e)

  [--------------
   (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”.

## Redex

Example: $\lambda$-calculus with call-by-value normal-order reduction.

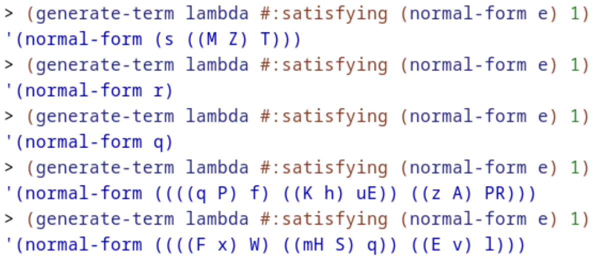


```
> (judgment-holds (normal-form y))
#t
> (judgment-holds (normal-form (y z)))
#t
> (judgment-holds (normal-form ((λ x x) z)))
#f
```

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

## Redex

Example: $\lambda$-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.
- Get static guarantees of correctness (only syntax checks).

## Redex

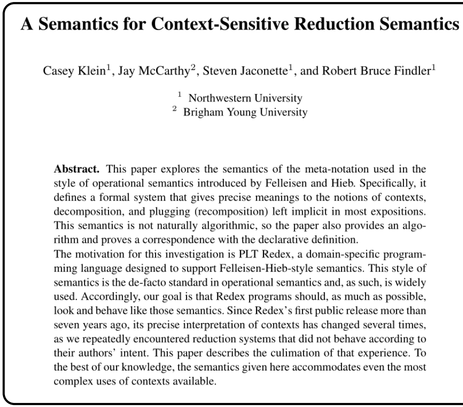
Problem:

- No facilities to export to proof assistants a (reasonable) complex fragment of a model.[4]

[4] Redex Plus being another effort in this direction, with some limitations imposed by its shallow embedding approach to pattern representation.

## Proposal

# 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.

$+$

Absent Redex features

Decision procedures

`Deep embedding`

## redex**2**coq

## RedexK[5]

[5] *A Semantics for Context-Sensitive Reduction Semantics*, Klein et. al

## RedexK

- Subset of Redex's language for patterns and terms.
  - Terms (t) are captured through grammars (g).
  - Productions of the grammars are specified through patterns (p).
    - Patterns can be: non-terminals of the grammar (n), literals, sequences of patterns, *named* patterns.
    - It is possible to specify context-dependent rules.

## RedexK

- Specification of matching of a term $t$ against some pattern $p$...
  - ...where $p$ could contain mentions to non-terminals from a grammar $g$:

$$g \vdash t : p \mid b$$

## RedexK

- Specification of matching of a term $t$ against some pattern $p$...
    - ...where $p$ could contain mentions to non-terminals from a grammar $g$:

      $g \vdash t : p \mid \textcircled{b} \leftarrow$ *bindings produced during matching of named patterns*

## RedexK

- Matching as an algorithm.
- Proofs of soundness and completeness...
    - ... of algorithmic matching with respect to its specification.

## Challenges for the mechanization in Coq.

## Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.
2. Reproduce soundness and completeness proofs.

## Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.
   - During matching occurs, either:
     - *Consumption* of the term.
     - *Consumption* of the pattern.
     - Productions of the grammar are *tested*, which might involve matching against a new pattern:

$$\{\ldots, (n, p), \ldots\} \vdash t : n \mid \ldots$$
$$\hookrightarrow$$
$$\{\ldots, (n, p), \ldots\} \vdash t : p \mid \ldots$$

## Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.
    - Observation: during matching we can *discard* already used productions from the grammar...
        - ...if the grammar is *non-left recursive*.
        - No problem: Redex only supports non-left recursive grammars.

## Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.
   - The previous allows us to generalize matching:
     - Original matching: $g \vdash t : p \mid b$
     - Generalized matching: $g \vdash t : p_{g'} \mid b$

# Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.
    - The previous allows us to generalize matching:

$$\{\ldots, (n, p), \ldots\} \vdash t : n_{\{\ldots,(n,p),\ldots\}} \mid b$$
$$\hookrightarrow$$
$$\{\ldots, (n, p), \ldots\} \vdash t : p_{\{\ldots\}} \mid b$$

# Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.

    - *Well-founded* relation over the tuples from $g \vdash t : p_{g'} \mid \ldots$, built by:
        - Preserving premise/judgement *order*.
        - Well-founded relation over the set of terms T (*term consumption*):

        $$<_T \subset T$$

        - Well-founded relation over the pairs P × G (pattern and/or production consumption):

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

## Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.
   - *Well-founded* relation over the tuples from $g \vdash t : p_{g'} \mid \ldots$, built by:
     - Lexicographic order on tuples from $<_T \times <_{P \times G}$:
       $$<^{\mathsf{G}}_{(<_T \times <_{P \times G})} \subset T \times \ (P \times G)$$
       is well-founded.

## Challenges for the mechanization in Coq.

1. Matching algorithm is not in a primitive recursive fashion.
    - We can capture a matching algorithm as a primitive recursion that *respects* the previous well-founded order.
        - Algorithm + its termination proof.

## Challenges for the mechanization in Coq.

2. Reproduce soundness and completeness proofs.
   - Soundness and completeness of algorithmic *generalized* matching with respect to its specification.
   - Soundness of our manipulation of grammars:
     $\mathsf{g} \vdash \mathsf{t} : \mathsf{p}_{g'} \mid \mathsf{b} \iff \mathsf{g} \vdash \mathsf{t} : \mathsf{p}_{g' \backslash n \to p} \mid \mathsf{b}$
   - Soundness and completeness of our specification of *generalized* matching, with respect to the original specification of matching.

Ongoing development.

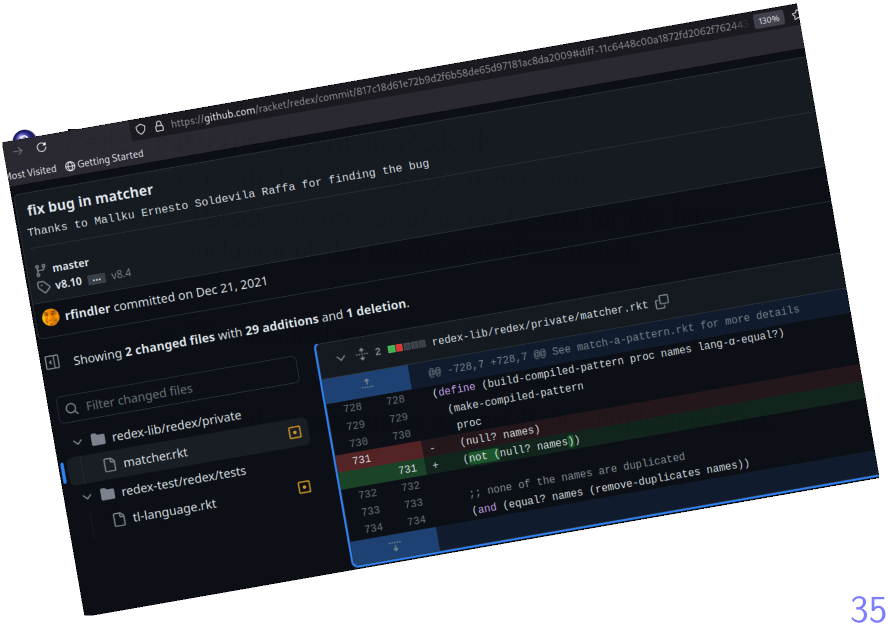
## Ongoing development.

1. Redex patterns absent in RedexK.
2. No previous work on decision procedures or specific tactics to prove properties over languages expressed through Redex patterns.
3. Transpiler.

## Ongoing development.

2. Redex patterns absent in RedexK.
    - We extended the language of patterns:
        - *Possible empty list of terms* (useful for the future inclusion of the Kleene-star of patterns).
        - We modified spec./matching/proofs.
        - Tested our semantics against Redex implementation.

# Ongoing development.


https://github.com/racket/redex/commit/817c18d61e72b9d2f6b58de65d97181ac8da2009#diff-11c6448c00a1872fd2062f762443

**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**.

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))
```

## Ongoing development.

2. No previous work on decision procedures or specific tactics to prove properties over languages expressed through Redex patterns.
    - In progress: Finite subset types of terms and patterns bounded in size (e.g., decidability of properties quantified over terms and/or patterns).

# Ongoing development.

2. No previous work on decision procedures or specific tactics to prove properties over languages expressed through Redex patterns.
   - In progress: Poset of patterns (ordered by language inclusion) $\rightarrow$ lattice over which we can perform equational reasoning about language intersection.

## Ongoing development.

3. Transpiler (in progress).
   - 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

## 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?