Building Lemmas Using Examples

Gabriel Infante-Lopez

Grupo de Procesamiento de Lenguaje Natural
FaMAF - UNC

International Workshop ACL2 - Nov. 2007

Overview

General Idea

1. Generates lemmas for helping proving theorems that state the equality between terms.

```
(defthm th1 (equal (f x) (g x)))
```

2. Selects a ground term a
3. Observes the computations of `(f a)` and `(g a)`
4. Suggests lemmas based on the observation.

Three Fundamental Questions:

Finding Ground Terms: How do we find interesting examples?

Three Fundamental Questions:

Finding Ground Terms: How do we find interesting examples?

Computational Model: Where to observe computations?

Three Fundamental Questions:

Finding Ground Terms: How do we find interesting examples?

Computational Model: Where to observe computations?

Meaningful Observations: What are we looking for on this models?

Three Fundamental Questions:

Finding Ground Terms: How do we find interesting examples? So far, by hand.

Computational Model: Where to observe computations?

Meaningful Observations: What are we looking for on this models?

Three Fundamental Questions:

Finding Ground Terms: How do we find interesting examples? So far, by hand.

Computational Model: Where to observe computations? We introduce the concept of Evaluation Graphs.

Meaningful Observations: What are we looking for on this models?

Three Fundamental Questions:

Finding Ground Terms: How do we find interesting examples? So far, by hand.

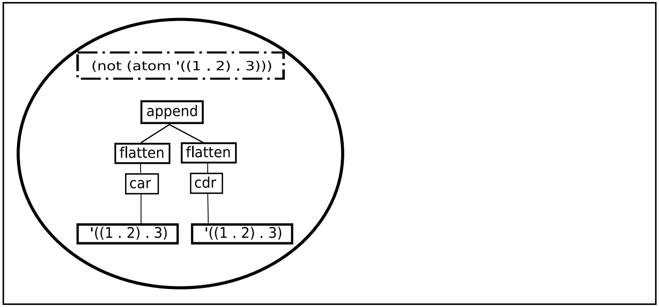
Computational Model: Where to observe computations? We introduce the concept of Evaluation Graphs.

Meaningful Observations: What are we looking for on this models? We look for a specific pattern while comparing Evaluation Graphs.

Basic Intuitions on EvalGraphs

EvalGraphs are directed graphs where

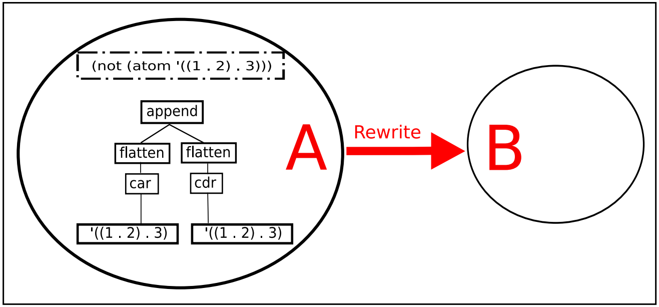
Vertices are pairs of terms. One of them is boolean.



Basic Intuitions on EvalGraphs

EvalGraphs are directed graphs where

Vertices	are pairs of terms. One of them is boolean.
Labeled Arcs	(A, B) is an arc, if vertex B is the result of *expanding* or *rewritting* A

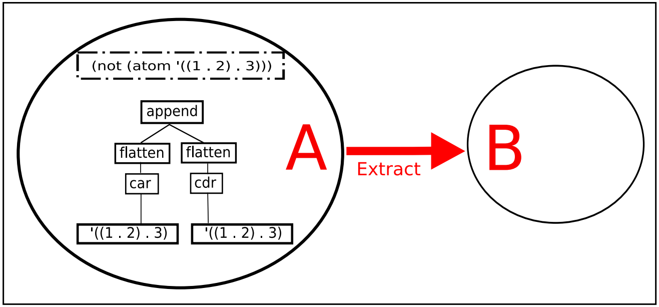


Basic Intuitions on EvalGraphs

EvalGraphs are directed graphs where

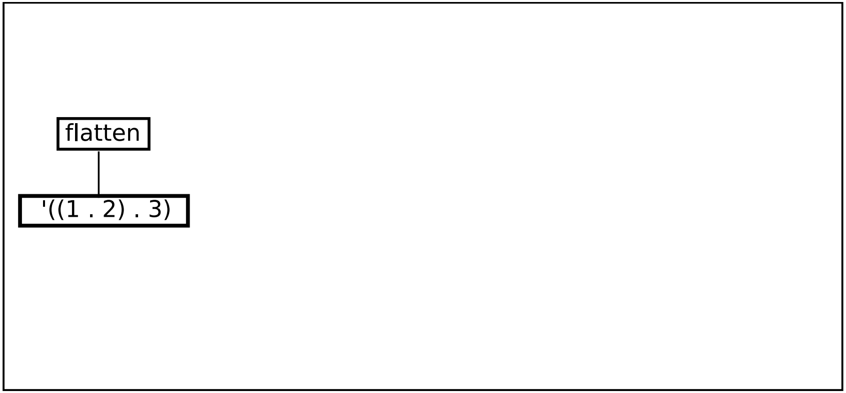
Vertices are pairs of terms. One of them is boolean.

Labeled Arcs (A, B) is an arc, if vertex B is the result of *expanding* or *rewritting* A



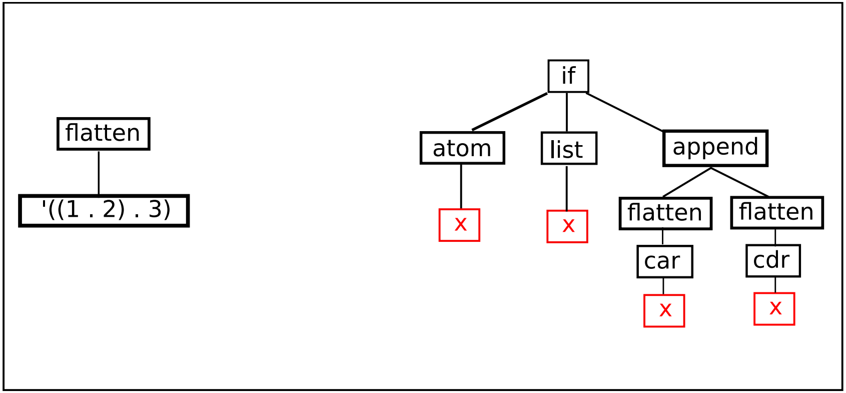
Operations on Terms: Rewrite

The *rewrite* operation introduces a new definiton:



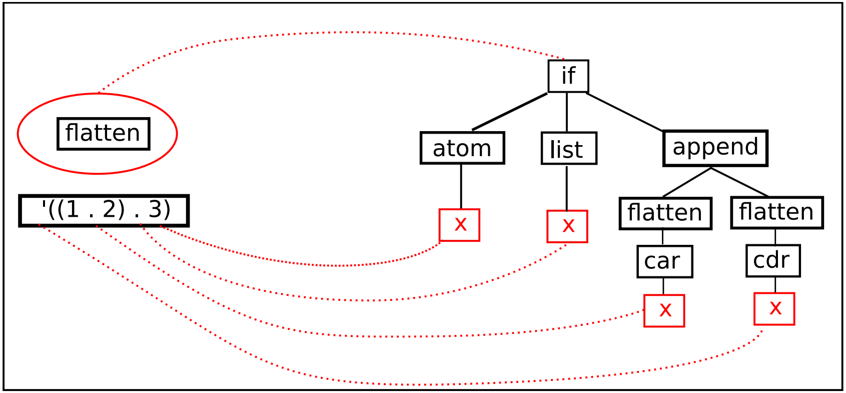
Operations on Terms: Rewrite

The *rewrite* operation introduces a new definiton:



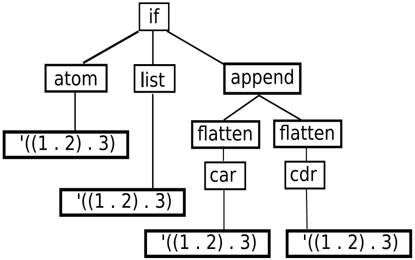
Operations on Terms: Rewrite

The *rewrite* operation introduces a new definiton:



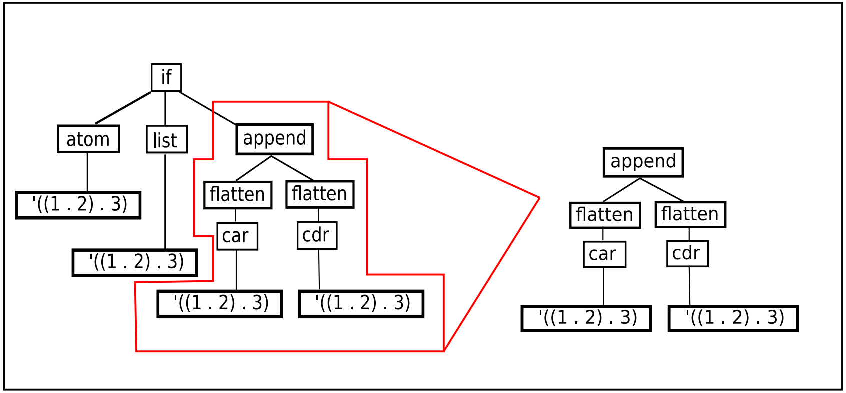
Operations on Terms: Rewrite

The *rewrite* operation introduces a new definiton:



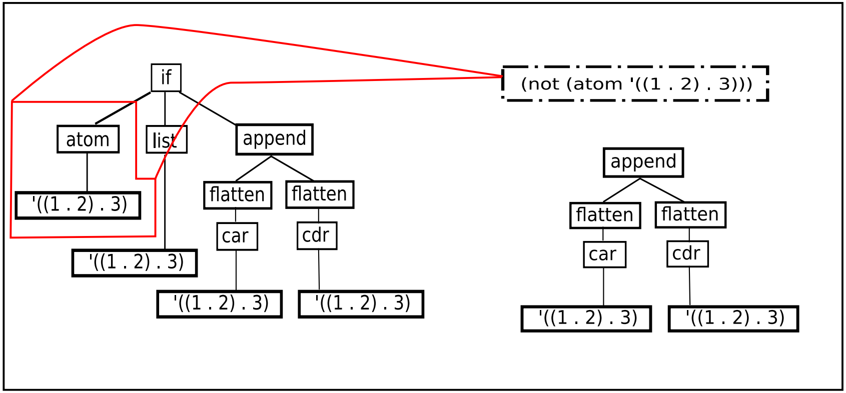
Operations on Terms: Rewrite

The *rewrite* operation introduces a new definiton:



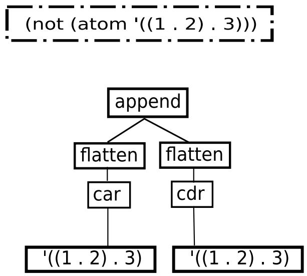
Operations on Terms: Rewrite

The *rewrite* operation introduces a new definiton:



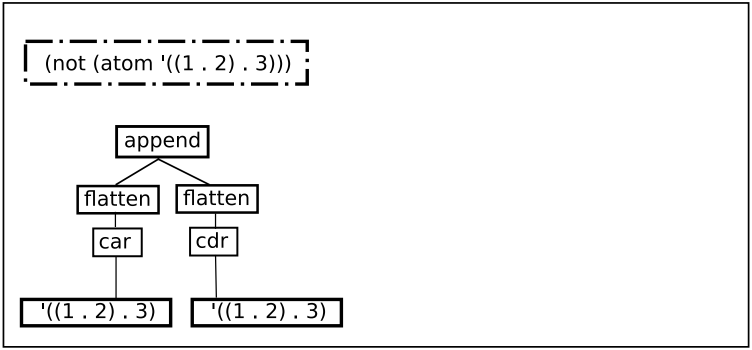
Operations on Terms: Rewrite

The *rewrite* operation introduces a new definiton:



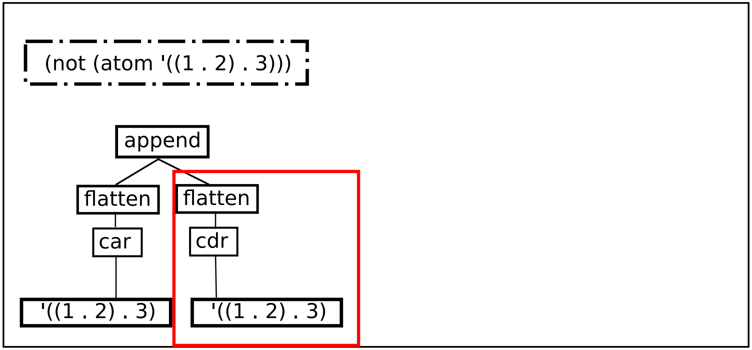
Operations on Terms: Extract

The *extract* operation extracts a subterm:



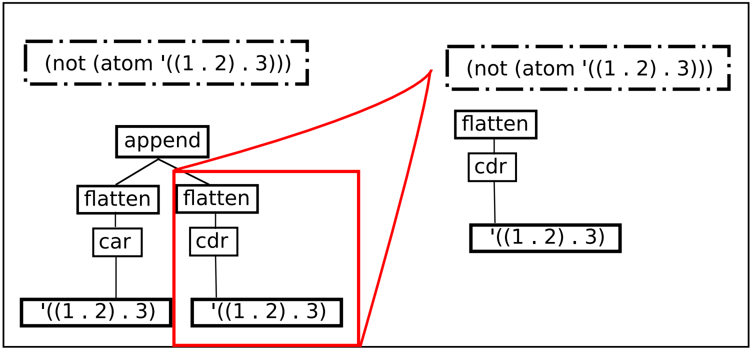
Operations on Terms: Extract

The *extract* operation extracts a subterm:

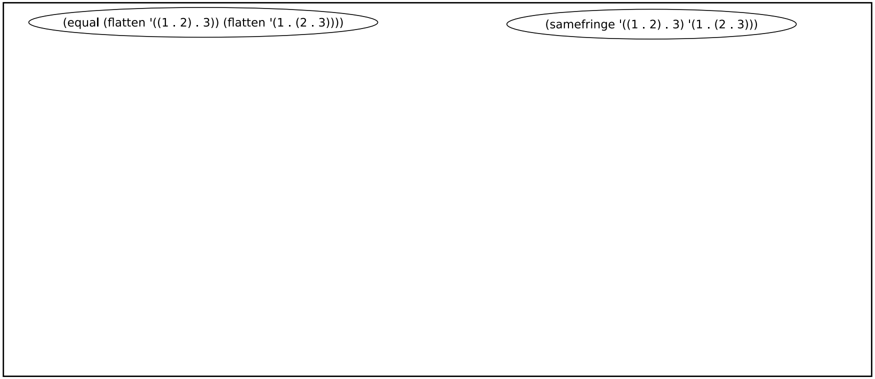


Operations on Terms: Extract

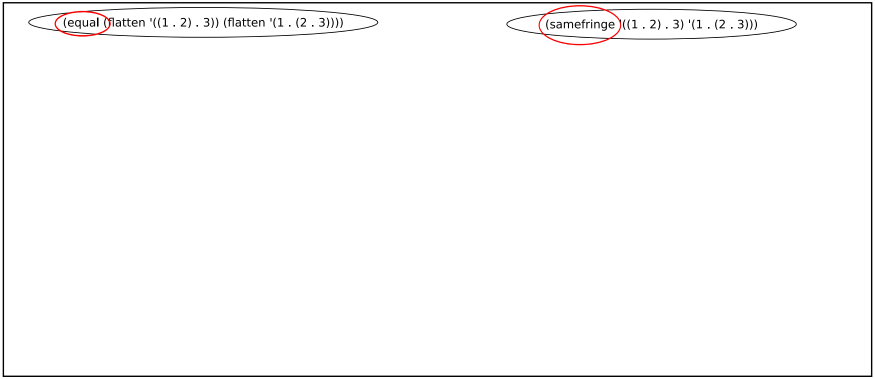
The *extract* operation extracts a subterm:



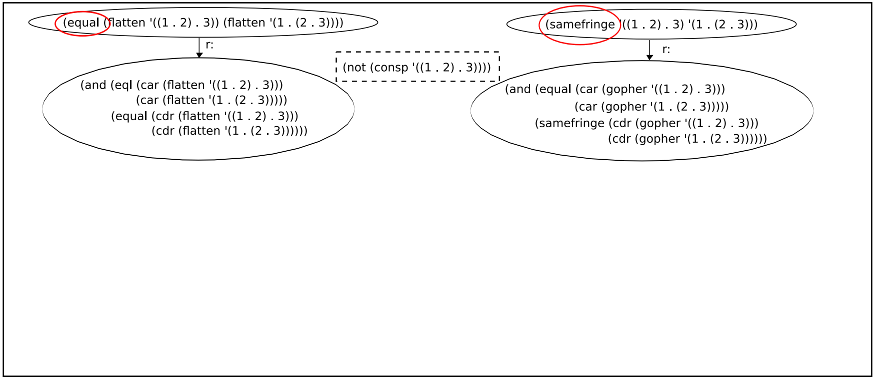
Meaningful Observations


(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))

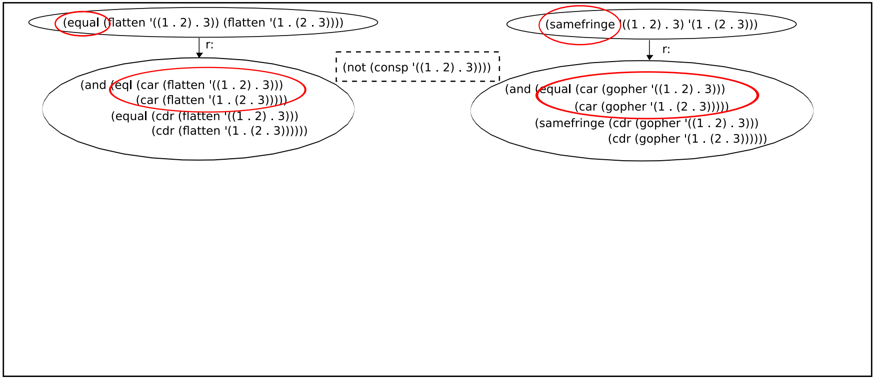
Meaningful Observations


(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))

Meaningful Observations

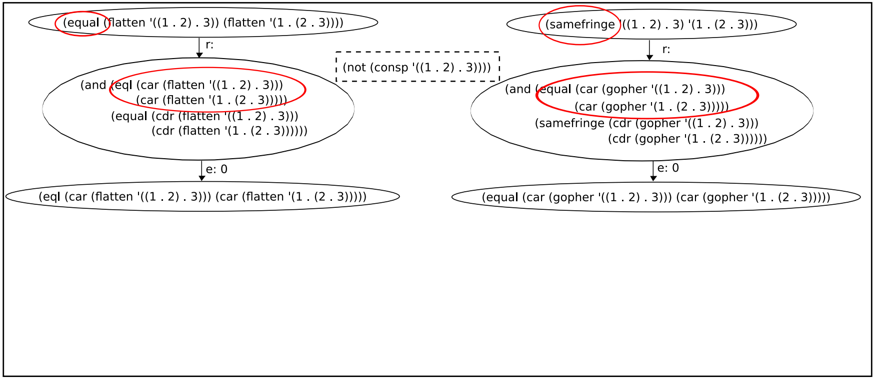

(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
r:
(and (eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3))))) (equal (cdr (flatten '((1 . 2) . 3))) (cdr (flatten '(1 . (2 . 3))))))
(not (consp '((1 . 2) . 3))))
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))
r:
(and (equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3))))) (samefringe (cdr (gopher '((1 . 2) . 3))) (cdr (gopher '(1 . (2 . 3))))))

Meaningful Observations

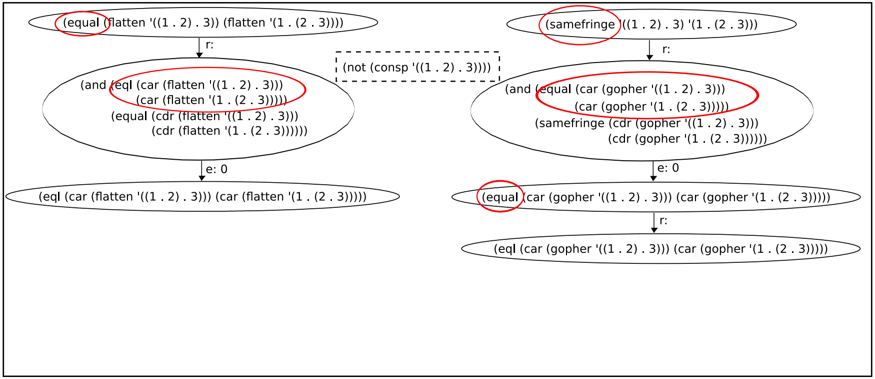


```
(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))

(and (eql (car (flatten '((1 . 2) . 3)))
          (car (flatten '(1 . (2 . 3)))))
     (equal (cdr (flatten '((1 . 2) . 3)))
            (cdr (flatten '(1 . (2 . 3))))))

(not (consp '((1 . 2) . 3))))

(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))

(and (equal (car (gopher '((1 . 2) . 3)))
            (car (gopher '(1 . (2 . 3)))))
     (samefringe (cdr (gopher '((1 . 2) . 3)))
                 (cdr (gopher '(1 . (2 . 3))))))
```

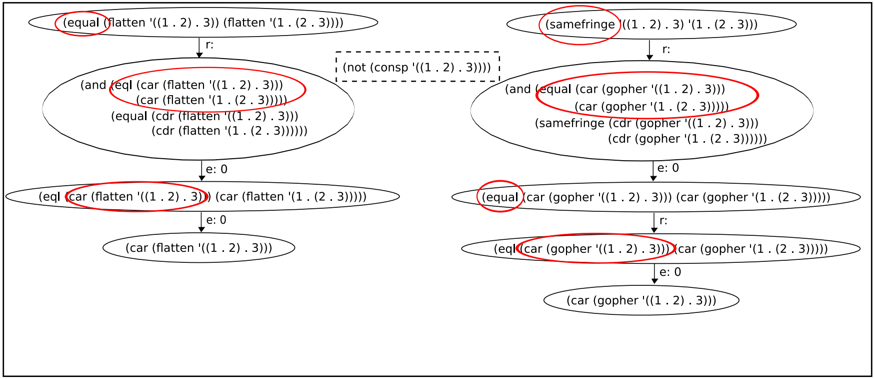
Meaningful Observations


(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
r:
(and (eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3))))) (equal (cdr (flatten '((1 . 2) . 3))) (cdr (flatten '(1 . (2 . 3))))))
e: 0
(eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3)))))
(not (consp '((1 . 2) . 3))))
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))
r:
(and (equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3))))) (samefringe (cdr (gopher '((1 . 2) . 3))) (cdr (gopher '(1 . (2 . 3))))))
e: 0
(equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))

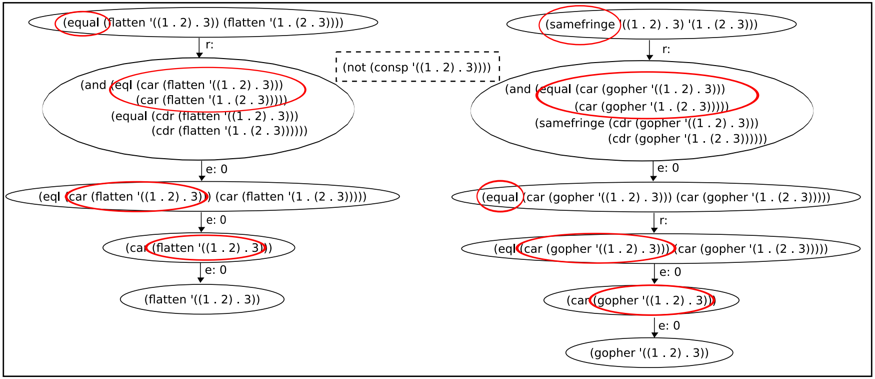
Meaningful Observations


(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
r:
(and (eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3))))) (equal (cdr (flatten '((1 . 2) . 3))) (cdr (flatten '(1 . (2 . 3))))))
e: 0
(eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3)))))
(not (consp '((1 . 2) . 3))))
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))
r:
(and (equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3))))) (samefringe (cdr (gopher '((1 . 2) . 3))) (cdr (gopher '(1 . (2 . 3))))))
e: 0
(equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))
r:
(eql (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))

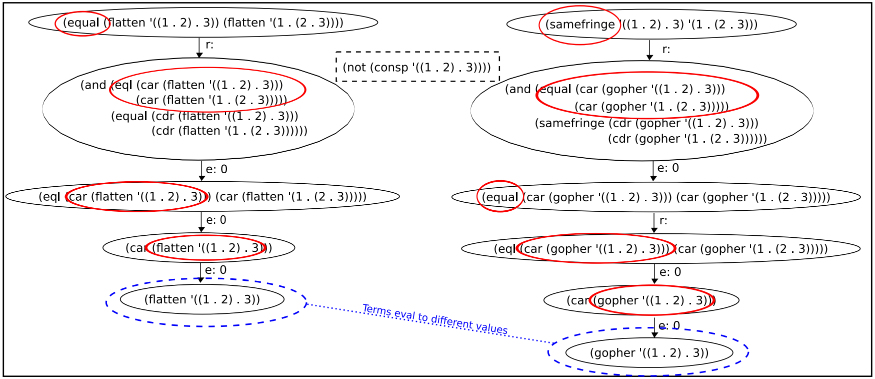
Meaningful Observations


(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
r:
(and (eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3))))) (equal (cdr (flatten '((1 . 2) . 3))) (cdr (flatten '(1 . (2 . 3))))))
e: 0
(eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3)))))
e: 0
(car (flatten '((1 . 2) . 3)))
(not (consp '((1 . 2) . 3))))
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))
r:
(and (equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3))))) (samefringe (cdr (gopher '((1 . 2) . 3))) (cdr (gopher '(1 . (2 . 3))))))
e: 0
(equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))
r:
(eql (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))
e: 0
(car (gopher '((1 . 2) . 3)))

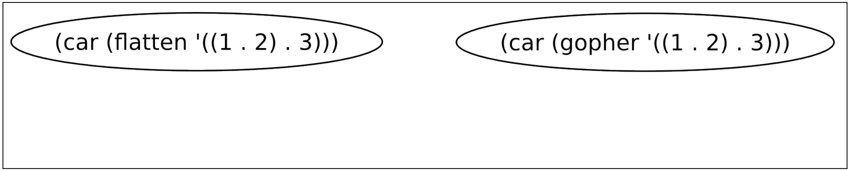
Meaningful Observations


(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
r:
(and (eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3))))) (equal (cdr (flatten '((1 . 2) . 3))) (cdr (flatten '(1 . (2 . 3))))))
e: 0
(eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3)))))
e: 0
(car (flatten '((1 . 2) . 3)))
e: 0
(flatten '((1 . 2) . 3))
(not (consp '((1 . 2) . 3))))
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))
r:
(and (equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3))))) (samefringe (cdr (gopher '((1 . 2) . 3))) (cdr (gopher '(1 . (2 . 3))))))
e: 0
(equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))
r:
(eql (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))
e: 0
(car (gopher '((1 . 2) . 3)))
e: 0
(gopher '((1 . 2) . 3))

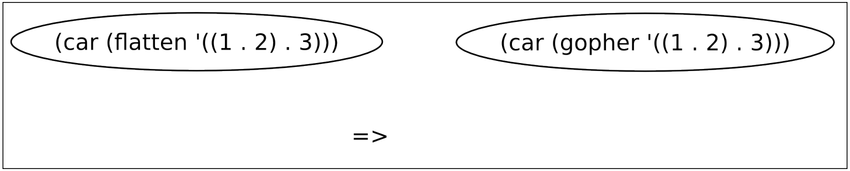
Meaningful Observations


(equal (flatten '((1 . 2) . 3)) (flatten '(1 . (2 . 3))))
r:
(and (eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3))))) (equal (cdr (flatten '((1 . 2) . 3))) (cdr (flatten '(1 . (2 . 3))))))
e: 0
(eql (car (flatten '((1 . 2) . 3))) (car (flatten '(1 . (2 . 3)))))
e: 0
(car (flatten '((1 . 2) . 3)))
e: 0
(flatten '((1 . 2) . 3))
(not (consp '((1 . 2) . 3))))
Terms eval to different values
(samefringe '((1 . 2) . 3) '(1 . (2 . 3)))
r:
(and (equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3))))) (samefringe (cdr (gopher '((1 . 2) . 3))) (cdr (gopher '(1 . (2 . 3))))))
e: 0
(equal (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))
r:
(eql (car (gopher '((1 . 2) . 3))) (car (gopher '(1 . (2 . 3)))))
e: 0
(car (gopher '((1 . 2) . 3)))
e: 0
(gopher '((1 . 2) . 3))

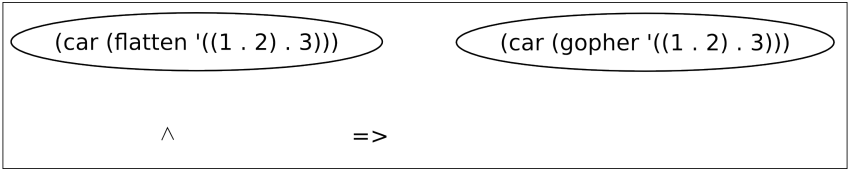
Building Lemmas



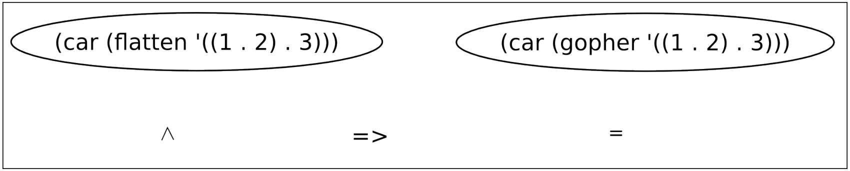
Building Lemmas



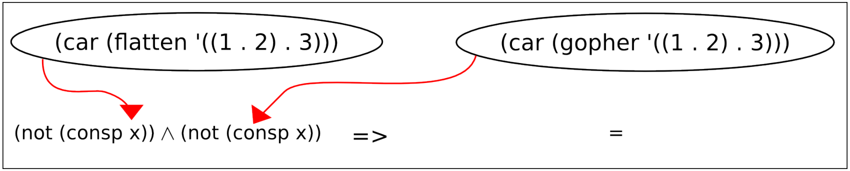
Building Lemmas



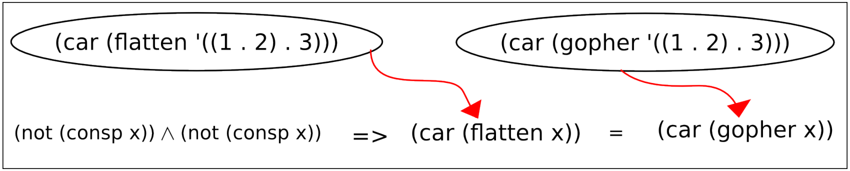
Building Lemmas



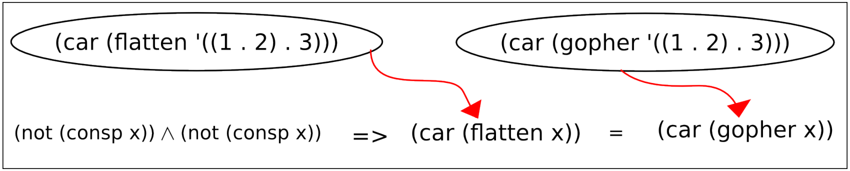
Building Lemmas


(car (flatten '((1 . 2) . 3)))
(car (gopher '((1 . 2) . 3)))
(not (consp x)) ∧ (not (consp x))
=>
=

Building Lemmas


(car (flatten '((1 . 2) . 3)))
(car (gopher '((1 . 2) . 3)))
(not (consp x)) ∧ (not (consp x))
=>
(car (flatten x))
=
(car (gopher x))

Building Lemmas


(car (flatten '((1 . 2) . 3)))
(car (gopher '((1 . 2) . 3)))
(not (consp x)) ∧ (not (consp x))
=>
(car (flatten x))
=
(car (gopher x))

Our new lemma:

```
(implies (consp x)
         (equal (car (flatten x))
                (car (gopher x))))
```

Current and Future Work

1. New patterns.
2. Heuristics for helping applying induction.
3. Filtering out lemmas.
4. Graphical interface.
5. Empirical evaluation of lemma discovery strategies.