Case Study
A Simple Polynomial System

- A univariate polynomial:
  $c_n x^n + c_{n-1} x^{n-1} + \cdots + c_0 x^0$
  c_i: constant integer coefficient
  x^i: variable with exponent
  $c_i x^i$: term

- Operations: $\left\{ \begin{array}{c} \text{Add} \\ \text{Subtract} \end{array} \right\}$ two polys

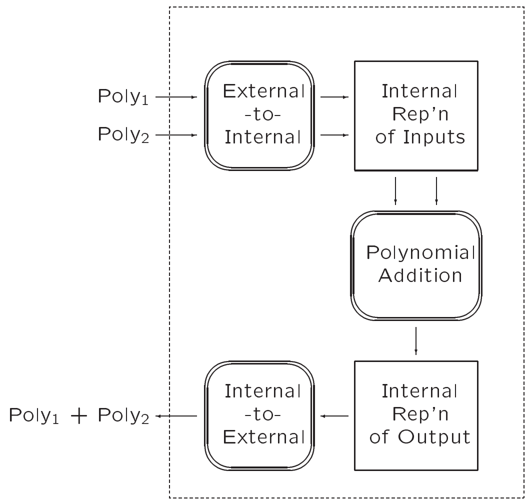
- Examples:
  - $(7x^3 + 2x + 2) + (4x^4 + x^2 + x)$
    $\rightsquigarrow (4x^4 + 7x^3 + x^2 + 3x + 2)$
  - $(7x^3 + 2x + 2) - (4x^4 + x^2 + x)$
    $\rightsquigarrow (-4x^4 + 7x^3 - x^2 + x + 2)$

Components

Given 2 polynomials
in "external representation"
Return their sum/difference
in "external representation"

1. Convert "external rep'n" into internal rep'n.
2. Perform requested operation [+/−]
   (using internal rep'n).
3. Convert resulting polynomial
   to external rep'n.

Overview of Process



Internal Form

Mathematical: $2x + x^4 + 7x^3 - 6$

User's Entry:

```
(+ (* 2 x) (** x 4) (* 7 (** x 3)) -6)
```

Exponent/Coefficient Pairs:

```
((1 . 2) (4 . 1) (3 . 7) (0 . -6))
```

Descending Exponent/Coefficient Pairs:

```
( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )
```

[Internal Representation]

Polynomial Operations

$(7x^3 + 2x + 2) \quad + \quad (3x^4 + x^2 + x)$

((3 . 7) (1 . 2) (0 . 2)) + ((4 . 3) (2 . 1) (1 . 1))

```
      7  ∅  2  2
+  3  ∅  1  1  ∅
----------------
   3  7  1  3  2
```

$\leadsto \quad 3x^4 \quad + 7x^3 \quad + x^2 \quad + 3x \quad + 2$

((4 . 3) (3 . 7) (2 . 1) (1 . 3) (0 . 2))

$(7x^3 + 2x + 2) \quad - \quad (3x^4 + x^2 + x)$

((3 . 7) (1 . 2) (0 . 2)) − ((4 . 3) (2 . 1) (1 . 1))

```
       7  ∅  2  2
-   3  ∅  1  1  ∅
-----------------
   -3  7 -1  1  2
```

$\leadsto \quad -3x^4 \quad + 7x^3 \quad - x^2 \quad + x \quad + 2$

$((4\ .\ -3)\ (3\ .\ 7)\ (2\ .\ -1)\ (1\ .\ 1)\ (0\ .\ 2))$

Assumptions

- Only one variable [named “x”]
- Integer exponents (non-negative)
  Integer coefficient
- At most 1 term with any exponent
- Each term is of form:

`(* ⟨coef⟩ (** x ⟨exp⟩))`	⟨coef⟩x$^{\langle exp \rangle}$	
`(** x ⟨exp⟩)`	x$^{\langle exp \rangle}$	
`(* ⟨coef⟩ x)`	⟨coef⟩x	(linear)
`⟨coef⟩`	⟨coef⟩	(constant)

- . . .

External to Internal Conversion

Given external form:

`(+ (* 2 x) (** x 4) (* 7 (** x 3)) -6)`

Return internal form:

`( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )`

- For each term in External Form:
  Determine its coefficient & exponent

  Forming list of
  (⟨exponent⟩ . ⟨coefficient⟩) pairs.

- Sort this list
  by (descending) order of exponents.

Needed Helper Functions

`(E-to-I ⟨term list⟩)`

Returns list of exponent/coefficient pairs, from ⟨term list⟩.

Convert “`((* 2 x) (** x 4) (* 7 (** x 3)) -6)`”
into “`( (1 . 2) (4 . 1) (3 . 7) (0 . -6) )`”

`(Term-to-Pair ⟨term⟩)`

Returns exponent/coefficient pair of ⟨term⟩.

```
(* 2 x)          ⇝ (1 . 2)
(** x 4)         ⇝ (4 . 1)
(* 7 (** x 3))   ⇝ (3 . 7)
-6               ⇝ (0 . -6)
```

`(Exp-of-Term ⟨term⟩)`

Extracts exponent of ⟨term⟩.

“`(** x 3)`” ⇝ “3”; “x” ⇝ “1”

Ext-to-Int

```
(defun ext-to-int (poly)
      ; converts (+ (* 2 x) (** x 4) (* 7 (** x 3)) -6 )
      ; into ( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )

  (sort (e-to-i (cond ((atom poly)            (list poly))
                      ((eq (car poly) '+)     (cdr poly))
                      ((eq (car poly) '*)     (list poly))
                      ((eq (car poly) '**)    (list poly))
                      ( t    (error "In Ext-to-Int
                                   - bad input"))))
        '(lambda (x y) (> (car x) (car y)))) )


(defun e-to-i (p)
      ; converts ((* 2 x) (** x 4) (* 7 (** x 3)) -6 )
      ; into ( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )

  (cond ((null p)  nil)
        ((atom p)  (error "In E-to-I - bad input"))
        ( t   (cons (Term-to-Pair (car p))
                    (e-to-i (cdr p)) ))))
```


Evaluating Ext-to-Int

```
→ (ext-to-int '(+ (* 2 x) (** x 4) (* 7 (** x 3)) -6 ))
  1.  Enter e-to-i:  ((* 2 x) (** x 4) (* 7 (** x 3)) -6 ))
  1.  Exit e-to-i:  ((1 . 2) (4 . 1) (3 . 7) (0 . -6))
( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )

→ (ext-to-int '(* 7 (** x 3)))
  1.  Enter e-to-i:  ((* 7 (** x 3)))
  1.  Exit e-to-i:  ((3 . 7))
((3 . 7))

→ (ext-to-int '(** x 9))
  1.  Enter e-to-i:  ((** x 9))
  1.  Exit e-to-i:  ((9 . 1))
((9 . 1))

→ (ext-to-int '22))
  1.  Enter e-to-i:  (22)
  1.  Exit e-to-i:  ((0 . 22))
((0 . 22))
```


Term-to-Pair

```
(defun Term-to-Pair (term)
  (cond ((atom term)
         (cond ((numberp term) (cons 0 term))
               ((eq term 'x)   (cons 1 1))
               ( t  (error "In Term-to-Pair - bad atomic input
        ((eq (car term) '*)
          (cond((not (numberp (cadr term)))
                 (error "In Term-to-Pair - non-number coeffic
               (t (cons (exp-of-term (caddr term))
                        (cadr term) ))))
        (t (cons (exp-of-term term) 1))))


→ (Term-to-Pair '-6)
(0 . -6)
→ (Term-to-Pair 'x)
(1 . 1)
→ (Term-to-Pair '(* 2 x))
(1 . 2)
→ (Term-to-Pair '(* 7 (** x 3)))
(3 . 7)
→ (Term-to-Pair '(* 52 x))
(52 . 1)
→ (Term-to-Pair '(** x 4))
(4 . 1)
```


Exp-of-Term

```
(defun exp-of-term (et)
      ; determines coefficient of exponent-term:
      ; (** x 3) ⇝ 3; x ⇝ 1
  (cond ((eq et 'x) 1)
        ((atom et)
          (error "In exp-of-term - non-x atom"))
        ((not (eq (car et) '**))
          (error "In exp-of-term - expected **"))
        ((and (numberp (caddr et)) (eq (cadr et) 'x))
          (caddr et))
        ( t  (error "In exp-of-term - bad form"))))


→ (exp-of-term 'x)
1
→ (exp-of-term '(** x 3))
3
→ (exp-of-term '(** x 2))
2
```

Polynomial Addition

```
(defun poly-add (pol1 pol2)
  (int-to-ext (p-add (ext-to-int pol1) (ext-to-int pol2))))

(defun p-add (ip1 ip2)
  (cond ((null ip1)     ip2)
        ((null ip2)     ip1)
        ((< (caar ip1) (caar ip2))
          (cons (car ip2) (p-add ip1 (cdr ip2))))
        ((> (caar ip1) (caar ip2))
          (cons (car ip1) (p-add (cdr ip1) ip2)))
        ( t      ; here (= (caar ip1) (caar ip2))
          (cons (cons (caar ip1) (+ (cdar ip1) (cdar ip2)))
                (p-add (cdr ip1) (cdr ip2)))) ))
```

```
→ (poly-add '(+ (* 2 x) (** x 4) (* 52 (** x 3)) -6 )
            '(+ (* 4 (** x 2)) (** x 5) 4 (* -1 (** x 4))))

  1.  Enter p-add:( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )
                  ( (5 . 1) (4 . -1) (2 . 4) (0 . 4) )

  1.  Exit p-add:  ((5 . 1) (4 . 0) (3 . 7) (2 . 4)
                   (1 . 2) (0 . -2) )

(+ (** x 5) (* 7 (** x 3)) (* 4 (** x 2)) (* 2 x) -2)
```

Polynomial Subtraction

```
(defun poly-sub (pol1 pol2)
  (int-to-ext (p-sub (ext-to-int pol1) (ext-to-int pol2))))

(defun p-sub (ip1 ip2)
  (cond ((null ip1)     (p-sub '((0 . 0)) ip2))
        ((null ip2)     ip1)
        ((< (caar ip1) (caar ip2))
          (cons (cons (caar ip2) (- (cdar ip2)))
                (p-sub ip1 (cdr ip2))))
        ((> (caar ip1) (caar ip2))
          (cons (car ip1) (p-sub (cdr ip1) ip2)))
        ( t      ; here (= (caar ip1) (caar ip2))
          (cons (cons (caar ip1) (- (cdar ip1) (cdar ip2)))
                (p-sub (cdr ip1) (cdr ip2))))))
```

```
→ (poly-sub '(+ (* 2 x) (** x 4) (* 52 (** x 3)) -6 )
            '(+ (* 4 (** x 2)) (** x 5) 4 (* -1 (** x 4))))

  1.  Enter p-sub:  { ( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )
                    { ( (5 . 1) (4 . -1) (2 . 4) (0 . 4) ) }

  1.  Exit p-sub:  ( (5 . -1) (4 . 2) (3 . 7) (2 . -4)
                   (1 . 2) (0 . -10) )

(+ (* -1 (** x 5)) (* 2 (** x 4)) (* 7 (** x 3))
   (* -4 (** x 2)) (* 2 x) -10)
```

Int-to-Ext's Utilities

```
(defun i-to-e (ip)
       ; converts ( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )
       ; into ((** x 4) (* 7 (** x 3)) (* 2 x) -6 )
  (cond ((null ip)    nil)
        ((eq (cdar ip) 0)  ; if coeff is 0, just ignore term!
          (i-to-e (cdr ip)))
        ((eq (caar ip) 0)  ; if exp is 0, just print coeff.
          (cons (cdar ip) (i-to-e (cdr ip))))
        ((eq (cdar ip) 1)  ; if coeff is 1, just ignore coeff
          (cons (built-exp (caar ip)) (i-to-e (cdr ip))))
        ( t (cons (list '* (cdar ip) (build-exp (caar ip)))
                (i-to-e (cdr ip))))))

(defun build-exp (c)
         ; converts 4 into (** x 4)
         ; 1 into x
  (cond ((eq c 1) 'x)
        ( t  (list '** 'x c))))
```

Evaluating I-to-E

```
→ (i-to-e '((4 . 0) (1 . 2))
((* 2 x))

→ (i-to-e '((4 . 0)))
nil

→ (i-to-e '((0 . 7)))
(7)

→ (i-to-e '((4 . 1)))
((** x 4))

→ (i-to-e '((1 . 1)))
(x)

→ (i-to-e '((3 . 7)))
((* 7 (** x 3)))

→ (i-to-e '((4 . 1) (3 . 7) (1 . 2) (0 . -6)))
((** x 4) (* 7 (** x 3))(* 2 x) -6 ))
```

Int-to-Ext

```
(defun int-to-ext (int)
       ; converts ( (4 . 1) (3 . 7) (1 . 2) (0 . -6) )
       ; into (+ (** x 4) (* 7 (** x 3)) (* 2 x) -6 )
       ; ( (4 . 1) ) ~> (** x 4)
  (let ((ext (i-to-e int)))
    (cond ((null ext) '0)
          ((cdr ext) (cons '+ (i-to-e int)))
          ( t   (car ext)))))
```

→ <u>(int-to-ext '((4 . 0)))</u>

1. Enter i-to-e: ((4 . 0))
1. Exit i-to-e: nil

0

Evaluating Int-to-Ext (con't)

→ <u>(int-to-ext '((4 . 1) (3 . 7) (1 . 2) (0 . -6)))</u>

1. *Enter* i-to-e: ((4 . 1) (3 . 7) (1 . 2) (0 . -6))
1. *Exit* i-to-e: ((** x 4) (* 7 (** x 3))(* 2 x) -6))

(+ (** x 4) (* 7 (** x 3))(* 2 x) -6))

→ <u>(int-to-ext '((3 . 7)))</u>

1. *Enter* i-to-e: ((3 . 7))
1. *Exit* i-to-e: ((* 7 (** x 3)))

(* 7 (** x 3)))

→ <u>(int-to-ext '((9 . 1)))</u>

1. *Enter* i-to-e: ((9 . 1))
1. *Exit* i-to-e: ((** x 9))

(** x 9)

→ <u>(int-to-ext '((0 . 22)))</u>

1. *Enter* i-to-e: ((0 . 22))
1. *Exit* i-to-e: (22)

22