Verifying the Transformation Rules of the High-Assurance Transformation System (HATS):

Taking the Class Loader Implementation as an Example

Fares Fraij

June 2, 2004

Outline

- Transformation-Oriented Programming (TOP)
- High-Assurance Transformation System (HATS)
- Sandia Secure Processor (SSP)
- SSP-classloader implementation in HATS
- The ACL2 model
- Sketch of the verification effort

Transformation-Oriented Programming (TOP)

- A formal software development paradigm
- Incremental refinment of formal specifications to implementations
- HATS is an example of TOP

HATS

- Goals
- High-level overview
- Architecture
- Transformation language program
- Example

HATS Goals

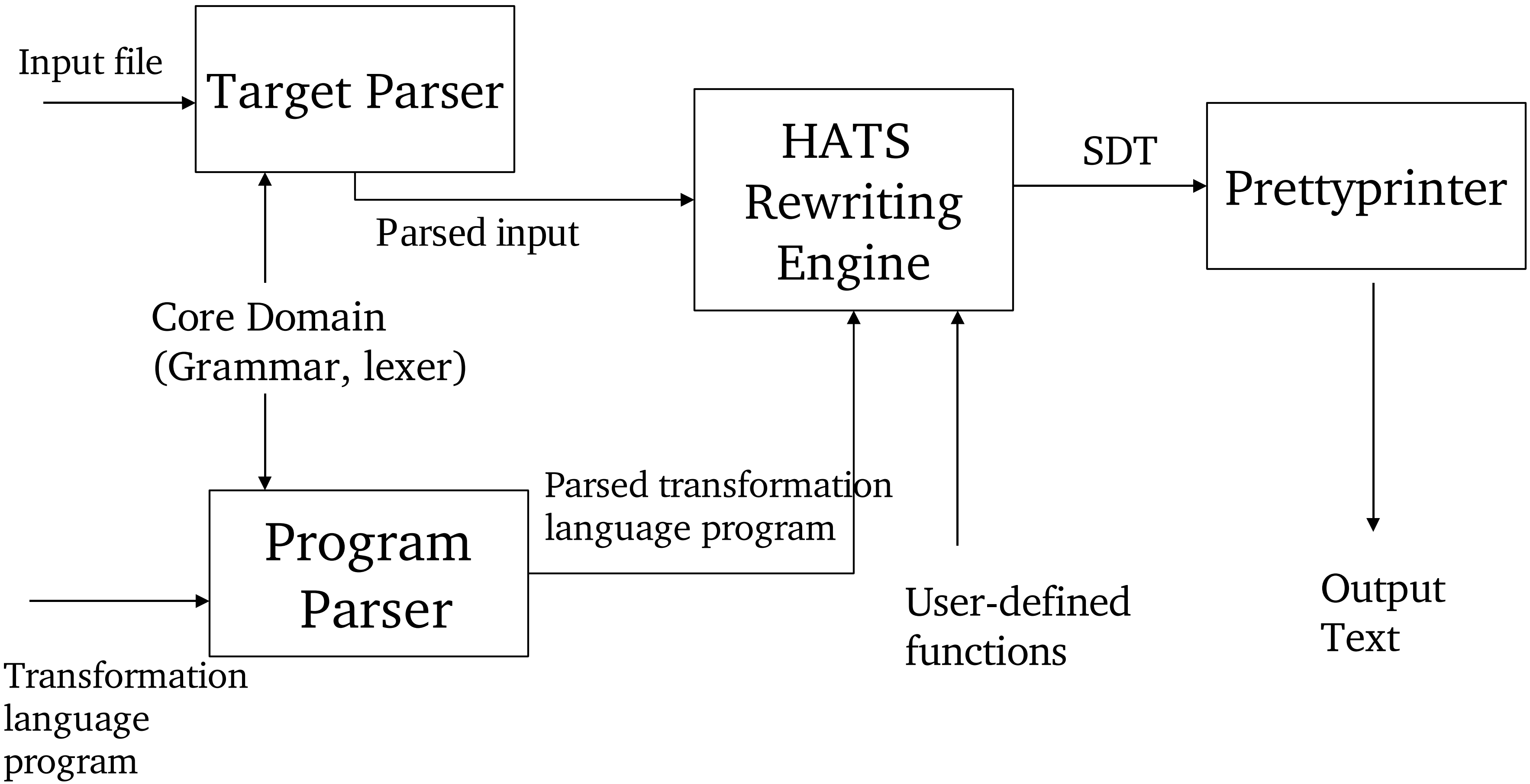
- Create a language-independent program transformation system
- Perform program transformation in a provable correct fashion

HATS High-Level Overview

Input program in specification language → HATS → Output program in implementation language

- Transforms input programs written in abstract languages to output programs in concrete languages
- Transformation language program (TLP) consists of sequence of transformation rules and a control strategy

HATS architecture



HATS Transformation Language Program

- Transformation rules:
  - General form

    left-hand side   *right-hand side if C*
  - Two types of transformation rules
    - First-Order
    - High-Order
- Control strategies

HATS Transformation Language Program

- Transformation rules
- Control strategies control the application of transformation rules to the input file
  - Control strategies
    - Once
    - Fix
    - Transient
    - Hide

Example

- Input expression: (* (* 2 (+ 3 4))
  (* 5 (+ 6 7))
  (* 8 (+ 9 10))
- Transformation rule:

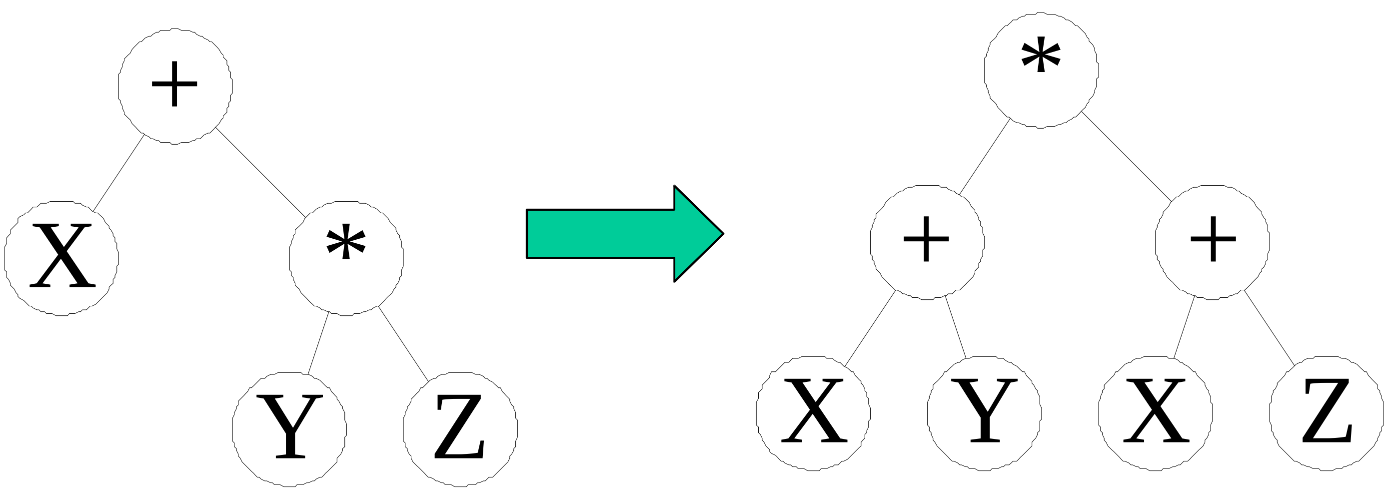
  (* X (+ Y Z)) ⟶ (+ (* X Y) (* X Z))

- Control strategy: *fix*

Example

HATS internal representation of the transformation rule

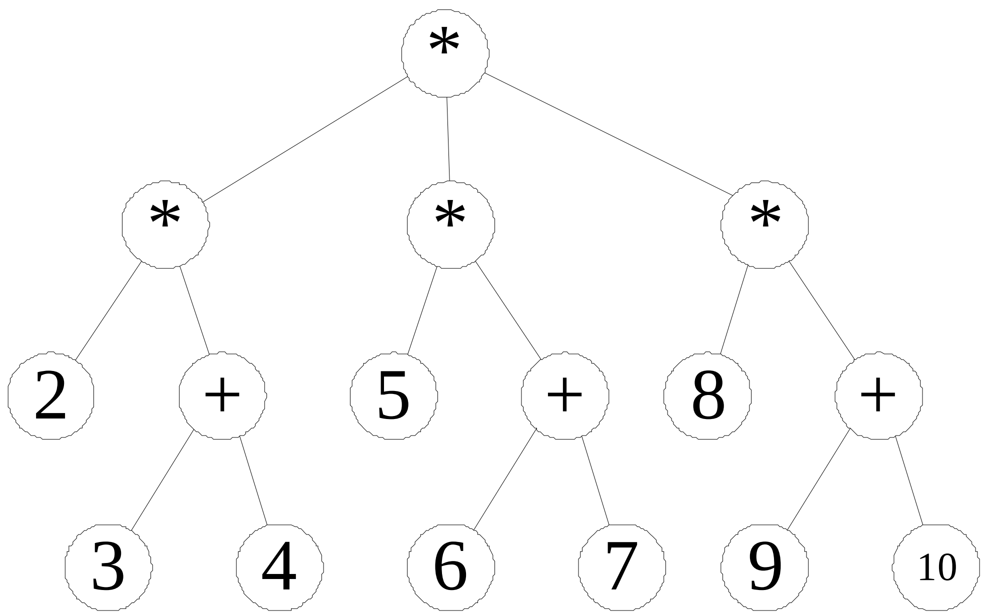
(* X (+ Y Z)) → (+ (* X Y) (* X Z))



Example

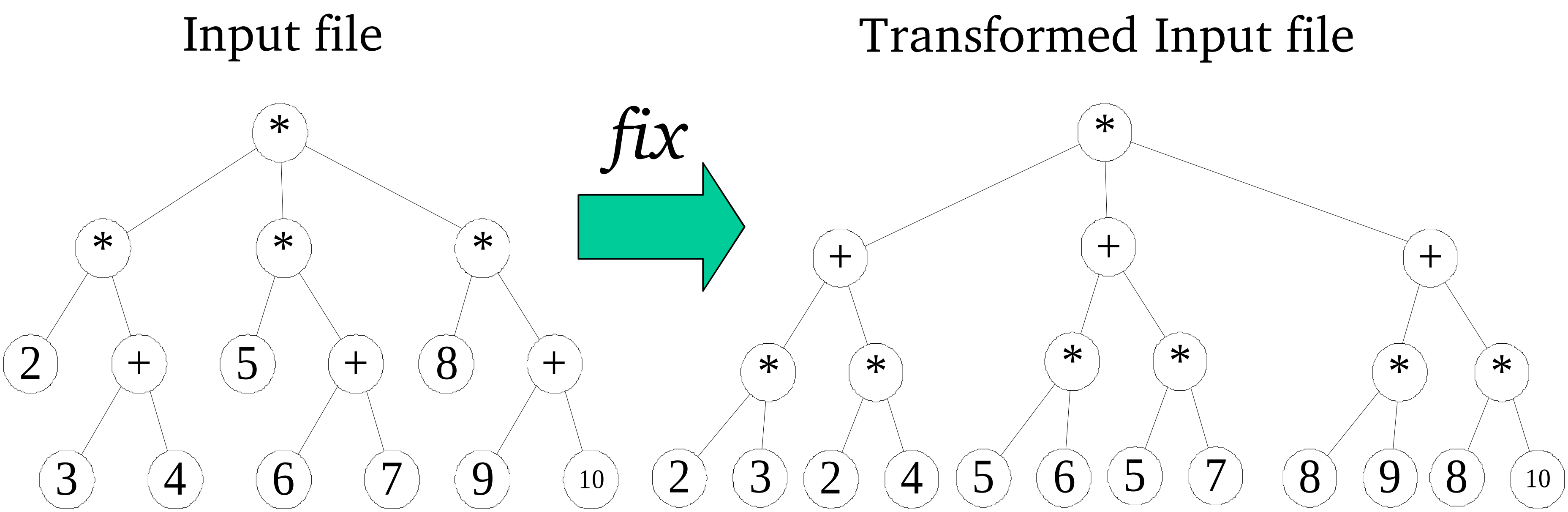
HATS internal representation of the input expression

(* (* 2 (+ 3 4))
(* 5 (+ 6 7))
(* 8 (+ 9 10))



Example

The result of applying the transformation rule to the input file using the control strategy *fix*



Example of Hide and Transient

static-addresses : c_0 TDL (lcond-tdl sfield-sum c_0) c_0

sfield-sum: addr[[i]]

transient(addr[[i]] addr[[i + 1]]) <+

hide(addr[[j]] addr[[j+1]])

The result of applying static-addresses to an input files, , that has three components is as follows:

(transient(addr[[i]] addr[[i + 1]]) <+ hide(addr[[j]] addr[[j+1]]))<+

(transient(addr[[i]] addr[[i + 1]]) <+ hide(addr[[j]] addr[[j+1]]))<+

(transient(addr[[i]] addr[[i + 1]]) <+ hide(addr[[j]] addr[[j+1]]))<+

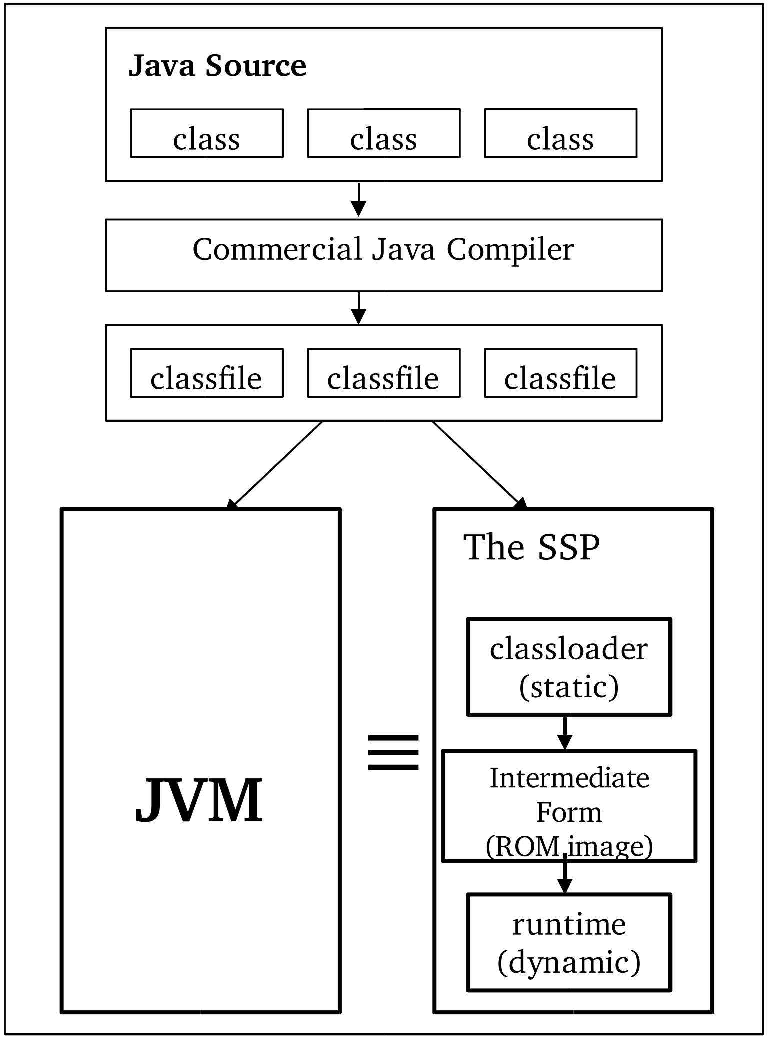
Sandia Secure Processor (SSP)

- Goals
- Components
- SSP and JVM
- *SSP-classloader* and HATS
- *First canonical* form of *SSP-classloader*

SSP goals

- Create Simplified Java processor for embedded systems
- Design is intended to be small, simple, and analyzable
- Provide a general-purpose computational infrastructure suitable for use in high-consequence embedded systems

SSP Components



SSP-classloader and HATS

- HATS is used to implement the *SSP-classloader*
- Functionality of the *SSP-classloader* is decomposed into five *canonical forms*
  - *Form1: index resolution*
  - *Form2:* static fields address calculation
  - *Form3:* offset address calculation
  - *Form4:* method table construction
  - *Form5:* inter-class distribution

Form1 - Index Resolution

Index-resolution: $class_0 \longrightarrow$ FIX_TDL (seq-tdl cp-normalize $class_0$) class0

cp-normalize: c-entry[[($index_1$, d_1)]] $\longrightarrow$ d[[$index_1$]] $\longrightarrow d_1$

Form2 - Static Fields Address Calculation

static-addresses : $class_0 \longrightarrow$ TDL (lcond-tdl sfield-sum $class_0$) $class_0$

sfield-sum: sfield [[key_1 @ $addr_1$]] $\longrightarrow$

transient(sfield[[key_1 @ $addr_1$]] $\longrightarrow$ sfield[[key_1 @ $addr_1$ + 1]]) <+

hide(sfield[[key_2 @ $addr_2$]] $\longrightarrow$ sfield[[key_2 @ $addr_2$+1]])

Form1

- Goal of *first canonical* form is to resolve all indirections in the constant pool of input class file, C_0.

C_0 = ((1 "Hello")
(2 "World")
(3 2)
(4 3))

→

C_1 = ((1 "Hello")
(2 "World")
(3 "World")
(4 "World"))

Modeling The Transformation Rules of The SSP-classloader in ACL2

- We simulate two control strategies:
  - *once: once-strategy*
  - *Fix* : *fix-strategy*

- We simulate the generation of *High-Order transformation rules* using the function *generate-rules*

Modeling The Transformation Rules of The SSP-classloader in ACL2

- $STF_{form\text{-}0}$ is the function that simulates the task of the *first canonical form, i*.e.,

$$STF_{form\text{-}0}\ (C_0) = C_1 ,$$

Where C_0 is the input file and C_1 is the result of resolving the indexes.

Sketch of The Verification Effort

- Construct a semantic function *S* for each *form*
- Main conjecture:

$$\forall (C),\ S_n (C) = S_n (STF_{form\text{-}n} (C)),$$

where S_n is the sematic function that corresponds to the form n, where n =1,2, …, 5, $STF_{form\text{-}n}$ is the a function that simulates the behavior of the form *n*, and *C* is the input file.

Example

- First canonical form can be abstracted by a table *resolution* problem as follows:

$C_0 =$ ((1 "Hello")

(2 "World")

(3 2)

(4 3))

Example

- Second-order transformation rule

TR-1 = (i j) ⟶ (x i) ⟶ (x j)

- Applying this rule to the table gives:

TR-1.0 = (x 1) ⟶ (x "Hello)

TR-1.1 = (x 2) ⟶ (x "World")

TR-1.2 = (x 3) ⟶ (x 2)

TR-1.3 = (x 4) ⟶ (x 3)

Example

We model form1 in ACL2 using function *fix-strategy*

$C_0 =$ ((1 "Hello")
(2 "World")
(3 2)
(4 3))

fix-strategy(C_0) =
((1 "Hello")
(2 "World")
(3 "World")
(4 "World"))

Example

- S_0

  get-constant (n C_0) ;; chaces a pointer, n, down in a

  ;; table, C_0.

  resolve-links (C_0) ;; resolves pointers in a given table,

  ;; C_0

- Conjecture

  (equal (get-constant n (fix-strategy C_0))

  (get-constant n C_0)))

Get-constant -Definition

```
(defun get-constant (n C0)
  (let ((temp (assoc n C0)))
    (cond ((null temp) nil)
          ((stringp (cadr temp)) (cadr temp))
          ((or (not (natp n))
               (not (natp (cadr temp)))
               (<= n (cadr temp)))
           nil)
          (t (get-constant (cadr temp) C0)))))
```

`get-constant` – Input Samples

- Example

```
(defconst *c0* '((1 "Hello")
                 (2 "World")
                 (3 2)
                 (4 3))))
```

ACL2 !>(get-constant 1 *c0*)

"Hello"

ACL2 !>(get-constant 2 *c0*)

"World"

ACL2 !>(get-constant 3 *c0*)

"World"

ACL2 !>(get-constant 4 *c0*)

"World"

ACL2 !>(get-constant 5 *c0*)

NIL

ACL2 !>

`resolve-links` -Definition

```
(defun resolve-links1 (tail C0)
 (cond ((endp tail) nil)
       (t (cons (list (car (car tail))
                      (get-constant (car (car tail))
                                    C0))
                (resolve-links1 (cdr tail) C0)))))

(defun resolve-links (C0)
  (resolve-links1 C0 C0))
```

`resolve-links` – Input Sample

```
ACL2 !>(resolve-links *c0*)
((1 "Hello")
 (2 "World")
 (3 "World")
 (4 "World"))
ACL2 !>
```

Lemma “assoc-resolve-links1“

```
(defthm assoc-resolve-links1
  (implies (and (natp n)
                (alistp tail))
     (equal (assoc n (resolve-links1 tail C0))
            (if (assoc n tail)
                (cons n (get-constant n C0))
                nil))))
```

Theorem “get-constant-resolve-links“

```
(defthm get-constant-resolve-links
  (implies (and (natp n)
                (alistp C0))
    (equal (get-constant n (resolve-links C0))
           (get-constant n C0))))
```