Verifying Transformation Rules of the HATS High-Assurance Transformation System: An Approach

Steve Roach Fares Fraij

Department of Computer Science

The University of Texas at El Paso

Fifth International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2-2004)

November 18, 2004

Goal

Develop models and techniques using ACL2 to prove the correctness of HATS transformation rules and apply them to a high-consequence system

Formal Approaches for Software Assurance

- Transformation-Oriented Programming (TOP)

  Incremental refinement of formal specifications to implementations

  – Correctness by construction

  – Examples: **HATS**, Maude, ELAN, Stratego, and ASF+SDF

- Automated theorem provers

  Model computing systems and their desired properties in the language of the of the theorem prover and prove the correctness of these properties using inference rules, axioms, and theorems

  – Correctness by verification

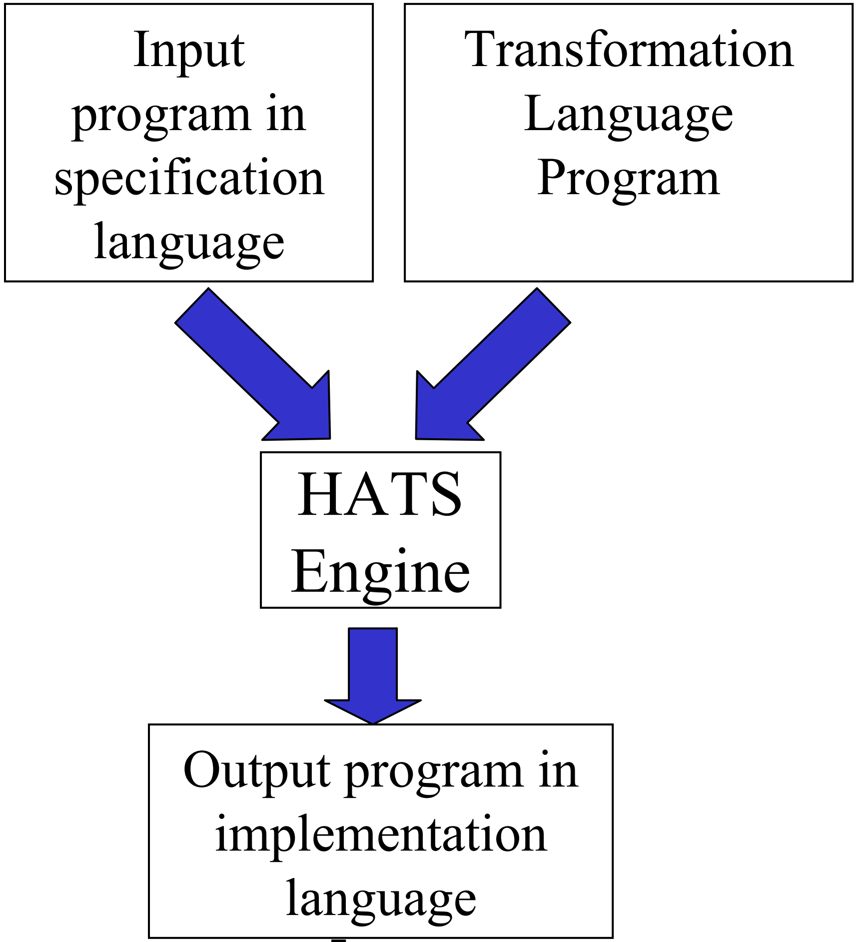
  – Examples: **ACL2**, HOL, PVS, Isabelle

HATS Goals

- Create a language-independent program transformation system
- Perform program transformation in a provably correct fashion
- Provide framework for experimenting with transformation techniques

HATS High-Level Overview

- 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 Transformation Language Program

Transformation rules	Combinators	Control strategies
• General form $LHS \rightarrow RHS \text{ if } C$ • Two types of transformation rules – First-Order – High-Order	• Types: – Seq (;) – Left-biased (<+) – Right-biased (+>)	Control the application of transformation rules to the input file • Types: – Once – Fix – Transient – Hide

Example: Once VS. Fix

Given the following table, T, the goal is to resolve the pointers in the second column to their respective string values *T* = ((1 "Hello") (2 "World") (3 2) (4 3))	To resolve the pointers in the table T, the following first-order transformation rules are needed: *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: Once VS. Fix

Rule-list

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)

Rule-List →, Once →

T = ((1 “Hello”)
(2 “World”)
(3 2)
(4 3))

Result → ; Rule-List →, Once →

NEW-T = ((1 “Hello”)
(2 “World”)
(3 “World”)
(4 2))

Result →

FINAL-T = ((1 “Hello”)
(2 “World”)
(3 “World”)
(4 “World”))

Rule-List →, Fix →

T = ((1 “Hello”)
(2 “World”)
(3 2)
(4 3))

Result →

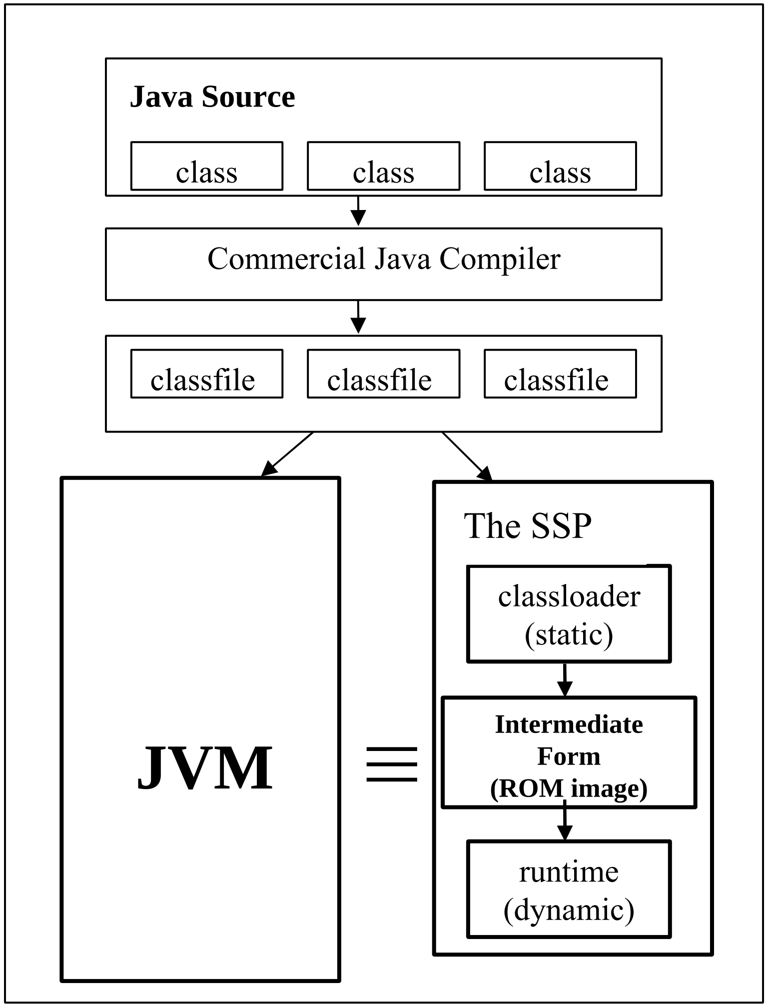
FINAL-T = ((1 “Hello”)
(2 “World”)
(3 “World”)
(4 “World”))

Verification Challenge

How do we know transformations are correct?

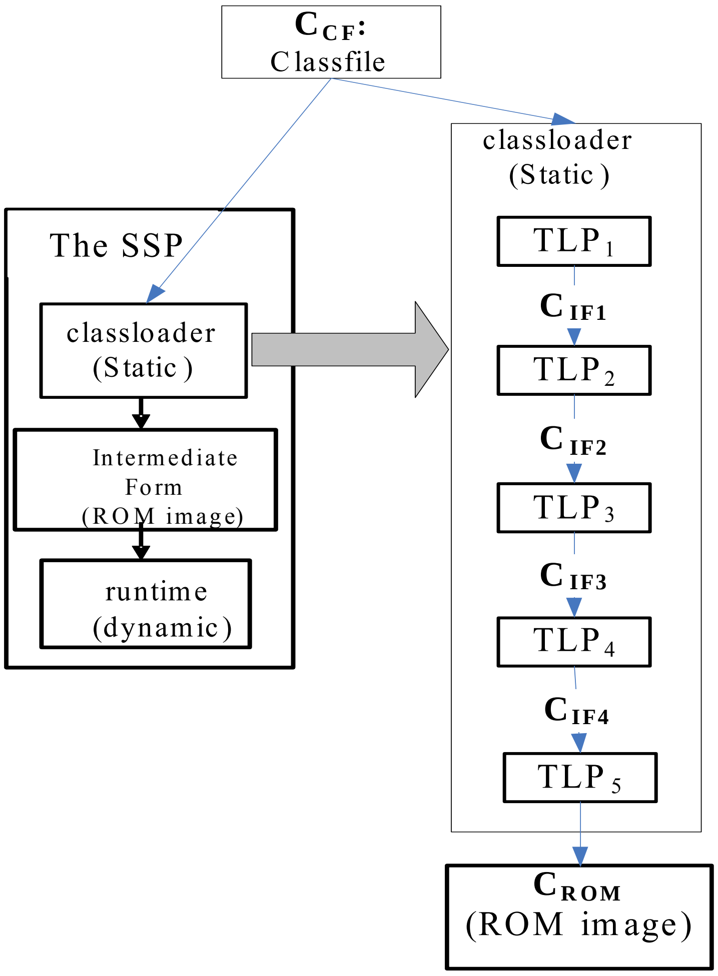
High-Consequence Application: Sandia Secure Processor (SSP)

- A general-purpose computational infrastructure suitable for use in high-consequence embedded systems
- A simplified Java processor designed to be small and analyzable
- Closed system



SSP-classloader and HATS

- HATS is used to implement the *SSP-classloader*
- Functionality of the *SSP-classloader* is decomposed into five *canonical forms*
  - TLP_1: index resolution
  - TLP_2: static fields address calculation
  - TLP_3: instance field offset calculation
  - TLP_4: method table construction
  - TLP_5: inter-class absolute address and offset address distribution



Methodology

- Model the HATS TLP_1 in ACL2
  - Modeling the control strategies and the combinators, $model_{TLP1}$
  - Defining semantic function, S_0
- Prove that the application of the transformation rules preserves the semantics

Methodology

- Model the behavior of TLP_1

  fix-strategy $(C_{CF}$, *rule-list*$)$

  – Applies the rule-list to C_{CF} exhaustively

- Construct a semantic function S_0 for TLP_1

  get-constant $(n\ C_{CF})$

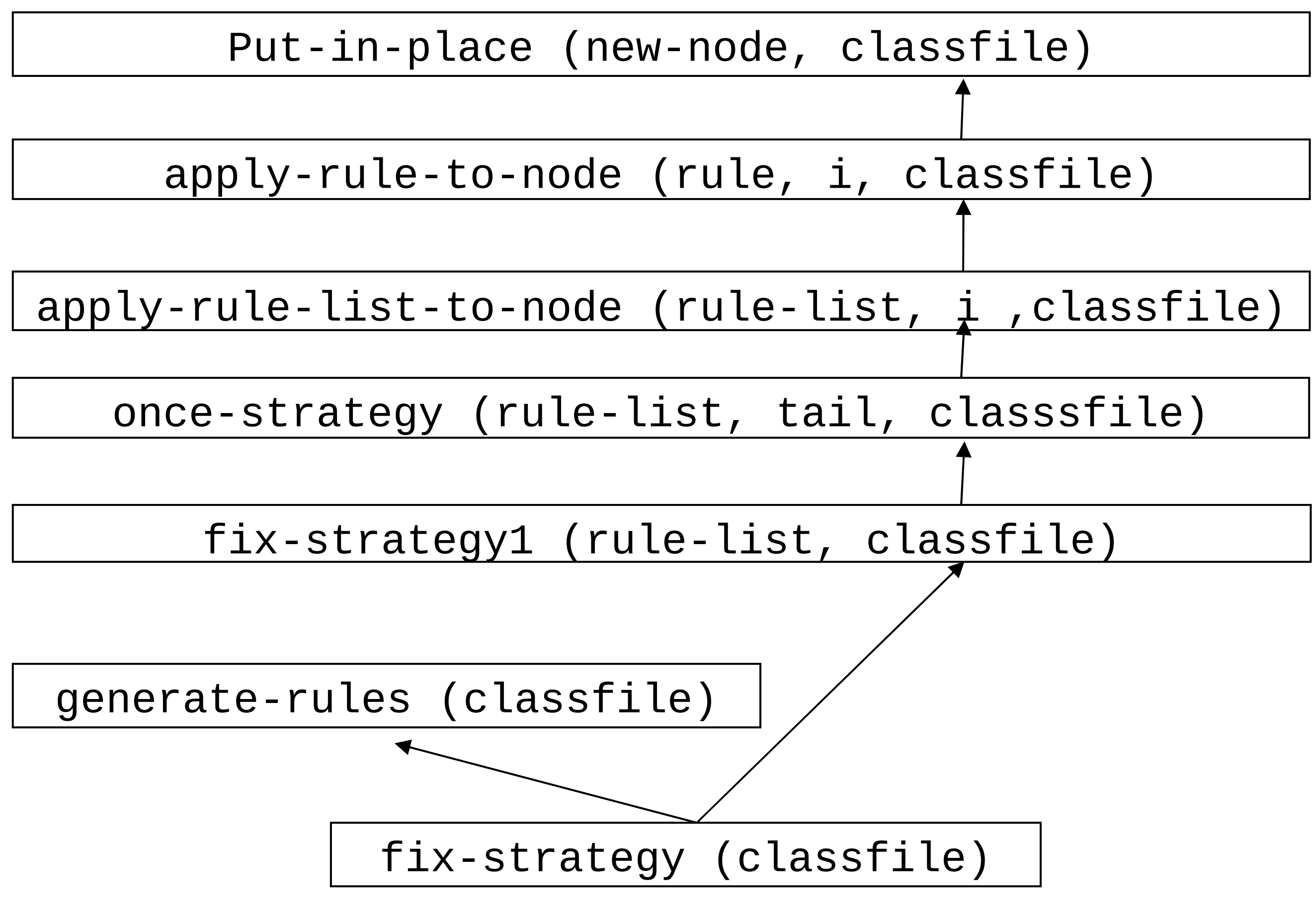
  – Chases a pointer n down in a table C_{CF}

- Main conjecture:

$\forall(C_{CF})\ S_0\ (model_{TLP1}\ (C_{CF})) = S_0\ (C_{CF})$, i.e.,

$$\forall(C_{CF}), \textit{get-constant}\ (n, (\textit{fix-strategy}\ (C_{CF}, \textit{rule-list}))) = \textit{get-constant}\ (n\ C_{CF})$$

Simplified ACL2 Model of TLP_1



Verification

- Proof of termination of *fix-staregy1*
- Proof of the main conjecture

Proof of Termination

```
(defthm sum-addr-once-strategy-strictly-<
    (implies
     (and (well-formed-classfilep classfile)
          (some-matchp rule-list tail classfile))
     (< (sum-addr-to-resolve
                  (once-strategy rule-list
                                 tail
                                 classfile))
        (sum-addr-to-resolve classfile))))
```

Proof of The Main Conjecture

$\forall (C_{CF})$ *(get-constant n (fix-strategy* C_{CF}*)) = (get-constant n* C_{CF}*)))*

- Main conjecture in ACL2

```
(defthm get-constant-n-fix-strategy1
 (implies (well-formed-classfilep classfile)
     (equal (get-constant n
                   (fix-strategy1 rule-list classfile))
            (get-constant n classfile))))
```