IsaRARE: Automatic translation of term rewrite rules into Isabelle/HOL lemmas

User and Reference Manual

Hanna Lachnitt

March 14, 2024

Contents

1 Introduction **1**

2 Set-up and Quick Usage **2**

3 The RARE language **2**

3.0.1 Rare Rules 2

3.0.2 Fixed-point Rules 4

3.0.3 Changes to Rare 4

4 Components **5**

5 Reference Manual **5**

1 Introduction

IsaRARE is a plugin for Isabelle that transforms rewrite rules in the RARE language into Isabelle lemmas. It serves two main purposes:

1. Verification: Proving a lemma generated by IsaRARE indicates that the corresponding rule is sound.
2. Reconstruction: If rule is used in a proof certificate by an external solver, the generated lemmas can be used by the smt method during the reconstruction of that proof inside of Isabelle.

2 Set-up and Quick Usage

IsaRARE itself does not require any prerequisites but to execute the bit-vector examples in the Tests/ folder a copy of Finite Machine Word Library from the Archive of Formal Proofs (AFP) is needed [1]. We have tested our tool with the version of the AFP from October 16, 2023.

IsaRARE can be used simply by importing IsaRARE.thy:

theory *IsaRARE*
  imports *HOL−CVC.Smtlib-String HOL−CVC.SMT-CVC*

begin

3 The RARE language

The Rare language[1] was introduced by Nötzli et al. [2]. As part of this work, we have extended the language to be able to represent more rewrite rules. We present the full updated language here and summarize the differences with [2] at the end of the section.

A Rare file contains a list of rules whose syntax is defined by the grammar in Figure 1. Expressions use SMT-LIB syntax with a few exceptions. These include the use of approximate sorts for parameterized sorts (e.g., arrays and bit-vectors) and the addition of a few extra operators (e.g., `bvsize`, described below). Rare uses SMT-LIB 3 syntax [?], which is very close to SMT-LIB 2 and mostly differs from its predecessor in that it uses higher-order functions for indexed operators.

We say that an expression e *matches* a match expression m if there is some *matching substitution* σ that replaces each variable in m by a term of the same sort to obtain e (i.e., $m\sigma$ is syntactically identical to e). For example, the expression `(or (bvugt x1 x2) (= x2 x3))`, with variables `x1`, `x2`, `x3`, all of sort `?BitVec`, matches `(or (bvugt a b) (= b a))` but not `(or (bvugt a b) (= c a))`, with `a`, `b`, and `c` bit-vector constant symbols of the same bit-width.

3.0.1 Rare Rules

A Rare rewrite rule is defined with the `define-rule` command which starts with a parameter list containing variables with their sorts. These variables are used for matching as explained below. After an optional *definition list* (see below), there follow two expressions that form the main body of the rule: the *match* expression and the *target* expression. The semantics of a rule with match expression m and target expression t is that any expression

[1] Rare comes from Rewrites, Automatically REconstructed.

$$
\begin{array}{lll}
\langle \textit{rule} \rangle & ::= & \texttt{(}\ \textbf{\texttt{define-rule}}\ \langle \textit{symbol} \rangle\ \texttt{(}\ \langle \textit{par} \rangle^*\ \texttt{)}\ [\langle \textit{defs} \rangle]\ \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle\ \texttt{)} \\
& | & \texttt{(}\ \textbf{\texttt{define-rule*}}\ \langle \textit{symbol} \rangle\ \texttt{(}\ \langle \textit{par} \rangle^*\ \texttt{)}\ [\langle \textit{defs} \rangle]\ \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle\ [\langle \textit{expr} \rangle]\ \texttt{)} \\
& | & \texttt{(}\ \textbf{\texttt{define-cond-rule}}\ \langle \textit{symbol} \rangle\ \texttt{(}\ \langle \textit{par} \rangle^*\ \texttt{)}\ [\langle \textit{defs} \rangle]\ \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle\ \texttt{)} \\
\langle \textit{par} \rangle & ::= & \langle \textit{symbol} \rangle\ \langle \textit{sort} \rangle\ [\textbf{\texttt{:list}}] \\
\langle \textit{sort} \rangle & ::= & \langle \textit{symbol} \rangle\ |\ \texttt{?}\ |\ \texttt{?}\langle \textit{symbol} \rangle\ |\ \texttt{(}\ \langle \textit{symbol} \rangle\ \langle \textit{numeral} \rangle^+\ \texttt{)} \\
\langle \textit{expr} \rangle & ::= & \langle \textit{const} \rangle\ |\ \langle \textit{id} \rangle\ |\ \texttt{(}\ \langle \textit{id} \rangle\ \langle \textit{expr} \rangle^+\texttt{)} \\
\langle \textit{id} \rangle & ::= & \langle \textit{symbol} \rangle\ |\ \texttt{(}\ \langle \textit{symbol} \rangle\ \langle \textit{numeral} \rangle^+\ \texttt{)} \\
\langle \textit{binding} \rangle & ::= & \texttt{(}\ \langle \textit{symbol} \rangle\ \langle \textit{expr} \rangle\ \texttt{)} \\
\langle \textit{defs} \rangle & ::= & \texttt{(}\ \textbf{\texttt{def}}\ \langle \textit{binding} \rangle^+\ \texttt{)}
\end{array}
$$

Figure 1: Overview of the grammar of RARE.

e matching m under some sort-preserving matching substitution σ can be replaced by $t\sigma$. With approximate sorts, the sort preservation requirement is relaxed as follows. In RARE, for any sort constructor S of arity $n > 0$, there is a corresponding approximate sort (S `?` $\cdots$ `?`) with n occurrences of `?` which is always abbreviated as `?`S. A variable x with sort `?`S (e.g., `?BitVec`) in a match expression matches all expressions whose sort is constructed with S (e.g., `(BitVec 1)`, `(BitVec 2)`, and so on). Variables with sort `?` match expressions of any sort.

An optional *definition list* may appear in a RARE rule immediately after the parameter list. It starts with the keyword **`def`** and provides a list of local variables and their definitions, allowing the rewrite rule to be expressed more succinctly. A rule with a definition list is equivalent to the same rule without it, where each variable in the definition list has been replaced by its corresponding expression in the body of the rule. For a rule to be well-formed, all variables in the match and target expressions must appear either in the parameter list or the definition list. Similarly, each variable in the parameter list must appear in the match expression (while this second requirement could be relaxed, it is useful for catching mistakes). Consider the following example.

```
(define-rule bv-sign-extend-eliminate ((x ?BitVec) (n Int))
  (def (s (bvsize x)))
  (sign_extend n x)   (concat (repeat n (extract (- s 1) (- s 1)
    x)) x))
```

In this rule, there are two parameters, `x` and `n`. The sort annotation `?BitVec` indicates that `x` is a bit-vector without specifying its bit-width. The latter is stored in the local variable `s` using the `bvsize` operator. The rule says that a `(sign_extend n x)` expression can be replaced by repeating `n` times the most significant bit of `x` and then prepending this to `x`.

The **`define-cond-rule`** command is similar to **`define-rule`** except that it has an additional expression, the *condition*, immediately after the parameter and definition lists. This restricts the rule's applicability to cases where the condition can be proven equivalent to true under the matching substitution. In the example below, the condition `(> n 1)` can be verified by evaluation since in SMT-LIB, the first argument of `repeat` must be a numeral.

```
(define-cond-rule bv-repeat-eliminate-1 ((x ?BitVec) (n Int))
  (> n 1)   (repeat n x)   (concat x (repeat (- n 1) x)))
```

Note that the rule does not apply to terms like `(repeat 1` *t*`)` or `(repeat 0` *t*`)`.

3.0.2 Fixed-point Rules

The **`define-rule*`** command defines rules that should be applied repeatedly, to completion. This is useful, for instance, in writing rules that iterate over the arguments of n-ary operators. Its basic form, with a body containing just a match and target expression, defines a rule that, whenever is applied, must be applied again on the resulting term until it no longer applies.

The user can optionally supply a *context* to control the iteration. This is a third expression that must contain an underscore. The semantics is that the match expression rewrites to the context expression, with the underscore replaced by the target expression. Then the rule is applied again to the target expression only. In the example below, the **`:list`** modifier is used to represent an arbitrary number of arguments, including zero, of the same type.

```
(define-rule* bv-neg-add ((x ?BitVec) (y ?BitVec) (zs ?BitVec
    :list))
  (bvneg (bvadd x y zs))   (bvneg (bvadd y zs))   (bvadd (bvneg x)
    _))
```

This rule rewrites a term `(bvneg (bvadd` *s* *t* `⋯))` to the term `(bvadd (bvneg` *s*`)` *r*`)` where *r* is the result of recursively applying the rule to `(bvneg (bvadd` *t* `⋯))`.

3.0.3 Changes to Rare

Here, we briefly mention the changes to Rare with respect to [2]. First, we have support for a richer class of approximate sorts, including approximate bit-vector and array sorts. Also, we replaced the `let` construct by the new `def` construct. The definition list is more powerful as it applies to the entire rest of the body (whereas `let` was local to a single expression).

Additionally, to aid with bit-vector rewrite rules, we added several operators: `bvsize`, which returns the width of an expression of sort `?BitVec`; `bv`, which takes a integer n and natural w, and returns a bit-vector of width w and value n mod 2^w; `int.log2` which returns the integer (base 2) logarithm of an integer, and `int.islog2`, which returns true iff its integer argument is a power of 2.

We also removed the `:const` modifier, which was used previously to indicate that a particular expression had to be a constant value. We found that this adds complexity and is usually unnecessary. For rules that actually manipulate specific constant values, we can specify those values explicitly, e.g., by using the `bv` operator above.

4 Components

All components of IsaRARE can be found in src/. In the following, we briefly summarize the function of each component on a high level. Additional information can be found as source code comments in the respective file.

ML-file ‹*src/isarare-config.ML*›

This file provides diagnostics and options for IsaRARE. The options and their usage are described in Section **??**

ML-file ‹*src/abstract-type-parser.ML*›

This file contains parser functionality for the extensions to SMT-LIB RARE offers. Mostly, this concerns parsing abstract types (see Section 3).

ML-file ‹*src/parse-rare.ML*›

This file provides functionality to parse a RARE rule into a AST. This datatype is called rewrite_tree and is defined in this file.

5 Reference Manual

end

References

[1] J. Beeren, M. Fernandez, X. Gao, G. Klein, R. Kolanski, J. Lim, C. Lewis, D. Matichuk, and T. Sewell. Finite machine word library. *Archive of Formal Proofs*, June 2016. https://isa-afp.org/entries/Word_Lib.html, Formal proof development.

[2] A. Nötzli, H. Barbosa, A. Niemetz, M. Preiner, A. Reynolds, C. Barrett, and C. Tinelli. Reconstructing fine-grained proofs of rewrites using a domain-specific language. In *CONFERENCE ON FORMAL METHODS IN COMPUTER-AIDED DESIGN–FMCAD 2022*, page 65, 2022.