A Systematic Incrementalization Technique and its Application to Hardware Design

Steven D. Johnson*, Yanhong A. Liu**, and Yuchen Zhang

Indiana University Computer Science Department
sjohnson@cs.indiana.edu

Abstract. A transformation method based on *incrementalization* and value *caching*, generalizes a broad family of loop refinement techniques. This method and *CACHET*, an interactive tool supporting it, are presented. Though highly structured and automatable, better results are obtained with intelligent interaction, which provides insight and proofs involving term equality. Significant performance improvements are obtained in many representative program classes, including iterative schemes that characterize Today's hardware specifications. Incrementalization is illustrated by the derivation of a hardware-efficient nonrestoring square-root algorithm.

KEYWORDS AND PHRASES: Formal methods, hardware verification, design derivation, formal synthesis, transformational programming, floating point operations.

1 Introduction

Incrementalization [3, 5, 4] is a generalization of program refinement techniques such as *strength reduction*, in which partial results are introduced to optimize looping computations. *CACHET* [2, 6] is a prototype refinement tool developed to explore incrementalization strategies. In this paper [1], we look at its application to a representative problem in hardware specification. A *nonrestoring integer square root* algorithm was previously used by O'Leary, Leeser, Hickey, and Aagaard [7] to illustrate the use of a theorem prover the step-wise refinement of a hardware implementation. We applied CACHET to the same problem in order to compare how the critical insights needed to justify an implementation are discovered and applied under deductive and derivational modes of formal reasoning. In either case, the implementation proof depends on just a few algebraic identities which must be provided by the (presumably human) external tool user. One of the problems inherent to formalized reasoning is the often overwhelming logical context in which relatively simple key facts must be applied,

* Supported, in part, by the National Science Foundation under grant MIP-9601358.
** Supported, in part, by the National Science Foundation under grant CCR-9711253

which includes not only the complex formal proof and design objects, but also the strategy being followed to achieve the verification goal. In 1993, Windley, Leeser, and Aagard pointed out that numerous hardware verification case studies have been found to follow a common proof plan [8]. Incrementalization might also be seen as a "super duper" derivation tactic, but it is one that is applicable a class of generally recursive specification patterns, of which hardware is a special subclass.

2 Systematic Incrementalization

The incrementalization method is an interplay between two kinds of function extension (Figure 1). $F\colon W \rightarrow V$ plays the role of the specification being transformed; $\oplus\colon Y \times W \rightarrow W$ is some *state mutator*, a combination of elementary operations applied to F's argument The *incrementalization of F with respect to* $\oplus$ is a function F' that computes $F(w \oplus y)$ given the value of $F(w)$. The idea is this: given a specification for F by which computing $F(w \oplus y)$ involves a recursive call to $F(w)$, we want to specify how $F(w)$ is used in calculating the final result. *Caching* extends a function to return auxiliary results. $F\colon W \rightarrow V$ is extended to $\overline{F}\colon W \rightarrow V^k$, so that $\overline{F}(w) = \langle F(w), v_2, \ldots, v_k\rangle$. What we are really after is $\overline{F}'$, the incrementalization of the caching extension of F, in which cached values are exploited to optimize across recursive calls.

3 Application to *sqrt* [7]

Incrementalization applied to a singly tail-recursive function (i.e., *while-loop*) is known as *strength reduction.* Take $\oplus$ to be the "body" of the loop, so that, unless F terminates, incrementalizing F with respect to $\oplus$ is yields $F'(\oplus(x), F(x)) = F(\oplus(\oplus(x))$. Thus, incrementalization is tantamount to loop unrolling, and caching accumulates partial values for use across iterations.

$\oplus\colon Y \times W \rightarrow W$	*original*	*incrementalized*
original	$F\colon W \rightarrow V$	$F'\colon W \times Y \times V \rightarrow V$ $F'(w, y, F(w)) = F(w \oplus y)$
caching	$\overline{F}\colon W \rightarrow V^n$ $F(w) = v_1$ **where** $\langle v_1, v_2, \ldots\rangle = \overline{F}(w)$	$\overline{F}'\colon W \times Y \times V^n \rightarrow V^n$ $\overline{F}'(w, y, \overline{F}(w)) = \overline{F}(w \oplus y)$

Fig. 1. Components of incrementalization and identities relating cached, incrementalized, and cached-incrementalized variants of F.

We applied CACHET to the specification of *sqrt* used by O'Leary, et. al, to obtain the same implementation. The source and target expression are shown in statement form in Figure 2, left and right respectively. The *sqrt* algorithm is expressed in the form $F(n, m, i) = \oplus\langle n, m, i\rangle = \langle n, M(n, m, i), i - 1\rangle$, where M is the state mutator incrementalized in Figure 3. At five points in this CACHET derivation, judgment was exercised that we would regard as requiring insight. These judgments were of two forms, the application of an algebraic identity (' $\stackrel{!?}{=}$ ') or the invocation of an invariant assertion (' $\stackrel{!?}{\Longleftrightarrow}$ '). Facts (d) and (e) are used in *after* incrementalization as the result is incorporated and the surrounding algorithm is simplified.

$$
\begin{array}{ll}
(a) & n - (m \pm u)^2 \stackrel{!?}{=} n - m^2 \mp 2mu - u^2 \\
(b) & w' = (u')^2 = \left(\frac{1}{2}u\right)^2 \stackrel{!?}{=} \frac{1}{4}w \\
(c) & 2m'n' = 2(m+u)(\frac{1}{2}u) \stackrel{!?}{=} \frac{2}{2}mu + \frac{2}{2}u^2 = \frac{1}{2}v + w \\
(d) & i' \geq 0 \iff i \geq 1 \stackrel{!?}{\Longleftrightarrow} u \geq 2 \iff u^2 \geq 4 \iff w \geq 4 \\
(e) & i' = -1 \iff i = 1 \stackrel{!?}{\Longleftrightarrow} u = 1
\end{array}
$$

```
n, i, m := input, (l − 2), 2^(l−1);
while i ≥ 0 do
  p := n − m^2;
  if p > 0 then
    m := m + 2^i
  else if p < 0 then
    m := m − 2^i;
  i := i − 1;
output := m
```

```
p, v, w := input, 0, 2^(2(l−1));
while (w ≥ 1) do
  if p > 0 then
    p, v, w := p − v − w; v/2 + w, w/4
  else if p < 0 then
    p, v, w := p + v − w, v/2 − w, w/4
  else
    v, w := v/2, w/4;
output := v
```

Fig. 2. Specification and implementation of nonrestoring *sqrt*

References

1. Steven D. Johnson, Yanhong A. Liu, and Yuchen Zhang. A systematic incrementalization technique and its application to hardware design. Computer Science Department Technical Report 524, Indiana University, June 1999.
2. Yanhong A. Liu. CACHET: An interactive, incremental-attribution-based program transformation system for deriving incremental programs. In *Proceedings of the 10th Knowledge-Based Software Engineering Conference*, pages 19–26, Boston, Massachusetts, November 1995. IEEE CS Press, Los Alamitos, Calif.
3. Yanhong A. Liu. Principled strength reduction. In Richard Bird and Lambert Meertens, editors, *Algorithmic Languages and Calculi*, pages 357–381. Chapman & Hall, London, U.K., 1997.

4. Yanhong A. Liu, Scott D. Stoller, and Tim Teitelbaum. Static caching for incremental computation. *ACM Trans. Program. Lang. and Syst.*, 20(2):1–40, March 1998.
5. Yanhong A. Liu and Tim Teitelbaum. Systematic derivation of incremental programs. *Sci. Comput. Program.*, 24(1):1–39, February 1995.
6. Yanhong Annie Liu. *Incremental Computation: A Semantics-Based Systematic Transformational Approach.* PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, January 1996.
7. John O'Leary, Miriam Leeser, Jason Hickey, and Mark Aagaard. Non-restoring integer square root: A case study in design by principled optimization. In Ramayya Kumar and Thomas Kropf, editors, *Proceedings of the 2nd International Conference on Theorem Provers in Circuit Design: Theory, Practice, and Experience*, volume 901 of *Lecture Notes in Computer Science*, pages 52–71, Bad Herrenalb (Black Forest), Germany, September 1994. Springer-Verlag, Berlin.
8. Phillip Windley, Mark Aagard, and Miriam Leeser. Towards a super duper hardware tactic. In Jeffery J. Joyce and Carl Seger, editors, *Higher-Order Logic Theorem Proving and its Applications*, volume 780 of *Lecture Notes in Computer Science.* Springer-Verlag, August 1993.

$\oplus$		
	$M(n,m,i) \stackrel{\diamond}{=}$ **let** $p = n - m^2$ **in** **if** $p > 0$ **then** $m + 2^i$ **else if** $p < 0$ **then** $m - 2^i$ **else** m	
	$\overline{M}(n,m,i) \stackrel{\diamond}{=}$ **let** $p = n - m^2$ **in** **if** $p > 0$ **then** **let** $u = 2^i$ **in** $\langle m+u, p, u, 2mu, u^2\rangle$ **else if** $p < 0$ **then** **let** $u = 2^i$ **in** $\langle m-u, p, u, 2mu, u^2\rangle$ **else** $\langle m, 0, \text{-}, \text{-}, \text{-}\rangle$	$\overline{M}'(m,p,u,v,w) \stackrel{\diamond}{=}$ **if** $p > 0$ **then** **let** $p = p - v - w$ **in** **if** $p > 0$ **then** $\langle m+\frac{u}{2}, p, \frac{u}{2}, \frac{v}{2}+w, \frac{w}{4}\rangle$ **else if** $p < 0$ **then** $\langle m-\frac{u}{2}, p, \frac{u}{2}, \frac{v}{2}+w, \frac{w}{4}\rangle$ **else** $\langle m, 0, \frac{u}{2}, \frac{v}{2}+w, \frac{w}{4}\rangle$ **else if** $p < 0$ **then** **let** $p = p + v - w$ **in** **if** $p > 0$ **then** $\langle m+\frac{u}{2}, p, u, \frac{v}{2}+w, \frac{w}{4}\rangle$ **else if** $p < 0$ **then** $\langle m-\frac{u}{2}, p, u, \frac{v}{2}+w, \frac{w}{4}\rangle$ **else** $\langle m, 0, \frac{u}{2}, \frac{v}{2}+w, \frac{w}{4}\rangle$ **else** $\langle m, 0, \frac{u}{2}, \frac{v}{2}+w, \frac{w}{4}\rangle$

Fig. 3. Incrementalization of *sqrt* [7]