# CS 745 Spring 2001
# Project Proposal

Nick Hopper (hopper@cs)
Dan Rosewater (drose@cs)
Project Web page: `http://www.cs.cmu.edu/~hopper/745project/`

March 13, 2001

## 1 Project Description

We intend to implement and use the Static Single Assignment representation in MachSUIF. Because the conversion of CFG representation to and from SSA form may already have been implemented by Mike Smith's group at Harvard, our proposal is split into two cases: one in which we implement the SSA conversion ourselves, and one where conversion to and from SSA is done for us. We hope to find out within the next week whether or not the SSA form is available.

### 1.1 No pre-existing SSA implementation

We will implement the conversion to SSA form in MachSUIF using the algorithm outlined in [2]. After discussions with Seth, we may instead implement thinned gated single assignment (TGSA) form described in [3], although this still needs to be evaluated. TGSA is a richer representation than SSA; the location of the branching instruction that determines which argument of each $\phi$-function will be selected is also noted. Some feedback regarding this decision is desirable.

There are two options for implementing SSA using MachSUIF. First, we could attempt to implement SSA using mostly annotations. This involves annotating most operands with their version numbers and annotating blocks with a list of the $\phi$ functions at the beginning of each block. However previous attempts at using such a strategy have failed due to implementation snags. The other possibility introduces a new subclass of operand which are created as temporaries for each version of a variable, and a new $\phi$ instruction type. In this case traditional SSA passes should work as planned, but the difficulty here is in conversion back to CFG form; algorithms for this conversion are given in [2].

### 1.2 Pre-existing SSA implementation

We will implement partial redundancy elimination using SSA as described in [1]. The basic algorithm for performing sparse PRE (on a single expression) has six components that can be implemented in a semi-independent manner. Then an overall PRE pass can be implemented using a worklist of expressions. If time permits, the use of an expression tree as described in [1] will also be implemented.

## 2 Logistics

### 2.1 Plan of Attack and Schedule

Our schedule is (obviously) determined by the availability of a MachSUIF pass. We present two schedules corresponding to the cases:

### 2.1.1 Case 1 Plan

| Week | Task |
|---|---|
| 1 | Devise representation appropriate for MachSUIF and SSA. |
| 2 | Implement representation code (new classes or instruction types) |
| 3 | Implement SSA construction algorithm |
| 4 | Testing and debugging |
| 5 | Implement aggressive dead code elimination |
| 6 | Implement SSA to CFG conversion algorithm (if time allows) |

### 2.1.2 Case 2 Plan

| | |
|---|---|
| 1 | Learn the API to the MachSUIF SSA library, make preliminary design decisions. |
| 2 | Implement SSAPRE steps 1–3 |
| 3 | Implement SSAPRE steps 3–4 |
| 4 | Implement SSAPRE steps 5–6 |
| 5 | Implement SSAPRE pass |
| 6 | Test, Debug, Document, Performance Measurement? |

## 2.2 Milestone

For Case 1, our milestone will be a functioning algorithm to convert a CFG representation to SSA or TGSA. For Case 2, our milestone will be a functioning algorithm to perform sparse SSAPRE on a single expression.

## 2.3 Literature Search

See the references section for a list of publications we will use in directing our project.

## 2.4 Resources Needed

Other than those previously mentioned, none.

## 2.5 Getting Started

The first and most important element is to determine whether or not the MachSUIF SSA pass is available. Aside from that we've had a few discussions regarding the design decisions outlined in Section 1 and have briefly reviewed the papers cited in the bibliography.

# References


[1] F.C. Chow, S. Chan, R. Kennedy, S. Liu, R. Lo, and P. Tu. A new alogrithm for partial redundancy elimination based on ssa form. In *Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation (PLDI)*, pages 273–286, June 1997.

[2] R. Cytron, J. Ferrante, B. K. Roseen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Tran. on Programming Languages and Systems*, 1991.

[3] Paul Havlak. Construction of thinned gated single-assignment form. In *1993 Workshop on Languages and Compilers for Parallel Computing*, number 768, pages 477–499, Portland, Ore., 1993. Berlin: Springer Verlag.