Sal/Svm—A Language and Virtual Machine for Computing with Non-Enumerated Sets
Outline

- Background and Example
- Architecture
- Implementation
- Preliminary Evaluation
- Summary
Research Context and Background

- **Research context**: Hardware architectures for future device technologies
  - Are microprocessors the best way to take advantage of tradeoffs in <<20nm CMOS?

- **Research problem**
  - **Challenge**: Can we represent compute problems independent of algorithms for their solution?
  - **Observation**: Can represent problems using set-theoretic properties their solutions obey

- **This talk**
  - Platform that represents compute problems using set-theoretic constructs and machine state
Sets and Non-Enumerative Set Representations

- Most people are familiar with these:

\[ S_1 \cap S_2, \quad \text{etc.} \]

- Generally, people often think of sets as these:

\[
\begin{align*}
S_4 &= \{1, 3, 7, 9\} \\
S_5 &= \{(1, \text{“science”}), \ (2, \text{“history”})\},
\end{align*}
\]

- Two main ways of representing sets
  - Enumerated: items in set captured in the representation, e.g., \{1, 3, 5, ...\}
  - Non-enumerated: properties of the elements captured in representation: e.g. “odd integers”
Sal (Set Assembly Language) and Svm (Set VM)

Sal/Svm: A Language and Virtual Machine for Computing with Non-Enumerated Sets
- Application: precisely representing compute problem definitions without algorithms
- Problems specified in an input language (Sal) for describing set relations
- A virtual machine (Svm) processes the set relations, returning a solution (if requested / possible)

A simple example: set of odd integers between 0 and 1000 that are not multiples of 3:

1. Universe
   - The universe
   \[ U1 \colon \text{Integers} = \langle 0 \ldots 1000 \rangle \]

2. Set
   - The set
   \[ S1 = (((v:U1[1] \mod 3) \neq 0) \land ((v:U1[1] \mod 2) \neq 0)) : U1) \]

3. Boolean predicate

Svm Output
\[ S1 = \{1, 5, 7, 11, 13, 17, 19, \ldots \} \]
(cardinality = 333, predicate tree size = 11)
Outline

- Background and Example
- Architecture
- Implementation
- Preliminary Evaluation
- Summary
Representation of Sets in SVM

Set = Predicate Tree (root node shown here) + (Multi-Dimensional) Universe

- Universe
- ID
- Reference Count

Predicate Tree
- op
- Universe Index
- Dimension Index
- Constant Element
- Reference Count
- left
- right

(Multi-Dimensional) Universe
- cmp()
- prn()
- enumerated
- refcount
- values[]

References to these dimensions are often shared across universes

Additional universe-specific state
Svm virtual machine has three sets of registers that underlie all computation

- **U** (universe) **registers**: hold the “basis” elements (scalar)
- **P** (predicate) **registers**: hold Boolean predicate trees (data structures)
- **S** (set) **registers**: hold information about specific pairings of predicates to universes

---

**The universe**

\[ U_{1} : \text{Integers} = <0 \ldots 1000> \]

**The set**

\[ S_{1} = (((v:U_{1}[1] \% 3) \neq 0) \& ((v:U_{1}[1] \% 2) \neq 0) : U_{1}) \]

**Set's predicate expression**

\[ S_{a} = (x : U_{1}[1] \neq 1) : U_{1}. \]

**Set's universe / type**

**free variable's type**

**free variable's type**

---

**P Registers**

- P0
  - \( = \)
  - \( \% \)
  - \( ... \)
  - \( 2 \)
  - \( v \)

**S Registers**

- **P0 : U1**
- **true : U2**
- **U1**

**U Registers**

- \(<1, 5, \ldots, 94>\)
- \(<0 \ldots 1000>\)
- \({2.1, \ldots, -1.3}\)
Larger Example: Convex Hull Compute Problem Definition

Definition (Convex Hull). The convex hull, $\text{CH}(S)$, of a set $S$ of points in the plane, is the smallest convex polygon for which each point in $S$ is either on the boundary thereof, or in its interior.

Convex hull computational problem specification in Sal

U0 : Integers = <1 ... 10 delta 2*iota>
U1 : Integers = <1 ... 10 delta (2*iota)+1>
U2 = U0 x U1

P10 = !((qy == py:U2[2]) & (qx == px:U2[1]))
P11 = ((qx*ry - qy*rx) - px:U2[1]*(ry - qy) + py:U2[2]*(rx - qx)) >= 0
S2 = (P1 : U2)
print enum S2
Boolean Predicate Trees

Sal predicate expression

\[ P_{10} = \neg((q_y == p_y:U2[2]) \land (q_x == p_x:U2[1])) \]

Predicate Tree

Compilation

Real example:

(Predicate tree in register \( P_{10} \) from convex hull example; this figure was auto-generated by runtime system)

Generated by Svm version 0.3-alpha (build 08-03-2010-20:08:47-pip@listener.local-Darwin), on Sun Aug 8 18:30:39 2010.
Outline

- Background and Example
- Architecture
- Implementation
- Preliminary Evaluation
- Summary
Implementation

- **Svm**
  - Core of runtime system implemented in a library, in ANSI C
  - Library also provides utilities such as rendering machine state (example in previous slide), etc.

- **Salc**
  - Implemented in C, with YACC-driven parser front-end
  - Takes Sal source assembler and outputs intermediate representation in an ELF container
  - A few optimizations implemented, potential for more

- **Interactive command console**
  - Accepts Sal statements, parses and injects into machine state
  - Interactive help system, commands to probe Svm state, etc.

- **Web interface to interactive console on server**
  - Runs interactive command console via web, on server
Internal Demo Site Examples

- Interactive interface is embedded in a content-management system
  - Contains examples that can be modified and run via web; users can post comments, questions

- A collection of example problems
Outline

- Background and Example
- Architecture
- Implementation
- Preliminary Evaluation
- Summary
Preliminary Performance Evaluation: Convex Hull Problem

Recall: given set of points in plane, compute “points on the hull” (black points below)

- Evaluation setup
  - All cases: compiler: gcc 4.2.1, -O3; CPU: 2.8GHz Intel® Core™ i7; OS: MacOS® 10.6.3

- Svm is within ~20× of optimized C code implementing same enumeration algorithm
  - Sal problem specification is significantly simpler (~11 lines) than C algorithm implementation

- Orders of magnitude slower than (provably) optimum algorithm (Graham’s Scan)
  - Paper provides a counterexample where Sal/Svm and C search can be more energy-efficient
Outline

- Background and Example
- Architecture
- Implementation
- Preliminary Evaluation
- Summary
Summary and Ongoing Work

- **Summary:** Research context
  - Architectures for future devices and device technologies

- **Summary:** Sal/Svm
  - A framework for specifying computational problems using set-theoretic constructs
  - Enables leaving algorithm choice undefined, to pick best hardware-dependent tradeoff

- **Ongoing:** Integrating Sal-based problem definitions in high-level languages
  - Ongoing work to use Sal problem definition as part of type structure in a high-level language
  - Specification of “arithmetic imprecision” in computation problems

- **Availability**
  - Currently available to users inside IBM Research
  - Working to make interactive web version available to general public in coming months