# Lab Week 9—Efficient Records in rePL

CS 4215: Programming Language Implementation

Martin Henz

March 13, 2012

Generated on Thursday 15 March, 2012, 22:43

# New Instructions for Record Construction

New instructions:

$$\frac{s}{\texttt{LDPS}\ q.s} \qquad \frac{s}{\texttt{RCDS}\ i.s}$$

where $q$ is a property and $i$ is an integer.

# Compilation of Record Construction

$$\frac{E_1 \hookrightarrow s_1 \qquad \cdots \qquad E_n \hookrightarrow s_n}{[q_1 : E_1 , \ldots , q_n : E_n] \hookrightarrow \mathrm{LDPS}\ q_1.s_1.\ldots.\mathrm{LDPS}\ q_n.s_n.\mathrm{RCDS}\ n}$$

# Execution of LDPS

$$\frac{s(pc) = \texttt{LDPS}\ q}{(os, pc, e, rs) \rightrightarrows_s (q.os, pc + 1, e, rs)}$$

# Execution of RCDS

Consider LDPS $q_1.s_1.\ldots.$LDPS $q_n.s_n.$RCDS $n$ resulting from
$[q_1 : E_1, \ldots, q_n : E_n]$.
After LDPS $q_1.s_1.\ldots.$LDPS $q_n.s_n$, instruction RCDS $n$ finds association list on operand stack.

$$\frac{s(pc) = \texttt{RCDS}\ n}{(v_n.q_n.\ldots.v_1.q_1.os, pc, e, rs) \rightrightarrows_s (\{(q_1, v_1), \ldots, (q_n, v_n)\}.os, pc + 1, e, rs)}$$

# Operations on Records

Operations: `empty`, ".", `hasproperty`

New instructions:

$$\frac{s}{\mathtt{EMPTY}.s} \qquad \frac{s}{\mathtt{DOT}.s} \qquad \frac{s}{\mathtt{HASP}.s}$$

# Compilation of Record Operations

$$\frac{E \hookrightarrow s}{\texttt{empty}\ E \hookrightarrow s.\texttt{EMPTY}}$$

$$\frac{E \hookrightarrow s}{E\ \texttt{.}\ q \hookrightarrow s.\texttt{LDPS}\ q.\texttt{DOT}}$$

$$\frac{E \hookrightarrow s}{E\ \texttt{hasproperty}\ q \hookrightarrow s.\texttt{LDPS}\ q.\texttt{HASP}}$$

# Executions of Record Operations

$$\frac{s(pc) = \texttt{EMPTY}}{(v.os, pc, e, rs) \rightrightarrows_s (true.os, pc+1, e, rs)} \text{ if } v = \emptyset$$

$$\frac{s(pc) = \texttt{EMPTY}}{(v.os, pc, e, rs) \rightrightarrows_s (false.os, pc+1, e, rs)} \text{ if } v \neq \emptyset$$

# Executions of Record Operations

$$\frac{s(pc) = \texttt{DOT}}{(q.v.os, pc, e, rs) \rightrightarrows_s (v'.os, pc+1, e, rs)} \text{ if } v(q) = v'$$

$$\frac{s(pc) = \texttt{HASP}}{(q.v.os, pc, e, rs) \rightrightarrows_s (true.os, pc+1, e, rs)} \text{ if } \exists v_i.(q, v_i) \in v$$

$$\frac{s(pc) = \texttt{HASP}}{(q.v.os, pc, e, rs) \rightrightarrows_s (false.os, pc+1, e, rs)} \text{ if } \nexists v_i.(q, v_i) \in v$$

# Built-in Exceptions

Division by zero and record access throw exceptions.
Idea: place instructions for raising these two exceptions at the end of the instruction sequence.

$$\frac{E \hookrightarrow s_1 \qquad \texttt{[divisionByZero:true]} \hookrightarrow s_2 \qquad \texttt{[invalidRecordAccess:true]} \hookrightarrow s_3}{E s_1.\texttt{DONE}.s_2.\texttt{THROW}.s_3.\texttt{THROW}}$$

beginning address of $s_2$: $addr_{divisionByZero}$
beginning address of $s_3$: $addr_{invalidRecordAccess}$

# Primitive Operations Throwing Exceptions

$$\frac{s(pc) = \texttt{DIV}}{(0.i_1.os, pc, e, rs) \rightrightarrows_s (os, addr_{divisionByZero}, e, rs)}$$

$$\frac{s(pc) = \texttt{DOT}}{(q.v.os, pc, e, rs) \rightrightarrows_s (os, addr_{invalidRecordAccess}, e, rs)}$$

if $\nexists v.(q, v') \in v$

# Programmer-defined Exception Throws

$$\frac{E \hookrightarrow s}{\texttt{throw}\ E\ \texttt{end} \ \hookrightarrow\ s.\texttt{THROW}}$$

# Use Runtime Stack for Catching Exceptions

- Use runtime stack to keep track of the `catch...with...` part of `try` expressions
- Exception from `try` part will pop stackframes, until it finds the appropriate `catch...with...` part

# Translation of `try` Statement

$$\frac{E_1 \hookrightarrow s_1 \qquad E_2 \hookrightarrow s_2}{\begin{array}{c}\texttt{try}\ E_1\ \texttt{catch}\ x\ \texttt{with}\ E_2\ \texttt{end} \\ \hookrightarrow \\ (\texttt{TRY}\ x\ |s_1| + 3).s_1.\texttt{ENDTRY}.(\texttt{GOTOR}\ |s_2| + 1).s_2\end{array}}$$

# Execution of TRY Instruction

$$\frac{s(pc) = \texttt{TRY}\ x\ i}{(os, pc, e, rs) \rightrightarrows_s (os, pc + 1, e, (\texttt{catch}, x, pc + i, os, e).rs)}$$

# Execution of ENDTRY Instruction

$$\frac{s(pc) = \texttt{ENDTRY}}{(os, pc, e, (\texttt{catch}, x, pc', os, e).rs) \rightrightarrows_s (os, pc + 1, e, rs)}$$

# Throwing of an Exception

$$\frac{s(pc) = \texttt{THROW}}{(os, pc, e, (pc', os', e').rs) \rightrightarrows_s (os, pc, e, rs)}$$

$$\frac{s(pc) = \texttt{THROW}}{(v.os, pc, e, (\texttt{catch}, x, pc', os', e').rs) \rightrightarrows_s (os', pc', e'[x \leftarrow v], rs)}$$

# Problems with Records in RVM

- representation of records as a set of pairs is inefficient
- properties are strings

## Observations

- Properties always appear literally
- Records are always constructed with `[...]`, which explicitly lists all properties

# Representing Properties

- compiler constructs set $Q$ of all properties in a given $E$
- compiler calculates a bijection *idp* between $Q$ and $[0 \ldots |Q| - 1]$
- compiler replaces every occurrence of a property $q$ in an instruction by $idp(q)$

# Compilation of Record Operations (revisited)

$$\frac{E \hookrightarrow s}{E\ .\ q \hookrightarrow s.\texttt{LDCI}\ \mathit{idp}(q).\texttt{DOT}}$$

$$\frac{E \hookrightarrow s}{E\ \texttt{hasproperty}\ q \hookrightarrow s.\texttt{LDCI}\ \mathit{idp}(q).\texttt{HASP}}$$

# Record Construction

- All records are constructed by `[...]`
- compiler calculates a bijection *idr* between the set $R$ of all property sets of records and $[0 \ldots |R| - 1]$.
- associate with each property $q$ of each record its alphabetical position in the corresponding property set: $p(idr(\{q_1, \ldots, q_n\}), idp(q))$, starting with 0. If a record with index $m$ does not have a property with index $n$, we set $p(m, n) = -1$.

## Example

- Let us say compiler assigns the number 13 to the set of properties $\{\texttt{a}, \texttt{b}\}$ (thus $idr(\{\texttt{a}, \texttt{b}\} = 13)$
- the number 55 to the property `a`
  $idp(\texttt{a}) = 55$
- the number 77 to the property `b`
  $idp(\texttt{b}) = 77$
- the position of property `a` in `[a:5 b:7]` is $p(13, 55) = 0$
- For any property identifier $n \neq 55, 77$: $p(13, n) = -1$.

# Representing Records

Represent a record with properties $q_1, \ldots, q_n$ as a pair consisting of identifier $idr(\{q_1, \ldots, q_n\})$ and array that maps the alphabetical position of each $q$ in the corresponding property list.

# Example

In the example above, since $p(13, 55) = 0$ and $p(13, 77) = 1$, we can represent the record `[a:5 b:7]` by the pair $(13, [0 : 5, 1 : 7])$.

# New Translation of Record Construction

$$\frac{E_1 \hookrightarrow s_1 \qquad \cdots \qquad E_n \hookrightarrow s_n}{\begin{array}{c}[q_1 : E_1 \,, \ldots \,, q_n : E_n] \\ \hookrightarrow \\ \texttt{LDCI}\ \mathit{idp}(q_1).s_1.\ldots.\texttt{LDCI}\ \mathit{idp}(q_n).s_n.\texttt{RCD}\ n\ \mathit{idr}(\{q_1, \ldots, q_n\})\end{array}}$$

# Efficient Records in RVM

- compiler passes the table $p$ to RVM
- `RCD` constructs an array, whose indices corresponding to the record properties are given by $p$.

# New Execution of Record Construction

$$\frac{s(pc) = \mathtt{RCD}\ n\ m}{(v_n.i_n.\ldots.v_1.i_1.os, pc, e, rs) \Rrightarrow_s ((m, \{(p(m, idp(q_1)), v_1), \ldots, (p(m, idp(q_n)), v_n)\}).os, pc + 1, e, rs)}$$

# New Execution of Record Operations

$$\frac{s(pc) = \texttt{DOT}}{(i.(m,a).os, pc, e, rs) \rightrightarrows_s (a(j).os, pc+1, e, rs)} \text{ if}$$

$$p(m,i) = j, j \geq 0$$

$$\frac{s(pc) = \texttt{DOT}}{(i.(m,a).os, pc, e, rs) \rightrightarrows_s (os, addr_{invalidRecordAccess}, e, rs)} \text{ if}$$

$$p(m,i) = -1$$

# New Execution of Record Operations

$$\frac{s(pc) = \mathtt{HASP}}{(i.(m,a).os, pc, e, rs) \rightrightarrows_s (true.os, pc+1, e, rs)} \text{ if } p(m,i) \geq 0$$

$$\frac{s(pc) = \mathtt{HASP}}{(i.(m,a).os, pc, e, rs) \rightrightarrows_s (false.os, pc+1, e, rs)} \text{ if}$$

$p(m,i) = -1$

# Summary of Record Implementation

Constant time record access achieved by:

- representing properties by integers using *idp*
- mapping record property sets to integers using *idr*
- record access through arrays using lookup table *p*

# Overview of Next Lecture

- Imperative Programming: The language imPL