Lightweight Records for SAC

Clemens Grelck

University of Amsterdam
Institute for Informatics
Computer Systems Architecture Group

7th SAC Developer’s Conference (DevCon-7)
Amsterdam, May 27, 2010
Why Records?

Isn’t SAC an array language?

- Complex numbers
- RGB values
- Sparse matrices
- Stacks and queues

More generally:
You do want to bundle heterogeneous data!!
Why Records?

Isn’t SAC an array language?

- Complex numbers
- RGB values
- Sparse matrices
- Stacks and queues

More generally:

You do want to bundle heterogeneous data!!
Lightweight Records ?

SAC is an array language indeed!

- Achieve with limited implementation effort
- Do not change the type system
- Do not re-implement type system on records either
- Do not change code generation
Lightweight Records?

SAC is an array language indeed!

- Achieve with limited implementation effort
- Do not change the type system
- Do not re-implement type system on records either
- Do not change code generation

Distinguish between:

- non-recursive records
- recursive records
One more requirement:

- Achieve with limited implementation effort
- Do not change the type system
- Do not re-implement type system on records either
- Do not change code generation
- Keep syntax reasonably close to C
Record Type Declarations

One more requirement:

- Achieve with limited implementation effort
- Do not change the type system
- Do not re-implementation type system on records either
- Do not change code generation
- Keep syntax reasonably close to C

Who is who?

- Concept: me 2008
Record Type Declarations

Using the familiar struct notation:

```c
struct s1 {
    int e1, e2;
    double e3;
};

struct s2 {
    char x1;
    struct s1 x2;
};
```
Using the familiar dot notation:

```c
struct s1 {
  int e1, e2;
  double e3;
};

struct s1 foo( struct s1 arg )
{
  arg.e1 = 2 * arg.e2;

  return( arg );
}
```
The C way?

```c
struct s1 {
    int e1, e2;
    double e3;
};

struct s1 foo(int x, int y, double z) {
    struct s1 res;

    res.e1 = x;
    res.e2 = y;
    res.e3 = z;

    return res;
}
```
Record Construction

The C way?

```c
struct s1 {
    int e1, e2;
    double e3;
};

struct s1 foo(int x, int y, double z) {
    struct s1 res;
    res.e1 = x;
    res.e2 = y;
    res.e3 = z;
    return(res);
}

NO!!
```
Record Construction

The functional way!

```c
struct s1 {
    int e1, e2;
    double e3;
};

struct s1 foo( int x, int y, double z) {
    res = s1( x, z, z);
    return( res);
}
```
How do we drag records through the compiler?

Scanner / Parser

Functionalisation

Type Inference
Type Specialisation

High-Level Optimisations

Memory Management

De-Functionalisation

Parallelisation

Code Generation

Function Inlining
Array Elimination
Dead Code Removal
Common Subexpression Elimination
Constant Propagation
Constant Folding
Copy Propagation
Algebraic Simplification
Loop Unrolling
Memory Reuse
Loop Invariant Removal
With-Loop Unrolling
With-Loop Invariant Removal
With-Loop Folding
With-Loop Scalarisation
With-Loop Fusion
Automatic Array Padding
Index Vector Elimination

Clemens Grelck, University of Amsterdam

Lightweight Records for SAC
Record Type Declarations

```c
// type definitions
typedef hidden _struct_s1;

typedef double _struct_elem_s1_e3;
typedef int _struct_elem_s1_e1;
typedef int _struct_elem_s1_e2;

// constructor functions
_struct_s1 s1(_struct_s1 s);
_struct_s1 s1(int e1, int e2, double e3);

// getter functions
int _struct_get_e1(_struct_s1 s);
int _struct_get_e2(_struct_s1 s);
double _struct_get_e3(_struct_s1 s);

// setter functions
_struct_s1 _struct_set_e1(int e, _struct_s1 s);
_struct_s1 _struct_set_e2(int e, _struct_s1 s);
_struct_s1 _struct_set_e3(double e, _struct_s1 s);
```

Clemens Grelck, University of Amsterdam
Lightweight Records for SAC
Record Type Declarations

```c
struct s1 foo( struct s1 arg )
{
    arg.e1 = 2 * arg.e2;

    return( arg );
}

_struct_s1 foo( _struct_s1 arg )
{
    tmp = 2 * _struct_get_e2( arg );
    arg = _struct_set_e1( tmp, arg );

    return( arg );
}
```
Now the Type System Strikes

The typechecker does all the consistency legwork!!
Even cooler: with no clue about records

▶ Non-existant record constructors
▶ Non-existant record field labels
▶ Type consistency across records
▶ Multiply used record field labels resolved by overloading
▶ ....
Now the Type System Strikes

The typechecker does all the consistency legwork!!
Even cooler: with no clue about records

- Non-existant record constructors
- Non-existant record field labels
- Type consistency across records
- Multiply used record field labels resolved by overloading
- ....

Tagging of record constructors/setters/getters allows for customised error message.
Record Resolution

// constructor functions
_struct_s1 s1( int e1, int e2, double e3);

// constructor functions
inline
_struct_elem_s1_e1, _struct_elem_s1_e2, _struct_elem_s1_e3
s1( int e1, int e2, double e3)
{
    return( e1, e2, e3);
}
// getter functions
int _struct_get_e1( _struct_s1 s);

// getter functions
inline
int _struct_get_e1( _struct_elem_s1_e1 _s_e1,
                   _struct_elem_s1_e2 _s_e2,
                   _struct_elem_s1_e3 _s_e3)
{
    return(_s_e1);
}
Record Resolution

// setter functions
_struct_s1 _struct_set_e1( int e, _struct_s1 s);

// setter functions
inline
_struct_elem_s1_e1, _struct_elem_s1_e2, _struct_elem_s1_e3
_struct_set_e1( int e, _struct_elem_s1_e1 _s_e1,
      _struct_elem_s1_e2 _s_e2,
      _struct_elem_s1_e3 _s_e3)
{
  return( e, _s_e2, _s_e3);
}
State of the Art

- Scanner, Parser: done!
- Record function generation: done!
- Record resolution: done!

Does it work now? No :-(
State of the Art

- Scanner, Parser: done!
- Record function generation: done!
- Record resolution: done!

Does it work now? No :-(

Clemens Grelck, University of Amsterdam
Where Do We Go From Here?

- SAC data types
- Recursive records