Migration to a Dynamic Typing System

Design Goals

Moto needs to be able to recognize and add new ‘dynamically defined’ types on the fly in order to implement higher order functions. Dynamically defined types are those types which are automatically valid whenever the types they depend upon are valid. An example of this is arrays. Given a type X, types X[], X[][], X[][][] are all valid types and should be added to the type system as needed.

Design Overview

The idea will be to actually add the dynamically defined types as real MotoTypes to the type system the first time they are used. Thus the getType() function must have the side effect of adding dynamic types behind the scenes.

Dynamically defined types should have canonical names so, like all other types, we will have no more than one MotoType with the same name in the system at any one time. This is an important performance benefit we want to retain because with it comparing types for equality is simply a pointer comparison.

Classes of Dynamically Defined Types

There are two main classes of DDTs we know of right now. Array types and HOF types

For array types the rule is:

if X is a type than X[] is a type

For HOF types the rules will be:

if X is a type than X() is a type
if X and Y are types than X(Y) is a type
if X is a type and Z is a comma delimited list of types than X(Z) is a type

This will allow for types of the form:

```
int ()
int (String)
int (String,char)

int () ()
int (String) ()
int () (String)
int (String) (char)
int (String,char) ()
int (String,char) (float)

int (String () )
int (String (char) )
```

Type Ancestry and Canonicalization

Its clear that these dynamic types will have one or more ancestor types. There are multiple ways to encode this ancestry even for arrays, for instance we could have the following inheritance path for 3 dimensional arrays of integers:

int -> int[] -> int[][] -> int[][][]

In the interest of being able to rapidly find the base type of arrays easily the implementation should be:

int -> int[]
int -> int[][]
int -> int[][][]

This way the immediate ancestor is always the base type. The mototype can store the dimension of the array.

For HOFs the situation is necessarily more complex. Here the type must somehow contain the return type AND the argument types. For storage simplicity we will use the ancestor type to store the return type and add a separate array for storage of the argument types

int -> int()
int -> int(String,int)
int -> int[][] -> int[][]()
int -> int() -> int()()
int -> int() -> int()[] -> int()[](String)

<u>Environment Changes</u>

The MotoType structure will need to be greatly enhanced to store both ancestry relations and argument types for HOF DDTs.

```
/* A Moto Type */
typedef struct motoType {
   TypeKind kind;
   char *name;
   MotoType *atype;
   int dimension;
   int argc;
   MotoType *argt;
   char isExternallyDefined;
} MotoType;
```

MotoVals (RefVals specifically) and MotoVars will have the subtype and dimension fields removed. This may pave the way for Val / Var unification at some point in the future.

We should take this opportunity to factor MotoType and MotoType related functions out of env.c/h and into mototype.c/h

We will want ways of getting / adding dynamic types:

```
MotoType *
moto_addArrayType(MotoEnv *env, char *name, int dim) {
      MotoType *type = NULL, *atype = NULL;
      int i;


      /* Make sure this array type doesn't already exist, if it does don't add
it, just return it */
      /* Retrieve the base type for the new array type, if it doesn't exist we
have a problem */
      /* If the type name passed is the name of another array then get the
true base type and add the dimensions of this type to those passed in */
      /* Allocate the storage for our new type */
      /* Construct the canonical name for this new type */
      /* Set the ancestor type to the base type this array type stores. Set
the dimension
      /* This is not a HOF type so zero out argc and argt */
}

MotoType *
moto_getArrayType(MotoEnv *env, char *name, int dim) {
      char typen[256];
      MotoType *type;
      int i;

      /* Construct the canonical name for the array type */

      /* Retrieve and return the array type */
}
```

And modify the existing ways to differentiate between base types and dynamic types:

moto_addType and moto_addBuiltInType should be modified to contain:

```
      /* This is a base type so set its atype, argc, argt, and
      dimension to 0 */

      type->dim = 0;
      type->atype = NULL;
      type->argc = 0;
      type->argt = NULL;
```

refval.subtype should become type->atype
refval.dim should become type->dim
moto_getType(X,Y) should become moto_getType(X,Y,0)

We will also want to get rid of the built in “Array” type ... that was a bad idea from the start

The isArray(MotoVal) function should be changed to

```
isArray(MotoVal *val) {
  return (val->type->dim > 0 && val->type->atype != NULL);
}
```

The motoTypeToCType function should be changed to output UnArray* when type->dim > 0

We can clean up the plethora of different calls to

```
moto_varTypeAssignError(lval->type->name, rval->type->name);
moto_varTypeReturnError(lval->type->name, rval->type->name);
```

Mototypes are about to become much much much more complex beasts. As such just passing dim to functions like createVal and createVar isn't going to cut it anymore. We should actually get the type ourselves and pass that to createVal and createVar

```
int moto_checkImplicitCast(MotoEnv *env, char *fromtype, char *totype,int fromdim,int todim);

MotoVal *moto_createVal(MotoEnv *env, char *typen,int dim);
MotoVar *moto_createVar(MotoEnv *env, char* varn, char *typen, int dim,char isMemberVar,void *address);
MotoVar *moto_declareVar(MotoEnv *env, char* varn, char *typen, int dimension, char isGlobal);
```

Changes to MCDs and MotoFunctions

Motod currently plays two important roles. First off it verifies that no function, method, or Class is defined more than once. Second it adds all the 'base types' the type system needs for use in later stages. This is critical because in moto types can be used prior to definition.

To allow for this moto function arguments and class variable types were only *loosly coupled* to the MotoType system. Types were just strings that may or may not have definitions anywhere. Its up to motov to verify that MotoDefined classes, functions, and methods don't depend on anything that is 'undefined'

Unfortunately function arguments and class methods were only base types. Dimension information was, unfortunately, stored elsewhere. Moreover with the addition of dynamic HOF types the simple types that are used in MotoFunctions and Classes require some major enhancements.

As was stated earlier the reason for loose coupling is so that types can be used prior to definition. The solution to dynamic type storage in these objects is to make the *string canonical form* of a type what gets stored and have functions handy for

1) Constructing the string canonical from of a type from a TYPE or DEF union cell
2) Retrieving a true MotoType for one in string canonical form.

motod_nameForTypeCell(UnionCell* type_or_def_cell) - Returns what the canonical name for the parsed type would be assuming all the base types actually existed.

mttab_typeForName(MotoTypeTable *table, char *typen) - Returns the canonical MotoType for the canonical type name (or null if it is not definable because base types are missing).

There are many many fringe benefits to this design:

1) This will have the immediate effect of removing all those nasty 'dimension' arrays and hashtables from MCDs and moto functions.

2)

Dynamic Loader Changes
- **mfn_createFromSymbol(char* sym, char* libname, void* handle)** - This method needs to be modified to construct the canonical form of array typed return values and array typed arguments. Perhaps in the future this change should be moved to mxc

Environment Changes
- **void moto_delete(MotoEnv *env, MotoVal *val) -** This method no longer needs to pass a dimensions array to ftab_get
- **moto_emitCStructures(MotoEnv *env, StringBuffer *out)** - We should just be passing 0 for the dimension to moto_createVar since the dimension information is now stored in the member variable type
- **void moto_emitCImplicitConstructors(MotoEnv *env, StringBuffer *out)** - We should just be passing 0 for the dimension to moto_createVar since the dimension information is now stored in the member variable type
- **char* moto_fnToCName(MotoEnv* env,char* classn, char* fname, int argc,char** argtypes)** -arg dimensions are no longer needed since they are built into the arg types. They do have to be derived from the argtype strings tho
- **char* moto_fnToCPrototype(MotoEnv* env, MotoFunction* f)** - Just pass 0 for the dimension field in createVal since the dimensions are built into the type name

FTable Changes
- **int ftab_cacheGet(FTable *table,MotoFunction **f,char *name,int argc, char **types)** - Don't need the dimensions array anymore, that information is now in the types array
- **char ftab_argsMatchExactly(int argc, char **candidateTypes, char **argTypes) -** Dimensions arrays are no longer necessary, this function will just strcmp the candidate types against the argtypes
- **char ftab_argsCanBeCast(int argc, char **candidateTypes, char **argTypes)** - Dimensions arrays are no longer necessary, this function will just check implicit casts between the candidate types against the argtypes
- **int ftab_getExactMatch(FTable *table, MotoFunction **f, char *name, int argc, char **types)** - Dimensions arrays are no longer necessary. Need to fix calls to other ftab functions so as not to pass it.
-**int ftab_get(FTable *table, MotoFunction **f, char *name, int argc, char **types)** - Dimensions arrays are no longer necessary. Need to fix calls to other ftab functions so as not to pass it

Moto Class Definition Changes
- **MotoClassDefinition* mcd_create(char* classn, const UnionCell* p)** - Don't need to create the member dimensions table anymore
- **void mcd_free(MotoClassDefinition* mcd)** - Don't need to free the member dimensions table anymore
- **void mcd_addMember(MotoClassDefinition* mcd,char* name, char* typen) -** Don't need to pass member dimension anymore
- **void mcd_computeOffsetsAndSize(MotoClassDefinition* mcd)** - Need to check that the type is not basic when computing offsets
- Remove **mcd_getMemberDimension** - don't need this anymore :)

Moto Compiler Changes
- **void motoc_define(const UnionCell *p) -**

    1) Get rid of the arg dimensions array ... its no longer needed

2) Be sure to use motoc_extractType to get at the argument types
3) When declaring member vars pass 0 for the dimension to moto_declareVar
4) Remove the dimension array from the call to ftab_cacheGet
5) When creating the return type pass 0 for the dimension to createVal

- **void motoc_dereference_rval(const UnionCell *p)** - Pass 0 for the dimension to createVal since dimension information is encoded in the member vars type
- **void motoc_fn(const UnionCell *p)** -
    1) Remove the dimensions array ... its no longer needed
    2) Remove the 'special handling' for array type conversions ... no longer needed
    3) Fix all the calls to ftab_cacheGet
    4) Pass 0 for dimension to createVal since dimension is encoded in f->rtype
- **void motoc_new(const UnionCell *p)** -
    1) Remove the dimensions array ... its no longer needed
    2) Remove the 'special handling' for array type conversions ... no longer needed
    3) Fix all the calls to ftab_cacheGet

<u>Motod Changes</u>
- **char* motod_nameForTypeCell(const UnionCell *p)**

```
    1) if opcode is TYPE
        1.1) construct basetype + "[]"(dim times)
    2) if opcode is DEF
        1.1) call motod_name for type on the return cell and put the return value in a buffer
        1.2) add '(' to the buffer
        1.3) for each argument cell
            1.3.1) if this is not the first argument add ',' to the buffer
            1.3.2) call motod_nameForTypeCell on the argument cell and att the return to the buffer
        1.4) add ')' to the buffer
        1.5) return the buffer
```

- **void motod_declare(const UnionCell *p) -** call motod_nameForTypeCell to get the string type name. Get rid of all the add member dimension stuff
- **void motod_define(const UnionCell *p) -** call motod_nameForTypeCell to get the string type names for return type and argument types. Get rid of all the dimension stuff

<u>Moto Function Changes</u>
- **MotoFunction* mfn_create(char* motoname,char* rtype, int argc,char** argtypes,char** argnames,char* fname, char* cname, char tracked,const UnionCell* opcell, char* libname,void* handle,void (*fptr)()) -** Get rid of the arg dimensions and rtype dimension.

<u>Moto Interpreter Changes</u>
- **static void motoi_fillArgs(int argc,void** argv,char** types) -** Don't need to pass the dimensions array anymore. Remove the 'special handling' for arrays
- **static void motoi_convertArgs(int argc, void **argv, char **at, char **vt,MotoVal** castvals) -** Don't need to pass the dimensions arrays anymore.
- **void motoi_fn(const UnionCell *p)** -
    1) Remove the argument dimensions array ... don't need it anymore
    2) Change the calls to fillArgs and ftab_cacheGet and convertArgs
    3) When initializing the return val set the dimension to 0
    4) Use motoi_extractType to get the types of the MDF arguments

5) replace all calls to getMemberDimensions with 0

- **void motoi_new(const UnionCell *p)** -
    1) Remove the argument dimensions array ... don't need it anymore
    2) Change the calls to fillArgs and ftab_cacheGet and convertArgs
    3) When initializing the return val set the dimension to 0
    4) Use motoi_extractType to get the types of the MDF arguments
    5) replace all calls to getMemberDimensions with 0

<u>MotoType Changes</u>
- **int moto_checkImplicitCast(MotoTypeTable *table, char *fromtype, char *totype) -** remove the fromtype dimension and the totype dimension arguments. Get them back by calling typeForName :)
- **MotoType * mttab_typeForName(MotoTypeTable *table, char *typen) -** Hand parse (recursively) canonical type names into MotoTypes adding dynamically defined types along the way

<u>Verifier Changes</u>
- **static void motov_fillArgs(int argc,void** argv,char** types) -** Don't need to pass the dimensions array anymore. Remove the 'special handling' for arrays
- **void motov_fn(const UnionCell *p)** -
    1) Remove the argument dimensions array ... don't need it anymore
    2) Change the calls to fillArgs and ftab_cacheGet
    3) When initializing the return val set the dimension to 0

- **void motoi_new(const UnionCell *p)** -
    1) Remove the argument dimensions array ... don't need it anymore
    2) Change the calls to fillArgs and ftab_cacheGet
    3) When initializing the return val set the dimension to 0

<u>Upgrades to the Moto Extension Compiler</u>

First off the output of mxcg should be modified to not output an arrays dimensions seperate from its base type in exports.mx.c like it does now:

```
  "Date::Date(2)|Date|0|Date|2|String|0|String|0|_Date_Date__SS|date_create|1",
```

should become

```
  "Date::Date(2)|Date|Date|2|String|String|_Date_Date__SS|date_create|1"
```

Instead the String canonical form of the return and argument types should be output e.g

```
  "Vector::toArray(0)|Object|1|-|0|-|_Vector_toArray__|vec_toArray|0"
```

should become

```
  "Vector::toArray(0)|Object[]|-|0|-|_Vector_toArray__|vec_toArray|0"
```