SimpMeta
Programmer's Manual

— Version 1.0 —

A Simple Compiler/Synthesizer Infrastructure for Teaching
Tech Report TR-01-01-01
January, 2001

Jianwen Zhu
Electrical and Computer Engineering
University of Toronto

Contents

1 Introduction

SimpMeta is a simple compiler infrastructure designed for teaching courses such as behavioral synthesis and compiler construction.

SimpMeta hides much of the implementation detail so that those who are interested in programming language design can focus on the frontend, while those who are interested in optimizations can focus on the backend. This desired separation of frontend and backend is achieved by defining two APIs, namely the SimpFrontend API for frontend developers, and the SimpIR for backend developers. Both APIs create, query and manipulate a set of cleanly-defined meta objects. Encapsulated in C++ classes, these meta objects include all the basic components needed for compiler intermediate representations.

SimpMeta also comes with a set of tools. One of the most useful is the C frontend, called `mcc`, which compiles C code into an intermediate form that can be stored on the disk and be further processed by the SimpIR API. Other tools are designed for debugging purpose, `mdump`, can print the intermediate representation in human readable form, `m2dots` can print the intermediate representation in a format that can be visualized using `dot` tool from AT&T.

To use SimpMeta tools, add the following line in the .cshrc file, assuming SimpMeta is installed at `installdir`:

```
>setenv IPSUITE installdir
>setenv PATH ${PATH}:${IPSUITE}/bin
```

Also make sure that the gnu C compiler (gcc) as well as the C preprocessor (cpp) are on the search path.

To use the SimpMeta APIs, add the following in your C++ compiler command line:

```
-I${IPSUITE}/include -L${IPSUITE}/lib -lsimpmeta \\
        -ldl -lm
```

The following example illustrate how `mcc` should be used:

```
>mcc -beh foo.c
```

The command generates foo.cls and foo.beh. Note that both foo.cls and foo.beh are needed in order for SimpIR API to load them into memory.

The following example illustrate how `mcc` can be used to generate SimpIR in SSA form:

```
>mcc -Wintra-O1 -beh foo.c
```

The following example illustrates how `mdump` should be used, assuming both foo.cls and foo.beh are present in your currrent directory.

```
>mdump native.foo
```

Note that the argument of `mdump` should be a complete type name. In SimpMeta infrastructure, all the C modules are within the `native` name space, and hence the prefix `native` is needed to complete the type specification.

The following example illustrates how `m2dots` can be used in conjunction with dot to create the visualization of the control-flow graph of a function bar in foo.c in postscript form.

```
>m2dots native.foo
>dot -Tps -o bar.ps bar.dot
```

2 SimpMeta Objects

simpmeta.h

Names

Author: Jianwen Zhu
Version: 1.0

2.1 enum **SimpKind**

kinds of SimpMeta object

Names

KIND_TYPE_VOID
void data type

KIND_TYPE_BOOL
boolean data type

KIND_TYPE_INT *signed integer data type*

KIND_TYPE_UNSIGNED
unsigned integer data type

KIND_TYPE_FLOAT
floating point data type

KIND_TYPE_CLASS
class type

KIND_CNST_VALUE
primitve value constant

KIND_CNST_STRING
string constant

KIND_CNST_MEMREF
member reference constant

KIND_FIELD *field member*

KIND_METHOD *method member*

KIND_VARIABLE *local variable or method parameter*

KIND_LABEL *normal label*

KIND_CASE_LABEL
case label

KIND_DEFAULT_LABEL
default label

KIND_DAG_LABEL
label instructions

KIND_DAG_BJ *branch and jump instructions*

KIND_DAG_MWB *multiway branch instructions*

KIND_DAG_FUNC *arithmetic or logical instructions*

KIND_DAG_CNST *constant instruction*

KIND_DAG_CNV *conversion instructions*

KIND_DAG_MEM *memory related instructions*

KIND_DAG_MISC *miscellaneous instructions*

KIND_BLK_BASIC
basic block

2.2

enum **SimpAcc**

access code

Names

ACC_FINAL *final modifier*

ACC_PUBLIC *public modifier*

ACC_PROTECTED
protected modifier

ACC_PACKAGE *package modifier*

ACC_PRIVATE *private modifier*

ACC_PARAM *parameter*

ACC_STATIC *static modifier*

ACC_SYNCHRONIZED
synchronized modifier

ACC_NATIVE *native modifier*

ACC_VOLATILE *valatile modifier*

2.3

enum **SimpOpcode**

instruction opcodes

Names

OP_LAB	*label*
OP_BA	*branch always (jump)*
OP_BT	*branch if true*
OP_BF	*branch if false*
OP_BEQ	*branch if equal*
OP_BNE	*branch if not equal*
OP_BGT	*branch if greater than*
OP_BGE	*branch if greater and equal*
OP_BLT	*branch if less than*
OP_BLE	*branch if less than and equal to*
OP_RET	*return*
OP_MWB	*mutiway branch multiway*
OP_CAL	*static dispatch*
OP_CALV	*dynamic dispath*
OP_CALR	*remote dispatch*
OP_CALP	*primitive dispatch*
OP_ARG	*argument*
OP_NEW	*object creation*
OP_DEL	*object deletion*
OP_CNV	*convertion*
OP_BEXT	*bitvector extraction*
OP_BREV	*bitvector reverse*
OP_BCONC	*bitvector concatenation*
OP_CPY	*copy*
OP_ADD	*addition*
OP_SUB	*subtraction*
OP_MUL	*multiplication*
OP_DIV	*devision*
OP_NEG	*negation*
OP_MIN	*minimum*
OP_MAX	*maximum*

OP_ABS	*absolute value*
OP_SHL	*left shifting*
OP_SHR	*right shifting*
OP_USHR	*unsigned right shifting*
OP_BAND	*bitwise and*
OP_BOR	*bitwise or*
OP_BXOR	*bitwise exclusive or*
OP_BINV	*bitwise inversion*
OP_SAND	*serial and*
OP_SOR	*serial or*
OP_SXOR	*serial xor*
OP_REM	*remainder (modulo)*
OP_SEQ	*set if equal*
OP_SGT	*set if greater than*
OP_SGE	*set if greater than or equal to*
OP_SLT	*set if less than*
OP_SLE	*set if less than or equal to*
OP_SNE	*set if not equal*
OP_LOR	*logical or*
OP_LAND	*logical and*
OP_LINV	*logical inversion*
OP_SEL	*selection*
OP_TUPLE	*tuple*
OP_LIST	*list*
OP_GEN	*generate*
OP_CNST	*constant*
OP_SREG	*symbolic register*
OP_LODF	*load field*
OP_STRF	*store field*
OP_LODL	*load local*
OP_STRL	*store local*
OP_LODP	*load parameter*

OP_STRP	*store parameter*
OP_LODA	*load array*
OP_STRA	*store array*
OP_LOD	*load pointer*
OP_STR	*store pointer*
OP_ADDRF	*address of field*
OP_ADDRL	*address of local*
OP_ADDRP	*address of paramter*
OP_ADDRA	*address of array*
OP_ADDRB	*address of a label(for jmp table)*
OP_PADD	*pointer addition*
OP_PSUB	*pointer subtraction*
OP_NOP	*nop*
OP_PHI	*phi instruction in ssa form*
OP_MRK	*mark for source coodinate*
OP_REP	*repeation*
OP_AST	*assertion*
OP_SPC	*memory space*
OP_LAST	*the last opcode*

2.4 SimpValue

place holder for constant values

Names

Jboolean	**b**	*boolean value*
Jlong	**i**	*signed integer value*
Julong	**u**	*unsigned integer value*
Jdouble	**f**	*floating-point value*
char*	**s**	*string value*

struct	*signed integer range value*
struct	*unsigned integer range value*
struct	*floating-point range value*

2.5 class **SimpMeta**

base class for all meta objects

Names

2.5.1 Jboolean **isNull** (void)

null detection

This method tests if the meta object is an null object.

Return Value: TRUE if it is null, FALSE otherwise

2.5.2

Jint **getID** (void)

ID query

This method gets a unique integer identifier of the meta object.

Return Value: the ID of the object

2.5.3

Jint **getKind** (void)

kind query

This method gets the kind (one of SimpKind values) of a meta object.

Return Value: the kind of the object.

2.5.4

void **dump** (FILE* fp)

dumping

This method dumps a meta object to human readable form.

Return Value: none
Parameters: `fp` — the file to which to dump.

2.6 class **SimpCell**

cell of a double-linked list

Names

2.6.1 Jboolean **isNull** (void)

null detection

This method tests if a list cell is null.

Return Value: TRUE if it is null, FALSE otherwise.

2.6.2 SimpCell **getPrev** (void)

This method retrieves the previous element of a list cell.

Return Value: a SimpCell object.

2.6.3

SimpCell **getNext** (void)

This method retrieves the next element of a list cell.

Return Value: a SimpCell object.

2.6.4

SimpMeta **getData** (void)

This method retrieves the meta object associated with a list cell.

Return Value: a SimpMeta object.

2.7

class **SimpList**

a double-linked list

Names

2.7.1

Jboolean **isNull** (void)

null detection.

This method tests if a list is null.

Return Value: TRUE if it is null, FALSE otherwise

2.7.2

SimpCell **getHead** (void)

head query

This method retreives the head of a list.

Return Value: a SimpCell object.

2.7.3

SimpCell **getTail** (void)

tail query

This method retreives the tail of a list.

Return Value: a SimpCell object.

2.7.4

Jint **getSize** (void)

size query

This method retreives the tail of a list.

Return Value: a SimpCell object.

2.7.5

void **insert** (SimpCell cell, SimpCell after)

insertion

This method insert a SimpCell into a list.

Return Value: none.
Parameters: `cell` — the SimpCell to insert.
`after` — the location after which to insert. If after is a NULL, then the new cell is inserted at the head of the list.

2.7.6

void **remove** (SimpCell cell)

removal

This method remove a SimpCell from a list.

Return Value: none.
Parameters: `cell` — the SimpCell to remove.

2.7.7

void **append** (SimpCell cell)

insertion

This method inserts a SimpCell at the end of a list.

Return Value: none.
Parameters: `cell` — the SimpCell to insert.

2.7.8

void **push** (SimpCell cell)

insertion

This method inserts a SimpCell at the head of a list.

Return Value: none.
Parameters: `cell` — the SimpCell to insert.

2.7.9

SimpCell **pop** (void)

insertion

This method removes the head of a list.

Return Value: the removed SimpCell.

2.8

class **SimpType** : public SimpMeta

type object (primitive type or class type)

Names

2.8.1

Jint **getSize** (void)

size retrieval

This method reteieves the size (in number of bits) of a primitive type.

Return Value: the size.

2.8.2

Jint **getAlign** (void)

size retrieval

This method reteieves the alignment (in number of bits) of a primitive type.

Return Value: the size.

2.8.3

Jlong **getAcc** (void)

access code retrieval

This method reteieves the access code of a class type.

Return Value: the access code.

2.8.4

Jint **getArgNum** (void)

type argument retreivel

This method reteieves the number of arguments of a type.

Return Value: the number of type arguments

2.8.5

SimpType **getArg** (Jint i)

type argument retreivel

This method reteieves the ith argument of a type.

Return Value: type argument

2.8.6

SimpList **getFields** (void)

field retrieval

This method reteieves the set of fields defined in a class type.

Return Value: the field list.

2.8.7

SimpList **getMethods** (void)

method retrieval

This method reteieves the set of methods defined in a class type.

Return Value: the method list.

2.9

class **SimpCnst** : public SimpMeta

constant object

Names

2.9.1

SimpValue **getValue** (void)

value retrival

This method retrives the constant value associated.

Return Value: the value.

2.9.2

SimpType **getType** (void)

type retrival

This method retrives the type of the constant.

Return Value: the type.

2.10

class **SimpVar** : public SimpMeta

variable (local or parameter) object

Names

2.10.1

char* **getName** (void)

name retrival

This method retrives the name of the variable.

Return Value: the name.

2.10.2

SimpType **getType** (void)

type retrival

This method retrives the type of the variable.

Return Value: the type.

2.10.3

Jboolean **isParam** (void)

parameter test

This method tests if the variable is a parameter.

Return Value: TRUE if it is a parameter, FALSE otherwise.

2.11

class **SimpLabel** : public SimpMeta

label object

Names

2.11.1

char* **getName** (void)

name retrieval.

This method retrieves the name of a label

Return Value: its name.

2.11.2

SimpValue **getCaseValue** (void)

value retrieval.

This method retrieves the value associated with a case label.

Return Value: its value.

2.12

class **SimpDag** : public SimpMeta

instruction object

Names

2.12.1

SimpType **getType** (void)

type retrieval.

This method retrieves the type of an instruction.

Return Value: its type.

2.12.2

Jint **getFormat** (void)

format retrieval.

This method retrieves the format of an instruction.

Return Value: its format (one of SimpKind value).

2.12.3

Jint **getOpcode** (void)

opcode retrieval.

This method retrieves the opcode of an instruction.

Return Value: its opcode (one of SimpOpcode value).

2.12.4

SimpMeta **getSymbol** (void)

symbol retrieval.

This method retrieves the symbol (constant, variable or label) associated with an instruction.

Return Value: its associated symbol.

2.12.5

SimpDag **getBase** (void)

base retrieval.

This method retrieves the base operand of an instruction.

Return Value: its base operand.

2.12.6

SimpDag **getSrc1** (void)

source operand retrieval.

This method retrieves the first operand of an instruction of KIND_FORMAT_2DAG/KIND_FORMAT_3DAG format.

Return Value: its first src operand.

2.12.7

SimpDag **getSrc2** (void)

source operand retrieval.

This method retrieves the second operand of an instruction of KIND_FORMAT_2DAG/KIND_FORMAT_3DAG format.

Return Value: its second source operand.

2.12.8

SimpDag **getSrc3** (void)

source operand retrieval.

This method retrieves the third operand of an instruction of KIND_FORMAT_3DAG format.

Return Value: its second source operand.

2.12.9

Jint **getOffset** (void)

source operand retrieval.

This method retrieves offset value associated with an instruction of KIND_FORMAT_2DAG format.

Return Value: offset

2.12.10

SimpList **getSrcs** (void)

source operands retrieval.

This method retrieves the list of source operands of an instruction of KIND_FORMAT_NDAG format.

Return Value: the list of all operands.

2.12.11

SimpList **getTargets** (void)

targets retrieval.

This method retrieves the list of targets of an instruction of KIND_FORMAT_NSYM format.

Return Value: the list of all targets.

2.12.12

Jerror **getCoord** (char**pfile, Jint*pcol, Jint *prow)

coordinate retrieval.

This method retrieves the coordinate of an instruction of KIND_FORMAT_COORD format.

Return Value: 0 if succeeded, negative if failed.
Parameters: `pfile` — a string pointer for the returned source file.
`pcol` — a inter pointer for the returned column.
`prow` — a inter pointer for the returned row (line number).

2.13

class **SimpBlk** : public SimpMeta

basic block object

Names

2.13.1

SimpList **getCodes** (void)

code retrieval.

This method retrieves the forest of instrutions in a basic block.

Return Value: the forest.

2.13.2

SimpList **getSuccs** (void)

successor retrieval.

This method retrieves the set of successors of a basic block in the control flow graph.

Return Value: the successors.

2.13.3

SimpList **getPreds** (void)

predecessors retrieval.

This method retrieves the set of predecessors of a basic block in the control flow graph.

Return Value: the predecessors.

2.14

class **SimpMember** : public SimpMeta

member object (field or method)

Names

2.14.1

SimpType **getType** (void)

type retrieval

This method retrieves the type (or return type) of a member.

Return Value: the type.

2.14.2

Jlong **getAcc** (void)

access code retrieval

This method reteieves the access code of a member.

Return Value: the access code.

2.14.3

char* **getName** (void)

name retrieval

This method retrieves the name of a member.

Return Value: the name.

2.14.4

SimpList **getBlks** (void)

control flow graph retrieval

This method retrieves the set of basic blocks of a member.

Return Value: the list of basic blocks.

2.14.5

SimpList **getVars** (void)

variable retrieval

This method retrieves the set of variables (parameters and locals) of a member.

Return Value: the list of variables.

2.14.6

SimpList **getLabels** (void)

label retrieval

This method retrieves the set of labels contained in a member.

Return Value: the list of labels.

2.14.7

Jint **getMaxVarID** (void)

maximum variable ID retrieval

This method retrieves the maximum variable identifier used.

Return Value: the ID.

2.14.8

Jint **getMaxLabelID** (void)

maximum label ID retrieval

This method retrieves the maximum label identifier used.

Return Value: the ID.

2.14.9

Jint **getMaxBlkID** (void)

maximum basic block ID retrieval

This method retrieves the maximum basic block identifier used.

Return Value: the ID.

2.14.10

Jint **getMaxDagID** (void)

maximum instruction ID retrieval

This method retrieves the maximum instruction identifier used.

Return Value: the ID.

2.14.11

void **dumpDot** (void)

visualization

This method dump the method foo in dot format (foo.dot) which can later be visualized by AT&T's dot utility.

Return Value: void.

3 SimpIR API

simpir.h

Names

Author: Jianwen Zhu
Version: 1.0

3.1 class **SimpIR**

SimpIR Name Space

Names

3.1.1

static Jerror **begin** (Jint argc, char* argv[])

initialization

This method initializes the SimpIR API. It must be called before any SimpMeta APIs are used.

Return Value: 0 if succeeded, negative if failed.
Parameters: `argc` — command line argument counts.
`argv` — command line arguments. option recogonized -Mpath: specifying class path.

3.1.2

static Jerror **end** (void)

finalization

This method finalizes the SimpIR API. It must be called before the program terminates.

Return Value: 0 if succeeded, negative if failed.

3.1.3

static SimpType **loadClass** (char* name)

loading

This method loads an intermediate representation (class file) from the disk into memory.

Return Value: a SimpType object.
Parameters: `name:` — the complete name of the class. the name follows the following convention: if it is a Java class: packageName.[outerClassName.]*className; if it is a C module: native.moduleName.

3.1.4

static Jerror **delClass** (SimpType m)

class deletion

This method removes an intermediate representation from memory

Return Value: 0 if succeeded, negative if failed.
Parameters: `name:` — a SimpType object

3.1.5

static Jerror **writeClass** (SimpType m)

loading

This method write a SimpType object from memory to disk (class file)

Return Value: 0 if succeeded, negative if failed.
Parameters: `m:` — a SimpType object. The object must be of kind KIND_TYPE_CLASS.

3.1.6

static Jerror **dumpClass** (SimpType m)

dumping

This method dump a SimpType object in human readable form

Return Value: 0 if succeeded, negative if failed.
Parameters: `m:` — a SimpType object. The object must be of kind KIND_TYPE_CLASS.

3.1.7

static SimpDag **newDag** (SimpMember mbr, SimpOpcode opcode, SimpType type, SimpDag base, SimpDag src1, SimpDag src2, SimpMeta symb, Jint offset)

instruction creation

This method creates a new instruction in KIND_FORMAT_2DAG format

Return Value: the created instruction.
Parameters:
`mbr` — the member in which the instruction should be created.
`gopcode` — the opcode of the instruction.
`type` — the type of the instruction.
`base` — the base operand of the instruction.
`src1` — the first source operand of the instruction.
`src2` — the second source operand of the instruction.
`symb` — the associated symbol (variable, label or constant).
`offset` — offset.

3.1.8

static SimpDag **newDag** (SimpMember mbr, SimpOpcode opcode, SimpType type, SimpDag base, Jint noperands, SimpDag* operands, SimpMeta symb)

instruction creation

This method creates a new instruction in KIND_FORMAT_NDAG format

Return Value: the created instruction.

Parameters:
`mbr` — the member in which the instruction should be created.
`gopcode` — the opcode of the instruction.
`type` — the type of the instruction.
`base` — the base operand of the instruction.
`noperands` — the number of source operands.
`operands` — the vector of operands.
`symb` — the associated symbol (variable, label or constant).

3.1.9

static SimpDag **newDag** (SimpMember mbr, SimpOpcode opcode, SimpType type, SimpDag base, Jint ntargets, SimpLabel* targets)

instruction creation

This method creates a new instruction in KIND_FORMAT_NSYM format

Return Value: the created instruction.

Parameters: `mbr` — the member in which the instruction should be created.
`gopcode` — the opcode of the instruction.
`type` — the type of the instruction.
`base` — the base operand of the instruction.
`ntargets` — the number of targets.
`targets` — the vector of targets.

3.1.10

static void **delDag** (SimpMember mbr, SimpDag dag)

instruction deletion

This method reclaims the memory for an unused instruction

Return Value: void
Parameters: `mbr` — the member to which the instruction belong.
`dag` — the instruction to delete.

3.1.11

static SimpVar **newVar** (SimpMember mbr, SimpType type, char* name)

variable creation

This method creates a new local variable.

Return Value: the created variable.
Parameters: `mbr` — the member in which the variable should be created.
`type` — the type of the variable.
`name` — the name of the variable.

3.1.12

static void **delVar** (SimpMember mbr, SimpVar var)

variable deletion

This method reclaims the memory of an unused variable.

Return Value: void
Parameters: `mbr` — the member to which the variable belong.
`var` — the variable to delete.

3.1.13

static SimpLabel **newLabel** (SimpMember mbr, Jint kind, char* name, SimpValue v)

label creation

This method creates a new label.

Return Value: the created label.
Parameters: `mbr` — the member in which the label should be created.
`kind` — the kind of the label.
`name` — the name of the label.
`v` — the value of the case label.

3.1.14

static void **delLabel** (SimpMember mbr, SimpLabel label)

label deletion

This method reclaims the memory of an unused label.

Return Value: void
Parameters: `mbr` — the member to which the label belong.
`label` — the label to delete.

3.1.15

static SimpCell **newCell** (SimpMember mbr, SimpMeta data)

variable creation

This method creates a new list cell.

Return Value: the created list cell.
Parameters: `mbr` — the member in which the cell should be created.
`data` — the associated meta object.

3.1.16

static void **delCell** (SimpMember mbr, SimpCell cell)

list cell deletion

This method reclaims the memory of an unused list cell.

Return Value: void.
Parameters: `mbr` — the member to which the cell belong.
`label` — the cell to delete.

4 SimpFrontend API

simpfront.h

Names

Author: Jianwen Zhu
Version: 1.0

4.1 class SimpFrontend

the SimpFrontend API

Names

4.1.1

static Jerror **begin** (int argc, char* argv[])

initialization

This method initializes the SimpFrontend API.

Return Value: 0 if succeeded, negative if failed.
Parameters: `argc` — argument count
`argv` — argument vector

4.1.2

static Jerror **end** (void)

finalization

This method finalizes the SimpFrontend API.

Return Value: 0 if succeeded, negative if failed.

4.1.3

static SimpType **newPrimitiveType** (Jint kind, Jint size, Jint align)

primitive type creation

This method creates a new primitive data type.

Return Value: the created type.
Parameters: `kind` — meta object kind, must be one of KIND_TYPE_BOOL, KIND_TYPE_INT, KIND_TYPE_UNSIGNED, KIND_TYPE_FLOAT.
`size` — the size (in number of bits) of the type.
`align` — the alignment (in number of bits) of the type.

4.1.4

static SimpType **newClassType** (Jlong acc, SimpType parent, char* name)

class type creation

This method creates a new class data type.

Return Value: the created type.
Parameters: `acc` — the access flag of the class, must be one of the SimpAcc value.
`parent` — parent (package or outer class) class.
`name` — name of the class.

4.1.5

static SimpCnst **newValueCnst** (Jint typekind, Jint typesize, SimpValue v)

value constant creation

This method creates a new primitive value constant.

Return Value: the created constant.
Parameters: `typekind` — type of the constant.
`typesize` — size of the constant.
`v` — value of the constant.

4.1.6

static SimpCnst **newStringCnst** (char* v)

string constant creation

This method creates a new string constant.

Return Value: the created constant.
Parameters: `v` — value of the constant

4.1.7

static SimpCnst **newMemberRefCnst** (Jint kind, SimpType parent, SimpType type, char *v, int nargs, SimpType* args)

string constant creation

This method creates a member reference constant.

Return Value: the created constant.
Parameters: `kind` — kind of the member, must be KIND_FIELD or KIND_METHOD.
`parent` — class type of which the member belongs to.
`type` — type (or return type) of the member.
`v` — name of the member.
`nargs` — the number of arguments of the member.
`args` — a vector of argument types.

4.1.8

static Jerror **beginClassDecl** (SimpType ty, Jlong acc)

class declaration initialization

This method signals the beginning of class declaration

Return Value: 0 if suceeded, negative if failed.
Parameters: `ty` — the class type to be declared

4.1.9

static Jerror **extend** (SimpType ty, SimpType superty)

class inheritance

This method adds a super type for a class type.

Return Value: 0 if suceeded, negative if failed.
Parameters: `ty` — the class type to be extended.
`superty` — the super class type.

4.1.10

static Jerror **implement** (SimpType ty, SimpType superty)

interface inheritance

This method adds a super interface type for a class type.

Return Value: 0 if suceeded, negative if failed.
Parameters: `ty` — the class type to be implemented.
`superty` — the interface class type.

4.1.11

static Jerror **endClassDecl** (SimpType ty)

class declaration finalization

This method signals the end of a class declaration.

@ty the class type declared.

Return Value: 0 if suceeded, negative if failed.

4.1.12

static Jerror **beginClassDef** (SimpType ty, char* file, Jint x, Jint y)

class definiton initialization

This method signals the beginning of a class definition.

Return Value: 0 if suceeded, negative if failed.
Parameters: `ty` — the class type to be defined.
`file` — the source file name where the class is defined.
`x` — the source file column number where the class is defined.
`y` — the source file line number where the class is defined.

4.1.13

static Jerror **endClassDef** (void)

class definiton finialization

This method signals the end of a class definition.

Return Value: 0 if suceeded, negative if failed.

4.1.14

static Jerror **referenceType** (SimpType ty)

type reference

This method signals the use of an external type.

Return Value: 0 if suceeded, negative if failed.
Parameters: `cnst` — the type constant to be referenced.

4.1.15

static Jerror **referenceCnst** (SimpCnst cnst)

type reference

This method signals the use of an external constant.

Return Value: 0 if suceeded, negative if failed.
Parameters: `cnst` — the constant to be referenced.

4.1.16

static SimpMember **beginField** (Jlong acc, SimpType ty, char* name, SimpVar thisArg, char* file, Jint x, Jint y)

field definition initialization

This method signals the beginning of a field definition

Return Value: the created field
Parameters:
`acc` — the access flag of the field.
`ty` — the type of the field.
`name` — the name of the field.
`file` — the source file name where the field is defined.
`x` — the source file column number where the field is defined.
`y` — the source file line number where the field is defined.

4.1.17

static Jerror **endField** (void)

field definition finalization

This method signals the end of a field definition

Return Value: 0 if suceeded, negative if failed.

4.1.18

static SimpVar **newParameter** (Jlong acc, SimpType type, char* name, char* file, Jint x, Jint y)

parameter creation.

This method creates a method parameter

Return Value: the created parameter.
Parameters: `acc` — the access flag of the parameter.
`ty` — the type of the parameter.
`name` — the name of the parameter.
`file` — the source file name where the parameter is defined.
`x` — the source file column number where the parameter is defined.
`y` — the source file line number where the parameter is defined.

4.1.19

static SimpMember **beginMethod** (Jlong acc, SimpType type, char* name, SimpVar thisArg, int nargs, SimpVar* args, char* file, Jint x, Jint y)

method definition initialization

This method signals the beginning of a method definition

Return Value: the created method

Parameters:
`acc` — the access flag of the method.
`ty` — the return type of the method.
`name` — the name of the method.
`nargs` — the number of parameters.
`args` — the vector of parameters.
`file` — the source file name where the method is defined.
`x` — the source file column number where the method is defined.
`y` — the source file line number where the method is defined.

4.1.20

static Jerror **endMethod** (void)

method definition finalization

This method signals the end of a method definition

Return Value: 0 if suceeded, negative if failed.

4.1.21

static SimpVar **newVariable** (Jlong acc, SimpType type, char* name, char* file, Jint x, Jint y)

variable creation.

This method creates a method local variable

Return Value: the created variable.

Parameters:
`acc` — the access flag of the variable.
`ty` — the type of the variable.
`name` — the name of the variable.
`file` — the source file name where the variable is defined.
`x` — the source file column number where the variable is defined.
`y` — the source file line number where the variable is defined.

4.1.22

static SimpLabel **newLabel** (Jint kind, char* name, SimpValue v)

label creation.

This method creates a method label.

Return Value: the created label.
Parameters:
`kind` — the kind of the label, must be one of KIND_LABEL, KIND_CASE_LABEL, KIND_DEFAULT_DEFAULT.
`v` — the case label.
`name` — the name of the label.

4.1.23

static SimpDag **newThisDag** (void)

instruction creation.

This method creates an OP_LODP instruction that references `this`.

Return Value: the created instruction.

4.1.24

static SimpDag **newDag** (Juint gopcode, SimpType type, SimpDag base, SimpDag src1, SimpDag src2, SimpMeta symb, Jint offset)

instruction creation.

This method creates an instruction in KIND_FORMAT_2DAG format.

Return Value: the created instruction.
Parameters:
`gopcode` — the opcode of the instruction.
`type` — the type of the instruction.
`base` — the base operand of the instruction.
`src1` — the first source operand of the instruction.
`src2` — the second source operand of the instruction.
`symb` — the associated symbol (variable, label or constant).
`offset` — offset.

4.1.25

static SimpDag **newDag** (Juint gopcode, SimpType type, SimpDag base, Jint noperands, SimpDag* operands, SimpMeta symb)

instruction creation.

This method creates an instruction in KIND_FORMAT_NDAG format.

Return Value: the created instruction.
Parameters:
`gopcode` — the opcode of the instruction.
`type` — the type of the instruction.
`base` — the base operand of the instruction.
`noperands` — the number of source operands.
`operands` — the vector of operands.
`symb` — the associated symbol (variable, label or constant).

4.1.26

static SimpDag **newDag** (Juint gopcode, SimpType type, SimpDag base, Jint ntargets, SimpLabel* targets)

instruction creation.

This method creates an instruction in KIND_FORMAT_NDAG format.

Return Value: the created instruction.
Parameters: `gopcode` — the opcode of the instruction.
`type` — the type of the instruction.
`base` — the base operand of the instruction.
`ntargets` — the number of targets.
`targets` — the vector of targets.

4.1.27

static SimpDag **newDag** (char* file, int x, int y)

instruction creation.

This method creates an instruction in KIND_FORMAT_COORD format.

Return Value: the created instruction.
Parameters: `file` — the source file name.
`x` — the source file column number.
`y` — the source file line number.

4.1.28

static Jerror **addDag** (SimpDag dag)

instruction creation.

This method adds an instruction tree to the method.

Return Value: the created instruction.
Parameters: `dag` — the instruction to be added.

Class Graph