

# SUIF2 安装与使用说明

张昱 张予现

2011-5-26 创建

SUIF(<http://suif.stanford.edu/>)提供一种树型中间表示以及这种中间表示上的一些基本控制流、数据流分析,提供 `c2suif` 和 `suif2c` 用于实现 C 程序与 SUIF 程序之间的相互转换,提供 SUIF 中间表示上的扩展机制。它定义了一套中间表示结构,定义模块(module)和遍(pass)完成编译器运行。用户可以使用驱动(`suifdriver`)运行一个或多个遍,从而进行编译器翻译,分析和优化。

可以开展的工作:

- (1) 编写新的前端或扩展现有前端。
- (2) 编写新遍分析和优化程序。

## 1 Suif2 编译和运行

所有源程序和相关文档可以在 <http://suif.stanford.edu/suif/suif2/index.html> 中下载。

### 1.1 下载文件包

The SUIF 2.2 release contains :

- The basic infrastructure: [basesuif-2.2.0-4.tar.gz](http://suif.stanford.edu/basesuif-2.2.0-4.tar.gz) (597k)
- Extra data structures, viewers and an intraprocedural dataflow framework [extras-2.2.0-4.tar.gz](http://suif.stanford.edu/extras-2.2.0-4.tar.gz) (152k)
- Interprocedural alias analysis and support libraries [ipanalysis-2.2.0-4beta.tar.gz](http://suif.stanford.edu/ipanalysis-2.2.0-4beta.tar.gz) (45k)
- The SUIF1 and SUIF2 conversion package [suifversion1support-2.2.0-4.tar.gz](http://suif.stanford.edu/suifversion1support-2.2.0-4.tar.gz) (57k)
- OSUIF - Object oriented SUIF extensions [osuif-2.2.0-4.tar.gz](http://suif.stanford.edu/osuif-2.2.0-4.tar.gz) (98k)
- A Java frontend for OSUIF updated to work with jdk 1.2.2 [j2s-2.2.0-4.tar.gz](http://suif.stanford.edu/j2s-2.2.0-4.tar.gz) (619k)
- Documentation sources. [suifdoc-2.2.0-4.tar.gz](http://suif.stanford.edu/suifdoc-2.2.0-4.tar.gz) (1092k)
- [Doxygen](http://suif.stanford.edu/doxygen)-generated documentation for the above packages [doxygen-html-2.2.0-4.tar.gz](http://suif.stanford.edu/doxygen-html-2.2.0-4.tar.gz) (2839k)

### 1.2 安装前的准备

我们是在 Linux 系统中编译 SUIF 的,在编译前需要一些辅助的工具,有些是必须要的,有些不是必须的:

- ! GCC: 不能使用太高的版本,最好使用 gcc-2.95 或 gcc-2.96 版。(必须)
- ! MAKE/GMAKE: 用于编译源文件的工具。(必须)
- ! AWK/GAWK: 用于编写脚本和源文件的工具。(必须)
- ! X11: 一个 linux 中的库,一般系统中都包含。
- ! PERL: 一种常用的脚本语言。(必须)
- ! Tcl/TK: 一种很通用的编程语言脚本,需要用到其中的头文件和库,必须安装,建议低版本 8.3。(必须)

- | LEX and YACC: 用于生成文档的工具。(不必须)
- | DOT: 用于产生有向图的程序。(不必须)
- | GV: to view and navigate through PostScript and PDF documents。(不必须)
- | AS: 轻便的汇编程序, 一般系统上已经安装。
- | DOXYGEN: 用于 C/C++、java 等的文档生成器。(不必须)

### 1.3 安装步骤

在我们的安装过程中提供了两种安装方式, 如果只是对 suif 进行研究, 可以采用基本安装, 它可以构建 suif 编译实验平台; 在我们课题组中, 一般采用基于课题组软件包的安装, 里面不仅包含了 suif 的基本框架, 还有课题组已经实现的程序分析与程序变换模块。

#### 1.3.1 基本包安装

- 1) 解压 basesuif-2.2.0-4.tar.gz  
命令: tar xvfz basesuif-2.2.0-4.tar.gz

- 2) 进入 nci 目录  
命令: cd nci

- 3) 执行安装文件  
命令: ./install

可以在后面加上一下变量的信息

```
--with-CXX=compiler
--with-CC=cc_compiler
--with-CXXLINK=command line options
--with-GC_INCLDIRS=-Ipath
--with-OMEGA_INCLDIRS=-Ipath
--with-LOCAL_BASE, local solib and bin directories will be added
to the LD_LIBRARY_PATH and PATH.
--with-TCL_LIBDIRS= -Ldirs and -llib needed to link TCL libs
一般我们需要设置 GCC 和 G++ 的值, 例如, 使用 GCC-2.95 版则
--with-CC=/usr/bin/gcc-2.95
--with-CXX=/usr/bin/g++-2.95
--with-CXXLINK=/usr/bin/g++-2.95
```

运行后会在 nci 目录中生成一个 Makefile.std 文件, 是用来配置系统的信息, 具体内容见目录中的 README 文件。另外, 会生成两个脚本文件 nci\_setup.sh、nci\_setup.csh, 用来配置系统的环境变量。

- 4) 执行 make setup 命令  
运行后会生成三个目录 bin、obj、solib, 分别存放执行文件、对象文件、库文件。
- 5) 执行 make 命令, 开始编译所有源文件。  
由于在每个目录和子目录里都有 Makefile 文件, 所以, 编译时会根据定义的文件目录逐层进入并编译里面的文件。
- 6) 如果编译通过后可以运行 make test 命令来测试编译是否成功。
- 7) 如果想用 suif2c 命令, 则需要将 nci-edg-bin-2000-06-13 包中的文件拷到 nci 相应的目录, 如果仍然不能使用则按照注意事项中的提示操作。
- 8) 如果要编译其它程序包中的内容有两种方法:
  - n 下载所需要的程序包并解压到 nci\sui2b 目录中, 在 nci\sui2b 目录中的 extra\_suif\_packages 文件中添加解压后的文件夹名。退到 nci 目录中运行

make。

n 直接进入所需编译的文件夹中运行 make。

### 1.3.2 基于课题组软件包的安装

(1) 执行 setup.sh 文件。

u 生成配置文件。

```
./install --with-CC=/usr/bin/gcc-2.95 --with-CXX=/usr/bin/g++-2.95  
--with-CXXLINK=/usr/bin/g++-2.95 --with-TCL_INCLDIRS='-I/usr/include/tcl8.3'  
--with-TCL_LIBDIRS='-L/usr/lib'
```

运行后会在 nci 目录中生成一个 Makefile.std 文件，是用来配置系统的信息，生成两个脚本文件 nci\_setup.sh、nci\_setup.csh，用来配置系统的环境变量。这两个文件是类似的。

u make setup 创建三个目录：bin、obj、solib，分别存放执行文件、对象文件、库文件。

u source nci\_setup.sh 设置环境变量。source 命令:source filename 或 . filename

u make >error.log 编译

u cp auxlib/\* solib/ 拷贝编译后的文件到 bin 文件夹

```
cp auxbin/* bin/
```

```
chmod +x solib/*
```

```
chmod +x bin/*
```

这些文件需要共享库 libstdc++-libc6.1-1.so.2，但现有库版本较高，需要做一个软链接映射到新版本的库。使用 ln -s libstdc++-libc6.2-2.so.3 libstdc++-libc6.1-1.so.2 命令,-s 表示软连接，前一个参数是你系统中现有的库，后一个是需要产生的软链接库。

l make test

(2) 执行“source nci\_setup.sh”命令，设置 NCIHOME 等环境变量。

(3) 进入 suif/suif2b/basesuif/samples 执行 make clean 命令。

(4) 执行 c2suif test.c 会输出 test.suif 文件。

### 1.4 注意事项

1) 编译时不能用版本过高的 GCC，推荐使用 2.95 和 2.96 版。否则可能出现编译不通过或错误的情况。

2) 由于需要拷贝几个编译后的文件到 bin 文件夹，这个文件需要的共享库是 libstdc++-libc6.1-1.so.2，但是由于现有的库中版本比较高，因此我们需要做一个软链接映射到新版本的库。可以使用 ln -s libstdc++-libc6.2-2.so.3 libstdc++-libc6.1-1.so.2 命令,-s 表示软连接，前一个参数是你系统中现有的库，后一个是我们需要产生的软链接库。

3) 环境变量的设置需要注意，首先需要 NCIHOME=nci 目录所在的位置；LD\_LIBRARY\_PATH= \$NCIHOME/solib 所在的位置；PATH= \$NCIHOME/bin 所在的位置。每次运行前都需要确定这几个环境变量被设置。设置时可以手动使用 export 命令添加，也可以执行“source nci\_setup.sh”来设置。建议运行 suif 程序前先执行“source nci\_setup.sh”命令。

## 2 总体结构

SUIF 提供一种树型中间表示以及这种中间表示上的一些基本控制流、数据流分析，提供 `c2suif` 和 `suif2c` 用于实现 C 程序与 SUIF 程序之间的相互转换，提供 SUIF 中间表示上的扩展机制。

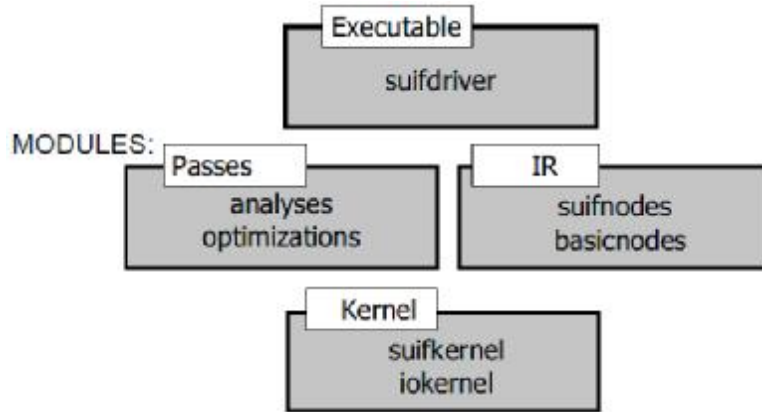


图 2-1 SUIF 总体结构图

- ①Kernel：提供基本的功能，如输入输出。
- ②Modules：中间表示，遍基础上的程序分析和优化。
- ③Suifdriver：使用脚本语言控制在模块和遍上的执行。

## 3 Suifdriver 的使用

驱动 `suifdriver` `- e / i`

```
suif> import basicnodes suifnodes
suif> import mylibrary
suif> load test.suif
suif> mylibrary_pass1
suif> print test.out
suif> save test.tsuif
```

将 c 文件编译成 suif 文件：

```
c2suif test.c
```

将 suif 软件包中的 `nci/suif/suif2b/dataflow` 下的文件进行 `make` 后，会在 `nci/bin` 目录下增加 `deadcode_driver` 可执行文件，会在 `nci/solib` 目录下增加 `libcopyprop.so`、`libdeadcode.so`、`libdfflowsolver.so`、`libsuiif_cfgraph.so` 4 个动态库。可以利用下面的命令加载 `libsuiif_cfgraph.so` 库，为指定的 suif 文件构造 CFG 并以 dot 格式输出到文件中：

```
suifdriver -e "import basicnodes suifnodes suif_cfgraph; load test.suif;
print_suif_cfgraph_to_dot" > test.dot
```

然后可以用 `graphviz` 软件包中的 `dot` 命令为指定的 dot 文件产生 ps 文件或 jpg 文件：

```
dot -Tps file.dot -o file.ps
dot -Tjpg file.dot -o file.jpg
```

## 4 SUIF IR 结构

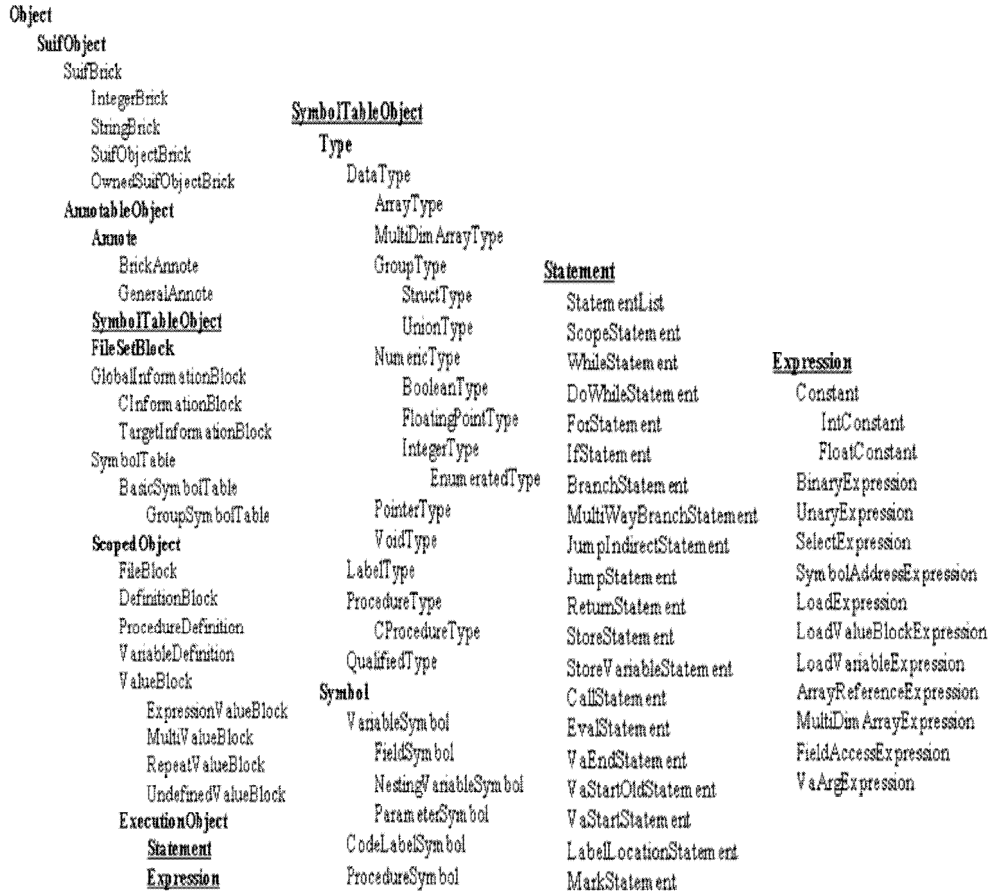


图 4-1 SUIF 基本对象的类层次图(1)

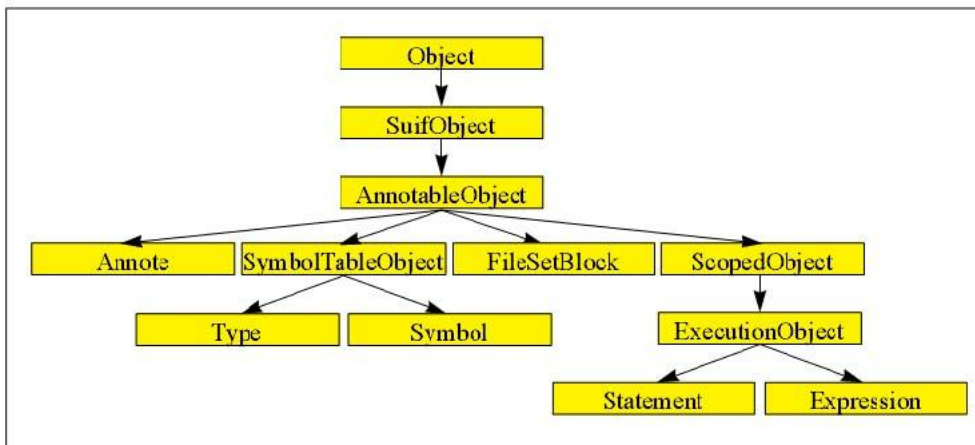


图 4-2 SUIF 基本对象的类层次图

SUIF IR 是 SUIF 系统的基础，它由相对简单的称为 hoof 文件的描述文件生成。Hoof 文件经 smgn (SUIF Macro Generator) 分析和翻译，得到一组头文件、一些用来创建该 hoof 文件所定义的类的对象的方法，并为该 hoof 文件中的类产生其在输入输出系统中的对应代码，使得用户无需为自己定义的类编写额外的输入输出代码。

## 4.1 由 hoof 生成的基本 SUIF IR 结构

基本的 SUIF IR 是由 hoof 文件生成的各种类，基本的 SUIF IR 描述定义在 **basic.hoof**、**suif.hoof** 和 **cfe.hoof** 3 个文件中，分别置于 nci/suif/suif2b/basic-suif 目录下的 basicnodes、suifnodes 和 cfenodes 3 个子目录中。

## 4.2 基本的 SUIF IR 结构 - FileSetBlock

任意一棵 **SUIF 树** 的顶层对象总是一个 FileSetBlock 实例，它包含两个符号表（外部符号表 `_external_symbol_table` 和文件符号表 `_file_set_symbol_table`）、一组类型为 `list<FileBlock*` 的文件块 `_fileblocks` 和一组类型为 `list<GlobalInformationBlock*` 的信息块 `_information_blocks`。

这些可以参见: `svn://parallel/users/yxzhong/suif/doc/main/main-suif.out` 文件。

### 4.2.1 外部符号表

外部符号表存储可以被该 suif 文件使用的**类型对象**、**符号对象**（即全局符号，包括变量符号和函数符号）。

### 4.2.2 文件符号表

文件符号表存储可以被多个 suif 文件使用的对象。

### 4.2.3 文件块

一个源程序文件（即一个编译单元）表示为一个文件块（类型为 FileBlock）。FileBlock 类中含有如下 3 个域：

- ①源程序文件名。
- ②作用域限于该文件块的全局符号表。
- ③该文件块的定义。

文件块定义包括：

#### 4.2.3.1 变量定义

#### 4.2.3.2 过程定义

##### 1、语句

CallStatement, IfStatement, WhileStatement, StoreStatement, StoreVariableStatement, ReturnStatement, JumpStatement, JumpIndirectStatement, BranchStatement, LabelLocationStatement。

##### 2、表达式

### 4.2.4 信息块

## 4.3 SUIF IR 的扩展

在 SUIF 中，除了顶层的 Object 和 SuifObject 两个类由系统定义之外，其他的类都是通过 hoof 文件扩展而来。

每一个 Hoof 文件，就是一个 SUIF 的模块(一个 Hoof 文件里面可以定义多个 module，但是只有第一个 module 会被处理)。Hoof 文件的内容格式如下：

```
module module_name{  
    include “要包含的头文件”;
```

```

....
import 要导入的模块
...
abstract/concrete class_name : super_class_name{
    类的属性域
};
...
}

```

一个 hoof 文件会产生 4 个 C++ 文件， module\_name.h、 module\_factory.h 和 module\_name\_forwarders.h 是头文件， module\_name.cpp 是它们三个的实现。

注意：模块名、文件名和 Makefile 文件中的 HOOF 变量的值(生成文件名的前缀)要一致。

- (1) Hoof 文件定义和使用。
- (2) Makefile 文件。

## 5 SUIF IR 的创建与访问

### 1.1 SUIF IR 对象的创建

#### 5.1.1 通过工厂方法直接创建

在 basic\_factory.h 和 suif\_factory.h 两个文件中，提供了两种创建类的方法接口，以语句类为例如下：

```

Statement* create_statement(SuifEnv *env)和
Statement* create_statement();

```

#### 5.1.2 通过库函数创建

在 SUIF 工程中，在文件 nci/suif/suif2b/utils/node\_builder.h 中，提供了一些方便的构造 SUIF IR 的方法

### 5.2 SUIF IR 对象的访问

SUIF 中提供了对象迭代器 Iterator 方法，SUIF IR 遍历的 Walker 方法，程序分析的遍 (Passs)方法。

#### 5.2.1 Iterator

迭代器是一组有序对象集合，且包含有遍历这些对象的接口。在 SUIF 工程的 nci/suif/suif2b/basesuif/suifkernel/iter.h 文件中，声明了迭代器 Iter 的接口，它是一个模板类。

#### 5.2.2 Walker

Walker 的作用是用来遍历结点树，并对传给 Walker 的某种类型及它的子类型结点进行访问处理。SUIF 提供了两种 Walker 的基类，在访问结点树的时候，可以通过需要来继承这两

者中的一个,或两者同时使用,它们的接口在 `nci/suif/suif2b/basesuif/suifkernel/group_walker.h` 头文件中。

### 5.2.3 Pass

Pass 是在程序分析中用来对整个 SUIF IR 的作一遍分析的接口类,它直接继承 Module。在 `nci/suif/suif2b/basesuif/suifpasses/pass.h` 头文件声明了 Pass 的接口。Pass 主要有以下几个方法:

```
void initialize()
void execute()
void do_file_set_block(FileSetBlock* file_set_block)
```

第一个是初始化 Pass,第二个方法是执行 Pass,第三个方法是用户想在这个 Pass 中所要做的分析工作,它会被第二个方法调用。从第三个方法的参数 FileSetBlock(它是一个 SUIF 文件的根)可以看出,这个遍是可以对整个 SUIF IR 表示进行操作的。

在 SUIF 中,为了方便程序分析工作,它提供了一个可以让多个遍连续执行的类 PipelinePass。PipelinePass 类继承了 Pass 类。

## 5.3 写一个自己的 pass

在设计程序分析遍的时,我们需要考虑下面三个步骤:

### I 设计分析遍类

它需要继承 PipelinePass,从而获得接口:

```
do_file_set_block
do_file_block
do_procedure_definition
do_variable_definition
```

因为这些接口,是对 SUIF IR 的某一部分的根。如 FileSetBlock 是整个 SUIF 的根,FileBlock 是文件块的根,ProcedureDefinition 是过程的根,VariableDefinition 是变量定义的根。这样的话,如果只想对变量进行访问,我们只需要在 `do_variable_definition` 中增加相应的访问结点的代码就可以了。显然,这样在使用时不仅方便,而且灵活。

### I 设计一个或多个 Walker

这些 Walker 需要继承 SelectiveWalker(也可以再设计一个 GroupWalker,将这些 SelectiveWalker 包装起来),用来对某一种类型或某几种类型的结点对象进行分析,如只想对变量符号进行访问,我们只需要将变量符号的类型传递给某个 Walker,则 Walker 在工作的时候,只会处理变量符号及其子类型的结点,而不去处理其他类型的结点。

### I 使用 Iterator

使用 Iterator 对相同类型的结点进行访问。

上面的三个步骤是相关的,一般情况下,Iterator 的使用是在 Walker 内,对一些具体的对象进行访问的,而 Walker 的是用来遍历结点的,它会被放在 PipelinePass 中的接口方法中,

并以 `PipelinePass` 方法的参数为根用 `Walker` 去遍历它的结点，当满足条件时，如果有需要，就使用 `Iterator` 器去迭代。