Compiler Principle

Prof. Dongming LU

Apr. 1st, 2024

Content

6 ACTIVATION RECORD

6.2 FRAMES IN THE Tiger COMPILER

What sort of stack frames should the Tiger compiler use?

The frame interface : abstract

```
/* frame.h*/
typedef struct F_frame_ *F_frame;
typedef struct F_access_ *F_access;

typedef struct F_accessList_ *F_accessList;
Struct F_accessList_  {F_access head; F_accessList tail; };

F_frame F_newFrame( Temp_label name, U_boolList formals );
Temp_label F_name(F_frame f);
F_accessList F_formals(F_frame f);
F_access F_allocLocal(F_frame f, bool escape)
```

- **Implemented by a module specific to the target machine.**
- **Accessed by the machine-independent parts of the compiler.**

The type F_frame

- **Holding the information about formal parameters and local variables allocated in the frame.**

- A three-argument function named g **with first argument escapes**.

  F_newFrame(g, U_BoolList(**TRUE**,
  U_BoolList(FALSE, U_BoolList(FALSE,
  NULL))))

The F_access type

- F_access type **describes formals and locals**

```
/*mipsframe.c*/
#include "frame.h"
struct F_access
        { enum {inFrame, inReg} kind;
          union {int offset;
                 Temp_temp reg;
                } u; };
static F_access InFrame(int offset);
static F_access InReg(Temp_temp reg)
```

The F_formals interface

- Parameters may be **seen differently** by the caller and the callee.

- The **“shift of view”** depends on the calling conventions of the target machine handled by the frame module.

- newFrame must calculate two things for each formal parameter.
  - **How** the parameter will be seen from inside the function(in a register, or in a frame location);
  - **What** instructions must be produced to implement the “View shift”

Representation of Frame Descriptions

- **Secret from** any clients of the Frame module.
- A data structure holding:
  - ✓The locations of all the formals,
  - ✓Instructions required to implement the “View shift”,
  - ✓The number of locals allocated so far,
  - ✓And the label at which the function’s machine is to begin

Representation of Frame Description

Formal parameters for g(x1, x2, x3) where x1 escapes **(for Pentium)**

- inFrame(8)
- InFrame(12)
- inFrame(16)

Formals 1
2
3

- M[sp+0] ← fp
- fp←sp
- sp←sp-K

View
Shift

Representation of Frame Description

Formal parameters for g(x1, x2, x3) where x1 escapes **(for MIPS)**

- InFrame(0)
- InReg(t_{157})
- InReg(t_{158})

Formals 1 2 3

- sp←sp-K
- M[sp+K+0] ← r2
- t_{157}←r4
- t_{158}←r5

View Shift

Representation of Frame Description

Formal parameters for g(x1, x2, x3) where x1 escapes **(for Sparc)**

- InFrame(68)
- InReg(t_{157})
- InReg(t_{158})

Formals 1 2 3

- save %sp,-K,%sp
- M[fp+68] ← i0
- $t_{157} \leftarrow i1$
- $t_{158} \leftarrow i2$

View Shift

Local Variables

- **Some local variables are kept in the frame while others are kept in registers.**

- Allocate a new local variable in a frame ***f***
  - F_allocLocal(f, TRUE)
  - An InFrame access with an offset from the frame pointer.
  - **Such as InFrame(-4)**

- Allocate a register for a local variable
  - F_allocLocal(f, FALSE)
  - **Such as InReg(t_{481})**

Calculating Escape

- **Look for escaping variables and record this information in the escape fields of the abstract syntax。**

  —— FindEscape

- Occur before semantic analysis

- **The traversal function for FindEscape: a mutual recursion on abstract syntax exp’s and var’s.**

Calculating Escape

```
/*escape.h*/
void Esc_findEscape(A_exp exp);
 /*escape.c*/
 static void traverseExp(S_table env, int depth, A_exp e);
 static void traverseDec(S_table env, int depth, A_dec d);
 static void traverseVar(S_table env, int depth, A_var v);
```

- **A variable or formal-parameter declaration at static function-nesting depth d**, such as A_VarDec{name=symbol(“a”), escape=r,…}

- **EscapeEntry(d, &(x->escape)) will set x->escape FALSE.**
- **When a is used at depth > d,** the escape field of x is set to TRUE

Temporaries and Labels

- Use the word temporary to mean a value held in a register;
  - ✓Temps are abstract names for local variables;
- Use the word label to mean some machine-language location;
  - ✓Labels are abstract names for static memory address.

- Temp_newtemp() : a new temporary from an infinite set of temps
- Temp_newlabel() : a new label from an infinite set of labels
- Temp_namedlabel(**string**) : a new label whose assembly-language name is string.

- **Call Temp_namedlabel("f") to process the declaration function f(…)**

Two Layers of Abstraction

semant.c

translate.h

translate.c

frame.h temp.h

*u*frame.c temp.c

- The frame.h and temp.h interfaces provide machine-independent views of memory-resident and register-resident variables.
- **The translate.h is not absolutely necessary. Translate manages local variables and static function nesting for semant.**

Two Layers of Abstraction

```
/*new versions of VarEntry and FunEntry*/
Struct E_enventry_{
        enum {E_varEntry, E_funEntry} kind;
        union {struct { Tr_access access; Ty_ty ty;} var;
                struct { Tr_level level; Temp_label label;
Ty_tyList formals; Ty_ty result;} fun} u; };

struct Tr_access_ {Tr_level level; F_acess access;};
```

Managing Static Links

- **The Frame module should be independent of the specific source language being compiled.**
- Frame should **not know anything about static links**. Translate will be responsible for this.
- **The static link behaves like a formal parameter**. For a function with k "ordinary" parameters, let l be the list of booleans

$$l' = \text{U\_BoolList}(\text{TRUE}, l)$$

Managing Static Links

- The newFrame(label ,l’) makes the frame with the “extra” parameter

- **An example** : function f (x,y) is nested inside function g

  Tr_newLevel(levelg, f, U_BoolList(FALSE),

  U_BoolList(FALSE,NULL))

  F_newFrame(label, U_BoolList(TRUE, fmls))

Keeping Track of Levels

- The level for the “main” Tiger program is obtained by calling **Tr_outermost()**

- The function transDec will **make a new level**
- Tr_newLevel must be told the enclosing function’s level

- transDec can pass a level to transExp.

The end of Chapter 6(2)