

Compiler Construction

~ Clever Translations ~

Translating expressions

How to translate the expression
 $\alpha < \beta$ in HIR?

Naive translation

```
eseq
  seq
    cjump ( $\alpha < \beta$ ) ltrue lfalse
    label ltrue
    move temp t const 1
    jump lend
    label lfalse
    move temp t const 0
    label lend
  seq end
temp t
```

Closer look to the naive translation

Naive translation is costly

- one cjump
- one jump
- two label
- one temporary

⇒ Can we do better?

Note: *jumps and cumps are costly is the microprocessor*

⇒ We must try to minimize them!

Can we exploit additional information?

```
let  
    ...  
in  
     $\alpha < \beta$ ,  
    ...  
end
```

In this situation we don't care about the translation of $<$

\Rightarrow We are only interested about side effect of α and β

Improved translation

```
seq  
  sxp  $\alpha$   
  sxp  $\beta$   
seq end
```

Improved translation

```
seq  
  sxp  $\alpha$   
  sxp  $\beta$   
seq end
```

- 0 cjump / 0 jumps
- 0 label
- 0 temporary

⇒ Better!

Yet another example

```
let
  ...
in
  if  $\alpha < \beta$ 
  then /*TRUE*/
  else /*FALSE*/
end
```

In this situation a naive translation would produce a lot of useless jump/cjump

Improved translation

```
cjump  $\alpha < \beta$  ltrue, lfalse  
label ltrue  
/* TRUE translation */  
label lfalse  
/* FALSE translation */
```

Only one cjump (and one jump at the end of ltrue)!

⇒ Better than the naive translation!

Translating Conditions

What is the right translation for $\alpha < \beta$,
with α and β two arbitrary expressions?

It depends on the *use*:

- 1 `if $\alpha < \beta$ then ...`
- 2 `a := $\alpha < \beta$`
- 3 `($\alpha < \beta$, ())`.

Problem statement

When the visitor is about to translate $\alpha < \beta$, it does not know the context.

Context Sensitive Translation

- The right translation depends upon the *use*. **This is context sensitive!**
- How to implement this?
 - ▶ When entering an *IfExp*, warn “I want a condition”,
 - ▶ then, depending whether it is an expression or a statement, warn “I want an expression” or “I want a statement”.
- Don't forget to preserve the demands of higher levels...
- Eek.

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)			
Cx(a < b)			
Nx(s)			

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)		
Cx(a < b)			
Nx(s)			

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	
Cx(a < b)			
Nx(s)			

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	cjump($e \neq 0$, t, f)
Cx(a < b)			
Nx(s)			

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	cjump($e \neq 0$, t, f)
Cx(a < b)	seq(sxp(a), sxp(b))		
Nx(s)			

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	cjump($e \neq 0$, t, f)
Cx(a < b)	seq(sxp(a), sxp(b))	eseq($t \leftarrow (a < b)$, t)	
Nx(s)			

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	cjump($e \neq 0$, t, f)
Cx($a < b$)	seq(sxp(a), sxp(b))	eseq($t \leftarrow (a < b)$, t)	cjump($a < b$, t, f)
Nx(s)			

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	cjump(e \neq 0, t, f)
Cx(a < b)	seq(sxp(a), sxp(b))	eseq(t \leftarrow (a < b), t)	cjump(a < b, t, f)
Nx(s)	s		

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	cjump(e \neq 0, t, f)
Cx(a < b)	seq(sxp(a), sxp(b))	eseq(t \leftarrow (a < b), t)	cjump(a < b, t, f)
Nx(s)	s	error	

Prototranslation, Expression Shells

Rather, delay the translation until the use is known:

- Ex** Expression shell, encapsulation of a proto value,
- Nx** Statement shell, encapsulating a wannabe statement,
- Cx** Condition shell, encapsulating a wannabe condition.

Then, ask them to finish their translation according to the use:

Exp	un_nx	un_ex	un_cx (t, f)
Ex(e)	sxp(e)	e	cjump(e \neq 0, t, f)
Cx(a < b)	seq(sxp(a), sxp(b))	eseq(t \leftarrow (a < b), t)	cjump(a < b, t, f)
Nx(s)	s	error	error

if 11 < 22 | 22 < 33 then print_int(1) else print_int(0)

```
cjump ne
  eseq seq cjump 11 < 22 name 10 name 11
    label 10 move temp t0 const 1
      jump name 12
    label 11 move temp t0
      eseq seq move temp t1 const 1
        cjump 22 < 33 name 13 name 14
        label 14
        move temp t1 const 0
        label 13
      seq end
      temp t1
    jump name 12
  label 12
    seq end
    temp t0
  const 0
  name 15
  name 16
label 15  sxp call name print_int const 1
          jump name 17
label 16  sxp call name print_int const 0
          jump name 17
label 17
```

A Better Translation: Ix

```
seq
  cjump 11 < 22 name 13 name 14
  label 13
    cjump 1 <> 0 name 10 name 11
  label 14
    cjump 22 < 33 name 10 name 11
  seq end
label 10
  sxp call name print_int const 1
  jump name 12
label 11
  sxp call name print_int const 0
label 12
```

Summary

Naive
translation

Proto
translation