



lab5

Otimizações



Lab5

No lab 4 nós criamos um programa na linguagem intermediária do LLVM.

Nesse lab nós vamos pegar os programas feitos pela nossa linguagem e vamos aplicar otimizações nele.

Para isso vamos usar as bibliotecas do LLVM para implementar um passe de otimização.



LLVM

Uma das ferramentas oferecidas pelo LLVM é o executável “opt”. Esse executável recebe um código na linguagem intermediária, executa uma otimização e retorna o resultado em um outro código na linguagem intermediária.

Ele também pode receber como parâmetro um binário que aplica uma otimização, e aplica ele em cima do código recebido.

Para não ter que alterar todo o llvm, nós vamos criar apenas esse binário que será usado pelo “opt”



Ambiente de desenvolvimento

Para facilitar a criação do ambiente de desenvolvimento nós estamos fornecendo o arquivo `llvm_doc.tar`

Esse arquivo é uma imagem para ser usada no “docker” que já contém o código do `llvm` com a versão que nós iremos trabalhar. O código também já passou por uma etapa de compilação para vocês não terem que re compilar ele todo do início. O ambiente também assume que o passe que vocês devem implementar está em uma pasta específica.



Docker

Já estão instalados nas máquinas do IC

Caso queiram instalar em uma máquina pessoal, talvez seja necessário executar

- `systemctl start docker`

Para usar o serviço é necessário estar no grupo docker.

- `usermod -aG docker <user_name>`

Talvez seja necessário reiniciar uma vez dentro do grupo.



Docker

- `docker load -i llvm_doc.tar`

Esse comando vai importar a imagem dentro do `llvm_doc.tar` para o docker. A imagem possui o identificador `llvm:0.2`



Docker

Para abrir o bash dentro do docker use o comando:

- `docker run -it -v SimpleMath:/usr/src/app/llvm-project/llvm/lib/Transforms/SimpleMath llvm:0.2`

Esse comando deve ser executado dentro do diretório que contém a pasta SimpleMath. -it Fará com que um terminal seja aberto para vocês poderem navegar. -v Fará a pasta SimpleMath ficar disponível dentro do diretório especificado depois dos ":". A pasta simple math estará disponível dentro da sua máquina e do docker, logo você pode editar os arquivos com o editor que quiser pela sua máquina.



Docker

Lembre-se que SEMPRE que você sair do docker, todas as mudanças que você fez FORA da pasta SimpleMath serão jogadas fora. Isso quer dizer que sempre que você usarem o docker vocês devem compilar o passe de novo!

Não criem arquivos importantes fora de SimpleMath!



Compilando o passe

Para compilar o passe, entre no diretório `/usr/src/app/build` e use o commando

- `cmake --build .`

O seu passe estará dentro do objeto em `/usr/src/app/build/lib/LLVMSimpleMath.so`



opt

Para executar o seu passe em um código .ll faça:

- `opt -load /usr/src/app/build/lib/LLVMSimpleMath.so -sm FILE_IN.ll > FILE_OUT.bc`

Dentro da pasta SimpleMath já existe SimpleMath.cpp. Esse passe é um exemplo para ajudar a te guiar pelas funcionalidades importantes para fazer esse lab. No comando acima, -sm é a flag que o programa pede para usar, ela não deve ser alterada.

A entrada do opt pode ser tanto um arquivo .ll quanto .bc, mas a saída é sempre .bc, vocês podem usar llvm-dis para transformar de volta em .ll



SimpleMath.cpp

Experimente executar o passe fornecido no laboratório. Esse passe não muda o código em nada, ele apenas imprime informações sobre o código.

Esse passe herda de `FunctionPass`, isso quer dizer que ele é feito em cima de funções isoladas (não precisa do conhecimento de outras funções). A parte principal do passe vai estar na função `runOnFunction`. O `opt` vai chamar esse método para todas as funções do código.

Nesse passe vocês vão encontrar exemplos de como iterar sobre as instruções do código, ver o tipo da instrução e acessar seus operadores.



SimpleMath.cpp

Para o laboratório vocês devem MODIFICAR SimpleMath.cpp

O passe agora deve fazer duas coisas:

Opasse deve imprimir para cada temporário, em qual instrução ele é usado (ver exemplos de entrada e saída). Isso deve ser feito para o código inicial, não o otimizado.

Encontrar casos de instruções algébricas que são feitas sobre duas constantes. Remover essa instrução e modificar as instruções que usavam o resultado dela como operador pelo valor resultante da operação.

Isso deve ser repetido até que não existam mais casos de operações aritméticas sobre duas constantes.



Arquivos de teste

Vários testeX.ll estão disponíveis junto com o lab.

Na pasta result existem arquivos chamados testeX.res, esses arquivos são o .bc que devem sair depois da otimização aplicada em cima do testeX.ll, perceba que em vários deles nenhuma otimização é feita.

Os arquivos testeX.txt contém o resultado da saída de erros do programa, que é o que vocês devem imprimir para verificar até qual instrução um temporário ainda precisa existir.



lab5.2

Na atividade que foi exposta até agora não existe nenhum algoritmo que se aproxime da complexidade dos que foram visto em aula. Ela serve para dar a oportunidade de vocês se acostumarem com o ambiente do LLVM e terem ideia de como funciona o código de um compilador real.

Esse lab vai possuir uma segunda parte, onde vocês vão usar o que aprenderam nesse para fazer um passe um pouco mais complexo. O enunciado da segunda parte do lab será disponibilizado depois.



Se guiando pelo código do LLVM

O código do LLVM é MUITO extenso. Uma forma de se guiar por ele é pelo site <http://llvm.org/doxygen/>

A versão do LLVM do docker é a mais recente e deve ser compatível com a documentação do site.

Se agente der azar do LLVM adiantar uma versão durante o tempo do lab é possível que elas fiquem incompatíveis.

Nesse lab deem prioridade por postar suas dúvidas no Classroom da turma. Como vocês terão que mergulhar um pouco no código do LLVM, é muito possível que outros alunos possuam a mesma dúvida depois.