SOPAS - Submissão Online Para Análise de Software (fase 3)

José Pedro Silva Pedro Faria Ulisses Costa

Engenharia de Linguagens
Projecto integrado

June 27, 2011

Index

Até agora:

Concretizado até ao ínicio da terceira fase:

- Estudo do language.C ✓
- Estudo das métricas ✓
- Maturação da WebApp ✓
- Inicio da implementação do acesso pelo Terminal ✓

Motivação e Objectivos

Objectivos para terceira fase:

- Terminar a aplicação web esteticamente e adicionar funcionalidades extra
- Início da implementação de uma *script* de instalação do sistema
- Implementação de algumas métricas
- Melhoramento no interface pelo Terminal

Index

Estado da aplicação Web

A aplicação está praticamente terminada, o que se pretende fazer daqui para a frente será melhorar o que está feito e adicionar alguns extras. O que estava implementado até à entrega da fase anterior:

- Criação de contas de utilizador (grupo)
- Associação de concorrentes a determinado grupo
- Criação de concursos
- Criação de enunciados (através da interface web ou submetendo em formato xml)

Estado da aplicação Web (parte 2)

- Inserção de baterias de teste para os enunciados
- Submissão de programas para avaliação
- Adição simplificada de novas funções de avaliação ou de novas linguagens de programação ao sistema
- Apresentação de resultados referentes às diversas tentativas

Novidades

- Limaram-se alguns aspectos na interface, a nível de acessibilidade
- Alterou-se o script de detecção de clones, escrito em perl, de forma a interagir melhor com a aplicação
- Adicionou-se à aplicação a funcionalidade de detecção de clones

Modo de utilização do script de detecção de clones (CloneDt.pl)

Modo de utilização

perl cloneDt.pl -file [path1] -comp [path2]

- em que [path1] representa o path do ficheiro que se acabou de submeter, e o path2 representa o path do ficheiro com o qual se pretende comparar o primeiro

Detalhes do script de detecção de clones (CloneDt.pl)

- utiliza o comando

Comando ctags

```
ctags -x [path]
```

de forma a obter as linhas referentes ao início de cada função

- lê o ficheiro e guarda o código de cada função, numa posição de array diferente
- remove espaços em branco e comentários
- substituí strings por 'S', números por '1' e variáveis por 'var'

Detalhes do script de detecção de clones (CloneDt.pl)

- repete o mesmo processo para o segundo ficheiro
- compara cada elemento do primeiro array gerado com os elementos do segundo array
- regista cada vez que encontra uma função que pode ser cópia de outra
- no fim imprime para o stdout a percentagem de funções que foram assinaladas como clones

Detecção de clones na aplicação

- cada vez que um utilizador submete uma proposta de resolução, o seu código é comparado com o código das tentativas mais recentes dos restantes grupos, para o mesmo enunciado
- quando encontra possíveis clones, adiciona uma entrada na base de dados

Index

Melhoramentos

- Implementação com sucesso de um sistema por comandos.
- Criação de um módulo, `Access.pm`, de comunicação entre a camada de dados.
  - Usando módulo `Moose`
- Aumento da área de cobertura entre a interface e o sistema.

Index

Instalação do sistema

- Implementar uma *script* de instalação de todo o *software* envolvido no sistema
- *Script* em *bash*

Vantagens

- Facilitar a manutenção sistema
- Facilidade em migrar o sistema
- Gerir versões de *software* usados pelo sistema
- Novas *skills* de administração de sistemas

Software a instalar

De momento, a *script* instala:

- `Perl` e módulos
- Bibliotecas `C`
- `Haskell` e derivados
- `Ruby`, `Rails` e `Gems`

Início I - Verificação do utilizador

A primeira função invocada:

Check_user_id

```
andlogfile="tee -a main.log"

function check_user_id {
        echo "`whoami` started this script installation file at `date`" |
             $andlogfile
        if [ ! "`whoami`" = "root" ]; then
                echo "Not running as root. Yes, this is an installation file..."
                      | $andlogfile
                exit 1 ;
        fi
}
```

Início II - Verificação do sistema

install_package

```
function install_package {
        case `uname -s` in
                "Darwin")       install_macosx
                                ;;
                "Linux")        case `uname -v` in
                                *"Ubuntu"*)   install_ubuntu
                                              ;;
                                *)            echo "Your Linux is not supported
                                      yet. If it does have a packet manager
                                      please send an email to $admin_email"
                                              exit 1;
                                              ;;
                                esac
                        ;;
                *) echo "Your operative system is not supported yet. Please send
                       an email to $admin_email"
                         exit 1;
                         ;;
        esac
}
```

Exemplo I - Invocação da função geral

install_macosx

```
function install_macosx {
        echo "Working on a MacOSX machine" | $andlogfile
        build_macosx
        $portins gd2
        install_perl_mac
        install_perl_modules
}
```

Exemplo II - `Haskell` e derivados

Função `build_macosx` trata de instalar:

- `GHCi` - Compilador de `Haskell`, juntamente com o interpretador.
- `Happy` - Gerador de *parsers* desenvolvido em `Haskell`
- `Alex` - Gerador de analisadores léxicos desenvolvido em `Haskell`
- `Language.C` - Biblioteca do `Haskell` para análise e geração de código `C`

Exemplo III - Instalação

build_macosx

```
function build_macosx {
        is_ghc_installed
        if [ $? -eq 1 ]; then
                echo "GHC is installed, I will continue..."
                is_ghc_package_installed "happy"
                if [ $? -eq 0 ]; then
                        echo "Happy is not installed, I will install"
                        $portins hs-happy
                fi
                is_ghc_package_installed "alex"
                if [ $? -eq 0 ]; then
                        echo "Alex is not installed, I will install"
                        $portins hs-alex
                fi
                is_ghc_package_installed "language"
                if [ $? -eq 0 ]; then
                        echo "Language.C is not installed, I will install"
                        build_language_c
                        cd Parser/language-c-0.3.2.1/
                        runhaskell Setup.hs install
                        cd -
                fi
```

Index

Strafunski

- SYB (Scrap Your Boilerplate[1]) - Programação genérica
- Implementa estratégias no paradigma funcional

Data.Data (SYB)

$$gfoldl :: (c\ (d \to b) \to d \to c\ b) \to (g \to c\ g) \to a \to c\ a$$

$$gunfold :: (c(b \to r) \to c\ r) \to (r \to c\ r) \to Constr \to c\ a$$

$$gmapT :: (b \to b) \to a \to a$$

[1] http://www.cs.uu.nl/wiki/GenericProgramming/SYB

Porquê Strafunski? - Language.C exemplo

void fun(int a, int b);

```
CTranslUnit
[CDeclExt
(CDecl
[CTypeSpec (CVoidType(NodeInfo("main.c", 2, 1) (Name { nameId = 1 })))]
[ (Just
(CDeclr (Just "fun")
[CFunDeclr (Right
([CDecl [CTypeSpec (CIntType (NodeInfo ("main.c",2,11) (Name {nameId = 4})))]
[(Just (CDeclr (Just "a")[] Nothing[] (NodeInfo("main.c", 2, 15)
(Name { nameId = 5 })))
,Nothing, Nothing)] (NodeInfo("main.c", 2, 11) (Name { nameId = 6 }))
,CDecl [CTypeSpec(CIntType(NodeInfo("main.c", 2, 18) (Name { nameId = 8 })))]
[(Just(CDeclr(Just "b") [] Nothing [] (NodeInfo ("main.c",2,22)
(Name {nameId = 9})))
,Nothing,Nothing)]
(NodeInfo ("main.c",2,18) (Name {nameId = 10}))],False))
[] (NodeInfo ("main.c",2,10) (Name {nameId = 11}))
]
Nothing [] (NodeInfo ("main.c",2,6) (Name {nameId = 2}))
)
,Nothing
,Nothing
)
```

Porquê Strafunski? - Como tratar arvores grandes (sem SYB)

Muito trabalhoso

- Ter uma função para cada tipo de dados (no caso do Language.C 28)
- Ter uma acção para cada constructor de tipos (no caso do Language.C 84)

Rápidamente se atinge perto de 200 linhas de código para fazer uma travessia na árvore e executar uma acção num constructor[2].

[2] https://github.com/ulisses/Static-Code-Analyzer/blob/95032a7e3eaf52e0dc6c36d3c1039519bc5fe689/Parser/Main.hs

Porquê Strafunski? - Como tratar arvores grandes (com SYB)

Pouco trabalhoso, genérico

- Definir a acção que se quer tomar sobre o nodo em causa
- Definir uma estratégia

Com pouco código definimos uma estratégia e uma acção sobre a árvore de parsing

Strafunski - Estratégias

Para a aplicação de uma estratégia a uma instância do nosso tipo de dados t iremos usar:

$$applyTP :: (Monad\ m, Term\ t) \Rightarrow TP\ m \rightarrow t \rightarrow m\ t$$

$$applyTU :: (Monad\ m, Term\ t) \Rightarrow TU\ u\ m \rightarrow t \rightarrow m\ u$$

Permite adicionar estratégias:

$$adhocTP :: (Monad\ m, Term\ t) \Rightarrow TP\ m \rightarrow (t \rightarrow mt) \rightarrow TP\ m$$

$$adhocTU :: (Monad\ m, Term\ t) \Rightarrow TUam \rightarrow (t \rightarrow mu) \rightarrow TU\ u\ m$$

Strafunski - Estratégias 2

Permite executar estratégias em sequência:

$$seqTP :: Monad\ m \Rightarrow TP\ m \rightarrow TP\ m \rightarrow TP\ m$$

$$seqTU :: Monad\ m \Rightarrow TP\ m \rightarrow TU\ u\ m \rightarrow TU\ u\ m$$

Para tentar usar estratégias diferentes:

$$choiceTP :: MonadPlus\ m \Rightarrow TP\ m \rightarrow TP\ m \rightarrow TP\ m$$

$$choiceTU :: MonadPlus\ m \Rightarrow TU\ u\ m \rightarrow TU\ u\ m \rightarrow TU\ u\ m$$

Strafunski - Estratégias 3

Aplica esta estratégia a todos os subtermos imediatos, para o **TU** os resultados são reduzidos com a função do monoid $+$:

$$allTP :: Monad\ m \Rightarrow TP\ m \rightarrow TP\ m$$

$$allTU :: (Monad\ m, Monoid\ u) \Rightarrow TU\ u\ m \rightarrow TU\ u\ m$$

Esta função permite aplicar a estratégia ao primeiro filho que aparecer da esquerda para a direita:

$$once\_tdTP, once\_buTP :: MonadPlus\ m \Rightarrow TP\ m \rightarrow TP\ m$$

$$once\_tdTU, once\_buTU :: MonadPlus\ m \Rightarrow TU\ u\ m \rightarrow TU\ u\ m$$

TU vs TP

```
full_tdTU   :: (Monad m, Monoid a) => TU a m -> TU a m
full_tdTU s =  op2TU mappend s (allTU' (full_tdTU s))


class Monoid a where
  mempty :: a
  mappend :: a -> a -> a
  mconcat :: [a] -> a
    -- Defined in Data.Monoid


full_tdTP   :: Monad m => TP m -> TP m
full_tdTP s =  s `seqTP` (allTP (full_tdTP s))
```

Strafunski - Exemplos

Contar o número de *ifs*, *switches* e ciclos (McCabeIndex−−).

```
testMcCabe :: IO Int
testMcCabe =  parse >>= mcCabeIndex . fromRight
    where fromRight = (\(Right prog) -> prog)
          parse = parseCFile (newGCC "gcc") Nothing ["-U__BLOCKS__"] "main.c"

instance Num a => Monoid a where
    mappend = (+)
    mempty = 0

mcCabeIndex :: Data a => a -> IO Int
mcCabeIndex =  applyTU (full_tdTU loopCond)

loopCond = constTU 0 `adhocTU` (return . action)

action :: Num a => CStat -> a
action (CIf _ _ _ _)     = 1
action (CSwitch _ _ _)   = 1
action (CWhile _ _ _ _) = 1
action (CFor _ _ _ _ _) = 1
action _                 = 0
```

Strafunski - Exemplos

Extrair as assinaturas de todas as funções.

```
getFunctionsSign :: IO [CTranslUnit]
getFunctionsSign = parse >>= return . getFunSign . fromRight
    where fromRight = (\(Right prog) -> prog)
          parse = parseCFile (newGCC "gcc") Nothing ["-U__BLOCKS__"] "main.c"

getFunSign :: Data x => x -> [x]
getFunSign = applyTP (topdown names1)
    where names1 = idTP `adhocTP` (return . fromFunctionToSign)

fromFunctionToSign (CFDefExt (CFunDef lCDeclSpec cDeclr _ _ _ ))
    = CDeclExt (CDecl lCDeclSpec [(Just $ cDeclr,Nothing,Nothing)] internalNode)
```

Index

Conclusão e trabalho futuro

- Implementação das restantes métricas descritas no relatório
- Melhorar a utilização pelo terminal (permitir escrita)

Perguntas

?