Equipe abnTEX2

Modelo Canônico de Trabalho Acadêmico com abnTEX2

Brasil

2018, v<VERSION>

Equipe abnTEX2

Modelo Canônico de Trabalho Acadêmico com abnTEX2

Modelo canônico de trabalho monográfico acadêmico em conformidade com as normas ABNT apresentado à comunidade de usuários LATEX.

Universidade do Brasil – UBr

Faculdade de Arquitetura da Informação

Programa de Pós-Graduação

Orientador: Lauro César Araujo

Coorientador: Equipe abnTEX2

Brasil

2018, v<VERSION>

Resumo

Segundo a **??**, 3.1-3.2), o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (…) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

Palavras-chave: latex. abntex. editoração de texto.

Abstract

This is the english abstract.

Keywords: latex. abntex. text editoration.

Sumário

1 Introdução

Ola Pits (PITTS, 2013)

Fixpoint

2 Referencial Teórico

3 Metodologia

Teoria de tipos dependentes.

Termos convertíveis são iguais no Coq. Tipo indutivo eq. "=" é a igualdade padrão no Coq "≡" igualdade *setoid*. Pode-se substituir termos iguais em qualquer expressão pois eles são convertíveis (igualdade de Leibniz). Mas ao introduzir uma noção diferente de equivalência, aonde termos não convertíveis são equivalentes, limitamos a reescrita apenas somente a subtermos que são argumentos de funções que respeitam a equivalência, chamadas de funções próprias em relação a equivalência. A igualdade é "forte" demais, em alguns casos temos objetos que são conceitualmente iguais, mas sintaticamente diferentes. A igualdade no Coq é sintática. Coq não tem tipos quocientes (e não é uma boa ideia implementar PROCURAR REFERÊNCIA checagem de tipos indecidível), utiliza-se *setoids* (conjuntos de Bishop). A substituição de subtermos equivalentes é chamado de reescrita de setoids, e é necessário o gerenciamento correto da aplicação de lemas próprios de funções e equivalências. Sem uma infraestrutura adequada as provas começam a ficar incontroláveis e grandes, aonde surge o setoid hell. Coq possui um mecanismo que da suporte a reescrita setoid baseado em classes de tipos.

Falar sobre classes de tipos. É semelhante a implementação em Haskell (falar pq isso é legal tipos de classes), mas no Coq são cidadões de primeira classe graças a teoria de tipos dependentes. Classes são implementadas como records (registros) aliado a tipos implícitos e busca de provas. Registros em Coq são dependentes, ou seja, um membro do registro pode referenciar um membro anterior. Instâncias das classes são instâncias ordinárias dos registros, mas cada instância é registrada em um banco de dados de "sugestões" para a busca de provas (semelhante ao Haskell), entretanto Coq permite mais de uma instância de classe (contrário do Haskell), pois pode-se ter mais de uma instância de um registro.

DIFERENÇA ENTRE SET E PROP

4 Formalização

Uma introdução a formalização

4.1 Átomos e Permutação

O interessante sobre nome é seu identificador, nome do nome cadeia de caracteres assim como ponteiros, em outras linguagens de programação. Sua estrutura (implementação) dos é irrelevante, por isto são denominados átomos, outros autores dão nome de "nomes puros". Entretanto, como sempre precisamos de nomes frescos, é preciso uma fonte inesgotável e uma forma de distingui-los: átomos é um conjunto infinito contável com igualdade decidível. Módulos do Coq permite fornecer uma interface, definindo um tipo de módulo, capaz de ocultar (como uma máscara) a implementação de um módulo. Abaixo definimos um tipo de módulo `ATOMIC` que esconde a implementação.

```
Module Type ATOMIC.
  Parameter t : Set.
  Axiom cnt : Countable t.
  Axiom dec : EqDecision t.
  Axiom inf : Infinite t.
End ATOMIC.
```

```
Module Atom : ATOMIC.
  Definition t := nat.
  Instance cnt : Countable t := nat_countable.
  Instance dec : EqDecision t := nat_eq_dec.
  Instance inf : Infinite t := nat_infinite.
End Atom.
```

```
Notation name := Atom.t
Notation nameset := (listset name).
```

Implementamos átomos como naturais. Assinaturas (tipo de módulos) têm duas utilidades: garantir que um módulo implementa corretamente alguma regra e esconder informação. Somente declarações presentes na assinatura podem ser acessadas externa ao módulo. Como não estamos omitindo nenhuma definição, estamos escondendo a implementação de átomo. portanto Atom.t é abstrato. À esquerda definimos um novo tipo de módulo, denominado `ATOMIC`, com um tipo abstrato `t`, com instâncias nas classes `EqDecision`, igualdade decidível, e `Infinite`. Construímos um novo tipo abstrato `Atom.t` com igualdade decidível `Atom.dec` e infinito `Atom.inf` no qual chamamos de `name`. `nameset` conjunto finito de nomes representado por uma lista. Explicar as type classes EqDecision e Infinite, que propriedades elas agregram ao desenvolvimento. Por exemplo EqDecision dá acesso a `decide` a partir de uma proposição, enquanto Infinite da acesso a `fresh` para gerar uma átomo novo.

Permutações é representado por uma lista de pares de nomes: Porque agora esse trem ficou longe?

```
Notation perm := (list (name * name)).
```

```
Notation "( a , b )" := (@cons (name * name) (a,b) (@nil _)).
Definition swap '(a,b) : name → name :=
  λ c, if decide (a = c) then b else if decide (b = c) then a else c.
Definition swap_perm (p: perm): name → name :=
  λ a, foldl (λ x y, swap y x) a p.
Instance perm_equiv: Equiv perm :=
  λ p q, ∀ a, swap_perm p a = swap_perm q a.
```

O comando @ desativa o mecanismo de variáveis implícitas do Coq para fornecermos manualmente todos os parâmetros.

Entretanto, precisamos garantir as algumas propriedades: um conjunto infinito contável distinguíveis.

nomes é um conjunto infinito e contável.

sendo sua estrutura irrelevante.

estamos interessados apenas nos identificadores que os nomes carregam

Nomes na teoria nominal, é um conjunto infinito e contável.

Nomes, uma das peças centrais da teoria nominal, tem uma implementação trivial. As características que nos interessam são:

O que nos interessa de nomes são seus identificadores, cadeia de caracteres, estrutura interna é irrelevante, por isso o chamamos de átomos, também são conhecidos como "nomes puros". Nomes são indivisíveis (opacos), infinitos contáveis com igualdade decidível. Não pode haver dúvidas quando dois nomes são iguais ou diferentes. Dado um conjunto **qualquer** de nomes **sempre** podemos obter um novo.

Implementação de permutação e átomos é a mesma para ambos.

```
Class GAction `(Group G) (X : Type) `{Act : Action G X, Equiv X} : Prop := {
        gact_setoid :> Equivalence(≡@{X});
        gact_proper :> Proper ((≡@{G}) ==> (≡@{X}) ==> (≡@{X})) ();
        gact_id : ∀ (x: X), ε@{G} • x ≡@{X} x;
        gact_compat: ∀ (p q: G) (x: X), p  (q  x) ≡@{X} (q + p)  x
}.
```

4.2 Copello

4.3 Pritam

5 Perspectivas Futuras

Referências

PITTS, A. M. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge: Cambridge University Press, 2013. (Cambridge Tracts in Theoretical Computer Science, 57). ISBN 9781139084673. Citado na página 7.