Criação de uma biblioteca de tipos de segunda ordem para HasCASL

Glauber Módolo Cabral – Orientando
Prof. Dr. Arnaldo Vieira Moura – Orientador

Universidade Estadual de Campinas
Instituto de Computação
Proposta de Mestrado em Ciência da Computação

Apoio: CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

Outubro de 2007

Roteiro

Contextualização

- CASL
- Haskell
- HasCASL
- HetCASL e Hets
- Isabelle

Proposta

Cronograma

CASL (Common Algebraic Specification Language)

- Linguagem de especificação algébrica;
- Criada por um grupo de pesquisadores para ser padrão na área;
- Une as boas características de outras linguagens;
- Permite extensões e sub-linguagens;
- Linguagem central de uma família de linguagens;
- Possui 4 partes semanticamente independentes;
- Possui uma biblioteca com exemplos de especificações.

CASL - 4 partes semanticamente independentes

- **Especificações Básicas:** declarações de tipos e operações; definições de operações; axiomas relacionando as operações;
- **Especificações Estruturais:** permitem a combinação de Especificações Básicas em especificações maiores;
- **Especificações Arquiteturais:** definem como combinar especificações em blocos de especificações dependentes entre si para serem implementadas e reusadas;
- **Especificações de Bibliotecas:** definem conjuntos de especificações, com controle de versão e distribuição via Internet.

Extensões e sub-linguagens são criadas alterando-se apenas as Especificações Básicas.

Haskell

- Linguagem funcional de programação: Implementa os conceitos de λ-Cálculo; programação através de aplicações de funções;
- Fortemente tipificada: todo elemento possui tipo (pré-definido ou calculado automaticamente no contexto);
- Avaliação preguiçosa: um argumento de uma função só é avaliado quando é usado no cálculo;
- Pura: não permite alterar o estado do sistema (efeitos colaterais) a menos das partes envolvidas no cálculo de uma função.

Haskell - Sintaxe e Semântica

- Tipos polimorfos: permitem sobrecarga através de parametrização;
- Construtores de tipos: permitem definir novos tipos de dados;
- Classes de tipos: permitem associar operadores sobrecarregados (*overloading*) a classes e definir, através de instâncias, quais tipos possuem as operações;
- Coincidência de padrões (*pattern matching*): permite o uso de padrões para verificar se uma estrutura tem as características desejadas;
- Mônadas são utilizadas para seqüenciar operações com efeitos colaterais, encapsulando as alterações.

Haskell - Prelude

- Biblioteca padrão implementada e distribuída por todo compilador Haskell;
- Possui funções de uso corriqueiro:
  - Tipos básicos (Integer, Char, String, Float, ...);
  - Manipulação de listas;
  - Manipulação de texto;
  - Mônadas para IO em tela e arquivo.

HasCASL

- Extensão de CASL com conceitos de lógica de segunda ordem: tipos que são funções; polimorfismo e construtores de tipos;
- Tem a linguagem de programação funcional Haskell como sub-conjunto;
  - Facilita a transformação da especificação em código Haskell executável;
- Possui uma lógica interna para permitir tipos e funções recursivos;
  - A lógica não é um conceito básico da linguagem (não precisa ser aprendido para o uso da linguagem);
- Possui avaliação estrita de parâmetros: parâmetros indefinidos sempre resultam em valores de retorno indefinidos;
- Não possui biblioteca com especificações prontas.

HetCASL (Heterogeneous CASL)

- Linguagem da família CASL que implementa as construções estruturais;
- Interliga as demais linguagens (interliga várias especificações básicas em linguagens diferentes);
- Preserva a ortogonalidade entre as lógicas de cada especificação básica;
- Possui construções para indicar as lógicas utilizadas ao traduzir-se uma especificação em outra.

Hets: Heterogeneous Tool Set

- Analisador sintático e gerenciador de provas.
- Implementado em Haskell;
- Gerencia ferramentas de provas para as lógicas utilizadas nas extensões e sub-linguagens de CASL:
  - SPASS: lógica de primeira ordem;
  - Isabelle: lógica de segunda ordem.
- Utiliza o editor Emacs como interface para:
  - formatação automática de código;
  - execução automática da ferramenta de grafos de desenvolvimento.

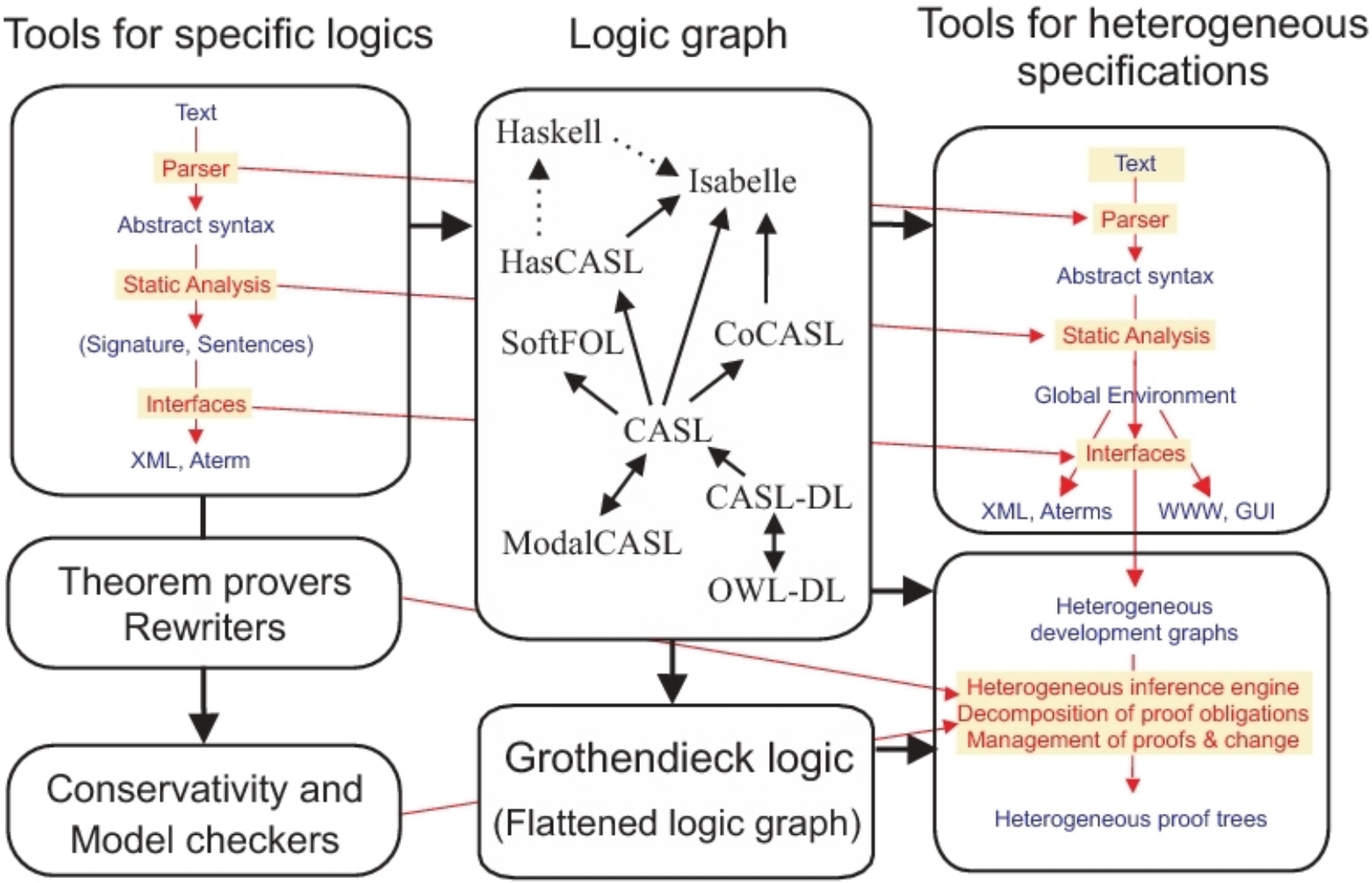
Hets - Grafos de desenvolvimento

- Grafo de lógicas e linguagens;
- Nós: especificações (completas ou não)
  - Assinatura;
  - Axiomas locais, herdados pelos nós dependentes através dos arcos de definição;
- Arcos:
  - de definição: dependência entre especificações e suas sub-especificações;
  - de teorema: criam relações entre teorias, indicando necessidades de provas que surgem no desenvolvimento;
    - globais: todos os axiomas válidos no nó fonte são válidos no nó alvo;
    - locais: apenas os axiomas definidos no nó fonte são válidos no nó alvo.

Isabelle

- Provador de teoremas genérico, semi-automático, que permite o uso de várias lógicas como cálculo formal;
- Automatiza alguns trechos repetitivos de provas: equações, aritmética básica e fórmulas matemáticas;
- Hets usa Isabelle com HOL (Higher-Order Language);
- A sintaxe de HOL assemelha-se à de Haskell;
- Hets traduz uma especificação em HasCASL para HOL de forma automática;

Architecture of the heterogeneous tool set Hets



Fonte: http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/index_e.htm

Proposta

Objetivo

Criar uma biblioteca para HasCASL e verificá-la através da ferramenta Hets.

Metodologia

- Reutilizar os tipos já especificados pela biblioteca de CASL;
- Incluir os tipos e funções presentes na biblioteca Prelude;
- Verificar formalmente as especificações criadas através da ferramenta Hets.

Motivação e Justificativa

- Criar a biblioteca contribui para difundir a linguagem;
- Permite o reuso em outras especificações;
- Tomar a biblioteca Prelude como base facilita a transformação de especificações em HasCASL para código executável em Haskell;
- Aprofundar conhecimentos em programação funcional e métodos formais.

Cronograma

1. Créditos obrigatórios do mestrado;
2. Estudo das linguagens HasCASL e CASL;
3. Estudo da ferramenta Hets e do provador Isabelle;
4. Estudo da bib. Prelude e de lógica de segunda ordem;
5. Implementação e verificação formal da biblioteca;
6. Escrita de um artigo para congresso;
7. Escrita da dissertação;
8. Defesa e revisão.

Atividade	2007					2008						2009
	3-4	5-6	7-8	9-10	11-12	1-2	3-4	5-6	7-8	9-10	11-12	1-2
1	•	•	•	•	•							
2		•	•									
3				•	•							
4						•	•					
5								•	•	•		
6										•		
7			•		•		•			•	•	
8												•

Agradecimentos

Obrigado!

Dúvidas?

Contato:
glauber.cabral@students.ic.unicamp.br

HasCASL - Modelo Henkin Intensional (1)

Modelo padrão

As funções parciais $s->?t$ são interpretadas pelo conjunto completo de funções parciais de s em t.

Modelo Henkin Extensional

As funções parciais são interpretadas por subconjuntos dos conjuntos anteriores de tal forma que todos os λ-termos possam ser interpretados (*comprehension*).

Modelo Henkin Intensional

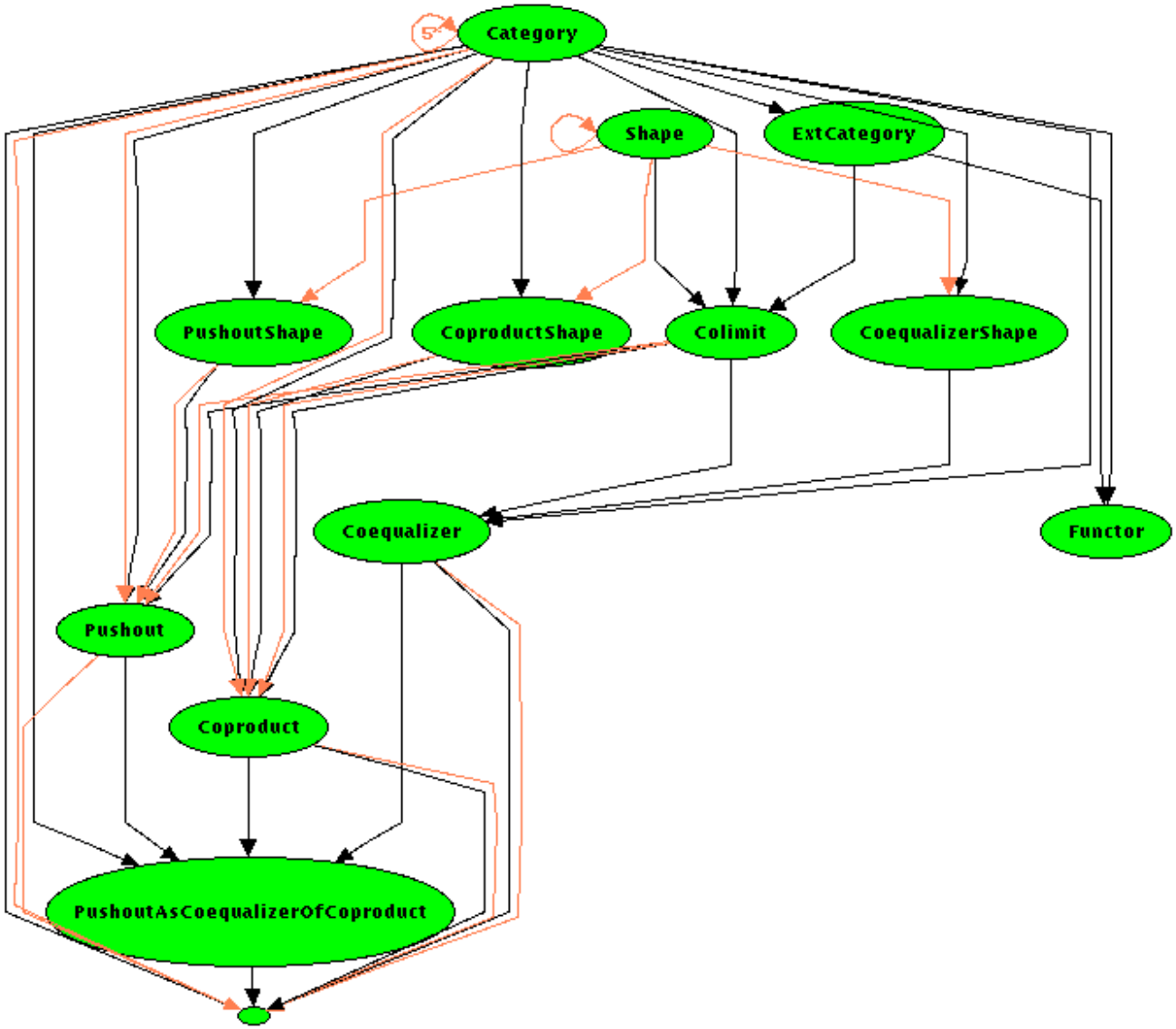
Um tipo de função é interpretado por um conjunto arbitrário equipado com operação de aplicação do tipo em questão. A interpretação dos λ-termos passa a ser parte da estrutura do modelo ao invés de um axioma existencial.

HasCASL - Modelo Henkin Intensional (2)

Vantagens

- Elimina os problemas de completude presentes em outros modelos;
- Permite funções parciais em modelos iniciais de assinaturas;
- Permite o uso da semântica operacional de Haskell ao invés do uso direto de lógica de segunda ordem.

Grafo de Desenvolvimento - Especificação de Categorias



State Monad - Haskell code

```
newtype State s a = State { runState :: (s -> (a,s)) }

instance Monad (State s) where
  return a         = State $ \s -> (a,s)
  (State x) >>= f =
    State $ \s -> let (v,s') = x s in runState (f v) s'
```

State Monad - HasCASL code

```
library State
logic HasCASL
spec State =
  sort S;
  op __>>=__ : forall a : Type . (S ->? (S * a)) *
                 (a ->? (S ->? (S * a))) ->? (S ->? (S * a));
     return  : forall a : Type . a ->? (S ->? (S * a));
  forall a : Type; x : a; m : S ->? (S * a);
         f : a ->? (S ->? (S * a))
    . return x = \ s:S . (s, x)                    %(return_def)%
    . m >>= f = \ s:S . let (s', a)
                            = m s in (f a) s'       %(bind_def)%
 then %implies
  forall a: Type; x: a; m: S ->? (S * a);
         f, g, h : a ->? (S ->? (S * a))
    . forall x:a
    . (f x >>= g) >>= h =
                      f x >>= (\ y:a . g y >>= h) %(monad_assoc)%
    . f x >>= return = f x                         %(monad_unit1)%
    . def (f x) => return x >>= f = f x           %(monad_unit2a)%
    . m >>= (\ x:a . return x >>= f) = m >>= f    %(monad_unit2b)%
```