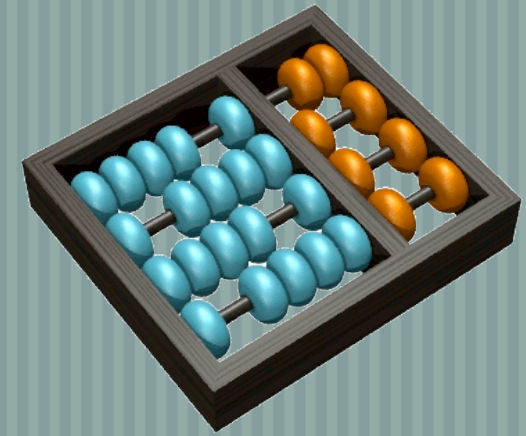


Universidade Estadual de Campinas
Instituto de Computação



Criação de uma Biblioteca Padrão para a Linguagem HasCASL

Glauber Módolo Cabral
Orientador: Arnaldo Vieira Moura

26 de Abril de 2010

Agenda

— [Introdução e Motivação

— [Linguagens envolvidas

— [Objetivo e Justificativa

— [Contextualização e Cenário de Pesquisa

— [Abordagem de Pesquisa

— [Exemplo de Especificação e Prova

— [Resumo das Especificações Realizadas

— [Contribuições

— [Trabalhos Futuros

— [Conclusões

— [Errata da Dissertação

— [Agradecimentos

Linguagens de Especificação

**Linguagens de
Especificação
Formal**

Z

Método B

Extended ML

CASL

HasCASL

Verificação e Tradução

**Linguagens de
Especificação
Formal**

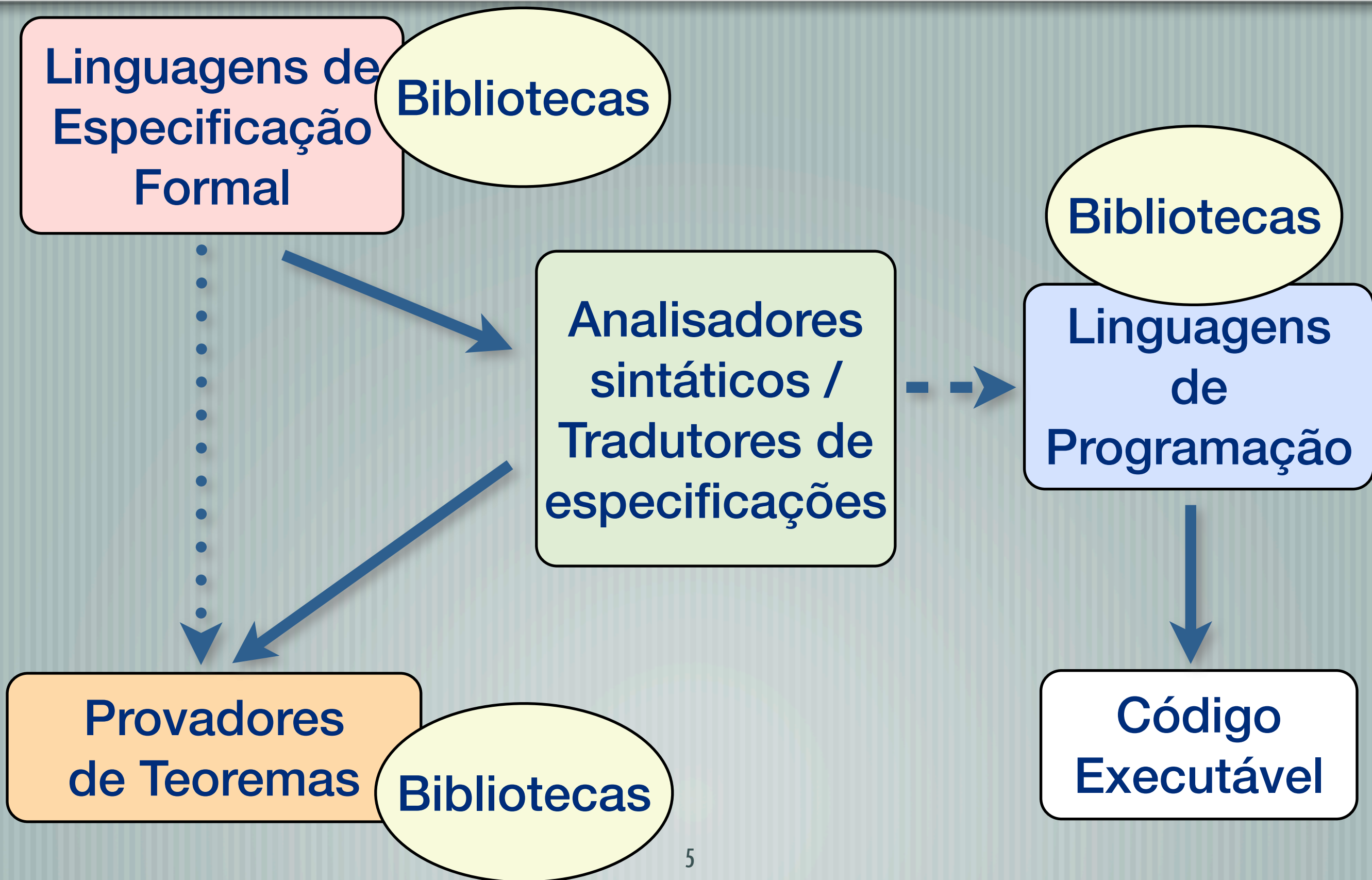
**Analísadores
sintáticos /
Tradutores de
especificações**

**Linguagens
de
Programação**

**Provadores
de Teoremas**

**Código
Executável**

Bibliotecas



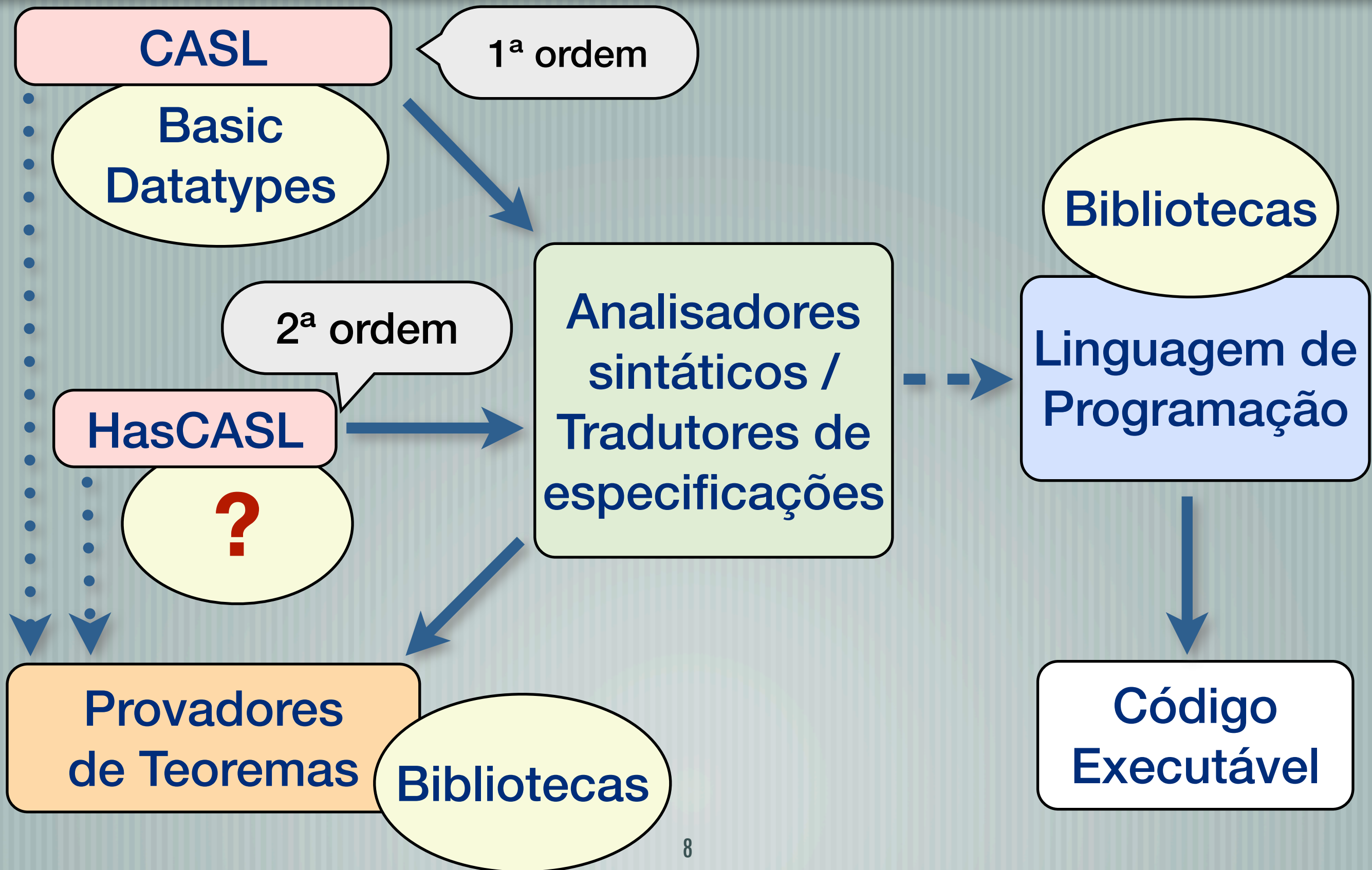
Linguagem CASL

- [Criada para ser um padrão na área de especificação formal
- [Possui as boas características das demais linguagens de especificação
- [Permite extensões para suportar novos paradigmas de linguagens
- [Semântica baseada em lógica de 1^a ordem
- [Possui biblioteca padrão de tipos de dados: Basic Datatypes

Linguagem HasCASL

- [Extensão da linguagem de especificação algébrica CASL para suportar lógica de 2ª ordem
- [Possui um subconjunto que pode ser traduzido diretamente para a linguagem de programação Haskell
- [Refinamento de especificações para este subconjunto permitirá geração automática de código
- [**Não possui biblioteca padrão de tipos de dados**

Estado da Arte



Objetivo

Especificar (e provar) uma biblioteca de tipos primitivos e funções de segunda ordem para a linguagem HasCASL

Justificativa

- [Tipos e funções da biblioteca Basic Datatypes:
 - Não possuem o poder de expressão esperado pela linguagem HasCASL por serem de 1ª ordem
 - Podem ser importados por especificações em HasCASL, mas suas especificações não possuem todos os lemas necessários para uso com o provador de teoremas
- [Escrever uma biblioteca com tipos de dados de segunda ordem colabora com a especificação de programas em HasCASL visando à tradução automática para Haskell

Ferramentas utilizadas

— [**Hets**

- Analisador sintático para as linguagens CASL e HasCASL
- Traduz especificações escritas em CASL e HasCASL para a linguagem HOL
- Gerencia provas através de um Grafo de Desenvolvimento

— [**Isabelle**

- Provedor de teoremas semiautomático
- Utiliza as linguagens HOL e HOLCF para escrever provas

Linguagem de Programação Alvo

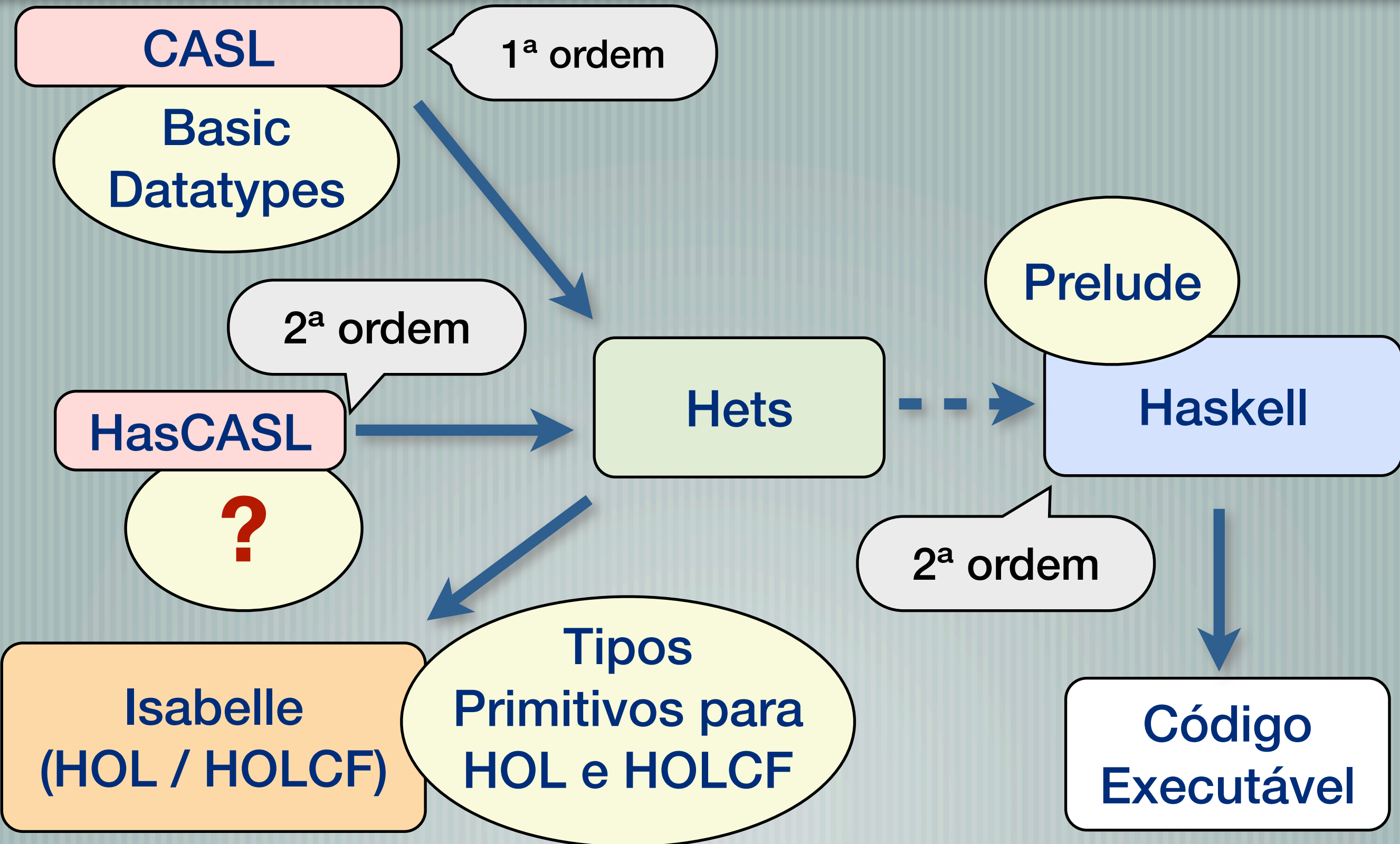
— [**Haskell**

- Linguagem de programação funcional
- Alvo da tradução automática de especificações HasCASL para código executável

— [**Prelude**

- Biblioteca padrão da linguagem Haskell
- Tipos de dados básicos: booleano, listas, caracteres, cadeias de caracteres, tipos numéricos, funtores e monadas
- Funções de manipulação de listas, de textos e de E/S

Cenário de Pesquisa



Abordagem de Pesquisa

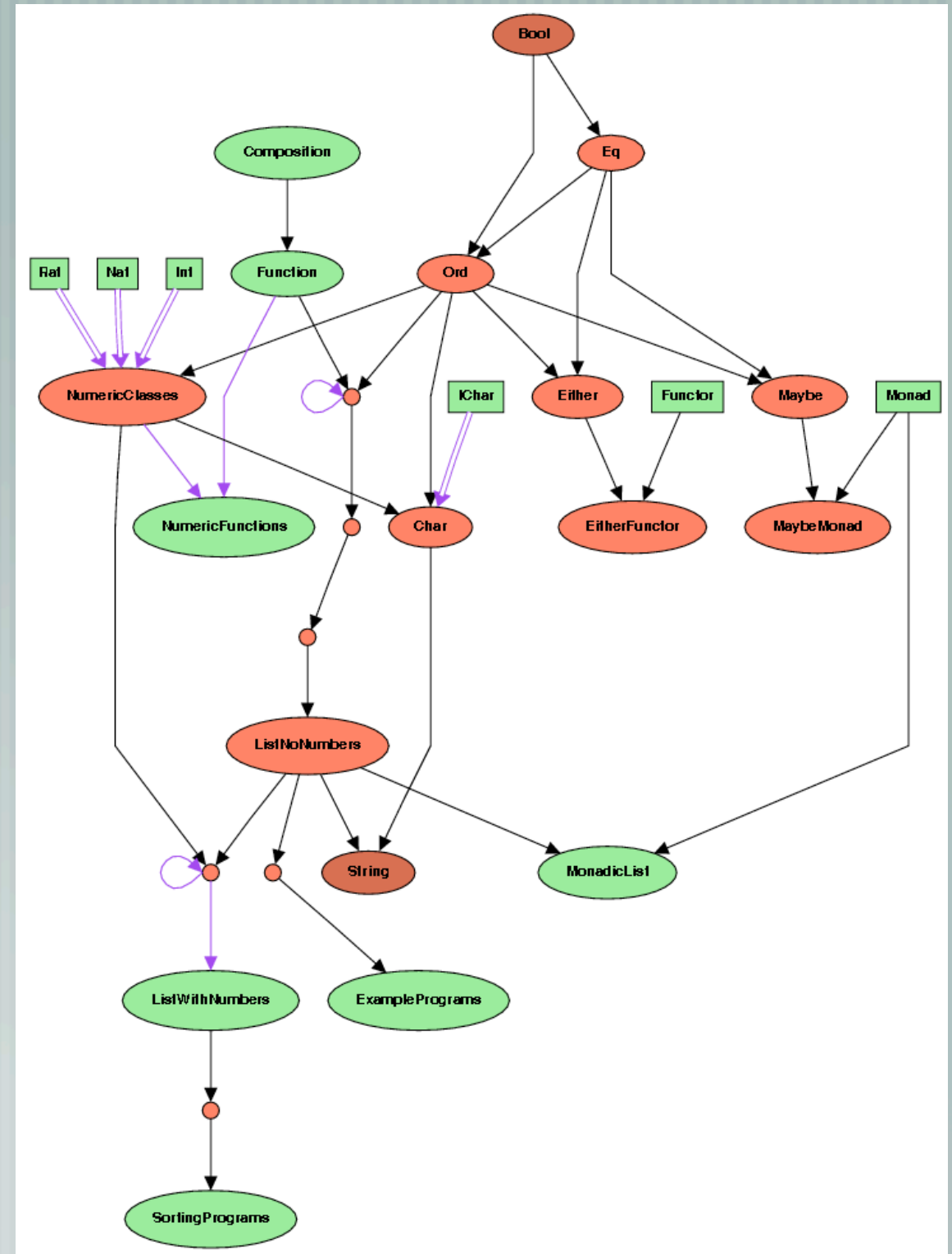
- [Especificar os tipos da biblioteca Prelude na linguagem HasCASL
- [Traduzir as especificações na linguagem HasCASL para a linguagem HOL com a ferramenta Hets
- [Escrever provas na linguagem HOL para propriedades das especificações
- [Verificar as provas escritas através do provador de teoremas Isabelle

Escolha inicial

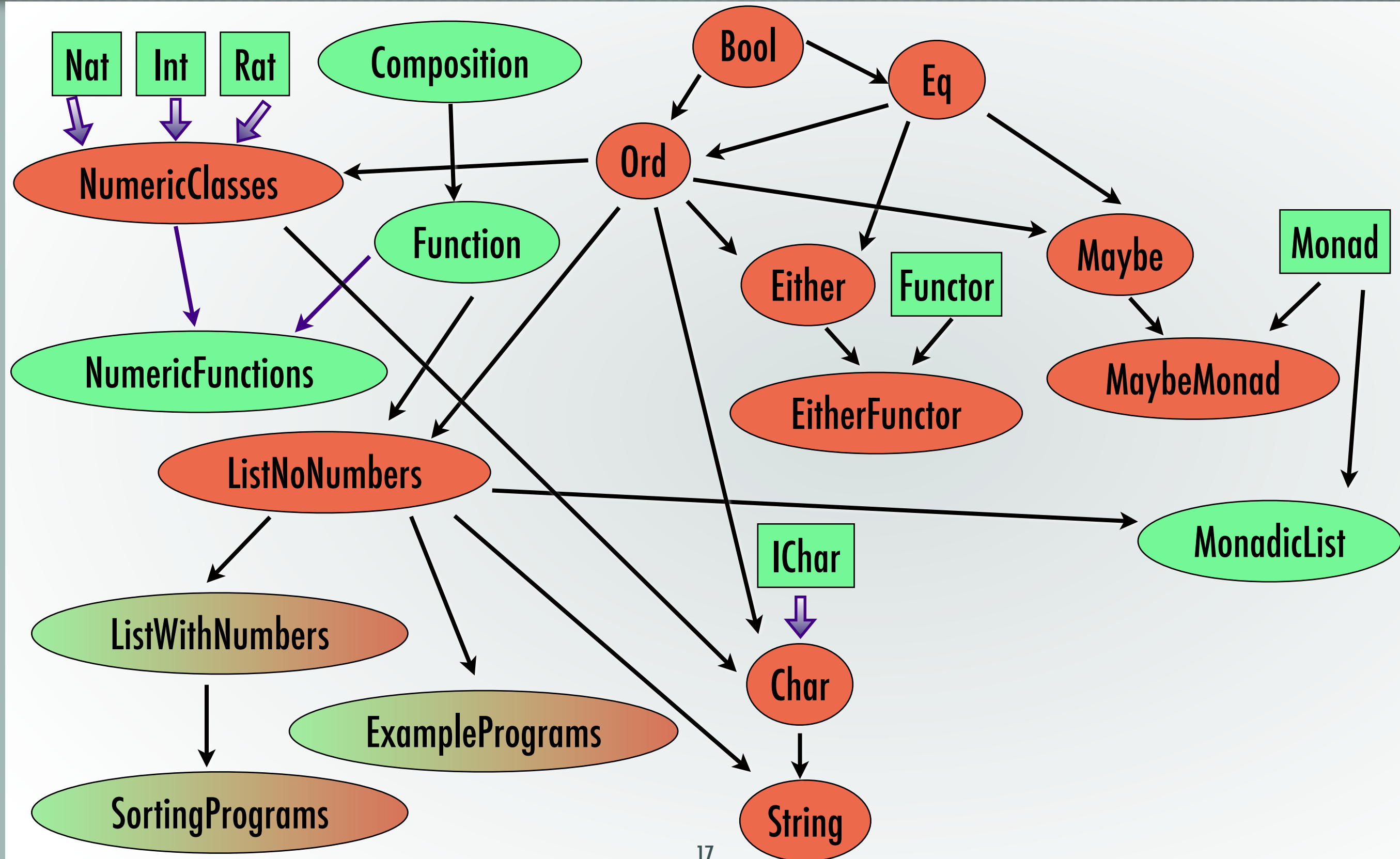
- [Iniciar com avaliação estrita e refinar para incluir avaliação preguiçosa
- [Avaliação estrita:
 - Emprega construções mais simples de HasCASL
 - Precisa de conhecimento básico de HOL para escrever as provas
 - Possui alguma documentação e alguns exemplos
- [Avaliação preguiçosa:
 - Emprega construções mais avançadas de HasCASL
 - Precisa de profundo conhecimento prévio das linguagens HOL e HOLCF para escrever as provas
 - Possui pouca documentação e exemplos

Especificações Desenvolvidas

- Nós elípticos /circulares representam especificações criadas
- Nós retangulares significam especificações importadas
- 18 especificações escritas
- Nós vermelhos possuem provas em aberto
- Nós verdes não possuem provas em aberto



Especificações Desenvolvidas



Especificação Ord em HasCASL

```
1 spec Ord = Eq and Bool then
2 free type Ordering ::= LT | EQ | GT
3 type instance Ordering: Eq
4 . (LT == LT) = True      %(IOE01)% %implied
5 . (LT == EQ) = False    %(IOE04)%
6 . (LT /= EQ) = True     %(IOE07)% %implied
7 class Ord < Eq {
8   fun __<__ : a * a -> Bool
9   var x, y, z, w: a
10
11 . (x==y) = True => (x<y) = False  %(LeIrreflexivity)%
12 . (x<y)=True /\ (y<z)=True => (x<z)=True %(LeTTransitive)%
13 . (x<y)=True => (y<x)=False %(LeTAsymmetry)% %implied
14 . (x<y)=True \/ (y<x)=True \/ (x==y)=True  %(LeTTotal)%
```

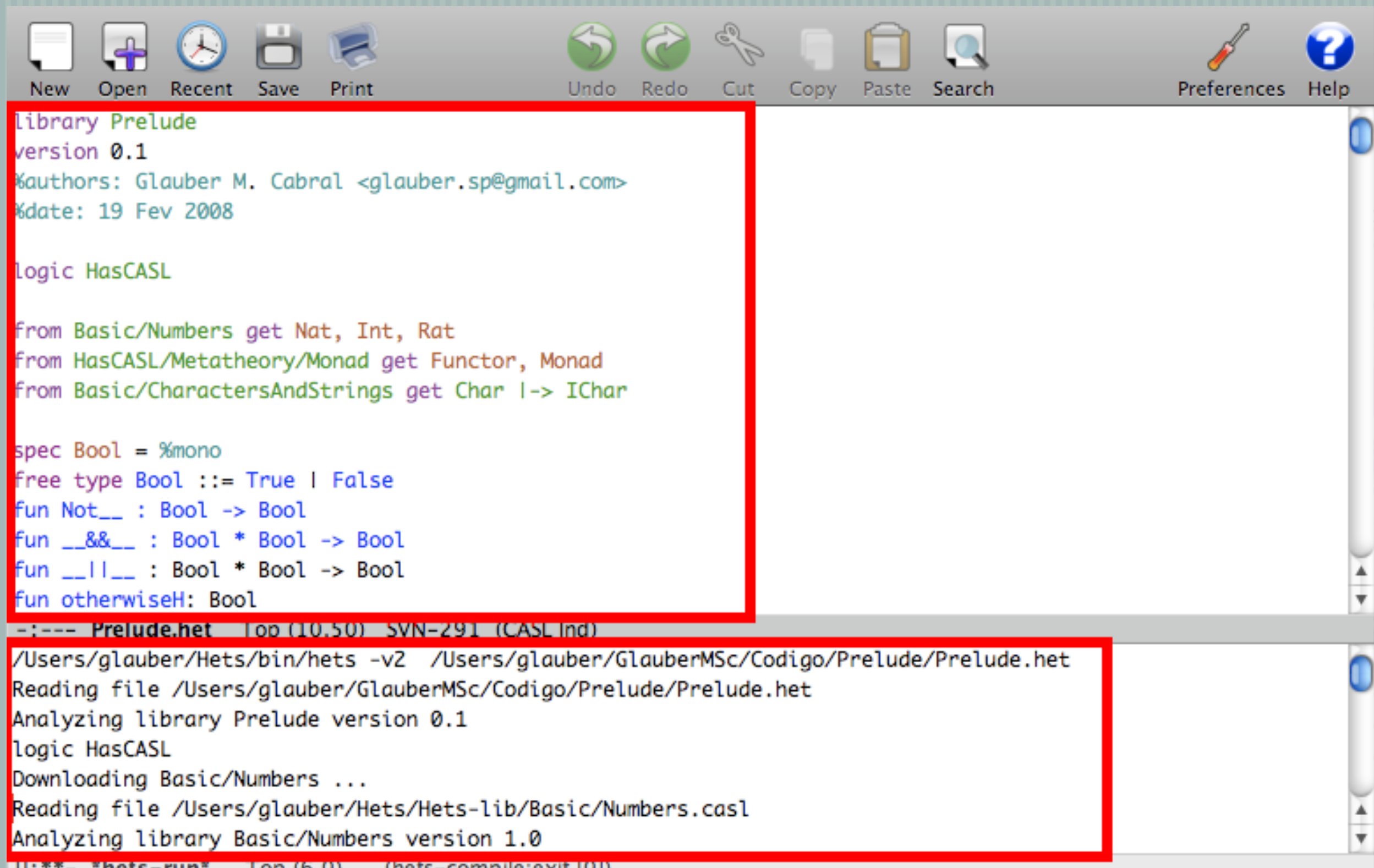
Tradução para HOL - Lemmas

```
1 LeIrreflexivity [rule_format] :  
2 "ALL (x :: 'a). ALL (y :: 'a).  
3  x == ' y = True' --> x < ' y = False' "  
  
4 Isabelle precisa provar:  
5 [| x < ' y = True'; y < ' x = True' |] ==> False  
  
6 lemma LeIrreflContra :  
7 " x < ' x = True' ==> False"  
8 by auto
```

Tradução para HOL - Prova

```
1 theorem LeTAsymmetry :  
2 "ALL (x :: 'a). ALL (y :: 'a).  
3  x < ' y = True' --> y < ' x = False' "  
4 apply(auto)  
5 apply(rule ccontr)  
6 apply(simp add: notNot2 NotTrue1)  
7 apply(rule_tac x="x" in LeIrreflContra)  
8 apply(rule_tac y = "y" in LeTTransitive)  
9 by auto
```

Passo a passo: Passo 1



The screenshot shows a Haskell IDE with a menu bar and a toolbar. The main editor window displays the source code of a file named `Prelude.het`. The code defines a library `HasCASL` with a version of 0.1, authored by Glauber M. Cabral. It imports `Nat`, `Int`, and `Rat` from `Basic/Numbers`, `Functor` and `Monad` from `HasCASL/Metatheory/Monad`, and `Char` from `Basic/CharactersAndStrings`. It then defines a `Bool` type and several functions: `Not`, `&&`, `||`, and `otherwiseH`. The bottom pane shows the output of the `hsets` command, which reads the file, analyzes the library, and downloads the `Basic/Numbers` library.

```
Library Prelude
version 0.1
%authors: Glauber M. Cabral <glauber.sp@gmail.com>
%date: 19 Fev 2008

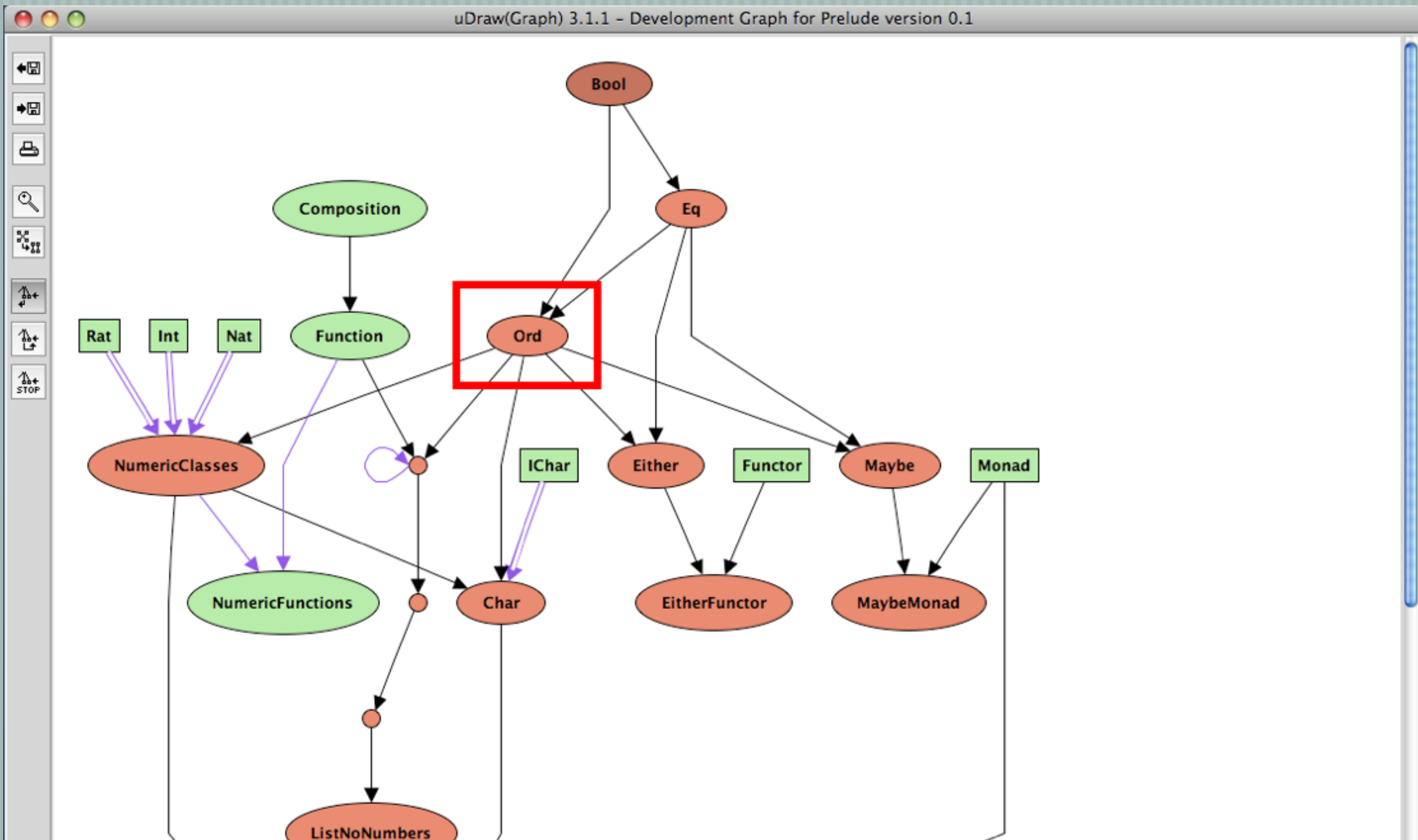
logic HasCASL

from Basic/Numbers get Nat, Int, Rat
from HasCASL/Metatheory/Monad get Functor, Monad
from Basic/CharactersAndStrings get Char |-> IChar

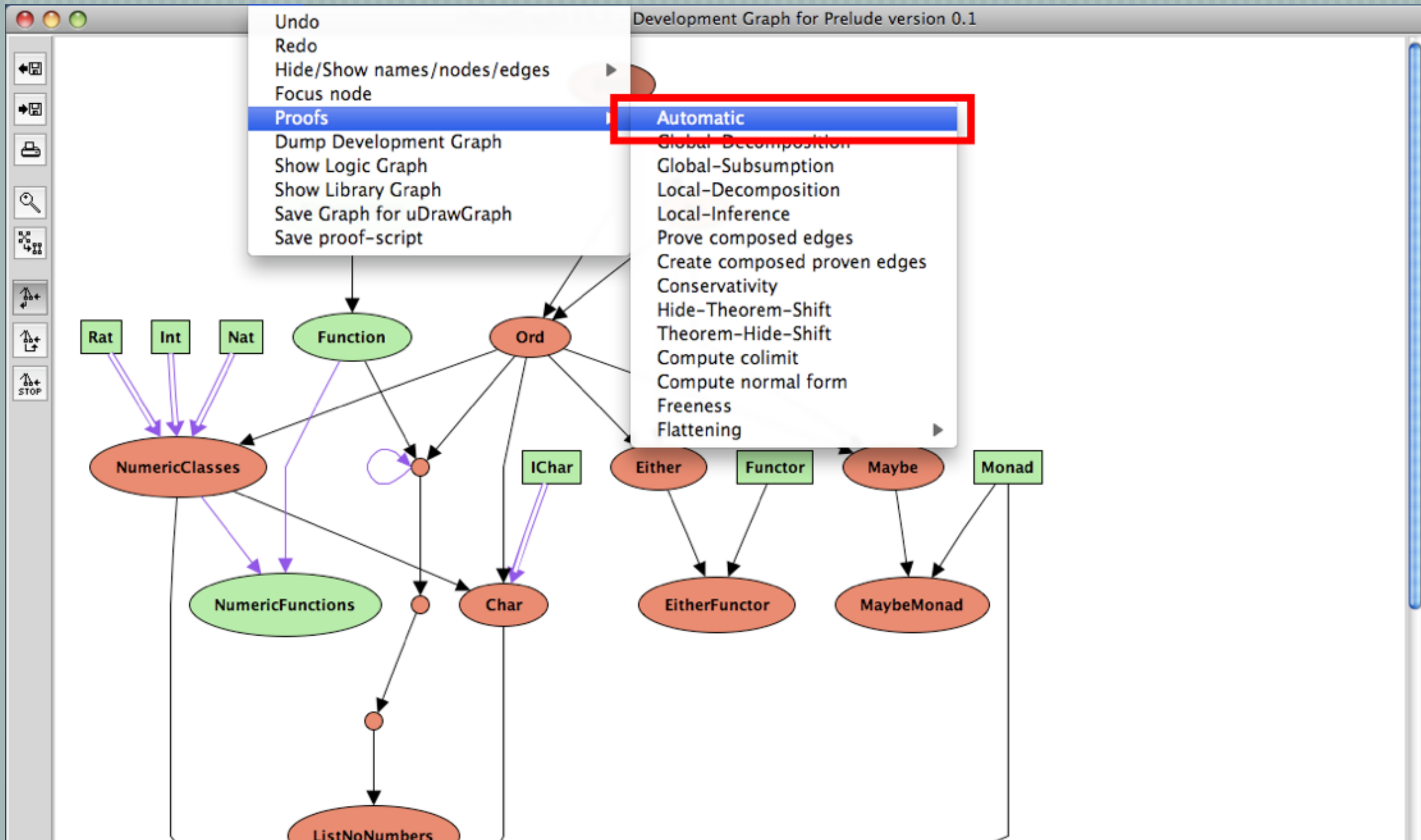
spec Bool = %mono
free type Bool ::= True | False
fun Not__ : Bool -> Bool
fun __&&__ : Bool * Bool -> Bool
fun __||__ : Bool * Bool -> Bool
fun otherwiseH: Bool

-:--- Prelude.het Top (10.50) SVN-291 (CASL Ind)
/Users/glauber/Hets/bin/hets -v2 /Users/glauber/GlauberMSc/Codigo/Prelude/Prelude.het
Reading file /Users/glauber/GlauberMSc/Codigo/Prelude/Prelude.het
Analyzing library Prelude version 0.1
logic HasCASL
Downloading Basic/Numbers ...
Reading file /Users/glauber/Hets/Hets-lib/Basic/Numbers.casl
Analyzing library Basic/Numbers version 1.0
H:*** *hets-run* Top (6.0) (hets-compile+exit 100)
```

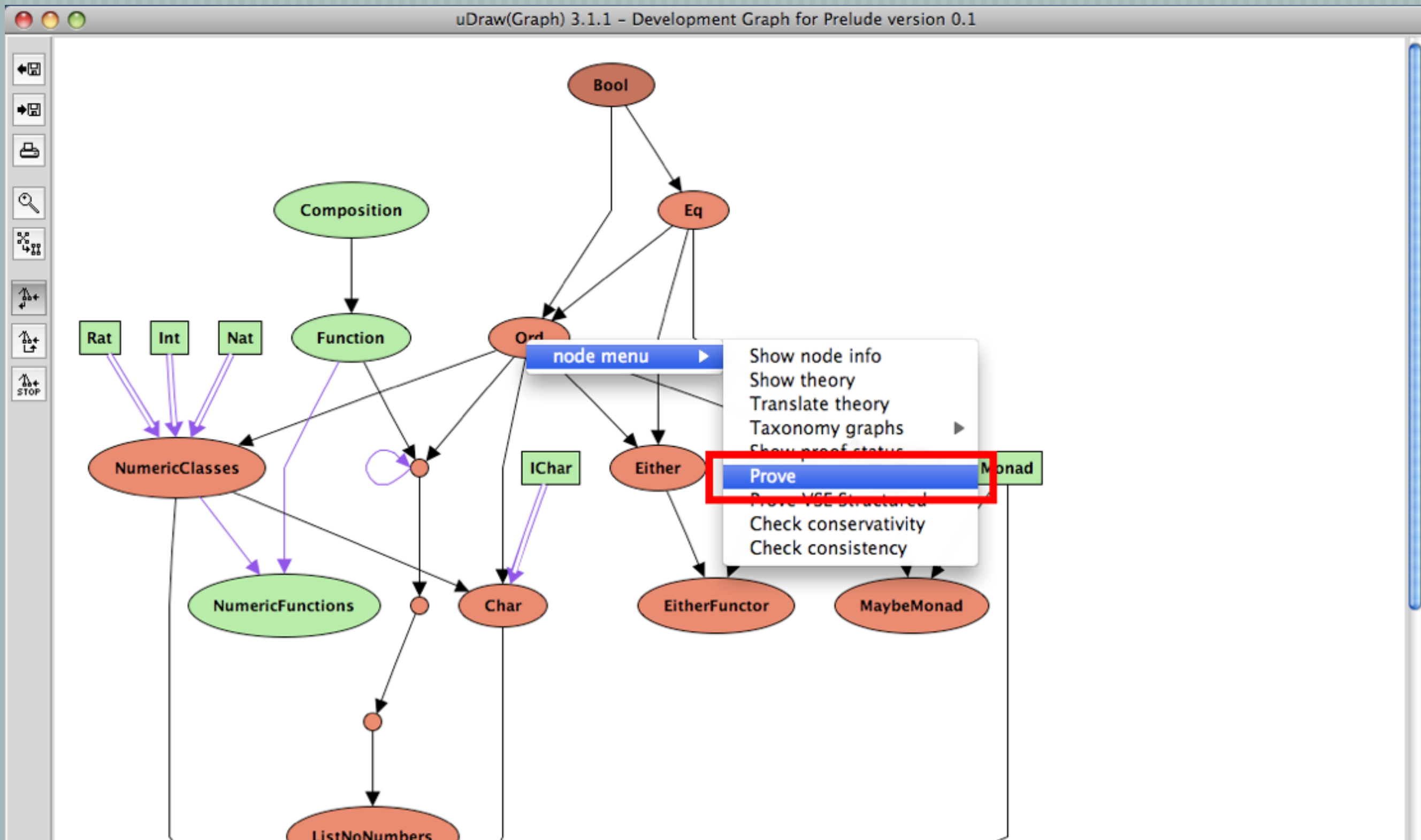

Passo a passo: Passo 2



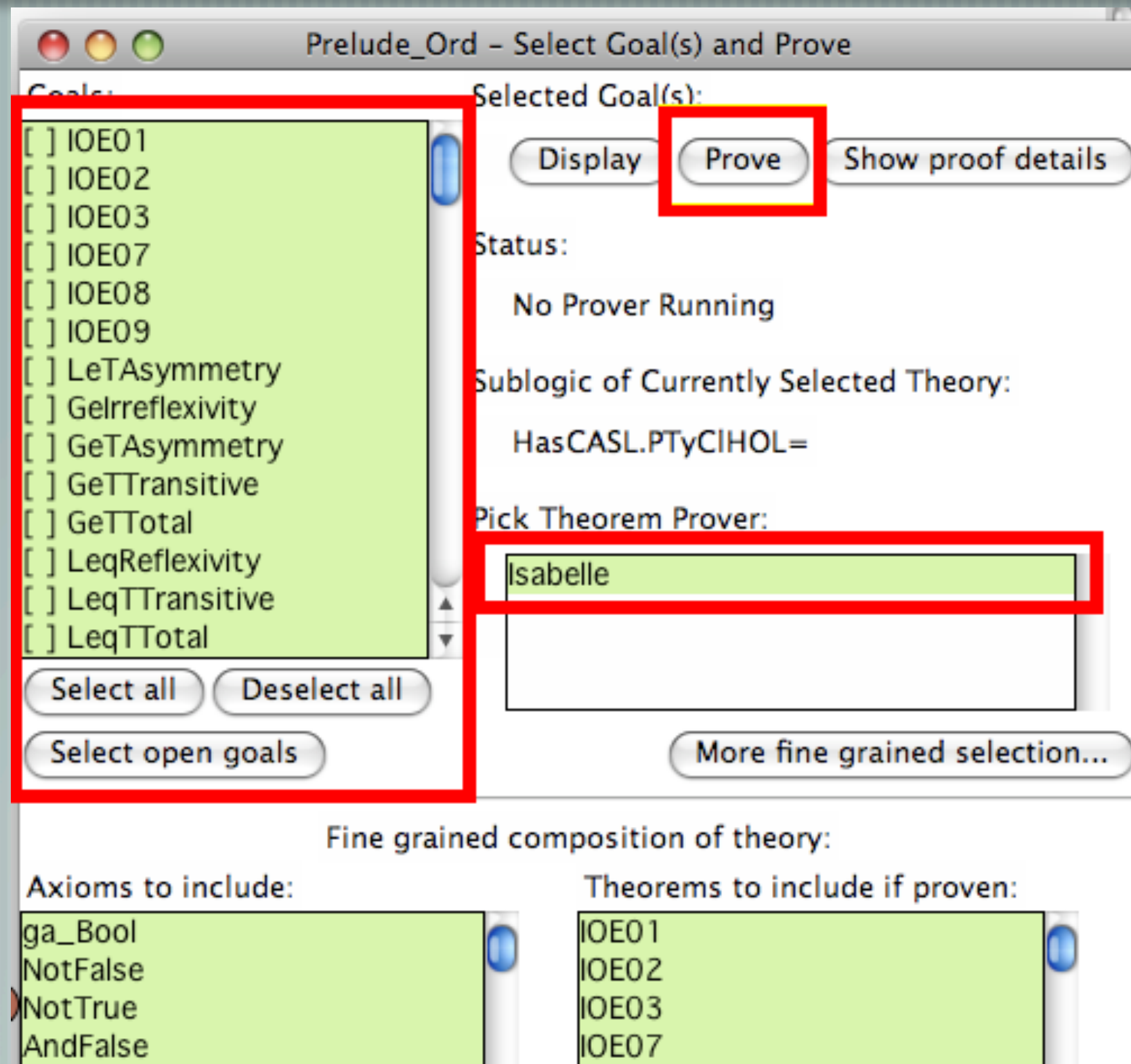
Passo a passo: Passo 3



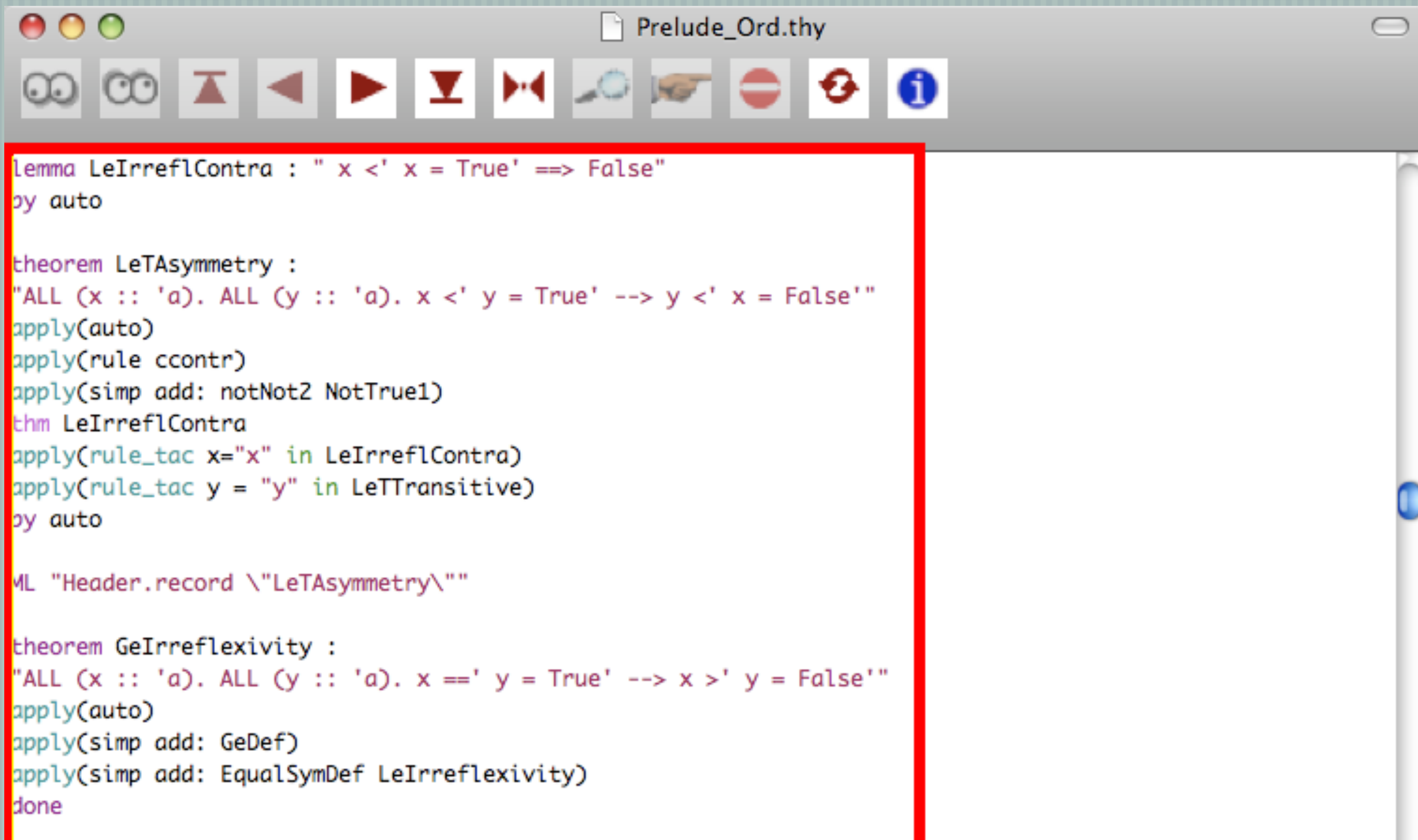
Passo a passo: Passo 4



Passo a passo: Passo 5



Passo a passo: Passo 6



```

Prelude_Ord.thy

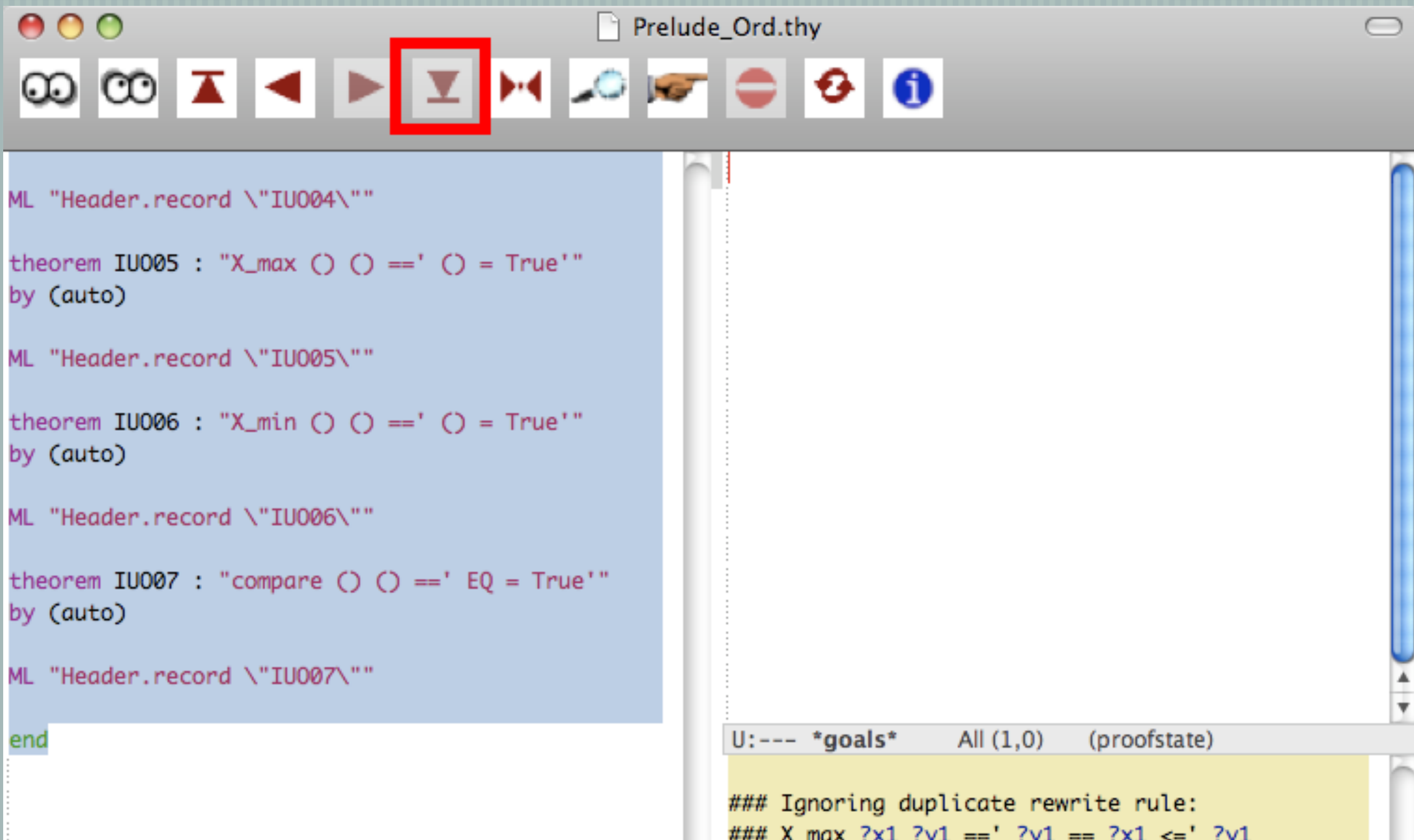
lemma LeIrreflContra : " x <' x = True' ==> False"
by auto

theorem LeTAsymmetry :
"ALL (x :: 'a). ALL (y :: 'a). x <' y = True' --> y <' x = False'"
apply(auto)
apply(rule ccontr)
apply(simp add: notNot2 NotTrue1)
thm LeIrreflContra
apply(rule_tac x="x" in LeIrreflContra)
apply(rule_tac y = "y" in LeTTransitive)
by auto

ML "Header.record \"LeTAsymmetry\""

theorem GeIrreflexivity :
"ALL (x :: 'a). ALL (y :: 'a). x ==' y = True' --> x >' y = False'"
apply(auto)
apply(simp add: GeDef)
apply(simp add: EqualSymDef LeIrreflexivity)
done
```

Passo a passo: Passo 7



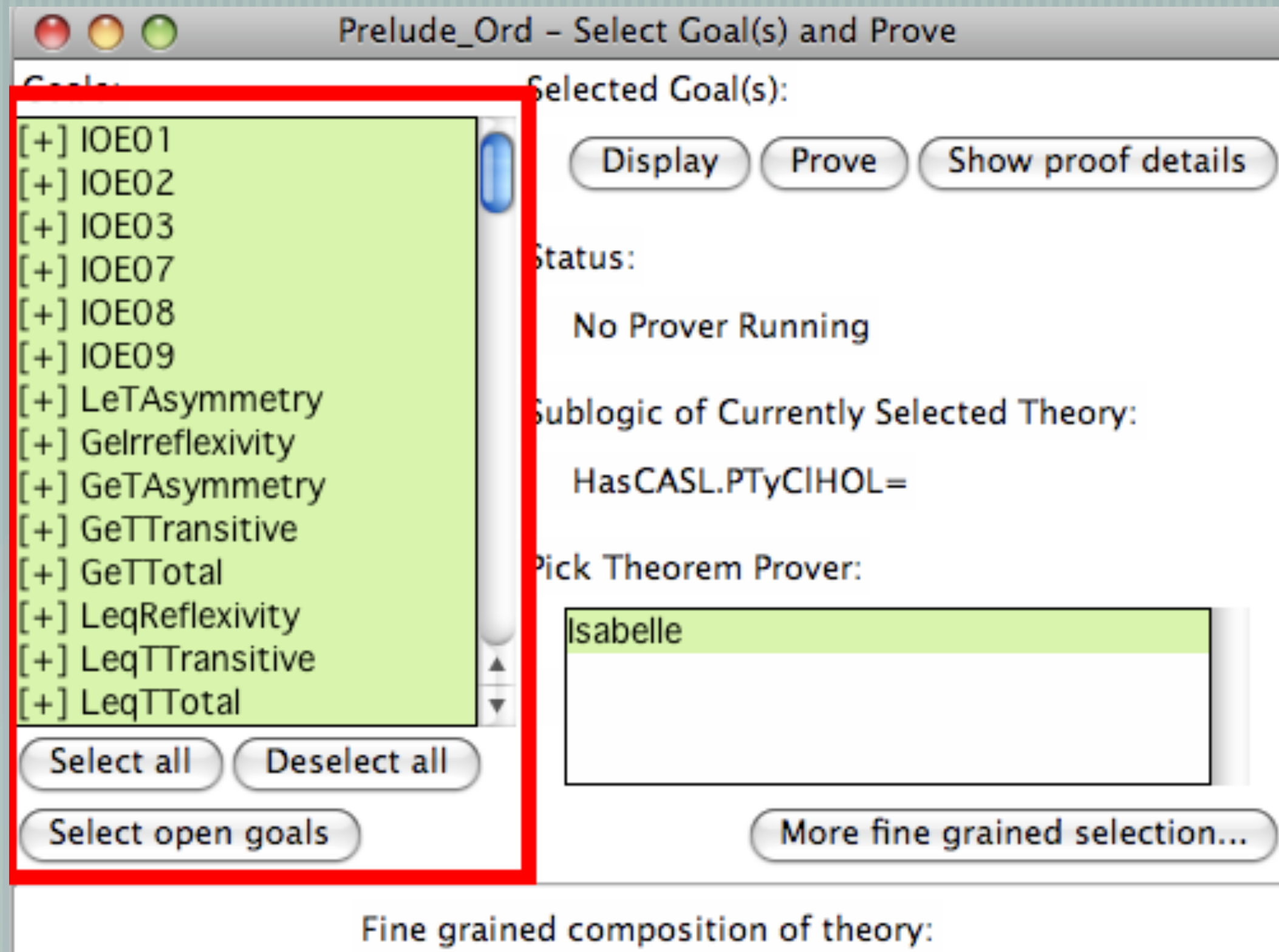
The screenshot shows the Isabelle/ML editor interface. The top toolbar contains various navigation and editing icons. The icon for 'Execute' (a downward-pointing triangle) is highlighted with a red square. The main text area on the left contains the following ML code:

```
ML "Header.record \"IU004\""  
  
theorem IU005 : "X_max () () == ' () = True'"  
by (auto)  
  
ML "Header.record \"IU005\""  
  
theorem IU006 : "X_min () () == ' () = True'"  
by (auto)  
  
ML "Header.record \"IU006\""  
  
theorem IU007 : "compare () () == ' EQ = True'"  
by (auto)  
  
ML "Header.record \"IU007\""  
  
end
```

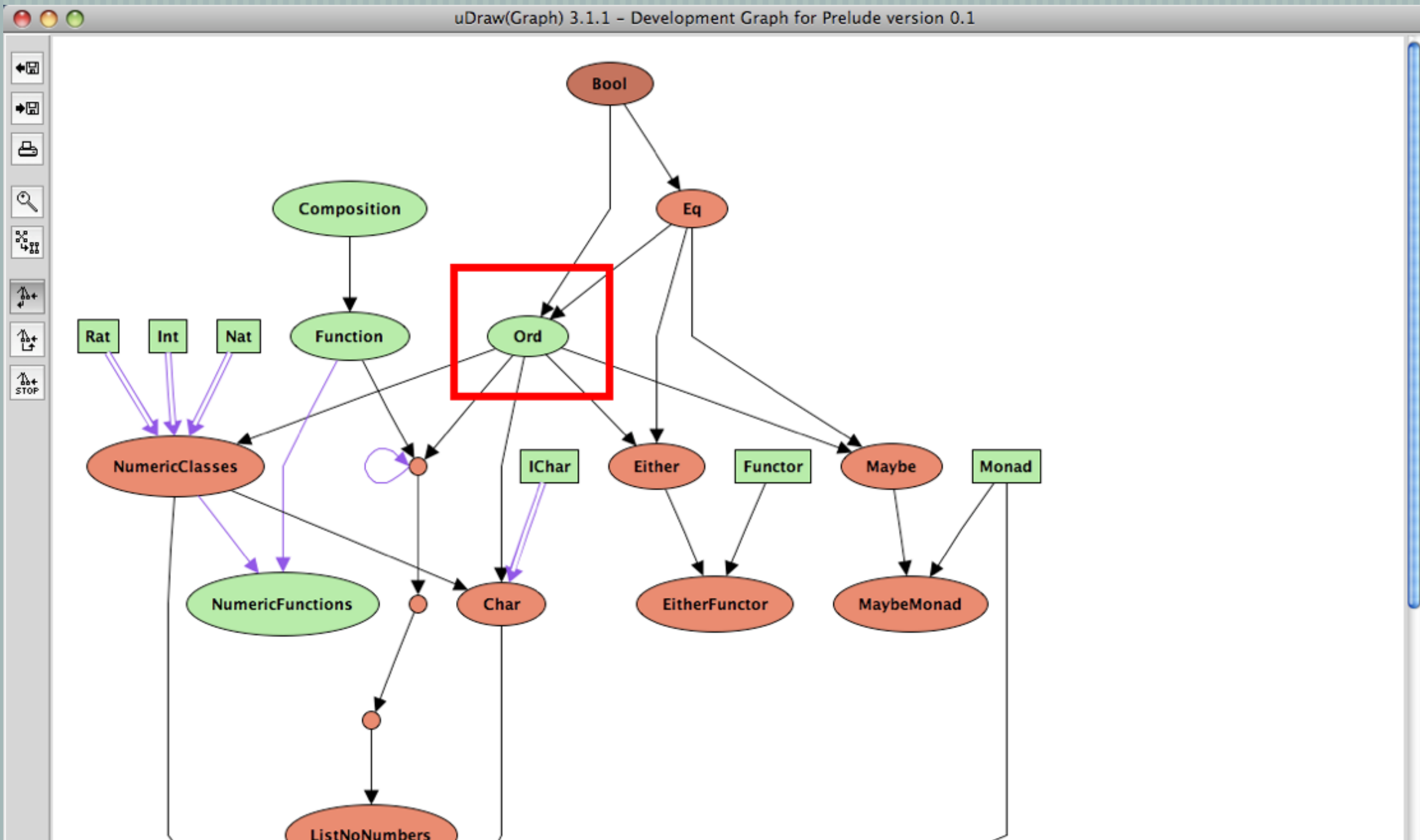
The right pane shows the current proof state:

```
U:--- *goals*      All (1,0)  (proofstate)  
  
### Ignoring duplicate rewrite rule:  
### X max ?x1 ?v1 == ' ?v1 == ?x1 <=' ?v1
```

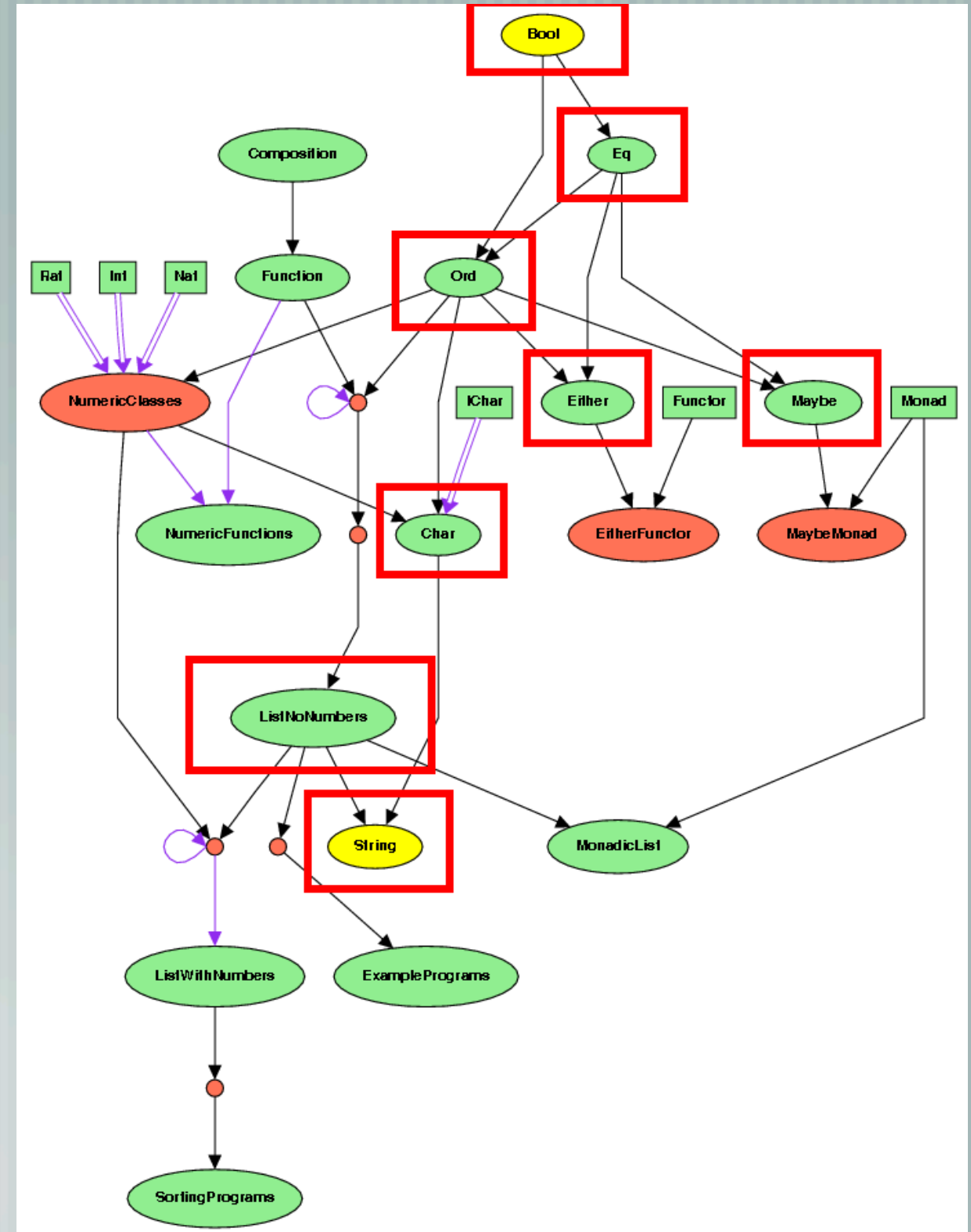
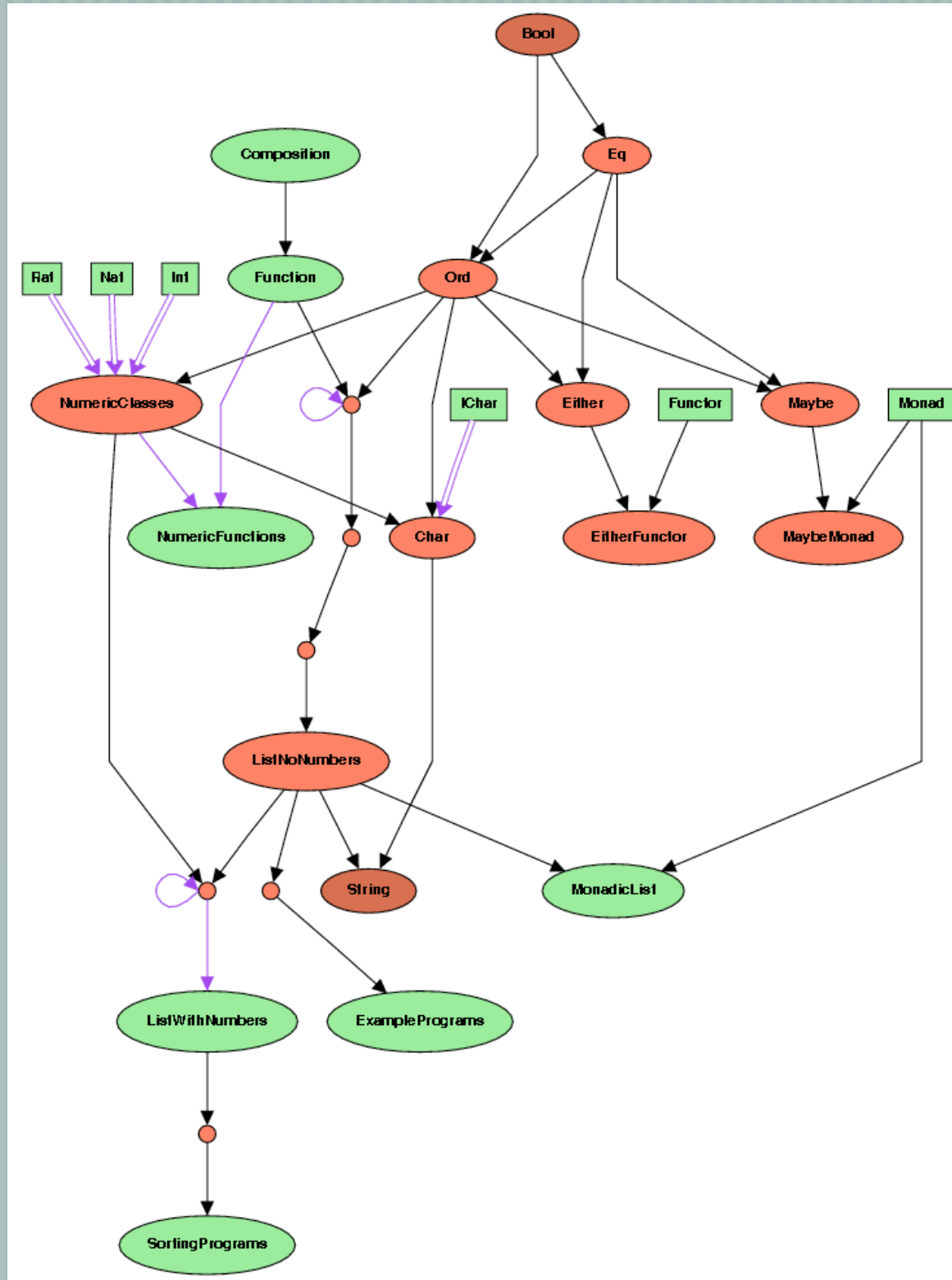

Passo a passo: Passo 8



Passo a passo: Passo 9



Estados inicial e final das provas



Contribuições

- [Biblioteca especificada possui os tipos de dados booleano, listas, caracteres e cadeias de caracteres
- [Especificações de exemplos empregam listas e booleanos
- [Duas versões para a biblioteca (aprox. 1000 LOC, cada):
 - 1ª Versão: Tipos com avaliação estrita devido à complexidade do uso de tipos com avaliação preguiçosa
 - 2ª Versão: Refinamento para suportar tipos com avaliação preguiçosa sem suporte a tipos infinitos

Contribuições

- [Verificação de propriedades:
 - 9 especificações verificadas totalmente
 - 8 especificações possuem alguns teoremas com provas incompletas
- [Tutorial introdutório da linguagem HasCASL em Português
- [Relatório técnico:
 - CABRAL, Glauber Módolo; MOURA, Arnaldo Vieira. **Creating a HasCASL library**. IC-09-03 Campinas: Instituto de Computação, 2009. 68 p. (Relatórios Técnicos do Instituto de Computação). Disponível em: <<http://www.ic.unicamp.br/~reltech/2009/abstracts.html>>. Acesso em: 26 abr. 2010.

Trabalhos Futuros

- [Escrever mapeamentos entre os tipos de dados da biblioteca da linguagem CASL e os tipos de dados da linguagem HOL
- [Melhorar o suporte a tipos de dados numéricos e verificar propriedades que os envolvem
- [Especificar tipos de dados contínuos (infinitos)
- [Especificar estruturas de dados mais complexas implementadas por alguns compiladores da linguagem Haskell, mas que estão fora da biblioteca Prelude
- [Divulgar o trabalho em evento científico

Conclusões

- [Este trabalho contribui na direção da tradução automática de especificações HasCASL para programas na linguagem Haskell
- [Este trabalho fornece um exemplo do processo de especificação e verificação de propriedades com as linguagens HasCASL, HOL e o provador de teorema Isabelle
- [Lições do mestrado:
 - Aprendi como é o método formal de desenvolvimento de software, quais são os seus usos e quais são as dificuldades envolvidas na sua utilização
 - Aprendi como lidar com o uso de softwares em desenvolvimento, quando a documentação ainda é escassa, e como contribuir para melhorar este cenário

Errata da Dissertação

- [No Capítulo 2, seção 2.1, o exemplo da linguagem Extended ML será removido porque replicá-lo nas outras linguagens foge ao escopo do trabalho
- [Na Conclusão, o 4º parágrafo será removido porque os subtipos já são parcialmente traduzidos para a linguagem HOL e as especificações numéricas puderam ter uma primeira versão especificada.
 - Incluir que cabe melhorar as especificações e verificar as propriedades que as envolvem.

Agradecimentos

— [Obrigado!

— [Apoio Financeiro: CNPq

— [Contato: glauber.sp@gmail.com