Система вывода в линейной логике с реализацией на языке Haskell

В. А. Панков

Направление подготовки: Прикладная математика и информатика
Руководитель: асс. каф. ИВЭ А. М. Пеленицын

Южный федеральный университет
Институт математики, механики и компьютерных наук имени И. И. Воровича

Кафедра информатики и вычислительного эксперимента

Постановка задачи

1. Реализовать вывод в линейной логике на языке Haskell.
2. Исследовать возможности его применения на примере игры «Рука Робота»

Базовые понятия линейной логики

Основные свойства

- все суждения в процессе доказательства можно использовать один раз.
- нельзя использовать одно и то же суждение дважды.
- суждения = ресурсы

Базовые понятия линейной логики

Общезначимые гипотезы

- $\cdot \vdash A \Rightarrow A$ общезначимая
- $\Gamma; \cdot \vdash A \Rightarrow A$ общезначимая

Базовые понятия линейной логики

Общезначимые гипотезы

- $\cdot \vdash A \Rightarrow A$ общезначимая
- $\Gamma; \cdot \vdash A \Rightarrow A$ общезначимая

Основное утверждение

С ресурсами Δ и общезначимыми гипотезами Γ можем доказать A:

$$\Gamma; \Delta \vdash A$$

Пример: одновременная конъюнкция

$$\frac{\Gamma; \Delta' \vdash A \qquad \Gamma; \Delta'' \vdash B}{\Gamma; \Delta', \Delta'' \vdash A \otimes B} \otimes R$$

Рис.: Правое правило для $\otimes$

Базовые понятия линейной логики

Пример: одновременная конъюнкция

$$\frac{\Gamma; \Delta' \vdash A \qquad \Gamma; \Delta'' \vdash B}{\Gamma; \Delta', \Delta'' \vdash A \otimes B} \otimes R$$

Рис.: Правое правило для $\otimes$

Пример: одновременная конъюнкция

$$\frac{\Gamma; \Delta, A, B \vdash C}{\Gamma; \Delta, A \otimes B \vdash C} \otimes L$$

Рис.: Левое правило для $\otimes$

Алгоритм фокусировки

Основные понятия

- Правило называется обратимым, если его применение не создаёт подзадач, зависящих друг от друга.
- Иначе правило называется необратимым.
- Правило, которое является обратимым при некотором наборе ресурсов называется частично обратимым.

Алгоритм фокусировки

Основные понятия

- Логическая связка называется правой / левой асинхронной, если у неё есть правое / левое обратимое правило.
- Логическую связку, у которой есть правое / левое необратимое или частично обратимое правило, называется синхронной.

Пример: синхронная логическая связка

$$\frac{\Gamma; \Delta' \vdash A \qquad \Gamma; \Delta'' \vdash B}{\Gamma; \Delta', \Delta'' \vdash A \otimes B} \otimes R$$

Алгоритм фокусировки

Пример: синхронная логическая связка

$$\frac{\Gamma; \Delta' \vdash A \qquad \Gamma; \Delta'' \vdash B}{\Gamma; \Delta', \Delta'' \vdash A \otimes B} \otimes R$$

Проблема: как поделить ресурсы между подзадачами

Возникает неопределенность в том, как много ресурсов от $\Delta = (\Delta', \Delta'')$ отдать для доказательства левой ветки, а сколько — для правой.

Алгоритм фокусировки

Пример: синхронная логическая связка

$$\frac{\Gamma; \Delta' \vdash A \qquad \Gamma; \Delta'' \vdash B}{\Gamma; \Delta', \Delta'' \vdash A \otimes B} \otimes R$$

Проблема: как поделить ресурсы между подзадачами

Возникает неопределенность в том, как много ресурсов от $\Delta = (\Delta', \Delta'')$ отдать для доказательства левой ветки, а сколько — для правой.

Аналогия с процессами.

Алгоритм фокусировки

Фазы алгоритма

1. *Инверсия*. Устранение асинхронных логических связок.
2. *Решение*. Принятие решение о фокусировке на цели или на посылке.
3. *Фокус*. Устранение синхронных логических связок

Алгоритм фокусировки

Правила вывода

Алгоритм схематично описывается правилами вывода

$$\frac{\Gamma; \Delta; \Omega, A, B \Uparrow\vdash C}{\Gamma; \Delta; \Omega, A \otimes B \Uparrow\vdash C} \otimes L$$

Рис.: Привило инверсии для $\otimes$

$$\frac{\Gamma; \Delta' \vdash A \Downarrow \qquad \Gamma; \Delta'' \vdash B \Downarrow}{\Gamma; \Delta', \Delta'' \vdash A \otimes B \Downarrow} \otimes R$$

Рис.: Правило фокуса для $\otimes$

Представление выражений

Data types à la carte

- Реализация конструкторов по-отдельности
- Рекурсивный супер-тип, который их связывает

Представление выражений

Пример: простое выражение

```
data SimConj f = SimConj f f

  data Unit f = Unit

  newtype Expr f = In { out :: f (Expr f) }

  data (f :+: g) a = Inl (f a) | Inr (g a)

  someExpr :: Expr (SimConj :+: Unit)
someExpr = In (Inl (SimConj
                 (In (Inr Unit))
                 (In (Inr Unit))))
```

Представление выражений

Пример: простое выражение

```
someExpr :: Expr (SimConj :+: Unit)
someExpr = In (Inl (SimConj
                    (In (Inr Unit))
                    (In (Inr Unit))))
```

Вставки *Inl* и *Inr* можно автоматизировать.

Реализация алгоритма

Описание

- Использование классов типов для реализации вывода
- Использования перекрытия классов для реализации контролирующих алгоритм классов.

Реализация алгоритма

Пример правила

```
instance RightAsync ResImpl where
  rightInvRule (ResImpl a b) (uniq, (vals,
      ress, ords)) =
     [(uniq, (vals, ress, a:ords) :>> b)]
```

Реализация алгоритма

Реализация управления ресурсами

```
data Channel =
      Channel { wait  :: [LinearFormula]
              , get   :: [(Int, LinSeq)] }

    instance RightSyncFocus SimConj where
      rightSyncFocus (SimConj a b) (uniq, (vals
         , ress)) =
        do leftChan <- applyRightFocus
         (uniq * 2, (vals, ress) :=> a)

           let (leftRess, leftChan') = receive
              leftChan

           rightChan <- applyRightFocus
```

Пример работы

Игра «Рука робота»

- На столе находятся блоки.
- Их состояния описываются суждениями.
- Задача состоит в доказательстве возможности перехода от одного состояния системы к другому.

Пример работы

Предикаты

```
emptyHand :: LinearFormula
emptyHand = atom "Empty" []

clearTop :: Term -> LinearFormula
clearTop x = atom "Clear" [x]

onTopOf :: Term -> Term -> LinearFormula
onTopOf x y = atom "On" [x, y]

handHolds :: Term -> LinearFormula
handHolds x = atom "Holds" [x]

onTable :: Term -> LinearFormula
onTable x = atom "Table" [x]
```

Пример работы

Возможные действия робота

```
getFromTop :: LinearFormula
getFromTop =
  forAll $ \ x ->
    forAll $ \ y ->
      (emptyHand &&: clearTop x &&: onTopOf x y
        ) ->: (handHolds x &&: clearTop y)

getFromTable :: LinearFormula
getFromTable =
  forAll $ \ x ->
    (emptyHand &&: clearTop x &&: onTable x)
      ->: (handHolds x)

putOnTop :: LinearFormula
putOnTop =
```

Пример работы

Возможные действия робота

```
getFromTop :: LinearFormula
getFromTop =
  forAll $ \ x ->
    forAll $ \ y ->
      (emptyHand &&: clearTop x &&: onTopOf x y
        ) ->: (handHolds x &&: clearTop y)

getFromTable :: LinearFormula
getFromTable =
  forAll $ \ x ->
    (emptyHand &&: clearTop x &&: onTable x)
      ->: (handHolds x)

...
```

Пример работы

Доступные блоки

```
blockA :: Term
blockA = constant "A"

blockB :: Term
blockB = constant "B"

blockC :: Term
blockC = constant "C"
```

Пример работы

Состояние стола и состояние-цель

```
tableState :: [LinearFormula]
tableState =
  [emptyHand, onTable blockA, onTopOf blockB
     blockA,
    clearTop blockB, onTable blockC, clearTop
       blockC]

goal :: LinearFormula
goal = onTopOf blockA blockB &&: top
```

Пример работы

Доказательство

```
sequent :: LinSeq
sequent = (handMoves, tableState) :=> goal


main = case prove sequent of
         Left  _ -> putStrLn "oops"
         Right chan -> putStrLn $
            showChannel chan
```

Пример работы

Результат

Доказываемые утверждения: *On(A, B)* и *T*.

Полученные результаты

1. Разработана система вывода в линейной логике
2. Исследованы и применены следующие возможности Haskell:
   - Data types a ` la carte
   - Перекрытие классов