ЛОГИЧЕСКИЕ МЕТОДЫ В ИНФОРМАТИКЕ
(Вычислительные системы)
1993 год Выпуск 148

УДК 519.68

ПРЕДСТАВЛЕНИЕ Σ-ПРОГРАММ В ИНТУИЦИОНИСТСКОЙ ТЕОРИИ ТИПОВ*)

А.В.Грасмик

В в е д е н и е

Работа посвящена представлению Σ-программ в интуиционистской теории типов Мартина-Лёфа (ITT) и исследованию процедуры логического вывода в ITT.

Определение списочной надстройки (не включающее определения $\in$, $\subseteq$) Σ-формулы взято из [1]. Понятия Σ-программы и обращения к Σ-программе используются так же, как и в [2]. Используется вариант интуиционистской теории типов Мартина-Лёфа, описанный в [5], но для обозначения того, что a является объектом типа A, вместо значка "$\in$" используется значок ":".

Представление Σ-программы в ITT проводится в три этапа. На первом этапе в ITT строится списочная надстройка, полностью аналогичная GES. На втором - определяется функция Trans, производящая перевод Σ-формул в типы ITT. На третьем рассматриваются представление Σ-программ и обращения к Σ-программам. Как результат рассмотрения вывода в ITT дается процедура автоматического вывода типов.

В заключение доказывается теорема об эквивалентности истинности Σ-формулы и выводимости ее представления в ITT.

*) Работа выполнена при финансовой поддержке Российского Фонда Фундаментальных исследований (93-011-1506).

Пусть $\mathcal{M}$ - модель сигнатуры σ_0 с системой основных множеств $\bigcup_{i\in I} M_i = M$. Построим в ITT списочную надстройку $S(\mathcal{M})$ над моделью $\mathcal{M}$. Добавим к базовым типам ITT $(N_0,\ldots,N_m,\ldots;N)$ типы, представленные множествами M_i, $i \in I$.

Определим тип list, который поэлементно совпадает со списочной надстройкой $S(\mathcal{M})$:

nil:list,

cons:$(\Pi(\alpha,\beta)$: $(\Sigma\ \alpha$: list) $(M+$list$))$ list,

conc:$(\Pi(\alpha,\beta)$: $(\Sigma\ \alpha$:: list) list) list,

head:$(\Pi\alpha$: list) $(M+$list$)$,

tail:$(\Pi\alpha$: list) list.

Определять отношения $\in$ и $\subseteq$ нет необходимости,так как при трансляции Σ-формул происходит и трансляция формул с ограни - ченными кванторами.

Представление списочной надстройки одним типом list не очень хорошо,так как не видна структура списков и их элементов, не ясно действие списочных операций. Проведем структуризацию списочной надстройки.

Определим типы list$(A_1,\ldots,A_n)$, где A_i - тип i-го элемента списка. Тогда определение списочной надстройки таково (везде $A,A_1,\ldots,A_n,B_1,\ldots,B_m$ - типы из V_1):

nil: list(N_0),

cons: $(\Pi(\alpha,\beta)$: $(\Sigma\ \alpha$: list$(A_1,\ldots,A_n)A)$ list$(A_1,\ldots,A_n,A)$,

conc: $(\Pi(\alpha,\beta)$: $(\Sigma\ \alpha$: list$(A_1,\ldots,A_n)$ list$(B_1,\ldots,B_n))$ list$(A_1,\ldots,A_n,B_1,\ldots,B_m)$,

head: $(\Pi\ \alpha$: list$(A_1,\ldots,A_n))A_n$,

tail: $(\Pi\alpha$: list$(A_1,\ldots,A_n))$ list$(A_1,\ldots,A_{n-1})$.

Трансляция Σ-формул

Опишем функцию Trans, переводящую Σ-формулы в типы теории ITT:

Trans(false) = N_0;

Trans(true) = N_1;

Trans(x) = x для любой переменной x;

Trans(T) = T для любого типа T (в смысле Σ);

Trans($P(x_1,\ldots,x_n)$) = $P(Trans(x_1),\ldots,Trans(x_n))$, где P - предикат из сигнатуры;

Trans(f(x)) = Trans(f)(Trans(x));

Trans(A & B) = $(\Sigma\ x\colon Trans(A))\ Trans(B)$, где x - новая переменная;

Trans($(\exists x\colon A)B(x)$) = $(\Sigma\ x\colon Trans(A))\ Trans(B(x))$;

Trans($\neg A$) = $(\Pi x\colon Trans(A))N_0$, где x - новая переменная;

Trans($A \to B$) = Trans($\neg A \vee B$);

Trans($(\forall x\colon A)B(x)$) = $(\Pi x\colon Trans(A))\ Trans(B(x))$;

Trans($A \vee B$) = Trans(A) + Trans(B);

Trans($t_1 = t_2$) = $I(Trans(t_1), Trans(t_2), T)$, если t_1 и t_2 - оба типа T;

Trans($(\forall x \in t)B$) = Trans(B(head(t)) & $(\forall x \in tail(t))B$);

Trans($(\exists x \in t)B$) = Trans(B(head(t)) $\vee$ $(\exists x \in tail(t))B$;

Trans($(\forall x \subseteq t)B$) = Trans(B(t) & $(\forall x \subseteq tail(t))B$);

Trans($(\exists x \subseteq t)B$) = Trans(B(t) $\vee$ $(\exists x \subseteq tail(t))B$).

Функция Trans осуществляет перевод всех Σ-формул. Кроме того, переводятся и формулы с неограниченным квантором существования $(\forall x\colon A)B(x)$, который конструктивен тогда, когда сильно конструктивен тип A.

Возможно расширение синтаксиса Σ-формул за счет использования имеющихся в ITT средств. Например, можно ввести λ-абстракцию и бесконечную рекурсию (реализуемую конструктором типов W).

Но исследование данного вопроса выходит за рамки данной работы и требует самостоятельного рассмотрения.

Трансляция Σ-программ

Будем считать, что Σ-программа представляет из себя последовательность определений вида:

$$P_1(\bar{x}_1) \text{ def } A_1(\bar{x}_1)$$

$$\vdots$$

$$P_n(\bar{x}_n) \text{ def } A_n(\bar{x}_n),$$

где A_i - Σ-формула, в которую символы $P_1,\ldots,P_n$ входят только позитивно и в которой все свободные переменные формулы A_i содержатся в списке $\bar{x}_i$. На самом деле можно использовать и Σ^+-формулы. Теория ITT позволяет делать это.

Трансляция Σ-программ такова:

$$P_1(\bar{x}_1) \text{ def } \mathrm{Trans}(A_1)$$

$$\vdots$$

$$P_n(\bar{x}_n) \text{ def } \mathrm{Trans}(A_n).$$

Исполнение запросов к Σ-программам осуществляется при помощи проверки истинности соответствующих формул в модели $S(\mathcal{M})$. После трансляции проверке истинности соответствует построение из элементов модели $S(\mathcal{M})$ объектов типов, соответствующих транслируемым формулам.

Построение объектов типов производится при помощи исчисления теории ITT, представленного в форме естественного вывода. Ниже будет дана система автоматического вывода в ITT.

Вывод в ITT

В ITT имеются четыре вида формул

$$A\ type, \tag{1}$$

$$A = B, \tag{2}$$

$$a: A, \tag{3}$$

$$a = b: A. \tag{4}$$

Исчисление включает основные правила, правила для типов $N_0,\ldots,N_m,\ldots,N$; U_i (универсумов) и правила для конструкторов типов $\Pi, \Sigma, +, I, W$. Для каждого конструктора имеются четыре пары (для формул вида (1),(3) и вида (2),(4)): образования, введения, удаления и равенства (только для формул вида (4)).

Пусть P - некоторый тип. Построим его объекты при помощи правил введения, удаления и, возможно, равенства.

Проведем некоторые предварительные действия. Представим тип в виде дерева (подобно способу, который используют для представления термов в виде дерева). Вершины дерева пометим конструкторами типов, концы ветвей - элементарными типами (т.е. не построенными из базовых с помощью конструкторов типов). Ветви соответствуют синтаксической структуре типа. Например, для типа

$$(\Pi\, x\colon A)(\Sigma y\colon B)C \to (\Sigma f\colon (\Pi\, x\colon A)B)(\Pi\, x\colon A)C(f(x)/y)$$

($A \to B$ - аббревиатура $(\Pi x\colon A)B$, где x не свободен в B).

Введем обозначения:

$D_1 = (\Sigma y\colon B)C,$

$D_2 = (\Pi\, x\colon A)D_1,$

$D_3 = (\Pi\, x\colon A)C(f(x)/y),$

$D_4 = (\Pi\, x\colon A)B,$

$D_5 = (\Sigma f\colon D_4)D_3,$

$D_6 = (\Pi z\colon D_2)D_5.$

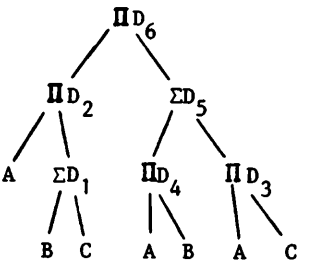
Известно:

A type,

B type (x: A),

C type (x: A, y: B).

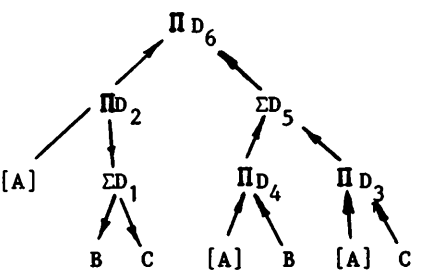
Тогда описанному типу соответствует дерево:



Зависимость типов B (от A) и C (от A и B) не известна. Следовательно, необходимо получить B и C. Предположив, что нам известен тип $\Pi D_2(f: \Pi D_2)$, и воспользовавшись правилом П-удаления, получаем тип ΣD_1, а из него по правилам Σ-удаления получаем типы B и C.

Теперь, зная типы A, B, C, мы можем последовательно получить типы ΠD_4, ΠD_3, ΣD_5 и, наконец, ΠD_6 (т.е. тот тип, который мы строили) с помощью правил введения.

Окончательно дерево доказательства можно изобразить с помощью следующего ориентированного графа:



Здесь стрелки вверх обозначают соответствующие использования правил введения, а стрелки вниз - правил удаления. Квадратными скобками отмечены типы, которые по предположению известны.

Равенство

Выше ничего не было сказано о равенстве. В связи с этим описанный вывод не полон.

Рассмотрим тип $D_3 = (\Pi x: A)\, C(f(x)/y)$. Видно, что в типе $C(f(x)/y)$ вместо переменной y подставляется терм $f(x)$. Однако это никак не отражено при выводе типа C. С помощью правил равенства можно показать, что дисциплина типов соблюдена.

Мы имеем:

$$q(z(x)) : C(p(z(x))/y),$$

$$(\lambda x)p(z(x))(=f) : (\Pi x: A)B.$$

В силу правила П-равенства:

$$((\lambda x)p(z(x)))(x) = p(z(x)) : B.$$

В силу правила симметрии:

$$(p(z(x)) = ((\lambda x)p(z(x))x : B;$$

подстановки:

$$C(p(z(x))/y) = C(((\lambda x)p(z(x)))(x)/y)$$

и равенства типов:

$$C(p(z(x))/y) = C(f(x)/y) \quad (f(x) = ((\lambda x)p(z(x)))(x)).$$

Такие проверки дисциплины типов необходимо проводить при построении типов.

Теперь перейдем к рассмотрению основной теоремы.

ТЕОРЕМА. $\mathcal{M} \vdash A \Leftrightarrow ITT \vdash Trans(A)$.

ДОКАЗАТЕЛЬСТВО проводится индукцией по числу логических связок. Индукционный шаг проводится разбором случаев (по видам логических связок). Для примера рассмотрим случай $A = (\exists x: T)B(x)$. Пусть A истинна, т.е. $x_0 : T$ такой, что истинна $B(x_0)$. По индукционному предположению $x_0 : Trans(T)$ и тип

$Trans(B(x_0))$ не пуст. Тогда по правилу Σ-введения тип $(\Sigma x: Trans(T))\ Trans(B(x))$ не пуст. А этот тип и есть $Trans(A)$. (Здесь под выводимостью типа в ITT понимается наличие объектов (не пустота) типа.) В обратную сторону доказательство проводится аналогично: используются правила удаления и определения истинности формулы на модели.

Заключение

В силу доказанной теоремы и наличия процедуры вывода типов можно рассматривать данное представление как операционную семантику Σ-программ. Однако практическое использование данной семантики проблематично.

Автор благодарен С.В.Котову за помощь при постановке задачи и ее решении.

Литература

1. ГОНЧАРОВ С.С., СВИРИДЕНКО Д.И. Σ-программирование //Логико-математические проблемы МОЗ. - Новосибирск,1985. -Вып.107: Вычислительные системы. - С.3-29.

2. ВОРОНКОВ А.А. Естественное исчисление для Σ-программ //Логические методы в программировании. - Новосибирск, 1987. - Вып. 120: Вычислительные системы. - С.14-23.

3. CONSTABLE R.L., ZLATIN D.R. The type theory of PL/CV3 //Lecture Notes in Comput. Sci. - 1982. - Vol. 131. - P. 72-93.

4. MARTIN-LÖF P. An intuitionistic theory of types: predicative part //Logic Coll. 73. - Amsterdam: North-Holland. - 1975. - P. 73-119.

5. MARTIN-LÖF P. Constructive mathematics and computer programming //Logic, Methodology and Philosophy of Science. - Vol. 1. - Amsterdam: North-Holland. - 1982. - P. 153-179.

Поступила в ред.-изд.отд.

10 июля 1993 года