计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
43(7):1304~1308, 2006

接续到直接的指称语义转换

吕江花[1,2] 马世龙[1] 潘 静[3] 金成植[2]
[1](北京航空航天大学软件开发环境国家重点实验室 北京 100083)
[2](吉林大学大学计算机科学与技术学院 长春 130012)
[3](北京科技大学管理学院 北京 100083)
(jhlv@nlsde.buaa.edu.cn)

Denotational Semantics Transform from Continuation to Direct

Lü Jianghua[1,2], Ma Shilong[1], Pan Jing[3], and Jin Chengzhi[2]
[1](*National Laboratory of Software Development Environment*, *Beihang University*, *Beijing* 100083)
[2](*College of Computer Science and Technology*, *Jilin University*, *Changchun* 130012)
[3](*School of Management*, *University of Science and Technology Beijing*, *Beijing* 100083)

Abstract The key of semantic transform is the discord of semantics functions' signature. A denotational semantics transform technology from continuation to direct is given based on the analysis between the two in the early research, and here continuations in different context are handled differently. Finally, the implementation system is described in Haskell, which tests the feasibility of the transform method.

Key words denotational semantics; continuation; signature; semantics transform

摘 要 接续和直接指称语义之间的转换的主要难点在于不保函数基调. 基于 Monad 思想推导出的接续语义函数和直接语义函数之间的关系,给出了基于规约的从接续指称语义描述形式到直接指称语义描述形式的转换技术,分别考虑了接续函数在不同情形下的处理. 最后给出了转换算法的 Haskell 实现系统,验证了转换的可行性.

关键词 指称语义;接续函数;基调;语义转换
中图法分类号 TP301

1 引 言

指称语义[1~4]分为直接指称语义和接续指称语义,其中直接指称语义适用于结构化语言,而接续指称语义由于定义了接续函数表示后续的计算,还适用于非结构化语言. 研究直接指称语义和接续指称语义之间的关系,有助于深入掌握语言的各种特性,同时对检验语义描述的等价性也具有重要的意义. 在文献[5]中,我们给出了基于 Monad[6] 思想的从直接指称语义到接续指称语义的转换方法,在该文中,我们详细分析了直接指称语义函数和接续指称语义函数之间的关系. 相比较而言,直接到接续的转换是扩展式的转换,而接续到直接的转换是收缩式的转换,而且其中的扩展和收缩是互为逆的过程. 其中最主要的难度还在于转换不是保函数基调的转换.

本文仍基于文献[5]中的基本概念,给出了基于规约的从接续指称语义描述到直接指称语义描述的转换,包括 LET 转换和 MIU 变换. 接着给出了转

收稿日期:2005-01-28;修回日期:2005-12-28
基金项目:国家"九七三"重点基础研究发展规划基金项目(G1999032701);中国博士后科学基金项目(2004036195)

换实例,最后给出了转换的 Haskell 实现系统.

2 转换原理

Monad 技术为计算机科学注入了一些新的思路和新的技术. 我们主要吸取了 Monad 化的思想. 将给定一函数 $f:A\to B$ 转换为形如 $f':A\to M(B)$ 的函数,并且使得当取 $M(\alpha)=\alpha$ 时 f' 和 f 恒等,则称 f' 为函数 f 关于 M 的 Monad 化函数,其中 M 是论域的一种转换. 显然,f' 是 f 的一种扩充. 若取 $M\alpha=(\alpha\to C)\to C$,并定义 $f'':A\to M(B)$,$r:B\to C$,$f''a\ r=r(f\ a)$,则当取 r 为 $Id_B:B\to B$ 时有 $f''a\ r=f\ a$,因此 f'' 是 f 的扩展函数.

假设 μ_s 和 m_s 分别为语句的直接和接续指称语义指派函数,则它们有下面关系:

$$m_s[\![S]\!]\varepsilon\sigma r = r(\mu_s[\![S]\!]\varepsilon\sigma),$$

其中 r 表示语句的接续函数,即有 $r\in State\to Ans$. 由此看,接续指称语义指派函数 m_s 是直接指称语义指派函数 μ_s 关于 M 的 Monad 化函数,其中 $M\ \alpha=(\alpha\to Ans)$. 故可用 Monad 化方法来实现直接指称语义与接续指称语义描述之间的自动转换.

假设给定接续指称语义描述:

$$m_s::\ Syn\to D_1\to D_2\to SC\to Out,$$
$$SC=R\to Out,$$
$$m_s[\![S]\!]d_1d_2c=\tau[\![d_1,d_2]\!],$$

则首先从方程左部产生;其次对方程右部 $\tau([\![d_1,d_2]\!]$ 进行等价变换,使得最后变成形式 $c(E^\mu)$,其中 E^μ 是不包含 m 的表达式,它可包含 μ,但也可不包含 μ,即为不包含语义函数调用的一般表达式. 最后产生对应的函数基调和直接指称语义方程:

$$\mu_s::\ Syn\to D_1\to D_2\to R,$$
$$\mu_s[\![S]\!]d_1d_2 = E^\mu.$$

在转换中主要用到的事实是等式关系:

$$m_s[\![S]\!]d_1d_2c = c(\mu_s[\![S]\!]d_1d_2),$$

即在转换中遇到形如 $m_s[\![S]\!]e_1e_2c$ 的表达式,并且 e_1 和 e_2 中不包含 m,c 是接续参数,则将其转换成 $\mu_s[\![S]\!]e_1e_2$.

语义指派函数基调的转换规则很简单. 假设 Syn 是一个语法域,m_{Syn} 为对应的语义指派函数,SC 为对应的接续函数域,Out 为程序的结果域,m_{Syn} 的基调式为

$$m_{syn}:Syn\to Dom_1\to Dom_2\to SC\to Out,$$
$$SC = Result\to Out,$$

并且令 Syn 的直接指称语义的语义指派函数名为 μ_{Syn},则 μ_{Syn} 的函数基调如下:

$$\mu_{syn}:Syn\to Dom_1\to Dom_2\to Result.$$

利用上面的基调转换规则,很容易从接续指称语义的语义指派函数基调和接续函数域出发自动导出直接指称语义的语义指派函数基调.

例如,如果语句的接续指称语义的语义指派函数 m_s 基调和接续函数域定义为

$$m_s:Statement\to SEnv\to DEnv\to SCont\to Out,$$
$$SCont = DEnv\to Out,$$

则按基调的转换规则,应推导出〈语句〉的如下直接指称语义的语义指派函数基调:

$$\mu_s:Statement\to SEnv\to DEnv\to DEnv.$$

3 语义方程的转换

在考虑语义方程的转换规则时,为了描述方便,我们在形式上给出了如下的假设:

(1) 语义方程都是满参形式. 例如

$$m_s[\![S_1;S_2]\!]\varepsilon\sigma\kappa = m_s[\![S_1]\!]\varepsilon\sigma(\lambda\sigma'.m_s[\![S_2]\!]\varepsilon\sigma\kappa)$$

是满参形式,而下面则是非满参形式

$$m_s[\![S_1;S_2]\!]\varepsilon = \lambda\sigma\kappa.m_s[\![S_1]\!]\varepsilon\sigma(\lambda\sigma'.m_s[\![S_2]\!]\varepsilon\sigma\kappa).$$

(2) 接续语义函数调用都是满参形式. 例如 $m_s[\![S]\!]\varepsilon\sigma\kappa$.

(3) 只有一种接续,即不考虑 break 和 continue 等转移.

(4) 假设语义指派函数的基调为

$$m_{Syn}:Syn\to Dom_1\to\cdots\to Dom_k\to SynCont\to Out,$$

则其中 $Dom_i(1\leqslant i\leqslant k)$ 中不含接续域 $SynCont$(不失一般性).

(5) 用 κ 表示参数类接续函数名,ϕ 表示循环重复类接续函数名,ψ 为过程体的语义函数;其目的只是为了减少转换规则描述中的说明,实际上的方程描述并不需要这些约束,因为转换系统会自动判断属于哪一类的接续.

(6) m 表示接续语义指派函数,μ 表示直接语义指派函数.

(7) E^0 表示无语义函数调用的表达式,E^m 表示包含 m 的表达式或 E^0 表达式;E^μ 表示(不含 m)包含 μ 的表达式或 E^0 型表达式.

3.1 转换规则

(1) LET 变换

在进行从直接指称语义到接续指称语义的转换,即进行 E^{μ} 到 E^{m} 的变换时,首先进行了到顺序式的转换 :$E^{\mu}\Rightarrow E^{s}$,其作用是把嵌套在内部的语义函数调用外提到前面,例如:

$$\mu_{s}【s_2】(\mu_{s}【s_1】\sigma)\Rightarrow$$
$$\text{let } \sigma_1 = \mu_{s}【s_1】\sigma \text{ in } \mu_{s}【s_2】\sigma_1$$

而这里说的 LET 变换是具有与此相反性质的变换,例如:

$$\text{let } \kappa_1 = \lambda\sigma'.m_{s}【s_2】\sigma'\kappa \text{ in } m_{s}【s_1】\sigma\kappa_1\Rightarrow$$
$$m_{s}【s_1】\sigma(\lambda\sigma'.m_{s}【s_2】\sigma'\kappa)$$

LET 变换的作用是把‘ let 出来 ’的接续表达式嵌入到内部. 下面是进行 LET 变换的转换规则:

$$\text{let } x = E_1^m \text{ in } E_2^m\Rightarrow E_2^m[E_1^m/x]$$

通过 LET 变换,原接续指称语义方程的右部表达式 E^{m},将被转换成没有 LET 表达式. 我们记这种表达式为 E^{mL}. 做这种变换的主要原因是,其中的 LET(计算)顺序恰好与直接指称语义时的顺序相反.

(2) MIU 变换

MIU 变换的输入是 E^{mL}型表达式,它是接续指称语义方程的右部表达式 E^{m} 经过 LET 变换得到的,而输出的是直接指称语义表达式 E^{μ}. 用到的最基本事实是直接指称语义和接续指称语义的关系式:

$$m_{s}【Syn】d_1d_2\cdots d_n\kappa = \kappa(m_{s}【Syn】d_1d_2\cdots d_n)$$

很显然,如果方程右部表达式 E^{m} 可转换成 $\kappa(E^{\mu})$,则得到关系式:

$$\mu_{s}【Syn】d_1d_2\cdots d_n = E^{\mu}$$

转换步骤:

① 产生直接指称语义方程左部:

$\mu_{s}【Syn】x_1x_2\cdots x_n$;

② 进行 LET 变换 :$E^{m}\Rightarrow E^{mL}$;

③ 进行 MIU 变换 :$E^{mL}\Rightarrow E^{\mu}$;

④ 产生直接指称语义方程右部 :E^{μ}.

我们约定:

① $m_{s}【Syn】_{E_1\cdots E_n}\kappa$ 计算出程序的执行结果,并停止计算.

② $m_{s}【Syn】_{E_1\cdots E_n}\kappa, E_i$ 不会含有 m_{s}.

③ 若有接续指称语义方程:

$$m_{\text{Syn}C}【\text{Syn}】\text{xy}\kappa = \text{E}^{\text{m}},$$

其中 κ 是 Syn 的接续参数,并且有推导

$$(E^{m},\kappa)\rightarrow E^{\mu},$$

则有直接指称语义方程:

$$\mu_{\text{Syn}}【Syn】xy = E^{\mu}.$$

下面是 $(E^{mL},\kappa)\rightarrow E^{\mu}$ 的推导规则,为简单起见,下面用 E^{m} 表示 E^{mL}.

转换规则Ⅰ:

$$((E^{\mu}\mapsto E_1^m,E_2^m),\kappa)\rightarrow$$
$$(E^{\mu}\mapsto(E_1^m,\kappa),(E_2^m,\kappa))$$

转换规则Ⅱ:

$$(\text{letrec } \phi x = E_1^m \text{ in } E_2^m,\kappa)\rightarrow$$
$$\text{letrec } \phi x = (E_1^m,\kappa) \text{ in } (E^m,\kappa)$$

转换规则Ⅲ:

$$(m_{\text{Syn}}【Syn】E_1^{\mu}\cdots E_n^{\mu}(\lambda v.E^m),\kappa)\rightarrow$$
$$(E_2^m[\mu_{\text{Syn}}【Syn】E_1^{\mu}\cdots E_n^{\mu}/v],\kappa)$$
$$(m_{\text{Syn}}【Syn】E_1^{\mu}\cdots E_n^{\mu}\kappa,\kappa)\rightarrow$$
$$(\kappa(\mu_{\text{Syn}}【Syn】E_1^{\mu}\cdots E_n^{\mu}),\kappa)$$
$$(m_{\text{Syn}}【Syn】E_1^{\mu}\cdots E_n^{\mu}\phi,\kappa)\rightarrow$$
$$\phi(\mu_{\text{Syn}}【Syn】E_1^{\mu}\cdots E_n^{\mu})$$
$$(E_1^{\mu}\cdots E_n^{\mu}\kappa,\kappa)\rightarrow E_1^{\mu}\cdots E_n^{\mu}$$

转换规则Ⅳ:
$$\frac{(\kappa(E^{\mu}),\kappa)\rightarrow E^{\mu}}{(\kappa(E^{\mu}\Theta[f\rightarrow proc(\lambda xy\kappa'.E^m)]),\kappa)\rightarrow E^{\mu}\Theta[f\rightarrow proc(\lambda xy\kappa'.(E^m,\kappa'))]}$$

转换规则Ⅴ:
$$\frac{(E^m,\kappa')\rightarrow E^{\mu}}{E_0^{\mu}\Theta[f\rightarrow proc(\lambda xy\kappa'(E^m,\kappa))]\rightarrow E_0^{\mu}\Theta[f\rightarrow proc(\lambda xy.E^{\mu})]}$$

转换规则Ⅵ:
$$\frac{(E_1^m,\kappa)\rightarrow E_1^{\mu}\quad(E_2^m,\kappa)\rightarrow E_2^{\mu}}{(E^{\mu}\rightarrow(E_1^m,\kappa),(E_2^m,\kappa))\rightarrow(E^{\mu}\rightarrow E_1^{\mu},E_2^{\mu})}$$

转换规则Ⅶ:
$$\frac{(E_1^m,\kappa)\rightarrow E_1^{\mu}\quad(E_2^m,\kappa)\rightarrow E_2^{\mu}}{\text{letrec } \phi x = (E_1^m,\kappa) \text{ in } (E^m,\kappa)\rightarrow \text{letrec } \phi x = E_1^{\mu} \text{ in } E_2^{\mu}}$$

转换规则Ⅷ:
$$\frac{E\rightarrow E\quad E\rightarrow E'E'\rightarrow E''}{E\rightarrow E''}$$

例如,条件语句的接续指称语义描述如下:

$$m_{s}【\text{if } e\ s_1\ s_2】\sigma\kappa =$$
$$m_{\text{E}}【e】\sigma(\lambda v.v\rightarrow m_{s}【s_1】\sigma\kappa,m_{s}【s_2】\sigma\kappa),$$

其中右部利用上述转换规则就有如下推导过程：

$m_E【e】\sigma(\lambda v.v \to m_s【s_1】\sigma\kappa, m_s【s_2】\sigma\kappa), \kappa$

转换规则Ⅲ

$\Rightarrow(\mu_E【e】\sigma \to m_s【s_1】\sigma\kappa, m_s【s_2】\sigma\kappa), \kappa$

转换规则Ⅰ

$\Rightarrow(\mu_E【e】\sigma \to (m_s【s_1】\sigma\kappa, \kappa), (m_s【s_2】\sigma\kappa, \kappa)$

而：

$m_s【s_1】\sigma\kappa, \kappa \Rightarrow \mu_s【s_1】\sigma$ 转换规则Ⅲ，

$m_s【s_2】\alpha\kappa, \kappa \Rightarrow \mu_s【s_2】\sigma$ 转换规则Ⅲ，

上式进一步推导为

$\Rightarrow(\mu_E【e】\sigma \to \mu_s【s_1】\sigma, \mu_s【s_2】\sigma)$ 转换规则Ⅵ.

因此，该语句对应的直接指称语义方程为

$$\mu_s【\text{if } e\ s_1\ s_2】\sigma = (\mu_E【e】\sigma \to \mu_s【s_1】\sigma, \mu_s【s_2】\sigma).$$

3.2 Haskell 实现系统描述

下面，我们用类 Haskell 语言描述了接续形式表达式到直接形式表达式的转换框架，Haskell 是一种高阶函数式语言，具有较好的模块性和通用性，算法 c-to-d 和数据结构定义如下：

(1) 数据结构定义

```
type Em = (Ec, Cont)
data Ec = App Cont E0u
        | Semcalc SemCc
        | Letrec Patt Patt Em Em
        | If E0u Em Em
        | Link E0u Env
type SemCc = (SemFunc, SynName, [Patt], Cont)
data Cont = N | K | F Patt | L Patt Em
data E0 = Pattern Patt
        | Func FuncName
        | Abs [Patt] E0
        | App E0 [E0]
        | Let Patt E0 E0
        | If E0 E0 E0
data E0u = General E0
         | Semcall SemCall
         | If E0u E0u E0u
         | Abs [Patt] E0u
         | App E0u [E0u]
         | Letrec Patt Patt E0u E0u
         | Link E0u Envu
data Envu = Map Patt Attru
data Attru = Lamb [Patt] E0u
type SemCall = (SemFunc, SynName, [Patt])
data Env = Map Patt Attr
data Attr = Lamb [Patt] Cont Em
type Patt = 省略
type SemFun = 省略
type FuncName = 省略
type SynName = 省略
```

(2) 转换算法框架描述

```
ctod :: Em→E0u
```

① 简单表达式处理

```
ctod ((App K (General e0)), K) = General e0
ctod ((App (F 'p') (General e0)), K) =
App (General (Pattern 'p')) [General e0]
```

② 接续指称函数调用处理

```
ctod ((Semcall ('m','S',el), K) =
Semcall ('u','S',el)
ctod ((Semcall ('m','S',el,(F 'f'))),K) =
App (General (Pattern 'p')) [Semcall ('u','S',el)]
ctod ((Semcall ('m','S',e_l, (L 'p' em))), K) = ctod ((update em 'p' (Semcall ('u','S',e_l))),K)
```

③ 含接续指称语义函数调用的接续形表达式的处理

```
ctod ((If e (ec1, N) (ec2, N)), K) =
  let eu1 = ctod(ec1,K) in
  let eu2 = ctod(ec2,K) in If e eu1 eu2
ctod ((Letrec 'p' 'x' (ec1, N) (ec2, N)), K) =
  let eu1 = ctod (ec1, K) in
  let eu2 = ctod (ec2, K) in Letrec 'p' 'x' eu1
  eu2
ctod ((Link eu (Map 'p' (Lamb pl K (ec, N)))), K) =
  let eu = ctod (ec, K) in
  Link eu (Map 'p' (Lamb pl eu))
```

4 结束语

本文给出了基于规约的从接续指称语义描述到直接指称语义描述的转换方法，并用高阶函数式语言 Haskell 实现了具体的转换系统. 最主要的难度还在于转换不是保函数基调的转换. 通过分析接续函数的语义含义，分别处理了不同情形下的接续函数，实现了从接续指称语义描述到直接指称语义描述的转换.

参 考 文 献

1 E. J. Neubold. The formal description of programming language [J]. IBM Systems Journal, 1971, 10(2): 86~112

2 M. J. C. Gordon. The Denotational Descriptions of Programming Languages [M]. Berlin: Springer-Verlag, 1979

3 R. D. Tennent. The denotational semantics of programming language [J]. Communications of the ACM, 1976, 19(8): 437~453

4 Jin Chengzhi. Program Theory and Technology [M]. Changchun: Jilin University Press, 1997 (in Chinese)
(金成植. 程序理论和技术[M]. 长春:吉林大学出版社, 1997)

5 Lü Jianghua, Jin Chengzhi. Semantics transformation: From direct denotational semantics to continuation denotational semantics [J]. Journal of Computer Research and Development, 2004, 41(6): 972~978 (in Chinese)
(吕江花, 金成植. 语义转换:直接指称语义到接续指称语义[J]. 计算机研究与发展, 2004, 41(6): 972~978)

6 Eugenio Moggi. Notions of computation and Monads [J]. Information and Computation, 1991, 93(1): 55~92

Lü Jianghua, born in 1975. Lecturer of the College of Computer Science and Technology in Jilin Unviersity. Now post-doctor in the National Lab of Software Development Environment of Beihang University. Her main research interests include theory and technology for software formalization, reflection and mobile computation.

吕江花,1975 年生,吉林大学计算机学院讲师,北京航空航天大学软件开发环境国家重点实验室博士后,主要研究方向为软件形式化、反射技术、移动计算语义描述.

Ma Shilong, born in 1953. Professor and doctor supervisor of the College of Computer Science and Technology in Beihang University, senior member of CCF. His main research interests include computation model in network and magnanimity information, logic and behaviors in computing.

马世龙,1953 年生,教授,博士生导师,中国计算机学会高级会员,主要研究方向为网络环境下计算模型、逻辑和计算行为研究、海量信息处理的计算模型研究(slma@nlsde.buaa.edu.cn).

Pan Jing, born in 1970. Lecturer of the School of Management in the Beijing University of Science and Technology, Her main research interests include software engineer, softare test technology.

潘静,1970 年生,讲师,主要研究方向为计算机软件工程、软件测试技术(panjing@mangement.ustb.edu.cn).

Jin Chengzhi, born in 1935. Professor and doctor supervisor in the College of Computer Science and Technology in Jilin University. His main research interests include software formalization, programming theory and technology.

金成植,1935 年生,教授,博士生导师,主要研究方向为软件形式化、程序理论与技术(jcz@mail.jlu.edu.cn.).

Research Background

Under the background of the National 973 Project and China Postdoctoral Science Foundation, we analyze the relationship between direct semantics and continuation semantics in Monad view, Based on Monad thought, a concept comes from category, we gives a transformation approach from semantic aspect including the transformation of semantic function. The object system concerned here is common rather than lambda calculus. Due to Monad 's highly abstract and strict mathematic characteristics, it is capable of unifying different systems, which brings advantage to reasoning about the systems. So our transform method based on Monad is believed to be understood and proven easily.