Inferência de tipos em Python *

Eva Maia Nelma Moreira Rogério Reis
{emaia,nam,rvr}@ncc.up.pt

DCC-FC & LIACC -UP

Resumo As linguagens dinamicamente tipificadas, como a linguagem Python, permitem ao programador uma maior flexibilidade, no entanto privam-no das vantagens da tipificação estática, como a detecção precoce de erros. Este artigo tem como objectivo descrever um sistema estático de tipos para um subconjunto do Python (RPython). Acreditamos que a definição de um este sistema de inferência de tipos, como este, é um passo importante para a construção de um sistema de verificação formal de programas Python.

1 Introdução

A verificação formal de programas é, hoje, de reconhecida importância devido ao aumento da necessidade de certificar o *software* como fiável. Em especial, é importante certificar o *software* para os sistemas críticos e embebidos. Quando o desempenho destas aplicações não é crítico, a necessidade de segurança, correcção e rapidez de desenvolvimento justificam a utilização de linguagens de alto nível, como o Python.

Nos últimos trinta anos, os sistemas de tipos têm sido desenvolvidos e usados com sucesso em diferentes linguagens de programação. Um sistema de tipos, é um componente das linguagens tipificadas, que define um conjunto de regras que associam tipos aos objectos do programa. O uso de um sistema de tipos permite prevenir a ocorrência de determinados erros durante a execução do programa.

O Python [?] é uma linguagem de programação de muito alto nível, orientada a objectos e dinamicamente tipificada. Possui uma sintaxe clara, que facilita a legibilidade do código e o desenvolvimento rápido de programas.

Neste trabalho apresentamos um sistema que permite a inferência estática de tipos em Python. Este, possui algumas características que impossibilitam a inferência de tipos, na ausência de execução. Deste modo, a linguagem alvo da inferência de tipos é um subconjunto do Python, designado como RPython [?], que foi definido informalmente no âmbito do projecto PyPy [?], cujo objectivo é a possibilidade de execução eficiente de Python e a construção de um compilador "Just-in-time".

O RPython é um subconjunto do Python para o qual é possível inferir tipos em tempo de compilação. As principais características da linguagem são:

* Trabalho parcialmente suportado pela Fundação de Ciência e Tecnologia e programa POSI, e pelo projecto RESCUE (PTDC/EIA/65862/2006)

1. as variáveis têm tipo estático.
2. os tipos complexos têm que ser homogéneos.
3. não possui características introspectivas nem reflexivas.
4. não permite o uso de métodos especiais (__*__), a definição de funções dentro de funções, a definição e uso de variáveis globais e apenas permite o uso de herança simples.

Existem alguns trabalhos relacionados com a inferência de tipos em Python. No entanto, nenhum deles procede à inferência de tipos, na ausência de execução, em Python, de modo formal.

2 Sistema de tipos

Um sistema de tipos define um conjunto de regras que associam tipos aos construtores de um programa.

A sintaxe abstracta do Python sobre a qual a inferência de tipos é efectuada é definida pela seguinte gramática, na qual não faremos distinção entre expressões e comandos:

$\texttt{e}, \bar{\texttt{e}}$ ::= n | l | x | ($\texttt{e}_1$, ..., $\texttt{e}_n$) (*tuplos*) | [$\texttt{e}_1$,..., $\texttt{e}_n$] (*listas*)
| {$\bar{\texttt{e}}_1$:$\texttt{e}_1$,...,$\bar{\texttt{e}}_n$:$\texttt{e}_n$}(*dicionários*) | x=e | e op e | e opc e | e opb e
| opu e | if e: e else e | e[n]| return | return e
| while e: e else e | def f($\texttt{x}_1$...$\texttt{x}_n$):e | f($\texttt{e}_1$... $\texttt{e}_n$)
| class c():[$\texttt{e}_1$,...,$\texttt{e}_n$]| c($\texttt{e}_1$, ..., $\texttt{e}_n$) | e.m($\texttt{e}_1$,..., $\texttt{e}_n$)| e.m

onde,

n $\in$ {int, float, long}, l $\in$ constantes, x $\in$ nomes de variáveis
f $\in$ nomes de funções, c $\in$ nomes de classes, m $\in$ nomes de métodos

op::= + | - | * | << | >> | | |^ | & | / | % | ** | //
opc::= == | != | < | $\leq$ | $>$ | $\geq$ | is | not is | in | not in
opb::= and | or
opu::= not | $\sim$ | + | -

Consideremos o contexto local a uma classe, Ω, definido do seguinte modo:

$$\Omega \ \texttt{::=} \ \{\texttt{m}_0\texttt{::}\eta_0 \ldots \ \texttt{m}_n\texttt{::}\eta_n\}$$

onde η_i se encontra definido abaixo.

O conjunto de tipos possíveis para a linguagem define-se pela seguinte gramática, onde TVar representa o conjunto das variáveis de tipo, τ e α os tipos monomórficos e η os tipos polimórficos:

τ ,α ::= eTop
| eInt | eFloat | eLong | eString | eBool | eNone | $\sigma \in$ TVar
| eTuple($\tau_1 \ldots \tau_n$) | eList(τ) | eDict(τ) | eArrow([$\tau_1 \ldots \tau_n$],α)
| eClass(c,Ω) | eCCla(l, [$\texttt{c}_1$,..., $\texttt{c}_n$]) | eCv(l, [τ_1,...,τ_n])

η `::=` τ
$|$ `eAll(`$[\sigma_1,\ldots,\sigma_n]$`, eArrow(`$[\tau_1\ldots\tau_n]$`,`$\alpha$`))`

2.1 Regras de inferência

Ao conjunto das atribuições de tipo a variáveis ou funções, distintas, chamamos contexto, e representamos por Γ. O contexto é global durante todo o processo de inferência. A definição deste conjunto, onde $t_i \in x, f$, é a seguinte:

$$\Gamma ::= \{t_0 :: \eta_0, ..., t_n :: \eta_n\}$$

Dado um contexto Γ, um construtor `e` e um tipo τ, $\Gamma \vdash$ `e` $::\tau$ significa que considerando o contexto Γ é possível deduzir que o construtor `e` tem tipo τ.

De seguida, vamos definir algumas das regras de inferência para o sistema de tipos.

$$\Gamma \vdash \mathtt{x} :: \tau,\ \mathtt{se}\ (\mathtt{x} :: \tau) \in \Gamma \quad (\textsc{Var})$$

$$\frac{\Gamma \vdash \mathtt{e_i} :: \tau_\mathtt{i}\ \ 1 \le \mathtt{i} \le \mathtt{n}}{\Gamma \vdash (\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eTuple}\ ([\tau_1, ..., \tau_\mathtt{n}])} \quad (\textsc{Tuplo})$$

$$\frac{\Gamma \vdash \mathtt{e_i} :: \tau\ \ 1 \le \mathtt{i} \le \mathtt{n}}{\Gamma \vdash [\mathtt{e_1}, ..., \mathtt{e_n}] :: \mathtt{eList}\ (\tau)} \quad (\textsc{Lst})$$

$$\frac{\Gamma \vdash \bar{\mathtt{e_i}} :: \alpha_\mathtt{i}\ \ \mathtt{hashable}(\alpha_\mathtt{i}) \qquad \Gamma \vdash \mathtt{e_i} :: \tau\ \ 1 \le \mathtt{i} \le \mathtt{n}}{\{\bar{\mathtt{e_1}} : \mathtt{e_1}, ..., \bar{\mathtt{e_n}} : \mathtt{e_n}\} :: \mathtt{eDict}\ (\tau)} \quad (\textsc{Dic})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \tau_1\ \ \Gamma \vdash \mathtt{x} :: \tau_2 \qquad \tau_1 <: \tau_2\ \mathtt{ou}\ \tau_2 <: \tau_1}{\Gamma \vdash \mathtt{x} = \mathtt{e} :: \mathtt{eNone}} \quad (\textsc{Atr})$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \tau_1\ \ \Gamma \vdash \mathtt{e_2} :: \tau_2\ \ \tau_1 <: \tau_2}{\Gamma \vdash \mathtt{e_1} + \mathtt{e_2} :: \tau_2} \quad (\textsc{Opb1})$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \tau_1\ \ \Gamma \vdash \mathtt{e_2} :: \tau_2\ \ \tau_2 <: \tau_1}{\Gamma \vdash \mathtt{e_1} + \mathtt{e_2} :: \tau_1} \quad (\textsc{Opb2})$$

$$\tau_1, \tau_2 \in \{eInt,\ eFloat,\ eLong,\ eString,\ eTuple(\alpha),\ eList(\alpha)\}$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \tau_1\ \ \Gamma \vdash \mathtt{e_2} :: \tau_2 \qquad \tau_1 <: \tau_2\ \mathtt{ou}\ \tau_2 <: \tau_1}{\Gamma \vdash \mathtt{e_1}\ \mathtt{opc}\ \mathtt{e_2} :: \mathtt{eBool}} \quad (\textsc{Opc1})$$

$$\frac{\Gamma \vdash \mathtt{e_1} :: \mathtt{eBool}\ \ \Gamma \vdash \mathtt{e_2} :: \mathtt{eBool}}{\Gamma \vdash \mathtt{e_1}\ \mathtt{opb}\ \mathtt{e_2} :: \mathtt{eBool}} \quad (\textsc{OpBool})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eList}(\tau)\ \ \mathtt{i} :: \mathtt{eInt}}{\Gamma \vdash \mathtt{e}[\mathtt{i}] :: \tau} \quad (\textsc{Alst1})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eList}(\tau)\ \ \mathtt{n} :: \mathtt{eInt}\ \ \mathtt{m} :: \mathtt{eInt}}{\Gamma \vdash \mathtt{e}[\mathtt{n} : \mathtt{m}] :: \mathtt{eList}(\tau)} \quad (\textsc{Alst2})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eDict}(\tau) \qquad \Gamma \vdash \mathtt{i} :: \alpha\ \ \mathtt{hashable}(\alpha)}{\Gamma \vdash \mathtt{e}[\mathtt{i}] :: \tau} \quad (\textsc{Adic1})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \mathtt{eDict}(\mathtt{eNone}) \qquad \Gamma \vdash \mathtt{i} :: \alpha\ \ \mathtt{hashable}(\alpha)}{\Gamma \vdash \mathtt{e}[\mathtt{i}] :: \mathtt{eTop}} \quad (\textsc{Adic2})$$

$$\Gamma \vdash \mathtt{return} :: \mathtt{eNone} \quad (\textsc{Return1})$$

$$\frac{\Gamma \vdash \mathtt{e} :: \tau}{\Gamma \vdash \mathtt{return\ e} :: \tau} \quad (\textsc{Return2})$$

$$\frac{\Gamma \vdash \mathtt{e_0} :: \mathtt{eBool}\ \ \Gamma \vdash \mathtt{e_1} :: \tau \qquad \Gamma \vdash \mathtt{e_2} :: \alpha\ \ \ \tau <: \alpha}{\Gamma \vdash \mathtt{if}\ \mathtt{e_0} :\ \mathtt{e_1}\ \mathtt{else}\ \mathtt{e_2} :: \alpha} \quad (\textsc{Cond1})$$

$$\frac{\Gamma \vdash \mathtt{e_0} :: \mathtt{eBool}\ \ \Gamma \vdash \mathtt{e_1} :: \tau \qquad \Gamma \vdash \mathtt{e_2} :: \alpha\ \ \alpha <: \tau}{\Gamma \vdash \mathtt{if}\ \mathtt{e_0} :\ \mathtt{e_1}\ \mathtt{else}\ \mathtt{e_2} :: \tau} \quad (\textsc{Cond2})$$

$$\frac{\bar{\Gamma} = \{\mathtt{x_i} :: \tau_{\mathtt{i}}\}\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}\ \bar{\Gamma} \cup \Gamma' \vdash \mathtt{e} :: \alpha}{\Gamma'' \vdash \mathtt{def\ f\ (x_1, ..., x_n) : e}\ ::\ \mathtt{eArrow}([\tau_1, ..., \tau_{\mathtt{n}}], \alpha)}\ (\textsc{DefFunc})$$

$\Gamma'' = \Gamma \cup f :: eArrow([\tau_1, ..., \tau_n], \alpha)$

$$\frac{\Gamma \vdash \mathtt{f} :: \mathtt{eArrow}([\tau_1, ..., \tau_{\mathtt{n}}], \alpha)\ \Gamma \vdash \bar{\mathtt{e}}_{\mathtt{i}} :: \alpha_{\mathtt{i}}\ \alpha_{\mathtt{i}}\ <:\ \tau_{\mathtt{i}}\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}}{\Gamma \vdash \mathtt{f}(\bar{\mathtt{e_1}}, ..., \bar{\mathtt{e_n}})\ ::\ \alpha}\ (\textsc{Aplicação})$$

$$\frac{\bar{\Gamma} = \{\ \mathtt{e_i} :: \tau_{\mathtt{i}}\}\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}\ \bar{\Gamma} \cup \Gamma' \vdash \mathtt{e} :: \alpha}{\Gamma'' \vdash \mathtt{def\ f(e_1, ..., e_n) : e}\ ::\ \mathtt{eAll}([\tau_{\mathtt{i}} \in \mathtt{TVar}], \mathtt{eArrow}([\tau_{\mathtt{i}}], \alpha))}\ (\textsc{Generalização})$$

$\Gamma'' = \Gamma \cup f :: eAll([\sigma_1, ..., \sigma_n], eArrow([\tau_1, ..., \tau_n], \alpha))$

$$\frac{\Gamma' \vdash \mathtt{e_i} :: \eta_{\mathtt{i}}\ \ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}}{\Gamma \vdash \mathtt{class\ c() : [e_1, ..., e_n]}\ ::\ \mathtt{eClass}(\mathtt{c}, \{\mathtt{m_1} :: \eta_1, ..., \mathtt{m_n} :: \eta_{\mathtt{n}}\})}\ (\textsc{DefCla})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{c} :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Gamma,\ \Omega \vdash \mathtt{\_\_init\_\_}(\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eNone}()\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eClass}(\mathtt{c}, \Omega)}\ (\textsc{Inst1})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{c}(\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Omega \vdash \mathtt{m}(\tau_1, ..., \tau_{\mathtt{n}}) :: \eta \\ \Gamma \vdash \bar{\mathtt{e_i}} :: \alpha_{\mathtt{i}}\ \alpha_{\mathtt{i}} <: \tau_{\mathtt{i}}\ \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e_1}, ..., \mathtt{e_n}).\mathtt{m}(\bar{\mathtt{e_1}}, ..., \bar{\mathtt{e_n}}) :: \eta}\ (\textsc{Acm1})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{c} :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Gamma,\ \Omega \vdash \mathtt{\_\_init\_\_}(\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eClass}(\mathtt{c}, \Omega)\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eClass}(\mathtt{c}, \Omega)}\ (\textsc{Inst2})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{c}(\mathtt{e_1}, ..., \mathtt{e_n}) :: \mathtt{eClass}(\mathtt{c}, \Omega) \\ \Omega \vdash \mathtt{m} :: \eta\end{array}}{\Gamma \vdash \mathtt{c}(\mathtt{e_1}, ..., \mathtt{e_n}).\mathtt{m} :: \eta}\ (\textsc{Acm2})$$

3 Conclusão

Actualmente a certificação de software, como correcto e seguro, é de extrema importância, especialmente para sistemas críticos e embebidos. Muitas das aplicações usadas nestes sistemas são desenvolvidas em linguagens de alto-nível, como o Python. Desejamos encadear este sistema de inferência de tipos com uma ferramenta de produção de obrigações de prova. Assim, o desenvolvimento deste sistema estático de inferência de tipos foi apenas o primeiro passo para um projecto futuro que implemente a certificação estática de programas em Python.