SimpleObject に対する型検査 — 型制約言語と推論アルゴリズム

1G－6

大久保弘崇　坂部俊樹　稲垣康善

名古屋大学工学部

1 はじめに

型なしのオブジェクト指向言語のモデル SimpleObject [1] に対する型推論系を構築する。プログラムの式が評価されたときにとる値のクラスの集合をその式の型と考え、構文から導かれるこの型に対する制約をもとに型を導出する。クラス定義に対して、そのメソッドやインスタンス変数の間の型の関係をクラス定義に対する型の情報として推論する。これをクラス制約と呼ぶ。モジュールをなすクラスの集まり、もしくはプログラムをなすクラスの集まりに対してはクラス制約をもとに型制約を解くことによって型の整合性が判定される。制約を解くには集合を値とする制約評価系を用いる。

2 クラス記述からの制約集合の導出

クラス定義の不完全な記述であるクラス記述を入力として、クラス制約の元となる制約集合を返す手続きについて述べる。この段階では継承についてクラス定義が不完全なため、制約集合は完全なものではない。

クラス記述の構文は

> **class** クラス名 [**inherits** 親クラス名]
> [**var** インスタンス変数宣言]
> [メソッド定義]*

である。これに対して制約式
`?-exist_class(` 親クラス名 `).`
を作る[1]。また各インスタンス変数 id の宣言に対して制約 `exist_variable`(id) を作る。

メソッド定義の構文は

> **method** メッセージ名 [(仮引数宣言)]
> [**var** 一時変数宣言]] 式

である。メソッド メッセージ名 の宣言に対して
`exist_method(`□. メッセージ名 `).`
argument(□. メッセージ名)
$= \langle$□. メッセージ名 .[仮引数宣言],...$\rangle$
その仮引数・一時変数 v の宣言に対して
`exist_variable(`□. メッセージ名 .v`).`
という制約を作る。

メソッド本体の式とその各部分式 E に対して、次のような型の変数を導入する。

> クラス名 . メッセージ名 .$addr$

ただし $addr$ はメソッド本体の式の構文木での部分式 E の位置を表す。これらの変数に関する制約を構文にしたがって以下のように求める。ただし式はメッセージ名 $\mathcal{M}$ のメソッドの記述であるとする。位置 $addr.i$ の部分式を E_i とする。□ はクラス名が後に入るための空欄である。

代入 $id := \mathrm{E}_1$
　variable(id) $\supseteq$ □.$\mathcal{M}$.$addr$.1
　□.$\mathcal{M}$.$addr$ = □.$\mathcal{M}$.$addr$.1

メッセージ送信 $\mathrm{E}_1 <= mes(\mathrm{E}_2,\ldots,\mathrm{E}_n)$
　□.$\mathcal{M}$.$addr$.1 $\subseteq$ HAS(mes)
　domain(□.$\mathcal{M}$.$addr$.1, mes)
　　$\supseteq \langle$□.$\mathcal{M}$.$addr$.2,...,□.$\mathcal{M}$.$addr$.n$\rangle$
　□.$\mathcal{M}$.$addr$ = range(□.$\mathcal{M}$.$addr$.1, mes)

継承メソッドの呼びだし[2] inherited mes($\mathrm{E}_1,\ldots,\mathrm{E}_n$)
　`?-exist_method(` 親クラス名 `.`mes`).`
　domain({ 親クラス名 }, mes)
　　$\supseteq \langle$□.$\mathcal{M}$.$addr$.1,...,□.$\mathcal{M}$.$addr$.n$\rangle$
　□.$\mathcal{M}$.$addr$ =
　　range({ 親クラス名 }, mes)

ブロック化 $\{\mathrm{E}_1,\ldots,\mathrm{E}_n\}$
　□.$\mathcal{M}$.$addr$ = □.$\mathcal{M}$.$addr$.n

条件分岐 if E_1 then E_2 else E_3
　□.$\mathcal{M}$.$addr$ = □.$\mathcal{M}$.$addr$.2 $\cup$ □.$\mathcal{M}$.$addr$.3

インスタンス生成 クラス名 new
　`?-exist_class(` クラス名 `).`
　□.$\mathcal{M}$.$addr$ = { クラス名 }

インスタンス生成 myClass new
　□.$\mathcal{M}$.$addr$ = {□}

自己参照 self　□.$\mathcal{M}$.$addr$ = {□}

Type Checking for SimpleObject — it's type constraint language and inference algorithm
Hirotaka OKUBO,Toshiki SAKABE,
Yasuyoshi INAGAKI (Nagoya University)

[1] 継承を用いずに定義されているクラスの場合この制約式は必要ない。

[2] 継承なしにクラスが定義されているならばこの時点でエラーである。

変数参照 id　　$\square.\mathcal{M}.addr = \text{variable}(id)$

空値 nil　　$\square.\mathcal{M}.addr = \{\}$

ここで variable は変数の出現をその型変数に対応させる方法を表し、
variable(id) =

$$\begin{cases} \texttt{exist\_variable}(\square.\mathcal{M}.id) \text{ が宣言されていれば} \\ \quad \square.\mathcal{M}.id \\ \text{さもなければ } \texttt{?-exist\_variable}(\square.id) \text{ を作り} \\ \quad \square.id \end{cases}$$

3　クラス制約の生成

クラス記述は継承がある場合にはクラスの定義として完全なものではない。これを完全なものとするために、継承により参照している全ての上位クラスの記述に対して前節で定義した制約集合を合わせてクラス定義に対する制約のまとまりであるクラス制約を作る。このときに空欄 □ が適当なクラス名で置き換えられ、正しい制約式となる。

クラス C のクラス制約を求める手続き

1. C とその全ての上位クラスに対する制約集合を前節の方法で求める。(それぞれは分けてとっておく)
2. 継承なしで定義されているもっとも大元のクラスを V とする。
3. V 中の全てのメソッド定義 $\square.\mathcal{M}$ をマークする。
4. 全てのマークされたメソッド $\mathcal{N}$ について、それが V の子クラスで再定義されているとき、V 中の全ての $\square.\mathcal{N}$ および $\square.\mathcal{N}.addr$ の □ に V を入れ、メソッド定義 $V.\mathcal{N}$ に対するマークを外す。
5. $V \neq C$ なら V を V の子クラスに変えて 3 へ
6. 残った全ての □ に C を入れる。
7. 全体を一つの制約集合として制約評価系によって評価して標準形を計算し、型パラメータを取り出す。
8. $\texttt{exist\_class}(C).$ を追加する。

ただし、必要ならば以下の定義を用いて制約を評価する。

HAS(メッセージ名)
$\stackrel{\text{def}}{=} \{\forall C | \texttt{exist\_method}(C.\ \text{メッセージ名}\ )\}$
domain($Type$, メッセージ名)
$\stackrel{\text{def}}{=} \bigcap_{C \in Type} argument(C,\ \text{メッセージ名}\ )$
range(Type, メッセージ名)
$\stackrel{\text{def}}{=} \bigcup_{C \in \text{Type}} C.\ \text{メッセージ名}$

型パラメータとは、$\mathcal{V} \subseteq$ HAS(mes) の形の制約を持つ $\mathcal{V} = C.\mathcal{M}.id$ or $C.id$ である。HAS 式の値は本来実行時に存在する全てのクラスが存在しない限り定まらないが、これを型パラメータとすることで、プログラマがそのパラメータに予想する型を代入して型の整合性を調べることができる。

4　モジュラな型検査

クラス定義をプログラミングした後、変数についた型制約にプログラマの想定していたクラスを型パラメータに代入してみることによって、「少なくともそのクラスで使う分には型が正しい」という意味での型の整合性の確認ができる。

具体的な手続きは次の通りである。モジュールとして考えるクラスのクラス制約を求め、それらをまとめて制約集合とする。クラス制約で導かれる型注釈を参照して、着目しているクラス制約の型パラメータに想定していた型を割り当てる。これを制約評価系で評価したときに矛盾が起きなければ、このモジュールはクラス間整合性が保たれているといえる。

5　プログラム型検査

実行に必要なクラスが全て揃ったとき、それはプログラムとなるが、このとき（そのプログラムの実行については）完全に型検査をすることができる。これは HAS 式の値が完全に求まるため、型パラメータへの値の割り当ても制約の評価として自動的に定まるからである。

プログラム型検査は、クラス制約を集めて制約集合を作った後、まず HAS 式の値を定めてから制約評価をすればよい。

6　まとめ

クラス定義単位に型の情報を取りだし、それらの整合性を確かめる手順のモジュラな型検査方法を提案し、型制約言語を定義した。集合を値とする制約評価系によって型の整合性が検査される。

メッセージ送信式の型を、その引数の型によって異なるような一次的な代入の機構を用いることによって、より厳密な型を付けることが可能であるが、これは今後の課題である。

参考文献

[1] SimpleObject におけるクラス定義単位の型制約導出と型検査、大久保弘崇、坂部俊樹、稲垣康善、プログラミング - 言語・基礎・実践研究会報告 14-7、pp.57-64、1993