社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

形式的手法に基づくJavaScriptプログラムの型検査系の実現

大久保弘崇† 山本晋一郎† 坂部 俊樹†† 稲垣 康善†

† 愛知県立大学 情報科学部
〒 480-1198 愛知県愛知郡長久手町大字熊張字茨ヶ廻間 1522–3
†† 名古屋大学大学院 情報科学研究科
〒 464-8603 名古屋市千種区不老町
E-mail: †{ohkubo,yamamoto,inagaki}@ist.aichi-pu.ac.jp, ††sakabe@is.nagoya-u.ac.jp

あらまし 我々は，形式的手法の実際的応用を目指し，次の 3 つの視点から研究を進めている．第 1 は，型のないオブジェクト指向言語である JavaScript に対する型検査手法を提案すること．この型検査により，JavaScript プログラムの安全性を高めることができる．第 2 は，ソフトウェア工学の問題に汎用の定理証明系を適用することにより，形式的手法を現実的な問題に直接適用する可能性を探ること．このため，型検査系を推論規則により形式的に定義し，これを極力そのままの形で実際のプログラムに適用する．第 3 は，上の目的に適合する定理証明系のための言語を提案すること．ソフトウェア工学の現実的な問題に定理証明系を適用する際に必要となるシンタックスシュガーや演算定義能力についての知見を得る．本稿では，JavaScript のサブセット言語に対する単純な型推論系に基づいて，モデル生成型の定理証明系による自動証明による型検査を試みる．
キーワード JavaScript, 型検査, 推論系

A Formal Approach to Type Inference System for JavaScript Programs

Hirotaka OHKUBO†, Shinichiro YAMAMOTO†, Toshiki SAKABE††, and Yasuyoshi INAGAKI†

† Aichi Prefectural University, Faculty of Information Science and Technology
1522–3, Ibaragabasama, Kumabari, Nagakute-cho, Aichi
†† Nagoya University, Graduate School of Information Science
Furo-cho, Chikusa-ku, Nagoya, Aichi
E-mail: †{ohkubo,yamamoto,inagaki}@ist.aichi-pu.ac.jp, ††sakabe@is.nagoya-u.ac.jp

Abstract We investigate application of formal methods to practical type inference problems from the following three view points: The first point is to propose a type checking method for programs in JavaScript, which is an untyped object-oriented language. Type checking improves the safety of JavaScript programs. The second is to explore the possibility of applying formal methods to practical problems through applying general purpose theorem prover to a problem of software engineering. The third is to propose a language for theorem provers suitable for our purpose. This paper reports our investigation of constructing a simple type inference system for a subset of JavaScript by using a theorem prover based on model generation.
Key words JavaScript, type inference, theorem prover

1. はじめに

社会のあらゆる場面で広く用いられている Web アプリケーションは，(1)XML や HTML などのコンテンツ記述，(2) レンダリングのための CSS や XSLT, (3) Java Server Page や Servlet などのサーバーサイド・プログラム, (4) JavaScript や Applet などのクライアントサイト・プログラムなどから構成される大規模でヘテロジニアスな分散ソフトウェアである．

信頼性が要求されないエンドユーザ向けの情報提供から始まった Web を基幹業務に使用する動きが広がりつつあるため，その信頼性を確保することは社会的課題である．しかし，サーバサイド・プログラムの信頼性を向上試みる研究は多い [1]ー[3] が，クライアントサイド・プログラムを対象とした研究は少ない [4]．システム全体の信頼性は，システム中の最弱のサブシス

```
P ∈ Program   ::=  D *
B ∈ Body      ::=  var y; e
D ∈ FuncDecl  ::=  function f (x)  B

e ∈ Exp       ::=  this              レシーバ
                   f                  関数
                   x                  引数
                   y                  局所変数
                   new f(e)           オブジェクト生成
                   null               空
                   e;e                順次
                   e.m(e)             メンバ関数呼び出し
                   e.m                属性読み出し
                   f(e)               大域関数呼び出し
                   v=e                代入

v ∈ Var       ::=  x | y | e.m
f ∈ 関数名
m ∈ メンバ名
```

図 1　BabyJ の文法

テムによって規定されるため，クライアントサイドで動作するスクリプト言語を対象とした信頼性向上は急務である．

本研究では，広く用いられているクライアントサイドのスクリプト JavaScript を対象として，型推論規則を素直な形で定理証明上に実現することにより，JavaScript プログラムの信頼性を向上することをめざす．ここでいう素直な形とは，型推論規則を Prolog に代表される反駁方式の推論規則に変換することではなく，モデル生成方式の定理証明を用いることをいう．モデル生成方式を用いる理由として，基となる型推論規則と素直な対応が取れて見通しが良いこと，証明結果を利用者に提示するなどの応用が容易であることが挙げられる．

本稿では，型検査系として文献 [5] で提案されている BabyJ^T をとりあげる．BabyJ^T は，JavaScript 言語のサブセットに対する型システムである．形式化の段階で，型検査系の定義のあいまいさを取り除く作業を行う．汎用の定理証明系として，文献 [6] で開発された JavaMGTP を用いる．単独で動作する証明系でなく一般的なプログラミング言語である Java と協調して動作するシステムを用いることで，既存の CASE ツールプラットフォームとの親和性が期待できる．

2. BabyJ 言語とその型推論系

2.1 BabyJ

BabyJ 言語は，JavaScript 言語のサブセットである [5]．機能を本質的な部分に限定し，その制限した文法の上でプログラムに対する型推論系の形式化，および型付けされたプログラムの Java プログラムへの変換手法について議論している．BabyJ の文法を図 1 に示す．型推論系に影響する JavaScirpt からの本質的な制限は，関数の引数は唯一 `x`，関数本体の局所変数は唯一 `y`，値としての関数はオブジェクトでない，メンバの削除はできない，文字列の eval ができない，の 5 点である．

2.2 BabyJ^T

BabyJ プログラムに型を付与したものを BabyJ^T と呼ぶ．BabyJ^T では，関数は `new` 演算子とともに用いられるコンストラクタ関数，(大域) 関数呼び出しをされる大域関数，そのいずれでもないメンバ関数に区分され，区分外の形で呼び出されることはないと仮定する．オブジェクトの型を区別するために，生成に用いたコンストラクタ関数の名前を用いる．これを，オブジェクト型と呼ぶ．メンバ関数がメンバ関数として動作するためには，コンストラクタなどでまずメンバに代入される必要がある．このように，メンバ関数は値として扱われるため，型をつける必要がある．関数型は，this の型，引数の型，返り値の型の 3 項組と定義される．this はオブジェクトであることから先頭の要素は必ずオブジェクト型になるが，後二者には，別の関数型が入ってもよい．BabyJ^T の型システムには，多相性はない．全ての部分式，全ての関数の this・引数・局所変数・返り値，全てのオブジェクト型と全ての属性名の組，に対して，プログラム全体で一意にオブジェクト型あるいは関数型を割り当てる．関数からその実行時の this・引数・局所変数・返り値の型への対応づけを写像 $\mathcal{P}(f) = (ts, t_{\mathrm{arg}}, t_{\mathrm{local}}, t_{\mathrm{ret}})$, オブジェクト型からそのメンバの型への対応づけを写像 $\mathcal{D}(ts, m) = t$ とする．BabyJ^T の型推論規則を図 2 に示す．ここで Γ は変数 `this`, `x`, `y` とその型との対応づけである．$t \approx t'$ は，t が t' と同じであるか，いずれかが初期値 * であるという関係である．推論の手続きはインクリメンタルに進行する．初期状態として全ての型割当を特殊な値 * に設定する．そして，推論規則を用いて，可能な部分から徐々に型割当を真の値に更新していく．その過程で，$\mathcal{P}$ と $\mathcal{D}$ も更新される．全ての * を型に置き換えることができたら，型推論は成功である．

2.3 BabyJ^T 型推論の問題点

「型割当の初期値を * として，型が判明した部分から確定して情報を付加していく」と手続きが説明されているが，式に対する型割当の推論結果を $\mathcal{P}, \mathcal{D}$ の写像に反映させる手続きが述べられていない．このため，制約の不動点としては解が存在するが，推論規則が型変数の更新を相互に待ち続けるデッドロック状態になり，手続きでは解が得られない形が存在する．

応用を考える際，組み込みデータ型の追加が必須となる．その方法は与えられていない．

3. JavaMGTP

JavaMGTP は，モデル生成法に基づく定理証明系である．値域限定性という制限をもつが，ホーン節に限らない一階述語論理の問題一般を扱える．特に，否定を含む推論を効率よく実行できる．MGTP の入力は `a1,a2,⋯,an -> b1;b2;⋯;bn.` という形の規則の並びである．これは「a1,a2,⋯,an がすべて成立するとき b1,b2,⋯,bn のいずれかが成立する」と読む．リテラルの前に `-` を置くと否定となる．前件，後件のいずれも否定できる．前件, 後件は 0 個でもよい．前者は正節と呼びリテラルが恒真であることを，後者は負節と呼び前件が成立したとき矛盾することを表す．前件が共通する複数の規則の後件をコンマで続けて記述することができる．

(Var)

$$\frac{}{\Gamma \vdash \mathtt{x} : \Gamma(\mathtt{x})\quad \Gamma \vdash \mathtt{y} : \Gamma(\mathtt{y})\quad \Gamma \vdash \mathtt{this} : \Gamma(\mathtt{this})}$$

(Null)

$$\frac{}{\Gamma \vdash \mathtt{null} : \mathtt{ts}}$$

(Seq)

$$\frac{\Gamma \vdash \mathtt{e}_1 : t' \quad \Gamma \vdash \mathtt{e}_2 : t}{\Gamma \vdash \mathtt{e}_1;\mathtt{e}_2 : t}$$

(New-cons)

$$\frac{\mathtt{f}\text{ はコンストラクタ} \quad \mathcal{P}(\mathtt{f}) = (\mathtt{f}, t, t'', t''') \quad \Gamma \vdash \mathtt{e} : t' \quad t \approx t'}{\Gamma \vdash \mathtt{new\ f(e)} : \mathtt{f}}$$

(Member-sel)

$$\frac{\Gamma \vdash \mathtt{e} : t' \quad \mathcal{D}(t', \mathtt{m}) = t}{\Gamma \vdash \mathtt{e.m} : t}$$

(Var-ass)

$$\frac{\Gamma \vdash \mathtt{e} : \mathtt{t} \quad \Gamma(\mathtt{v}) \approx t}{\Gamma \vdash \mathtt{v} = \mathtt{e} : t}$$

(Member-ass)

$$\frac{\Gamma \vdash \mathtt{e1} : t' \quad \Gamma \vdash \mathtt{e2} : t'' \quad \mathcal{D}(t', \mathtt{m}) = t \quad t \approx t''}{\Gamma \vdash \mathtt{e1.m} = \mathtt{e2} : t''}$$

(Member-call)

$$\frac{\Gamma \vdash \mathtt{e1} : t' \quad \Gamma \vdash \mathtt{e2} : t'_1 \quad \mathcal{D}(t', \mathtt{m}) = (t', t_1, t'') \quad t_1 \approx t'_1}{\Gamma \vdash \mathtt{e1.m(e2)} : t''}$$

(Global call)

$$\frac{\mathtt{f}\text{ は大域関数} \quad \Gamma \vdash \mathtt{e2} : t'' \quad \mathcal{D}(t', \mathtt{m}) = t \quad t \approx t''}{\Gamma \vdash \mathtt{e1.m} = \mathtt{e2} : t''}$$

(Member-func)

$$\frac{\mathtt{f}\text{ はメンバ関数} \quad \mathcal{P}(\mathtt{f}) = (t_1, t_2, t_3, t_4)}{\Gamma \vdash \mathtt{f} : (t_1, t_2, t_4)}$$

図 2　BabyJ^T の型推論規則

評価述語という特徴的な機能がある．{ } で囲って数値演算や，ユーザ定義の Java クラスメソッドの呼び出しが行える．この機能を用いて，項の上の演算をユーザサイドで拡張できる．また，JavaMGTP のクラスを取り込むことで，一般の Java アプリケーションから JavaMGTP の推論機能を利用することができる．CASE ツールプラットフォームと連携した実際的な応用を考える際，これらの特徴は大きな利点となる．

4. BabyJ^T 型推論の MGTP による実現

JavaMGTP を用いて，BabyJ^T 型推論規則を直接適用することを試みる．適当なプリプロセッサを用いて，入力する JavaScript プログラムを MGTP の正節に変換する．これに，MGTP の文法に変換した推論規則を合わせたものを JavaMGTP で処理し，モデルが得られれば，入力プログラムに正しく型が割り当てられる．以下，4.2 節–4.5 節でプリプロセッサでの処理，4.6 節–4.8 節で BabyJ^T 推論規則の MGTP への変換について述べる．

4.1　基本データ型の追加

整数，文字列などの基本データ型を構文に追加する．これらの値は，その型を一意に求めることができ，型推論においてはオブジェクト型とほぼ同様の扱いになる．ここでは，integer 型を追加することにする．

4.2　抽象構文木

BabyJ 式の構文規則を，MGTP で処理可能な項の形に変換する．関数記号の対応を以下に示す．

	BabyJ 式	MGTP 項
レシーバ	this	this
関数	f	f
引数参照	x	x
局所変数参照	y	y
オブジェクト生成	new f(e)	new(f,e)
空	null	null
順次	e1;e2	seq(e1,e2)
メンバ関数呼び出し	e1.m(e2)	mcal(e1,m,e2)
属性読み出し	e.m	sel(e,m)
大域関数呼び出し	f(e)	gcal(f, e)
代入	x=e	xass(e)
	y=e	yass(e)
属性代入	e1.m=e2	mass(e1,m,e2)
整数	n	intval

値として使われる関数は，関数名そのものの定数項で表す．BabyJ には変数は関数の引数 x と局所変数 y しかないので，これらへの代入は xass と yass とした．整数の値そのものは型推論には必要ないので，定数項 intval で代表させる．

4.3　関数の区分，関数名と本体の対応づけ

オブジェクト生成式で使用されている関数はコンストラクタと決める．大域関数呼び出しで使用されている関数は大域関数と決める．そのいずれでもない関数をメンバ関数とする．判定の結果，全ての関数は 3 種のいずれかになるので，これを

-> cf(f).　f はコンストラクタ
-> gf(f).　f は大域関数
-> mf(f).　f はメンバ関数

とする。

関数本体は 4.5 節で述べる述語 e を用いて

-> e(関数本体の MGTP 項, 関数名).

で対応づける．関数本体に含まれずにトップレベルに位置する式は，無名の main 関数の本体に出現するものとみなし，同様に対応づけをする．

4.4　型ドメインの要素の列挙

4.4.1　関数型の制限

MGTP は、推論規則が与えられると，正負リテラルの集合を生成する，この集合が推論規則を充足する場合，モデルとみなして結果とする．規則の左辺のリテラルがモデル候補に存在するとき，右辺のリテラルをモデルの候補に追加していく．BabyJ^T の型推論規則を実現するにあたって，関数型の空間が無限である点が問題になる．関数型の 3 項組を ft(F,A,B) という項で，項 T が型のドメインに含まれることを type(T) という述語で表すとする．関数型がまた型であることは

cf(F),type(A),type(B) -> type(ft(F,A,B)).

と書けるが，JavaMGTP はこの規則から導かれる無限個の関数型を全てモデルに組み込もうとする．関数型の入れ子の段数を定数で制限することで，関数型の数を有限に制限できる．JavaScript では関数はファーストクラスオブジェクトであるが，その上の演算は限られているため，実行時に任意の段数の関数型が必要になることはない．また，JavaScript の利用目的からみても，極端な段数の関数型への対応は重要な問題でない．本稿では，入れ子の段数は 0 段，すなわち関数型はオブジェクト型の 3 項組と再定義する．

4.4.2 存在限量

MGTP において,「ある述語を満たす要素がドメインに存在する」という命題はドメインの要素があらかじめ列挙可能な場合にのみ記述できる．ドメインの要素が a1,a2,···,an のとき，

```
-> dom(a1),dom(a2),···,dom(an).
-> p(a1);p(a2);···;p(an).
```

でドメインの要素のいずれかで p が真であることを，さらに

```
p(a1) -> -p(a2),···,-p(an).
p(a2) -> -p(a1),···,-p(an).
:
p(an) -> -p(a1),···,-p(a(n-1)).
```

でドメインのいずれか 1 要素のみで p が真であることを記述できる．ドメインが整数である場合に限り，後半は

```
p(A),dom(B),{A != B} -> -p(B).
```

と表現することもできるが，一般の項に対してこのような等価性の判定機能は JavaMGTP には組み込まれていない．

4.4.3 全ての型の生成

型の上の not equal 関係が後段の処理で必要になる．また，$\mathcal{P},\mathcal{D}$ といった写像を表現するために，唯一存在するという関係が必要になる．このためには前節で述べた通り，型ドメインの要素を列挙しなければならない．

基本データ型，オブジェクト型，そしてそれらを部分要素としてとる関数型すべてを，手続き的に列挙することができる．関数型の入れ子を許す段数を定数で定めれば，0 段に限らず列挙できるが，その数は爆発的に増加するので現実的でないと考えられる．列挙した全ての型のリストを利用して，4.4.2 節の方法を使うことで型であるという述語 `type(t)`．異なる型であるという述語 `ne(t1,t2)` を宣言する．$\mathcal{P},\mathcal{D}$ は型からの関数なので，これを表す述語もここで宣言するべきであるが，$\mathcal{P}$ は列挙の数が爆発的なので，半関数となる近似表現を用いる，$\mathcal{D}$ は本質的に半関数であるのでやはりここでは宣言しない．

4.5 出　　現

順次実行などの複式に関する型推論規則をそのまま MGTP 規則として記述すると，その規則によってモデル生成が爆発してしまう．これを防ぐため,「型を推論する必要のある式 a が出現 o に存在する」ことを表す述語 `e(a,o)` を導入する．入力プログラムに対して必要な `e(a,o)` 項は，プリプロセッサで生成することもできるし，図 3 に示す推論規則を用いて，トップの式から展開して得ることもできる．ここで `mem` はメンバ名であることを表す述語である．

```
e(seq(E1,E2),O) -> e(E1,l(O)),e(E2,r(O)).
e(new(F,E),O) -> e(E,l(O)).
e(sel(E,M),O) -> e(E,l(O)),mem(M).
e(xass(E),O) -> e(E,l(O)).
e(yass(E),O) -> e(E,l(O)).
e(mass(E1,M,E2),O) -> e(E1,l(O)),e(E2,r(O)),mem(M).
e(mcal(E1,M,E2),O) -> e(E1,l(O)),e(E2,r(O)),mem(M).
e(gcal(F,E),O) -> e(E,l(O)).
```

図 3　全ての出現を得る推論規則

4.6 写　像 $\mathcal{D}$

オブジェクトのメンバの型を定義する半関数 $\mathcal{D}$: オブジェクト型 × メンバ名 ⇀ 型 を，述語 `dee`(オブジェクト型, メンバ名, 型) で表現する．半関数なので，定義域の要素に対して，対応する値域の要素は存在しないか、存在するならば一つに定まる．このことは MGTP では次のように表現できる．

```
cf(F),mem(M),type(T) -> dee(F,M,T);-dee(F,M,T).
dee(C,M,T1),type(T2),ne(T1,T2) -> -dee(C,M,T2).
```

4.7 写　像 $\mathcal{P}$

関数の実行時の環境を定義する関数 $\mathcal{P}$: 関数名 → this のオブジェクト型×引数の型×局所変数の型×返り値の型は，値域の全ての要素を列挙して一意性を宣言することが困難なので，$\mathcal{D}$ と同じ方法で半関数として宣言する．よって，MGTP が返したモデルが本当に求めるものかどうかを，$\mathcal{P}$ が全関数になっているか否かで再チェックする必要がある．大域関数については this は空なので，特別な定数 `nulltype` を導入する．述語 `pee` の定義を図 4 に示す．

4.8 型推論規則

以上の準備の下で，BabyJ^T の型推論規則を MGTP 規則に書き直すことができる．$\Gamma \vdash e : t$ すなわち「環境 Γ の下で式 e (出現 o にあるとする) に型 t がつく」という表明を，述語 `ta` を用いて `ta(o,t,Γ(this),Γ(x),Γ(y))` と記述する．型推論規則を図 5 に示す．式そのものの代わりに式の出現を用いていること以外，図 2 の推論規則がそのまま MGTP 推論規則に対応している．

4.9 推論の実行

以上で導かれた全ての MGTP 推論規則を JavaMGTP で証明させることで，入力 BabyJ プログラムを BabyJ^T 型検査することができる．証明が成功したモデルで，`pee` が全関数であるモデルにあるリテラルが，入力プログラムへの型割当を表現している．モデル生成により，インクリメンタルな手続きを踏むことなく型割当の導出が行われる．

5. 考　　察

5.1 モデル生成法を用いた理由

BabyJ^T 型推論系の推論は，$\mathcal{P},\mathcal{D}$ の対応づけで関数本体式の型割当が矛盾なく行えるものを探す，という流れで進行する．$\mathcal{P},\mathcal{D}$ の対応づけが，`pee,dee` リテラルとしてモデル候補に含まれることから，モデル生成法による証明系と BabyJ^T 型推論

```
cf(F),type(B),type(C),type(D) -> pee(F,F,B,C,D);-pee(F,F,B,C,D).
gf(F),type(B),type(C),type(D) -> pee(F,nulltype,B,C,D);-pee(F,nulltype,B,C,D).
mf(F),cf(A),type(B),type(C),type(D) -> pee(F,A,B,C,D);-pee(F,A,B,C,D).
pee(F,A1,B1,C1,D1),cf(A2),type(B2),type(C2),type(D2),ne(A1,A2) -> -pee(F,A2,B2,C2,D2).
pee(F,A1,B1,C1,D1),cf(A2),type(B2),type(C2),type(D2),ne(B1,B2) -> -pee(F,A2,B2,C2,D2).
pee(F,A1,B1,C1,D1),cf(A2),type(B2),type(C2),type(D2),ne(C1,C2) -> -pee(F,A2,B2,C2,D2).
pee(F,A1,B1,C1,D1),cf(A2),type(B2),type(C2),type(D2),ne(D1,D2) -> -pee(F,A2,B2,C2,D2).
cf(F),pee(F,F,B,C,D) -> ta(F,F,B,C,D).
gf(F),pee(F,nulltype,B,C,D) -> ta(F,nulltype,B,C,D).
mf(F),pee(F,A,B,C,D) -> ta(F,A,B,C,D).
```

図 4　述語 pee の定義

```
/* Integer */
e(intval,O),cf(A),type(B),type(C) -> ta(O,integer,A,B,C).
/* Var */
e(this,O),cf(A),type(B),type(C) -> ta(O,A,A,B,C).
e(x,O),cf(A),type(B),type(C) -> ta(O,B,A,B,C).
e(y,O),cf(A),type(B),type(C) -> ta(O,C,A,B,C).
/* null */
e(null,O),cf(A),type(B),type(C),type(T) -> ta(O,T,A,B,C).
/* Seq */
e(seq(E1,E2),O),ta(l(O),T1,A,B,C),ta(r(O),T2,A,B,C) -> ta(O,T2,A,B,C).
/* New-cons */
e(new(F,E),O),pee(F,F,T,U,V),ta(l(O),T,A,B,C) -> ta(O,F,A,B,C).
/* Member-select */
e(sel(E,M),O),ta(l(O),T,A,B,C),dee(T,M,S) -> ta(O,S,A,B,C).
/* Var-ass */
e(xass(E),O),ta(l(O),T,A,T,C) -> ta(O,T,A,T,C).
e(yass(E),O),ta(l(O),T,A,B,T) -> ta(O,T,A,B,T).
/* Member-ass */
e(mass(E1,M,E2),O),ta(l(O),T1,A,B,C),ta(r(O),T2,A,B,C),dee(T1,M,T2) -> ta(O,T2,A,B,C).
/* Member-call */
e(mcal(E1,M,E2),O),ta(l(O),T1,A,B,C),ta(r(O),T2,A,B,C),dee(T1,M,ft(T1,T2,T)) -> ta(O,T,A,B,C).
/* Global-call */
e(gcal(F,E),O),ta(l(O),T,A,B,C),pee(nulltype,T,D,E) -> ta(O,E,A,B,C).
/* Member-func */
mf(F),e(F,O),pee(F,A,B,C,D),type(X),type(Y),type(Z) -> ta(O,ft(A,B,D),X,Y,Z).
```

図 5　MGTP で記述した型推論規則

は相性がよいと考えられる．この型推論を Prolog で記述するには，$\mathcal{P}, \mathcal{D}$ の対応づけを総当たりで探し，それが関数本体式の型割当を矛盾なく行うかどうか確認するホーン節で部分項の型を確認する、という手続き的な動作になる．ユニフィケーションによって $\mathcal{P}, \mathcal{D}$ の対応づけを得ようとすると，その対応づけと式の型付けとが相互に依存してしまい，推論は完了しない．

否定が扱えることも MGTP の大きな特徴であるが，正リテラルがモデル候補に存在しないだけでは前件の負リテラルは有効にならず，負リテラルがモデル候補に存在する必要がある．この動作が直観に反するところがあるといえる．

5.2 BabyJ^T 型推論系の整理

BabyJ では関数の引数や局所変数が必ず 1 つであるが，JavaScript ではその個数は任意である．このような言語に対する型推論規則を直接書き下すことは難しい．この問題について，任意個の並びを直接指定できる演算あるいはシンタックスシュガーを導入することも考えられる．しかし JavaScript では，関数の引数や局所変数は単一の Arguments オブジェクトの属性と定義されている．任意個の引数や局所変数を用いるプログラムを，プリプロセッサを用いて Arguments オブジェクトを利用する形に書き換えることができる．この変換を前提にすれば，BabyJ 唯一の局所変数 y は不要となる．

5.3 BabyJ^T 型推論系の拡張

BabyJ^T では，オブジェクト指向プログラムの型の多相性が全て排除されている．例えば，メンバ関数に対して，this の型を一意に定めるため，一つのメンバ関数を異なるコンストラクタから生成されるオブジェクト間で共用することさえできない．$\mathcal{P}$ を $\mathcal{P}$: (関数名) × (*this* の型) ⇀ (局所変数の型) × (返り値の型) と変形させることで，型変数を増やし，メンバ関数の型の多相性が表現できるようになる．

我々は [7] で，クラスベースのオブジェクト指向言語に対する型検査アルゴリズムを提案している．そこでは，クラスの集合をオブジェクトの型として，制約解消に基づく型推論を行う．BabyJ^T ではオブジェクト型はコンストラクタであるが，これをコンストラクタの集合に拡張することで，型体系を自然に拡張してオブジェクト指向プログラムの型の多相性を扱えると期待している．

5.4 定理証明系に必要な機能

4.4.2 節でも述べた通り、限量子が利用できれば記述力は大幅に向上する．関数記号にソートを定義できれば、限量子のドメインが決定できるだけでなく，推論規則を記述するときの型エラーを検出できる．

より複雑な問題に定理証明系を応用するためには，項の上の演算を再定義する機能，Standard ML のようにデータ型をユーザサイドで定義する機能，あるいはリストなどの標準的なデータ型のサポートの強化が必須であると考えられる．その記法，推論システムへの影響の検討は今後の課題である．

6. ま と め

BabyJ^T 型推論システムを JavaMGTP 定理証明系で判定可能な形式に変換し，形式的な型推論システムを直接実際のプログラムに適用する方法を示した．モデル生成法に基づく証明系を用いたことにより，問題の記述が自然な形で行えた．記述力を向上させるため，定理証明系に求められる機能について考察した．

JavaMGTP は Java プログラムに組み込むことができるので，今までの手続き的プログラミングから脱した推論を用いた CASE ツールが開発できる．Java あるいは他の手続き型言語と連携する機能を持った定理証明系は他に [8] などがある．今後，これらの利用も検討する必要がある．

文　献

[1] Shoji YUEN, Keishi KATO, Daiju KATO, Shinichiro YAMAMOTO, Seiji AGUSA, “A Testing Framework for Web Applications based on the MVC model with Behavioral Descriptions,” International Conference on Information Technology & Applications 2004, Harbin, China, Jan. 2004.

[2] 根路銘 崇, 小野 康一, 田井 秀樹, 安部 麻里, “Web アプリケーションモデルに基づく JSP の動的検証,” ソフトウェアテストシンポジウム 2004, pp.61–67, Jan. 2004.

[3] Filippo Ricca, Paolo Tonella, “Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions,” Proc. of TACAS’2001, Tools and Algorithms for the Construction and Analysis of Systems, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS’2001, LNCS 2031 pp.373–388, Genova, Italy, April 2001.

[4] 鷲尾 和則, 松下 誠, 井上 克郎, “JavaScript を含んだ HTML 文書に対するデータフロー解析を用いた構文検証手法の提案,” 電子情報通信学会技術研究報告, SS2002-22, Vol.102, No.370, pp.13–18, 2002.

[5] Christopher Anderson and Sophia Drossopoulou, “BabyJ: from object based to class based programming via types,” WOOD2003 Workshop on Object Oriented Developments, Warsaw, Poland, April 2003.

[6] 長谷川 隆三, 藤田 博, “Java によるモデル生成型定理証明系 MGTP の開発,” 情報処理学会論文誌, vol.41, No.6, pp.1791–1798, June 2000.

[7] 大久保 弘崇, 坂部 俊樹, 稲垣 康善, “オブジェクト指向プログラムに対する Message Not Understood フォールト検知のための型検査アルゴリズム,” コンピュータソフトウェア, Vol.17, No.3, pp.63–76, 2000.

[8] 番原 睦則, 姜 京順, 田村 直之, “線形論理型言語の Java 言語による処理系の設計と実装,” 情報処理学会論文誌, Vol.40, No.SIG 10 (PRO 5), pp.1–16, Dec. 1999.