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.

モデル生成に基づく JavaScript プログラム 型検査のためのフロントエンド

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 JavaScript に対する型推論規則を，現実のプログラムに定理証明系を用いて直接適用する手法について研究を進めている．本稿では，JavaScript プログラムを構文解析し，型推論実行系への入力形式を生成するフロントエンド部の設計について述べる．その設計の過程で，使用していた型推論規則では型の不整合を検出する条件が不足していることが明らかになった．この条件を追加することで，推論規則を満たしているが型割り当てとしては不充分な解を排除できるようになった．フロントエンドの自動化と連携させて型推論体系を拡張することにより，関数の引数及び局所変数の扱いを強化できた．

キーワード JavaScript, XML, 型検査，定理証明系，モデル生成法

Front-end for JavaScript Type Checker based on Model Generation Theorem Prover

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 how to apply type inference rules directly to practical programs based on a theorem prover. In this paper, we discuss constructing the front-end system which prepares the input forms to the type inference system for given JavaScript programs, and retrieves the infered output from the theorem prover. First we point out that the type inference rule set for this purpose needs additional type checking rules, which rejects incomplete models of type assignment. Also we extended the type inference rules to handle more than one formal arguments and local variables in first-order predicate logic.

Key words JavaScript, XML, type checking, model generation theorem prover

1. はじめに

Web アプリケーションにおけるクライアントサイドスクリプト言語として登場した JavaScript は，XUL [6] や Konfabulator [7] に見られるようにアプリケーション記述言語などへその用途を広げて普及している．より重要な場面で使用されるにつれ，信頼性の向上が要求されるようになる．JavaScript は型のないオブジェクト指向言語であるため，型検査を行うことにより一定の信頼性を保証することができる．JavaScript を対象とした型体系として BabyJ^T 型推論系が提案されている [2]．ここでは，推論規則の集合として型推論系が提示されている．型検査系を実装する際，型推論規則が定める型割り当てを求める手続き的プログラムを作成する方法が一般的であるが，我々は定理証明系を用いてこの型推論系を JavaScript プログラムに直接適用することを試みた [1]．このアプローチは，形式的手法を現実的な問題に直接適用する可能性を追求し，またその目的

に要求される定理証明系の機能や言語体系を明らかにするという視点から選択されたものである．

定理証明系にはモデル生成法に基づく JavaMGTP [3] を用いた．モデル生成法は，推論規則を充足するモデルが自動的に取得できる．型推論規則を充足するモデルとは型割り当てであるので，CASE ツールとしての応用を考える際に都合がよい．反駁方式の推論系で同様に充足解を取り出すためには，推論規則を基にした論理型プログラミングを行う必要があり，形式的な推論規則を直接適用するという目的に反する．

前回の報告 [1] では，BabyJ^T 型推論規則を JavaMGTP で動作させるため，推論手続き中の曖昧な部分を改良し，探索空間を規定する型のドメインの表現および，BabyJ^T 型推論規則をこの表現を用いて JavaMGTP 文法で記述する方法を与えた．これにより，BabyJ^T 型推論が JavaMGTP で実際に行えることを示した．

本稿では，JavaMGTP をバックエンドとして使用し，JavaScript プログラムの型検査を行うためのフロントエンドの設計について述べる．入力されたプログラムに応じて，そのプログラムに型割り当てを行う JavaMGTP 規則を生成することと，JavaMGTP が返すモデルから得られた型割り当てを取り出すことの 2 つが主な仕事である．我々の推論規則の基となっている BabyJ^T 型体系では，関数の引数と局所変数が 1 つしか許されないという制限があった．推論規則を，任意個の引数をもつプログラムに対応する形に汎用性を持たせて拡張すると，引数の数に応じて推論規則の形が変化するような高階の推論体系が必要になる．我々は，入力プログラムを指定して，そのプログラムに対する型推論規則を生成するとき，引数の個数は特定の場合しか存在しないことに着目した．推論規則に高階の記述で汎用性を持たせる代わりに，フロントエンド側で必要な形の型推論規則のみを生成する方法を採用する．これにより，JavaMGTP の扱える一階述語論理の範囲で，任意個の引数や局所変数を持つプログラムに対する型推論を行う方法が得られる．

フロントエンドの入出力のフォーマットには XML 技術で構成された JSX-model を用いる．これは，JavaScript プログラムに対する CASE ツール・プラットフォームによるマークアップであり，プログラムの細粒度リポジトリとなる．型推論により得られた型割り当てを式・文の要素の追加属性に書き出すことで，JSX-model を用いる CASE ツールで本システムの推論結果が応用できる．

2. BabyJ 言語とその型推論系

本稿で用いる型体系は，BabyJ^T [2] を基にしたものである．我々はこの型推論系を JavaMGTP で直接実行することを試みている [1] が，この際に体系の厳密化を行った．本節では文献 [1] で修正した BabyJ^T 型体系について概説する．

2.1 BabyJ

BabyJ 言語は JavaScript 言語のサブセットである．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．

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 ∈ *FuncID*			
m ∈ *MemberID*			

図 1　BabyJ の文法

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そのオブジェクトの型として用いる．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 の型体系には多相性はない．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．`noobject` は大域関数の実行環境の this に何も割り当てられないことを示す特別な型である．プログラムの全ての部分式に対して，推論規則により型が割り当てられる

(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)

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

(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)

$$\frac{\mathtt{f}\text{ はコンストラクタ} \quad \mathcal{P}(\mathtt{f}) = (\mathtt{f}, t_a, t_l, t_r) \quad \Gamma \vdash \mathtt{e} : t_a}{\Gamma \vdash \mathtt{new\ f(e)} : \mathtt{f}}$$

(Member-sel)

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

(Var-ass)

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

(Member-ass)

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

(Member-call)

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

(Global call)

$$\frac{\mathtt{f}\text{ は大域関数} \quad \Gamma \vdash \mathtt{e2} : t_a \quad \mathcal{P}(\mathtt{f}) = (\mathtt{noobject}, t_a, t_l, t_r)}{\Gamma \vdash \mathtt{f(e)} : t_r}$$

(Member-func)

$$\frac{\mathtt{f}\text{ はメンバ関数} \quad \mathcal{P}(\mathtt{f}) = (t_o, t_a, t_l, t_r)}{\Gamma \vdash \mathtt{f} : (t_o, t_a, t_r)}$$

図 2　BabyJ^T の型推論規則

ような $\mathcal{P}, \mathcal{D}$ の対応付けを，制約の解消によって決定する．型推論の結果は $\mathcal{P}, \mathcal{D}$ および部分式への型割り当てであり，$\mathcal{P}$ は関数の引数および局所変数への型割り当てを，$\mathcal{D}$ はコンストラクタが作るオブジェクトの各属性の型を与える．

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．JavaMGTP 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．

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．

非終端要素

`Program`	プログラム全体で XML のルート要素
`FunDec`	関数宣言
`Param`	仮引数宣言
`VarDec`	変数宣言
`Stmt`	文
`Expr`	式

終端要素

`ident`	識別子
`literal`	定数表記
`comment`	コメント
`kw`	予約語
`op`	演算子
`sp`	空白文字
`nl`	改行

文要素の sort 属性

`Block, Empty, Expr, If, For, While, DoWhile, Continue, Break, Return, With, Labelled, Switch, Throw, Try`

式要素の sort 属性

`FunCall, ArrayAccess, Allocation, Paren, VarRef, Literal, This, Empty, Argument, Void, TypeOf, Delete, StrictEqual, StrictNotEqual, Assign`，各種整数演算，演算を伴う代入，各種論理演算

図 3　JSX-model の要素

4. Sapid と JSX-Model

Sapid [4], [5] は，細粒度のソフトウェア・リポジトリに基づいた CASE ツール・プラットフォームである．さまざまな言語を処理対象にしているが，この中に JavaScript に対する XML リポジトリである JSX-model がある.JSX-model では，ECMAScript 3rd Edition の文法規則によるプログラムを 7 種類の終端要素と 6 種類の非終端要素よってモデル化する．JSX-model は元のプログラムテキストの内容をテキスト要素として完全に含み，マークアップにより情報を追加しているという点で細粒度である．図 3 にタグとその意味を示す．

解析器は Java で実装されている．JavaScript ソースプログラムを入力として，JSX-model によりタグ付けされた XML ファイルを出力する．

5. BabyJ^T の改良

5.1 型検査規則の導入

図 2 の型推論規則を用いることで，型の正しいプログラムに対してその型割り当てを推論することができる．一方，これらの規則は型の誤りがあるプログラムに対して充分な情報を与えない．ある式で型が正しく推論できないとき，その式の形に対応する推論規則が適用できず推論が進行せず，証明系は推論が成功したとしてモデルを返す．しかしこのモデルには求めるべき全ての型割り当てが含まれておらず，型割り当ての正しい解ではない．

その原因は，一部の型推論規則には『この条件を満たすとき

型割り当てが推論される、さもなくば型割り当ては破綻している』という暗黙の条件があり，図 2 の型推論規則にはそのうちの前者しか記述されていないことである．

後者の条件を明示し，前述の状況で推論が途中で成功とみなされて停止する代わりに，矛盾を引き起こして現在のモデル候補を棄却させなければならない．よって，型割り当てが破綻しないための後者の条件を型推論規則に明示的に追加する．これにより，型が付けられるプログラムに対して不十分な型割り当てしか持たない「別解」を証明系が返すことがなくなり，また型の誤りがあるプログラムの判定が正しく行えるようになる．

実際に追加する条件を以下に示す．*(New-cons)*, *(Global call)* 規則に関して，$\mathcal{P}(\mathtt{f})$ の与える関数の引数型と，実引数の型は同じでなければならない．*(Var-ass)* 規則に関して，代入される値と変数の型が等しくなければならない．*(Member-ass)* 規則に関して，代入される値と代入先の属性の型が等しくなければならない．*(Member-call)* 規則に関して，選択しているメンバにはメンバ関数が格納されており，さらにその this および引数の型が実レシーバ，実引数の型と等しくなければならない．

これらの条件を JavaMGTP 項で記述するには，条件が言及する変数を束縛せず自由にしておき，後件に成立するべき条件を追加する．

```
pre1(X),pre2(X) -> post(X).
```

は `pre1(x)` かつ `pre2(y)` のとき適用されないが，

```
pre1(X),pre2(Y) -> eq(X,Y), post(X).
```

は`-eq(x,y)` であるような `pre1(x)`, `pre2(y)` が成立しているときに矛盾を導く．

5.2 型ベクトルの導入

BabyJ では，型推論系の単純化のため、変数の扱いを単純化している．関数の引数と局所変数はそれぞれ 1 つしかなく，x,y という特定の名前である．JavaScript 型検査の実用化のためには，この制限を緩和する必要がある．

JavaScript 本来の動作意味では，関数の引数は Arguments オブジェクトの配列要素として渡される．また，局所変数はこのオブジェクトの属性として格納される．BabyJ 文法にこの動作をあてはめると，JavaScript の関数呼び出しは暗黙に，

（1） Arguments オブジェクトを生成

（2） 評価した実引数の値を配列要素に格納

（3） Arguments オブジェクトを引数 x に渡して関数呼び出し

（4） 局所変数は y の代わりに x の属性を使用

という計算を行っているとみなせる．JavaScript のための型体系であれば，この計算を直接型割り当てできるべきだが，BabyJ^T 型体系ではそれは難しい．関数の引数や局所変数の型は関数ごとに当然異なる．型は多相性がなく単一のオブジェクトコンストラクタであるため，関数ごとに異なる Arguments オブジェクトの型を表すには，コンストラクタを関数の数だけ用意し，正しく使い分ける必要がある．しかし，メンバ関数は動的束縛されるため，型割り当てを行う前に正しい Arguments コンストラクタを決定することはできない．そこで本稿では，関数の引数と局所変数の型の並びを扱う型ベクトルを BabyJ^T に導入してこの問題に対処する．一般的な表現を用いてこの方法を記述すると，任意サイズの型ベクトルに関する表現が必要になり高階の述語論理を用いることになるが，ここでは入力された JavaScript プログラムに対して型推論を行う規則を生成するため，型ベクトルはそのプログラムに出現するだけの有限種のものしか必要ない．

型ベクトルドメインを定義する述語には `tvec`，型ベクトルの構成子には `tv` を用いることにする．型ベクトルから任意の要素を取り出す述語 `nth` も必要である．

6. フロントエンドの構成

JavaScript プログラムを入力とし，5. 節で述べた改良を施した BabyJ^T 型を推論する JavaMGTP 規則集合を出力するフロントエンドの構成を述べる．また，型推論が成功した後は JSX-model XML ファイルへ得られた型割り当てを出力する．

基本的な解析は JSX-model 解析器にゆだねる．これにより，JavaScript プログラムは JSX-model DTD に基づきタグ付けされた XML ファイルとなる．以降，フロントエンドはこの XML を DOM を介して操作する．

a） 全関数名の取得

`FunDec` 要素により全て関数定義を取り出す．関数名に，仮引数リスト (`Param` 要素)，局所変数リスト (`VarDec` 要素)，を対応づけて記録する．

b） 関数の分類

BabyJ では，関数はコンストラクタ，大域関数，メンバ関数のいずれかであり，それ以外の呼び方をしてはならない．全ての関数呼び出し (`Expr` 要素で `sort="FunCall"`であるもの) を探し，関数を分類する．new 演算子とともに (`sort="Allocation"`要素に入る) 用いられていればコンストラクタである．そうでない関数は大域関数である．また，`sort="FunCall"`で呼び出されない関数はメンバ関数である．呼び出し方法が重複している場合はエラーとする．複数の呼び出し方法で使われている関数を複製し，呼び出し方法で区別して使い分けることでこの制限を回避することができる．フロントエンドの拡張により，この回避を内蔵することも可能である．

c） 型ドメインの要素の列挙

モデル生成法に基づく推論法には，値域限定性という制限がある．型推論においては，型というドメインの要素を数え上げることがこれに対応する．BabyJ^T 型体系における型は基本型，オブジェクト型，関数型からなるが，この中で関数型は再帰構造により無限の要素を持つ．JavaMGTP で推論を実行するため，関数型の項のサイズを定数で制限して近似の数え上げを行う．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と考えられる．初期値のサイズで制限した関数型を用いた型推論が失敗したとき，サイズを拡大して推論を再試行するようフロントエンドを構成する方法も考えられる．

実際の手続きでは，まず基本型 `integer` と `string` を宣言，

上で取得したコンストラクター覧を用いてオブジェクト型を宣言する．次にこれらの型を利用して関数型を設定したサイズ内で宣言する．

5.1 節で述べた条件判定を行うために，型ドメイン上の等価性が必要になる．JavaMGTP には項の上の等価演算子がないため，型の等価性も明示的に生成する．ここで，JavaMGTP では否定が使用できるが，あるリテラルが否定されるためには，そのリテラルの証明が失敗することではなく，明示的に否定したリテラルがモデルに追加される必要があることに注意する必要がある．言い換えると，推論規則から`-pred(a)` が導出されたときは否定が成立するが，推論規則から `pred(a)` が導出できないだけでは否定にならない．このため，`eq(T,T)` の形の公理を宣言するだけではなく，`-eq(S,T)` の否定形の公理も，全ての組み合わせに対して宣言しなくてはならない．5.1 節の型エラーの検出は，後者の形の公理と矛盾するリテラルの導出によって行われる．

d） 型ベクトル要素の列挙

5.2 節で述べたように BabyJ^T 型体系を拡張し，関数の引数の型および局所変数の型を管理する型ベクトルを導入する．上述のステップ a) で全ての関数の引数と局所変数の数が得られているので，これらから必要な要素数の型ベクトルの要素だけを列挙する．等価性についてもステップ c) と同様に宣言する．

e） 式の出現の列挙

図 2 の推論規則では式に対して型を割り当てているが，実際に型を割り当てる対象はプログラムテキスト中の部分式の出現である．関数 F 中の出現 O に式 expr が出現していることを `e(expr,F,O)` というリテラルで表現する．DOM 木をトラバースし，全ての部分式・文に対してこのリテラルを生成する．

引数・変数の参照および代入式については，変数名ではなくステップ a) で得られた変数リストを用いて，何番目の変数かという番号を用いて無名化する．この番号は，後に型ベクトルの対応する要素を取り出すインデックスとして用いられる．

f） 型推論の実行

ステップ c) で生成した型ドメインを定義する公理，ステップ d) で生成した型ベクトルドメインを定義する公理，図 2 の推論規則を JavaMGTP 文法で記述した型推論規則本体，そして前ステップ e) で生成したプログラムの内容を与える公理を連結して，JavaMGTP への入力とする．JavaMGTP に渡される推論規則の概観を図 4 に示す．

クラス MGTP のインタフェースでは以下のコンストラクタ及びメソッドを用いる．

```
public MGTP MGTP(String clauses, String[] options)
public void runMgtp()
public boolean sat()
public UList getModel()
```

MGTP のインスタンスを生成する際に推論規則を `String` で与える．メソッド `runMgtp()` で推論を実行させ，充足できたかどうかを `sat()` メソッドで調べる．充足できた場合は，`getModel()` メソッドにより `Term` クラスのオブジェクトのリストとしてモデルの全てのリテラルが得られる．

g） 結果の出力

JavaMGTP が返すモデルの中で，部分式の出現 O への型 T の割り当ては `ta(O,T)` というリテラルで表される．ステップ e) の逆を行い，それぞれの型割り当てが示す出現 O に対応する JSX-model 中の式 `Expr` と文 `Stmt` の XML 要素に対して，新たな属性値として割り当てられた型 T を追加する．また，コンストラクタ f が生成するオブジェクトの属性 m とその型 t は，`dee(f,m,t)` という形のリテラルで表される．関数の実行環境の型は写像 $\mathcal{P}$ に対応する述語 `pee` で表される．特に，`pee` の示す引数の型ベクトルと局所変数の型ベクトルについて，ステップ e) で行った変数の無名化の逆を行い，本来の変数宣言へ割り当てられた型を付記する．XML 形式を用いていることで，JSX-model を扱える他の CASE ツールに本方式で推論された型割り当てを受け渡すことができる．

7. ま と め

拡張 BabyJ^T 型推論を JavaMGTP 推論系で実行するために，JavaScript プログラムを入力として，必要な推論規則を生成するフロントエンドを構成した．フロントエンドはまた，推論の結果として得られる型割り当てを，他の CASE ツールで利用できるように XML を用いた細粒度リポジトリである JSX-model に格納する．型推論系に不十分なモデルを報告させないために，正しい型を推論する推論規則に加え，推論途中の型割り当てが矛盾していないことを確認する型検査規則を追加する必要があることを明らかにした．高階の述語論理の定理証明系を用いることなく，複数の引数を持つ関数を含むプログラムの型推論ができるようにフロントエンドで生成する推論規則の形を工夫した．

BabyJ^T 型推論規則は BabyJ 文法に依存しており，BabyJ 文法は JavaScript から多くの構文要素を削除している．より広範囲の JavaScript プログラムに対して型推論を実行するため，BabyJ 文法と BabyJ^T 型推論規則を拡充させていく必要がある．

はじめに述べた通り，JavaScipt は単体で用いられる言語ではなく HTML や XUL といった他の文書に埋め込んで利用される．このとき，外側の文書により定義されるデータオブジェクトが JavaScript から可視になっている．これら外部オブジェクトを操作する JavaScript プログラムの信頼性を型検査によって保証するためには，本稿で JavaScirpt プログラムから型推論の公理を生成したように，同時に外側の文書を読み込んで必要な公理を生成する機能がフロントエンドに必要である．

BabyJ^T 型体系では，オブジェクト指向プログラムが本来持つべき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．我々は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 [8]．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置き換えること

```
/* Var */
e(this,F,O),pee(F,S,X,Y,R) -> ta(O,S).
e(arg(N),F,O),pee(F,S,X,Y,R),nth(X,N,T) -> ta(O,T).
e(local(N),F,O),pee(F,S,X,Y,R),nth(Y,N,T) -> ta(O,T).
/* Seq */
e(seq,O),ta(l(O),T1),ta(r(O),T2) -> ta(O,T2).
/* New-cons */
e(new(CO),F,O),pee(CO,CO,X,Y,R),ta(arg1(O),T1),···,ta(argn(O),Tn) -> eqv(X,tv(T1,···,Tn)),ta(O,CO).
/* Member-select */
e(sel(M),F,O),ta(l(O),T),dee(T,M,S) -> ta(O,S).
/* Var-ass */
e(argass(N),F,O),ta(l(O),T),pee(F,S,X,Y,R),nth(X,N,U) -> eq(T,U),ta(O,T).
e(localass(N),F,O),ta(l(O),T),pee(F,S,X,Y,R),nth(Y,N,U) -> eq(T,U),ta(O,T).
/* Member-ass */
e(mass(M),F,O),ta(l(O),T1),ta(r(O),T2) -> dee(T1,M,T2),ta(O,T2).
/* Member-call */
e(mcal(M),F,O),ta(l(O),TS),dee(TS,M,TSM) -> ftype(TSM). /* typecheck */
e(mcal(M),F,O),ta(l(O),TS),dee(TS,M,ft(S,X,R)),ta(arg1(O),T1),···,ta(argn(O),Tn)
                                                    -> eq(TS,S),eqv(X,tv(T1,···,Tn)),ta(O,R).
/* Global-call */
e(gcal(G),F,O),pee(G,S,X,Y,R),ta(arg1(O),T1),···,ta(argn(O),Tn) -> eq(X,tv(T1,···,Tn)),ta(O,R).
/* Member-func */
e(F1,F,O),mf(F1),pee(F1,S,X,Y,R) -> ta(O,ft(S,X,R)).
```

図 4　MGTP で記述した型推論規則 (一部)

で, [8] の手法を応用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型体系を構築する予定である.

文　　献

[1] 大久保 弘崇, 山本 晋一郎, 坂部 俊樹, 稲垣 康善, “形式的手法に基づく JavaScript プログラムの型検査系の実現,” 信学技報 Vol.104, No.47, (ISSN 0913-5685), pp.13–18, April 2004.

[2] 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.

[3] 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.

[4] 福安 直樹, 山本 晋一郎, 阿草 清滋, “細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid,” 情報処理学会論文誌, Vol.39, No.6, pp.1990–1998, June 1998.

[5] Sapid : Sophisticated APIs for CASE tool Development, http://www.sapid.org/

[6] XUL : XML User Interface Language, http://www.mozilla.org/projects/xul/

[7] Konfabulator, http://www.konfabulator.com/

[8] 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.