修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工学　研究科　情報・通信工学　専攻　博士前期課程		
氏　　名	丹治　将貴	学籍番号	1531069
論 文 題 目	Ruby に対する Gradual typing の導入		

要　旨

プログラミング言語 Ruby は，広く使用されている動的型付け言語である．
Ruby は実行時に型チェックを行うため，プログラムを実行し型エラーを含む部分に実行が及ばなければ型エラーは報告されない．
このことは，プログラム中に型エラーによるバグが潜在的に残り，バグの発見が遅れたり，バグを見逃したりする要因となっている．
このような動的型付けの欠点を補う方法として，Gradual typing が提案されている．
Gradual typing とは，型注釈の有無により型付けの手法が異なるような型システムである．
型注釈がある部分については静的に型付けをし，実行前に型チェックを行う．
型注釈がない部分については動的型として型付けをし，実行時に型チェックを行う．
本研究の目的は，Ruby の動的型付けによる柔軟性を残しつつ，プログラムの型エラーによるバグの発見を容易にすることである．
その方針として，Ruby に Gradual typing を導入することで，静的型エラーの検出と，Ruby の柔軟性を両立させることを目指す．
本論文では, Gradual typing に基づく動的型と型注釈の構文を加えた Ruby のサブセットを考え，そのサブセットについて型付け規則と評価規則を与え，実装に向けての基本的な考え方を示す．
さらにその正当性を，進行定理と保存定理を証明することで示す．

平成 28 年度修士論文

Ruby に対する Gradual typing の導入

電気通信大学大学院情報理工学研究科
情報・通信工学専攻
コンピュータサイエンスコース

学籍番号　　: 1531069
氏名　　: 丹治将貴
主任指導教員 : 岩崎 英哉 教授
指導教員　　: 中山 泰一 准教授
提出日　　: 平成 29 年 1 月 30 日

要旨

プログラミング言語 Ruby は，広く使用されている動的型付け言語である．Ruby は実行時に型チェックを行うため，プログラムを実行し型エラーを含む部分に実行が及ばなければ型エラーは報告されない．このことは，プログラム中に型エラーによるバグが潜在的に残り，バグの発見が遅れたり，バグを見逃したりする要因となっている．このような動的型付けの欠点を補う方法として，Gradual typing が提案されている．Gradual typing とは，型注釈の有無により型付けの手法が異なるような型システムである．型注釈がある部分については静的に型付けをし，実行前に型チェックを行う．型注釈がない部分については動的型として型付けをし，実行時に型チェックを行う．本研究の目的は，Ruby の動的型付けによる柔軟性を残しつつ，プログラムの型エラーによるバグの発見を容易にすることである．その方針として，Ruby に Gradual typing を導入することで，静的型エラーの検出と，Ruby の柔軟性を両立させることを目指す．本論文では，Gradual typing に基づく動的型と型注釈の構文を加えた Ruby のサブセットを考え，そのサブセットについて型付け規則と評価規則を与え，実装に向けての基本的な考え方を示す．さらにその正当性を，進行定理と保存定理を証明することで示す．

目次

1 はじめに

1.1 背景

プログラミング言語 Ruby は，広く使用されている動的型付け言語であり，Ruby on rails で有名である．Ruby の特徴のひとつは，強力なメタプログラミングが可能という点である．その例として，既存クラスへのメソッド追加・変更などを行えるオープンクラスや，未定義メソッド呼び出しの際の処理を定義できる method_missing がある．Ruby は動的型付け言語であるため，プログラム中の型エラーは実行時に報告される．そのため，プログラムを実行しなければ型エラーは報告されない．このことは，プログラム中に型エラーによるバグが潜在的に残り，バグの発見が遅れたり，バグを見逃す要因となる．例えば次のような Ruby プログラムを考える．

```
1  def string_mul(str, n)    # str は文字列型，n は整数型を想定
2   str * n
3  end
4  string_mul(3.to_s, 2) * 2        # => "3333" (3.to_s => "3")
5  string_mul(3, 2) * 2             # => 12
6  string_mul(3, 2) + "2"           # => 型エラー
```

string_mul は第一引数に文字列 str，第二引数に整数 n を受け取り，str を n 回繰り返した文字列を返すメソッドと想定しているとする．4 行目では string_mul に正しい型の実引数を渡しており，型エラーとはならない．ここで，5 行目は 4 行目と同じ動作を期待しているが，第一引数の整数 3 に to_s を書き忘れ，第一引数に整数型の実引数を渡している．string_mul(3,2) は整数 6 を返すため，5 行目は整数と整数の乗算を行うことになり，返り値は 12 となり，これも型エラーとはならない．しかし，6 行目では整数と文字列の加算を行っており，これは Ruby においては型エラーとなる．6 行目のようなプログラムが実行されなければ，型エラーは報告されないため，このプログラムが 5 行目までだった場合は，バグが潜在的に残ってしまう．

この問題点を解決するひとつの方法として，Gradual typing [1] が提案されている．Gradual typing は部分的に静的型チェックと動的型チェックを行うので，静的型付けと動的型付けの両方の利点を得ることができる．

1.2 目的

本研究の目的は，Ruby に Gradual typing を導入することで，Ruby の動的型付けによる柔軟性を残しつつ，プログラムの型エラーによるバグの発見を容易にすることである．さらに Ruby のオープンクラスによるメソッド追加とメソッド更新に対応させることと，method_missing に

対応させることも目指す．Gradual typing を導入した Ruby を使用すれば，ユーザは静的に型エラーを検出したい部分に型注釈を入れ，Ruby の動的型付けによる柔軟な表現を残したい部分には型注釈を入れないことで，静的型エラーの検出と，Ruby の柔軟性を両立させることができる．

1.3 論文の構成

本論文の構成は次の通りである．第 2 章では Gradual typing について詳しく述べる．第 3 章では，Ruby に Gradual typing を導入した際のプログラムの振る舞いについて述べ，第 4 章では，本研究で扱う Ruby サブセットや，それに対する型付け規則，評価規則を述べる．第 5 章では Ruby サブセットが型健全性を満たすことを証明し，第 6 章で実装の一例を示し，第 7 章で関連研究について述べ，第 8 章で本論文をまとめる．

2 Gradual typing

Gradual typing とは，型注釈の有無により型付けの手法が異なるような型システムである．Gradual typing は静的型システムであるが，扱う型の一部に動的型を導入しており，この型に対応する部分は実行時に型チェックされ，それ以外の型は実行前に型チェックされる．したがって，動的型であるか否かにより型付けの手法が異なり，それは型注釈の有無による．

型注釈がある部分については静的に型付けを行い，実行前に型チェックを行う．例えば，以下のような Ruby プログラムがあったとする．

```
1    x ::: Fixnum = 3
2    "abc" + x
```

ここで，「:::」は型注釈である．1 行目では，変数 x は Fixnum クラス (Ruby における整数型の 1 つ) の型という型注釈を挿入している．そのため，x は Fixnum 型の変数として静的に型付けられる．2 行目では，文字列"abc" と変数 x の加算を行っているが，Ruby では文字列と整数の加算は型エラーとなる．x は 1 行目において Fixnum 型として静的に型付けされているため，2 行目のプログラムは静的型エラーとなる．

型注釈が無い部分は動的型として型つけられ，実行時に型チェックを行う．例えば，以下のような Ruby プログラムがあったとする．

```
1    y = 3
2    "abc" + y
```

1 行目では，変数 y に型注釈をつけずに値を代入している．このとき，変数 y は動的型として型付けされる．そのため，2 行目の変数 y は，静的型チェックが行われず，その型は実行時にチェックされる．したがって，2 行目は型注釈有りの場合の例と同様に型エラーとなるが，それは実行時に判明する実行時型エラーとなる．

以上のように，Gradual typing では，型注釈により部分的に静的型チェック・動的型チェックを行うことが可能となる．ユーザが型エラーの発見を容易にしたい部分に型注釈を入れることで，静的型チェックによる型エラーを発見しやすくできる．一方，ユーザが動的型付けによる利点を活かしたい部分には型注釈を入れないことで，動的型付けによる利点を残すことができる．また，プログラムすべてに完全に型注釈を入れた場合は，完全な静的型チェックが行われるため，すべての型エラーを実行前にチェックできる．

3　Ruby における Gradual typing の設計

本章では，Ruby における Gradual typing の動作を，プログラム例を用いて説明する．本研究で設計する Ruby サブセットでは，変数への代入式と，メソッド定義の部分に型注釈を挿入できる．この 2 つに型注釈を挿入できるようにすることで，Gradual typing を Ruby のオープンクラスに対応させる．また，型注釈が無いことを動的型？と表記する．ここでは，変数への代入式，メソッド定義の振る舞い，その他に特徴的な if 式とメソッド呼び出しの振る舞いについて説明する．

3.1　変数への代入式

図 3.1 にプログラム例を示す．

最初に変数への型注釈について説明する．本研究における Ruby サブセットでは，Ruby のクラス名を型として定義する．例として，Ruby には整数を表す Fixnum クラスや浮動小数を表す Float クラス，文字列を表す String クラスなどがあり，そのクラス名を用いて，整数は Fixnum 型，浮動小数は Float 型，文字列は String 型と定義する．本研究における Gradual typing の型注釈は，“:::” の後に型名を続けて表現する．例えば，変数への代入式は，

変数 ::: 型名 ＝ 式

と書く．型注釈をつけた場合，その変数は静的に型付けされる．図 3.1 の 1 行目から 3 行目がその例であり，x は String 型，y は Fixnum 型となる．z については，Float 型と型注釈しているのに対して，String 型の値を代入しているため，静的型エラーとなる．

```
1    x:::String = "abcde"      # xはString型
2    y:::Fixnum = 5            # yはFixnum型
3    z:::Float = "3.14"        # 静的型エラー
4    x:::Fixnum = 3            # xはFixnum型
5    y:::String = "fghij"      # yはString型
6    x = 100                   # xはFixnum型のまま
7    x = "foo"                 # 静的型エラー
8    w = 4                     # wは?型　?型 <= Fixnum型
9    x = w                     # 代入は成功
10   y = w                     # 動的型エラー
```

図 3.1　代入式における Gradual typing

すでに型がつけられている変数に対して，? 型以外の別の型注釈をつけて代入を行った場合，その変数の型を新しい型注釈の型に変更する．4 行目と 5 行目がその例であり，String 型だった変数 x が Fixnum 型，Fixnum 型だった変数 y が String 型となっている．

型注釈をつけずに代入式を記述すると，“::: ?” が省略されているものとみなす．すでに型がつけられている変数に型注釈を省略して代入すると，元の型をそのまま引き継ぐ，という意味になる．6 行目と 7 行目がその例となっており，変数 x は 4 行目で Fixnum 型として型付けされているため，Fixnum 型の値は代入できるが，String 型の値は代入できない．

初出の変数に型注釈を省略して代入すると，その変数の型は ? 型として型付けされる．8 行目において，変数 w は初出のため，? 型となる．ここで，? 型の変数に Fixnum 型の値 4 を代入しているため，右辺の値にキャストが挿入される．キャストとは，実行時に変換する型に関する情報を表すための構文であり，

$$\langle \text{キャスト先の型名} \Leftarrow \text{キャスト元の型名} \rangle \text{式}$$

と表現する．キャストは左辺の変数の型と右辺の式の型が異なる場合に式の前に挿入される．したがって，8 行目の式は w :::? = $\langle ? \Leftarrow \text{Fixnum} \rangle 4$ となり，このキャストは実行時に検査され，このキャストは成功する．続いて 9 行目では，Fixnum 型の変数 x に ? 型の変数 w を代入しているので，この場合においてもキャストが挿入され，x = $\langle \text{Fixnum} \Leftarrow ? \rangle$w となる．w の値は，$\langle ? \Leftarrow \text{Fixnum} \rangle 4$ だったため，9 行目の代入式は，x = $\langle \text{Fixnum} \Leftarrow ? \rangle \langle ? \Leftarrow \text{Fixnum} \rangle 4$ となる．これらのキャストは，最終的には Fixnum 型から Fixnum 型へのキャストとなり，キャストを行わないことと同等になるため，9 行目の代入式は動的型エラーを起こすことなく成功する．続いて 10 行目では，String 型の変数 y に w を代入しているので，9 行目と同様にキャストが挿入され，10 行目の代入式は y = $\langle \text{String} \Leftarrow ? \rangle \langle ? \Leftarrow \text{Fixnum} \rangle 4$ となる．これらのキャストは，最終的には Fixnum 型から String 型へのキャストとなり，矛盾するため，動的型エラーを起こす．

3.2 メソッド定義とメソッド呼び出し

図 3.2 にメソッド定義とメソッド呼び出しのプログラム例を示す．

1 行目から 6 行目で，既存のクラス Numeric に対して，オープンクラスによって新たなメソッド myadd を追加している．ここでは，myadd に対して，Fixnum 型を受け取り Numeric 型を返すという型注釈を挿入している．Ruby では，最後に評価された式の結果が返り値となるので，このメソッド定義では 4 行目の self+n が返り値となる．self は Numeric 型であり，引数 n が Fixnum 型と型注釈されているため，4 行目は Numeric 型の値と Fixnum 型の値和となり，これは Numeric 型の値となる．そのため，型注釈の返り値の型と一致している．したがって，このメソッド定義は静的型エラー・動的型エラーを起こすことなく成功する．

次にメソッド呼び出しについての振る舞いを，7 行目以降のプログラムで説明する．変数 x は Fixnum 型，y は ? 型，z は Numeric 型である．本来の Ruby では Numeric.new に引数を渡す

```
1   class Numeric
2    def myadd(n) ::: Fixnum -> Numeric
3                                        # メソッドへの型注釈
4     self + n
5    end
6   end
7   x:::Fixnum = 3
8   y = 5
9   z:::Numeric = Numeric.new(3.14)  # Numeric型
10  z.myadd(x)                       # Numeric型
11  z.myadd(y)                       # 引数にキャストが発生
12  z.myadd("abc")                   # 静的型エラー
13  w = Numeric.new                  # ?型
14  w.myadd(x)                       # レシーバにキャストが発生
```

図 3.2　メソッド定義，メソッド呼び出しにおける Gradual typing

ことはできないが，ここでは例を簡単にするために引数を渡せることができるものとする．z のメソッド myadd を 10 行目から 12 行目で呼び出している．10 行目のメソッド呼び出しは，定義時の型注釈の引数の型と実引数 x の型が Fixnum 型と一致しているので，Numeric 型の返り値が得られる．11 行目のメソッド呼び出しでは，実引数 y の型が？型のため，引数に，？型から型注釈の引数の型である Fixnum 型へのキャストが挿入される．その結果，11 行目の式は，z.myadd(⟨Fixnum ⇐?⟩y) となる．y は，⟨? ⇐ Fixnum⟩5 という値であるため，myadd の実引数は ⟨Fixnum ⇐?⟩⟨? ⇐ Fixnum⟩5 となり，このキャストは成功する．

12 行目のメソッド呼び出しは，String 型の実引数を与えており，型注釈と矛盾するため，静的型エラーとなる．

13 行目では，w にクラス Numeric の生成式を代入しているが，型注釈がないので w は？型となる．14 行目で w のメソッド myadd を呼び出しているが，レシーバである w の型が？型のため，w に対してキャストが挿入される．したがって，14 行目の式は，(⟨[myadd:Fixnum →?] ⇐?⟩w).myadd(x) となる．ここで，[myadd:Fixnum →?] は，「Fixnum 型の値を受け取り？型の値を返すメソッド myadd を持つクラス」の型を意味する．14 行目のレシーバ z の型が？型であるため，z の myadd の型が不明となることから，キャストにおける myadd の引数と返り値の型は？型となる．しかし，14 行目の引数 x の型が Fixnum と判明しているため，キャストにおける myadd の引数の型は Fixnum 型となる．したがって，実行時において 13 行目の w は Numeric クラスのインスタンスと判明するが，14 行目の返り値の型はキャストにおける myadd の型に従い，？型となる．

ここで，図 3.2 の後に図 3.3 のようなプログラムが存在したとする．

```
15      class Float
16       def myadd(n) ::: String -> String
17        ...
18       end
19      end
20      # 静的型エラー発生
21      class Float
22       def myadd(n) ::: Fixnum -> Fixnum
23        ...
24       end
25      end
26      # 静的型エラーなし
```

図 3.3　子孫クラスにおける先祖クラスメソッドの再定義

Numeric クラスの子孫クラスである Float クラスにおいて，同名のメソッド myadd を定義している．先祖クラスに存在するメソッドを再定義する場合は，再定義前のメソッドの型と，再定義する際の型注釈の型の関係により，静的型エラーになりうる．詳しい定義については 3 節で述べる．図 3.3 の 1 つ目のメソッドの再定義では，同名のメソッド myadd に対して，String → String と型注釈している．再定義前は引数の型が Fixnum 型，返り値の型が Numeric 型であるので，それらを String 型とすることはできず，静的型エラーとなる．2 つ目の再定義では，引数と返り値の型が Fixnum 型となっている．Fixnum クラスは，Numeric クラスの子孫クラスであるため，この場合は型の矛盾がないものとみなされ，静的型エラーとはならず，再定義は成功する．ただし，再定義するメソッドの型は先祖クラスの型に合わせるため，Float クラスの myadd の返り値の型は Numeric 型となる．そのため，返り値の部分に Fixnum から Numeric 型へのキャストが挿入される．

3.3　if 式

図 3.4 に if 式のプログラム例を示す．

最初に，if 式の結果の型について説明する．if 式の型は，then 節と else 節の内容に関わらず，必ず Nil 型とする．したがって，図 3.4 の 1 行目から 7 行目の if 式の型は Nil 型である．

次に，then 節と else 節のそれぞれの中で，変数への代入があった場合について説明する．この場合，then 節を通過した後に変数に与えられている型と else 節を通過した後に変数に与えられている型が等しくなければならない．図 3.4 の 1 行目から 7 行目の if 式では，then 節と else 節でそれぞれ x は Fixnum 型，y は?型が等しく与えられているため，静的型エラーは発生せず，if 式には型がつけられる．しかし，図 3.4 の 11 行目から 16 行目の if 式では，then 節で z は

```
1   if e then                    # eは適当な式
2    x:::Fixnum = 3              # xはFixnum型
3    y = "abc"                   # yは?型
4   else
5    x:::Fixnum = 5              # xはFixnum型
6    y = 3                       # yは?型
7   end
8   #if式の型はNil型
9   #xはFixnum型
10  #yは?型
11  if e then                    # eは適当な式
12   z:::String = "def"          # zはString型
13   w = 4                       # wは?型
14  else
15   z:::Fixnum = 5              # zはFixnum型
16  end
17  #z, wの型の不一致のため, 静的型エラー
```

図 3.4　if 式における Gradual typing

String 型，w は?型が与えられているが，else 節で z は Fixnum 型が与えられ，w については代入式が存在しない．z に与えられた型が，then 節と else 節で一致せず，また，w についても，then 節のみで型が与えられているため，型が一致しないとみなす．したがって，この if 式は成功せず，静的型エラーとなる．

4 設計

本節では，本研究で扱う Ruby のサブセットとそれに対する型付け規則と評価規則を示す．

4.1 Ruby のサブセット

本研究で扱う Ruby のサブセットを図 4.1 に示す．このサブセットは，Ren らの Ruby-Like 言語 [4] を参考にして定めた．

ξ は値であり，nil またはクラスのインスタンス $[A]$ である．v も値であり，ξ と同じであるが，キャスト挿入の構文追加の時に拡張する．x は変数の識別子である．self はレシーバを表す．A はクラスであり，A.new でそのクラスのインスタンスを生成する．$e; e$ は連続した式を表している．その他には，if 式とメソッド呼び出し $e.m(e)$ が式の内容に含まれている．ここで m はメソッド名である．

さらに，Gradual typing を導入するために，型注釈の構文を式として定義している．今回の研究では，代入式における左辺の変数と，メソッドの引数と返り値に型注釈を与える．$x ::: \tau = e$ が変数への型注釈であり，τ は変数が取り得る型である．また，def $A.m = \lambda x.e ::: \tau_s$ がメソッドへの型注釈である．ここで，τ_s の右矢印の左が引数の型，右が返り値の型である．型 τ の定義には，Nil 型とクラス型 A と，Gradual typing で扱う動的型を表す ? が含まれる．なお，代入式とメソッド定義を型注釈無しで書いた場合は，?型あるいは ? $\rightarrow$? 型と型注釈した場合の構文糖衣となる．c_m はメソッドであり，クラス A のメソッド m という意味となる．$vars$ は self と変数の集合となる．

expressions e	::=	$v \mid x \mid$ self $\mid$
		$e; e \mid A$.new $\mid$
		if e then e else $e \mid$
		$e.m(e) \mid x ::: \tau = e \mid$
		def $A.m = \lambda x.e ::: \tau_s$
ground values ξ	::=	nil $\mid [A]$
values v	::=	ξ
value types τ	::=	Nil $\mid A \mid$?
method types τ_s	::=	$\tau \rightarrow \tau$
methods c_m	::=	$A.m$
variables $vars$	::=	$x \mid$ self

method table MT	::=	$(c_m \mapsto \tau_s)*$
type env Γ	::=	$(vars \mapsto \tau)*$
TT	::=	$(c_m \mapsto \tau_s)*$
dynamic table DT	::=	$(c_m \mapsto \lambda x.e ::: \tau_s)*$
dynamic env E	::=	$(vars \mapsto v)*$
CT	::=	$(c_m \mapsto \lambda x.e ::: \tau_s)*$

extend	to	Cast Insertion
expressions e	+=	$\langle \tau_d \Leftarrow \tau \rangle e$
values v	+=	$\langle \tau_d \Leftarrow \tau \rangle \xi$
τ_d	::=	$\tau \mid [m ::: \tau_s]$

図 4.1　本研究で扱う Ruby サブセット

メソッドテーブル MT と型環境 Γ はそれぞれメソッドと型，変数と型の対応関係を表すものであり，型付け規則で用いる．今回のサブセットにおいては，クラスの親子関係に基づくメソッドの継承を扱っているが，MT には継承によるメソッドの型を含まない．したがって，$A.m \in dom(MT)$ である場合は，メソッド m はクラス A で必ず定義されている．また TT は，継承を含めたメソッドと型の対応を表す．TT は MT を参照する．$A.m \notin dom(MT)$ かつ $A.m \in dom(TT)$ であれば，クラス A におけるメソッド m は，先祖クラスから継承していることを表す．

DT，E は動的テーブル，環境であり，それぞれメソッドと定義本体 (型も含む)，変数と値の対応関係を表す．DT は MT と同様に，継承によるメソッド定義は含まないものとする．CT は，継承を含むメソッドとの対応関係を表す．CT は DT を参照する．これらは評価規則で用いる．

さらに，Gradual typing では，? 型を明らかにするためのキャストを挿入する必要があるため，キャストのための構文を追加している．キャストは $\langle \tau_d \Leftarrow \tau \rangle$ で表され，τ がキャスト前の型，τ_d がキャスト先の型である．また，ξ にキャストを挿入したものが値 v に追加されている．ξ は，キャストが含まれない基本型を表すので，基本型にキャストがひとつだけ付けられたものは値であることを意味する．キャスト先の型には，τ に加えて $[m ::: \tau_s]$ が含まれている．これは，メソッド m を持つクラスを表現する型を意味しており，レシーバが ? の時のメソッド呼び出しで用いる．なお，$TT([m ::: \tau_s].m) = \tau_s$ と定義する．

また，値やキャストに関連して，$Value$ と $badcast$ と $BadCast$ と $typeof$ を図 4.2 に定義する．

$Value$ は，値である式を意味している．

$badcast$ は，両辺の型の間に，Gradual typing で用いられる consistent-subtyping 関係 $\lesssim_{TT}$ を満たしていないキャストを値に適用した式を意味している．$\lesssim_{TT}$ については，3.2 節で詳しく説明する．

$BadCast$ は，部分式に $badcast$ を含む式を意味する．

$$
\begin{array}{rcl}
Value\ e & \equiv & \exists v, e = v \\
badcast\ e & \equiv & \exists v\ \tau_0\ \tau_1\ \tau_1'\ \tau_d \\
 & & e = \langle \tau_d \Leftarrow \tau_1' \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v\ \wedge \\
 & & \tau_0 \not\lesssim_{TT} \tau_d \\
BadCast\ e & \equiv & \exists e' : e \text{ の部分式}, badcast\ e'
\end{array}
$$

図 4.2　$Value$，$badcast$，$BadCast$ の定義

4.2 consistent-subtyping 関係

consistent-subtyping 関係 $\lesssim_{TT}$ には，Gradual typing で新たに定義されている consistent 関係 $\sim_{TT}$ と，subtyping 関係 $<:_{TT}$ が関わっている．右下に添字 TT があるのは，これらの関係は，TT により変化するためである．$\sim_{TT}$ 関係とは，両方の型において，?型でない既知の部分において等価である関係のことである．クラス間の $\sim_{TT}$ 関係をチェックするときは，クラス A に属するメソッドの引数における $\sim_{TT}$ 関係と返り値における $\sim_{TT}$ 関係をチェックする．図 4.3 に $\sim_{TT}$ 関係を示す．ここで M_{TT} は，TT に含まれるクラス A で利用できるメソッドの集合であり，

$$M_{TT}(A) = \{m \mid A.m \in dom(TT)\}$$

のように定義できる．

次に，$<:_{TT}$ 関係を図 4.4 に示す．同じ型どうしにおいては $<:_{TT}$ 関係にある．また，クラス間の $<:_{TT}$ 関係 $A <:_{TT} A'$ は，クラス A' に属すメソッドすべてがクラスに属しており，それぞれのメソッドの型が $<:_{TT}$ 関係を満たす場合に， $A <:_{TT} A'$ であると定義している．

$\lesssim_{TT}$ 関係は，$\sim_{TT}$ 関係と $<:_{TT}$ 関係により，推移できる型間の関係である．$\lesssim_{TT}$ の定義を

$$\frac{}{\tau \sim_{TT} \tau} \quad \frac{}{\mathrm{Nil} \sim_{TT} ?} \quad \frac{}{? \sim_{TT} \mathrm{Nil}}$$

$$\frac{}{? \sim_{TT} A} \quad \frac{}{A \sim_{TT} ?}$$

$$\frac{\tau_1 \sim_{TT} \tau_1' \quad \tau_2 \sim_{TT} \tau_2'}{\tau_1 \to \tau_2 \sim_{TT} \tau_1' \to \tau_2'}$$

$$\frac{M_{TT}(A) = M_{TT}(A') \quad \forall m \in M_{TT}(A), TT(A.m) \sim_{TT} TT(A'.m)}{A \sim_{TT} A'}$$

図 4.3　consistent $\sim_{TT}$ 関係

$$\frac{}{\tau <:_{TT} \tau}$$

$$\frac{\tau_1' <:_{TT} \tau_1 \quad \tau_2 <:_{TT} \tau_2'}{\tau_1 \to \tau_2 <:_{TT} \tau_1' \to \tau_2'}$$

$$\frac{\forall m \in M_{TT}(A), m \in M_{TT}(A') \wedge TT(A.m) <:_{TT} TT(A'.m)}{A <:_{TT} A'}$$

図 4.4　subtyping $<:_{TT}$ 関係

$$\frac{\exists \tau', \tau_1 \sim_{TT} \tau' \wedge \tau' <:_{TT} \tau_2}{\tau_1 \lesssim_{TT} \tau_2}$$

図 4.5　consistent-subtyping $\lesssim_{TT}$ 関係

図 4.5 に示す.

4.3　型付け規則

3.1 節で定義した Ruby のサブセットにおける型付け規則を図 4.6，4.7 に示す.

この型付け規則では，式から型へのジャッジメントとしており，その際にメソッドテーブルと

$\boxed{(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e')}$

$$\frac{}{(MT, \Gamma, [A]) \rightsquigarrow (MT, \Gamma, A, [A])}\ (\text{G\_Object}) \qquad \frac{}{(MT, \Gamma, \text{nil}) \rightsquigarrow (MT, \Gamma, \text{Nil}, \text{nil})}\ (\text{G\_Nil})$$

$$\frac{\Gamma(\text{self}) = \tau}{(MT, \Gamma, \text{self}) \rightsquigarrow (MT, \Gamma, \tau, \text{self})}\ (\text{G\_Self}) \qquad \frac{\Gamma(x) = \tau}{(MT, \Gamma, x) \rightsquigarrow (MT, \Gamma, \tau, x)}\ (\text{G\_Variable})$$

$$\frac{}{(MT, \Gamma, A.\text{new}) \rightsquigarrow (MT, \Gamma, A, A.\text{new})}\ (\text{G\_New}) \qquad \frac{(MT, \Gamma, e_1) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1') \quad (MT_1, \Gamma_1, e_2) \rightsquigarrow (MT_2, \Gamma_2, \tau_2, e_2')}{(MT, \Gamma, e_1; e_2) \rightsquigarrow (MT_2, \Gamma_2, \tau_2, e_1'; e_2')}\ (\text{G\_Sequence})$$

$$\frac{(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, \tau_0, e_0') \quad (MT_0, \Gamma_0, e_1) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1') \quad (MT_0, \Gamma_0, e_2) \rightsquigarrow (MT_2, \Gamma_2, \tau_2, e_2') \quad MT' = MT_1 = MT_2 \quad \Gamma' = \Gamma_1 = \Gamma_2}{(MT, \Gamma, \text{if } e_0 \text{ then } e_1 \text{ else } e_2) \rightsquigarrow (MT', \Gamma', \text{Nil}, \text{if } e_0' \text{ then } e_1' \text{ else } e_2')}\ (\text{G\_If})$$

$$\frac{(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau', e') \quad \Gamma'(x) = \tau \quad \tau' \lesssim_{TT} \tau}{(MT, \Gamma, x :::? = e) \rightsquigarrow (MT', \Gamma', \tau, x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e')}\ (\text{G\_Assign1})$$

$$\frac{(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau', e') \quad \tau' \lesssim_{TT} \tau \quad x \notin dom(\Gamma') \vee \tau \neq ? \quad \Gamma'' = \Gamma'[x \mapsto \tau]}{(MT, \Gamma, x ::: \tau = e) \rightsquigarrow (MT', \Gamma'', \tau, x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e')}\ (\text{G\_Assign2})$$

図 4.6　本研究における型付け規則 1

$\boxed{(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e')}$

$$\frac{(MT, \Gamma[\text{self} \mapsto A, x \mapsto \tau_1], e) \rightsquigarrow (MT', \Gamma', \tau', e') \quad \tau' \lesssim_{TT} \tau_2 \quad TT(A.m) = \tau_1 \to \tau_2}{(MT, \Gamma, \text{def } A.m = \lambda x.e :::? \to ?) \rightsquigarrow (MT, \Gamma, \text{Nil}, \text{def } A.m = \lambda x.\langle\!\langle \tau_2 \Leftarrow \tau' \rangle\!\rangle e' ::: \tau_1 \to \tau_2)} \text{(G\_Def\_Mth1)}$$

$$\frac{\begin{array}{c}(MT, \Gamma[\text{self} \mapsto A, x \mapsto \tau_1], e) \rightsquigarrow (MT', \Gamma', \tau', e') \quad \tau' \lesssim_{TT} \tau_2 \quad TT(A.m) = \tau_1 \to \tau_2 \\ \exists C \in Dec(A), C.m \in dom(MT)\end{array}}{(MT, \Gamma, \text{def } A.m = \lambda x.e ::: \tau_1 \to \tau_2) \rightsquigarrow (MT, \Gamma, \text{Nil}, \text{def } A.m = \lambda x.\langle\!\langle \tau_2 \Leftarrow \tau' \rangle\!\rangle e' ::: \tau_1 \to \tau_2)} \text{(G\_Def\_Mth2)}$$

$$\frac{\begin{array}{c}(MT', \Gamma[\text{self} \mapsto A, x \mapsto \tau_1], e) \rightsquigarrow (MT'', \Gamma', \tau', e') \quad \tau' \lesssim_{TT} \tau_2 \\ A.m \notin dom(TT) \quad MT' = MT[A.m \mapsto \tau_1 \to \tau_2] \\ \forall C \in Dec(A), C.m \notin dom(MT)\end{array}}{\begin{array}{c}(MT, \Gamma, \text{def } A.m = \lambda x.e ::: \tau_1 \to \tau_2) \\ \rightsquigarrow (MT', \Gamma, \text{Nil}, \text{def } A.m = \lambda x.\langle\!\langle \tau_2 \Leftarrow \tau' \rangle\!\rangle e' ::: \tau_1 \to \tau_2)\end{array}} \text{(G\_Def\_Mth3)}$$

$$\frac{\begin{array}{c}(MT', \Gamma[\text{self} \mapsto A, x \mapsto \tau_1'], e) \rightsquigarrow (MT'', \Gamma', \tau', e') \quad \tau' \lesssim_{TT} \tau_2 \\ TT(A.m) = \tau_1' \to \tau_2' \quad MT' = MT[A.m \mapsto \tau_1' \to \tau_2'] \\ \forall C \in Dec(A), C.m \notin dom(MT) \\ (\tau_1 \to \tau_2) \lesssim_{TT} (\tau_1' \to \tau_2') \quad e'' = e'[x \mapsto \langle \tau_1 \Leftarrow \tau_1' \rangle x]\end{array}}{\begin{array}{c}(MT, \Gamma, \text{def } A.m = \lambda x.e ::: \tau_1 \to \tau_2) \\ \rightsquigarrow (MT', \Gamma, \text{Nil}, \text{def } A.m = \lambda x.\langle\!\langle \tau_2' \Leftarrow \tau' \rangle\!\rangle e'' ::: \tau_1' \to \tau_2')\end{array}} \text{(G\_Def\_Mth4)}$$

$$\frac{\begin{array}{c}(MT, \Gamma, e_0) \rightsquigarrow (MT', \Gamma', \tau_d, e_0') \\ (MT', \Gamma', e_1) \rightsquigarrow (MT'', \Gamma'', \tau, e_1') \\ TT(\tau_d.m) = \tau_1 \to \tau_2 \quad \tau \lesssim_{TT} \tau_1 \\ \tau_d = A \vee \tau_d = [m ::: \tau_1 \to \tau_2]\end{array}}{\begin{array}{c}(MT, \Gamma, e_0.m(e_1)) \\ \rightsquigarrow (MT'', \Gamma'', \tau_2, e_0'.m(\langle\!\langle (\tau \leftharpoonup \tau_1) \Leftarrow \tau \rangle\!\rangle e_1'))\end{array}} \text{(G\_Ivk1)}$$

$$\frac{\begin{array}{c}(MT, \Gamma, e_0) \rightsquigarrow (MT', \Gamma', ?, e_0') \\ (MT', \Gamma', e_1) \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')\end{array}}{\begin{array}{c}(MT, \Gamma, e_0.m(e_1)) \\ \rightsquigarrow (MT'', \Gamma'', ?, (\langle [m ::: \tau_1 \to ?] \Leftarrow ? \rangle e_0').m(e_1'))\end{array}} \text{(G\_Ivk2)}$$

図 4.7　本研究における型付け規則 2

型環境も変化し得るようになっている．また，右辺の e' は，型付け後のキャストを挿入した式を表す．キャスト挿入に関わる部分以外では，e' は e と同等である．ここで，キャストに関する関数 $\langle\!\langle \tau_d \Leftarrow \tau \rangle\!\rangle e$ を以下のように定義している．

$$\langle\!\langle \tau_d \Leftarrow \tau \rangle\!\rangle e \equiv \begin{cases} e & (\tau = \tau_d \text{の場合}) \\ \langle \tau_d \Leftarrow \tau \rangle e & (\tau \neq \tau_d \text{の場合}) \end{cases}$$

これは，キャスト元の型とキャスト先の型が一致していれば，キャストを挿入しないことを意味する．

G_Object は，あるクラスのインスタンスは，そのクラス型であることを定義しており，G_New も，クラス生成式はそのクラス型であると定義している．nil についても同様であり，Nil 型に型付けされる (G_Nil)．self と変数は，型環境においてそれらに対応する型を与える (G_Self および G_Variable)．G_Sequence は連続した式 $e_1; e_2$ に対する型付けを定義している．e_1，e_2 を順番に型付けするたびにメソッドテーブルと型環境も変化し得る．最終的には，e_1 を型付けした後のメソッドテーブル MT_1 と型環境 Γ_1 において式 e_2 を型付けした結果の型が付けられる．

G_If は，if 式への型付けを定義している．型付けの順番は，最初に条件式 e_0 を型付けし，その後に e_1 と e_2 を型付けする．if 式の型は，e_1 と e_2 の型に関わらず，必ず Nil 型となる．型環境 Γ については，e_1 を型付けした型環境 Γ_1 と，e_2 を型付けした型環境 Γ_2 が一致していなければならない．MT についても，e_1 を型付けした環境 MT_1 と，e_2 を型付けした環境 MT_2 が一致していなければならない．したがって，if 式内で変数への代入とメソッド定義をする場合は，e_1 と e_2 の両方で同じ型を持つように定義しなければならない．

G_Assign1，G_Assign2 では，変数への代入式に対する型付けを定義している．基本的に，代入式自体の型は，代入後に与えられる変数の型と等しい．代入式の型付けは，右辺式の型となっており，右辺式の型付けにより，メソッドテーブルと型環境が変化し得る．さらに，変数の型が更新された場合は，型付け後の型環境 Γ' には，左辺の変数と右辺式の型の対応関係が加えられる．また，代入式においては，どの場合においても型付け後の代入式の右辺にキャストが挿入される可能性がある．

G_Assign1 は，定義済みの変数に対する代入式で，その変数に ？型と型注釈した場合 (すなわち型注釈がない場合) の型付けを定義している．この場合は，変数の型は代入前の型のまま更新されない．この時，右辺式に対しては，右辺式の型 τ' から変数の元の型 τ へのキャストが，必要ならば挿入される．

G_Assign2 は，未定義変数への代入式に対する型付けを定義している．代入式の右辺式の型と，左辺において注釈した型が $\lesssim_{TT}$ 関係を満たしていれば，変数の型は型注釈した通りの型になる．また，ここでは，変数が未定義または型注釈の型 τ が ？型でないことが仮定されている．変数が未定義の場合は，τ が ？型であってもこの規則が適用される．また，変数が定義済みの場合は，τ が ？型でないことが条件となる．キャスト挿入については，G_Assign1 と同様であ

り，右辺式の型 τ' から，型注釈の型 τ に (必要ならば) キャストする．

G_Def_Mth1，G_Def_Mth2，G_Def_Mth3，G_Def_Mth4 は，メソッド定義における型付けを定義している．定義式の結果自体は Nil 型としている．ここで使用している $Dec(A)$ は，クラス A の子孫クラスの集合を意味する．G_Def_Mth1 は，自身で定義しているか継承しているかに関わらず定義済みのメソッド m を，型注釈なしで再定義した場合である．この場合は，再定義前の型と同じ型とするため，MT は変化しない．また，e の型 τ' から，返り値の型 τ_2 へのキャストを e' へ挿入する．G_Def_Mth2 は，自身で定義しているか継承している，かつ子孫クラスに定義済みのメソッド m を既存の定義と同じ型注釈で再定義した場合である．この場合も，再定義前の型と同じ型とするため，MT は変化しない．G_Def_Mth3 は，未定義のメソッド m を定義した場合である．ただし，クラス A の先祖クラスと子孫クラスに m が定義されていないことを仮定している．この場合，型注釈通りの型を持つクラス A 上のメソッド m を，MT に追加する．キャストについては，G_Def_Mth1 と同様で，e の型 τ' から新しいメソッド定義の型注釈における返り値の型 τ_2 へのキャストを挿入する．G_Def_Mth4 は，(継承を含み) 定義済みであるメソッド $A.m$ を型注釈を挿入して再定義した場合であり，G_Def_Mth3 と同様に，子孫クラスで m が定義されていないことも前提としている．さらに，型注釈の型 $\tau_1 \to \tau_2$ と，再定義前の型 $\tau'_1 \to \tau'_2$ の間に $\lesssim_{TT}$ 関係があることも前提としている．この仮定は，継承において矛盾を生じさせないための制約である．この型付けでは，再定義するメソッドの型は，先祖クラスでのメソッドの型 $\tau'_1 \to \tau'_2$ として，MT に型情報を追加する．キャストについては，先祖クラスのメソッドの型へのキャストとなる．したがって，メソッド本体の式 e' の中の引数 x に τ'_1 から τ_1 へのキャスト，e' に τ_2 から τ'_2 へのキャストが挿入される．

G_Ivk1，G_Ivk2 はメソッド呼び出しにおける型付けを定義しており，レシーバ e_0，引数 e_1 の順に型付けする．

G_Ivk1 は，レシーバが ? 型ではないクラス型であるメソッド呼び出しの場合である．レシーバの型は，A またはメソッド m を持っているクラスの型 $[m ::: \tau_1 \to \tau_2]$ となる．この型付けでは，実引数の型が呼び出すメソッドの仮引数の型と $\lesssim_{TT}$ 関係にあれば，メソッド呼び出し式はメソッドの返り値の型に型付けられる．その際に，実引数に，実引数の型 τ から，$\tau \leftharpoonup \tau_1$ へのキャストを挿入する．ここで用いられている $\tau \leftharpoonup \tau_1$ は，τ と $\sim_{TT}$ 関係にあり，かつ，τ_1 の部分型である型 (τ' とする) を，すなわち $\tau \sim_{TT} \tau' \wedge \tau' <:_{TT} \tau_2$ となるような τ' を表している．

G_Ivk2 は，レシーバが ? 型の場合である．この型付けでは，引数の型に関わらず必ず ? 型とする．キャストについては，レシーバが ? であり，メソッド m を呼び出しているため，? からメソッド m を持っているクラスの型 $[m ::: \tau_1 \to ?]$ へのキャストをレシーバに挿入している．m の型は，τ_1 型となった式 e_1 を受けとり，返り値の型が ? となっているため，$\tau_1 \to ?$ としている．

図 4.6，4.7 の型付け規則を式 e に対して適用すると，必要に応じてキャストが挿入された式 e' が得られる．4 節での型健全性の証明においては，e' に対する型付けが必要となる．e' と e の違いはキャストの有無だけなので，キャスト以外の部分については，図 4.6，4.7 における型付

$$\frac{(MT,\Gamma,e_0) \rightsquigarrow (MT',\Gamma',\tau,e_0) \quad e = \langle \tau_d \Leftarrow \tau \rangle e_0}{(MT,\Gamma,e) \rightsquigarrow (MT',\Gamma',\tau_d,e)} \qquad \text{(G\_Cast)}$$

図 4.8　キャスト挿入された式への型付け規則

け規則を e' にそのまま適用することができる．さらに，キャストのある式については，図 4.8 に示す型付け規則 G_Cast を新たに導入する．ここで，G_Cast は図 4.6，4.7 に合わせるため，$\rightsquigarrow$ の右側は 4 つ組を書いているが，実際には 4 つ目の要素は使わない．G_Cast は，キャストが挿入された式の型は，キャスト先の型としているだけである．

4.4　評価規則

3.1 節で定義した Ruby のサブセットに対して，3.3 節で述べた型付け規則を適用した結果の式の評価規則を，図**??**に示す．

この評価規則では，式の 1 ステップ評価をジャッジメントとしており，その際に，DT と E が変化し得る．

E_Self は self の評価であり，self は，それに対応する値に評価される．E_Variable は変数についての評価で，self と同様である．

E_New は，A.new の評価である．インスタンスの生成のため，これは $[A]$ に評価される．

E_Assign1，E_Assign2 は，変数への代入式 $x ::: \tau = e$ を評価している．E_Assign1 は，右辺式が値ではなく，1 ステップ評価できる場合である．E_Assign2 は，右辺式が値になった場合であり，代入式の値を右辺式の値とする．この時，環境 E に左辺の変数と右辺の値の対応関係を追加する．

E_Sequence1，E_Sequence2 は，連続した式 $e_1; e_2$ についての評価であり，1 つ目の式 e_1 について場合分けしている．E_Sequence1 は e_1 を 1 ステップ評価できる場合の評価である．E_Sequence2 は e_1 が値になった場合であり，引き続き e_2 を評価する．

E_If，E_If_True，E_If_False は，if 式についての評価であり，条件式 e_0 について場合分けしている．ここで SV は，values から (あれば) キャストを除去するものであり，次のように定義される．

$$\begin{cases} SV(\xi) = \xi \\ SV(\langle \tau_d \Leftarrow \tau \rangle \xi) = SV(\xi) \end{cases} \tag{4.1}$$

E_If は，e_0 を 1 ステップ評価できる場合の評価である．E_If_True は，e_0 が型注釈の有無に関わらず値 $[A]$ になった場合である．この場合は，条件成立という扱いとなり，e_1; nil を評価する．E_If_False は， e_0 が型注釈の有無に関わらず値 nil となった場合である．この場合は，条件が不成立の扱いとなり，e_2; nil を評価する．G_If において，if 式の型は必ず nil 型となると

$\boxed{(DT, E, e) \to (DT', E', e')}$

$$\frac{E(\text{self}) = v}{(DT, E, \text{self}) \to (DT, E, v)}\ (\text{E\_Self}) \qquad \frac{E(x) = v}{(DT, E, x) \to (DT, E, v)}\ (\text{E\_Variable})$$

$$\frac{(DT, E, e) \to (DT', E', e')}{(DT, E, x ::: \tau = e) \to (DT', E', x ::: \tau = e')}\ (\text{E\_Assign1}) \qquad \frac{E' = E[x \mapsto v]}{(DT, E, x ::: \tau = v) \to (DT, E', v)}\ (\text{E\_Assign2})$$

$$\frac{}{(DT, E, A.\text{new}) \to (DT, E, [A])}\ (\text{E\_New}) \qquad \frac{(DT, E, e_1) \to (DT', E', e')}{(DT, E, (e_1; e_2)) \to (DT', E', (e'; e_2))}\ (\text{E\_Sequence1})$$

$$\frac{}{(DT, E, (v; e)) \to (DT, E, e)}\ (\text{E\_Sequence2}) \qquad \frac{(DT, E, e_0) \to (DT', E', e')}{\begin{array}{c}(DT, E, \text{if } e_0 \text{ then } e_1 \text{ else } e_2) \\ \to (DT', E', \text{if } e' \text{ then } e_1 \text{ else } e_2)\end{array}}\ (\text{E\_If})$$

$$\frac{SV(v) = [A]}{\begin{array}{c}(DT, E, \text{if } v \text{ then } e_1 \text{ else } e_2) \\ \to (DT, E, e_1; \text{nil})\end{array}}\ (\text{E\_If\_True}) \qquad \frac{SV(v) = \text{nil}}{\begin{array}{c}(DT, E, \text{if } v \text{ then } e_1 \text{ else } e_2) \\ \to (DT, E, e_2; \text{nil})\end{array}}\ (\text{E\_If\_False})$$

$$\frac{DT' = DT[A.m \mapsto \lambda x.e ::: \tau_1 \to \tau_2]}{\begin{array}{c}(DT, E, \text{def } A.m = \lambda x.e ::: \tau_1 \to \tau_2) \\ \to (DT', E, \text{nil})\end{array}}\ (\text{E\_Def\_Mth}) \qquad \frac{(DT, E, e_0) \to (DT', E', e_0')}{\begin{array}{c}(DT, E, e_0.m(e_1)) \\ \to (DT', E', e_0'.m(e_1))\end{array}}\ (\text{E\_Ivk1})$$

$$\frac{(DT, E, e_1) \to (DT', E', e_1')}{(DT, E, v.m(e_1)) \to (DT', E', v.m(e_1'))}\ (\text{E\_Ivk2}) \qquad \frac{\begin{array}{c}SV(v_0) = [A] \\ CT(A.m) = \lambda x.e ::: \tau_1 \to \tau_2 \\ E' = E[\text{self} \mapsto [A], x \mapsto v_1]\end{array}}{(DT, E, v_0.m(v_1)) \to (DT, E', e)}\ (\text{E\_Ivk3})$$

$$\frac{SV(v_0) = [A] \quad CT(A.m) = \lambda x.e ::: \tau_1 \to \tau_2}{\begin{array}{c}(DT, E, (\langle [m ::: \tau' \to ?] \Leftarrow ? \rangle v_0).m(v_1)) \\ \to (DT, E, \langle\!\langle ? \Leftarrow \tau_2 \rangle\!\rangle([A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1)))\end{array}}\ (\text{E\_Ivk4}) \qquad \frac{(DT, E, e) \to (DT', E', e')}{\begin{array}{c}(DT, E, \langle \tau_d \Leftarrow \tau \rangle e) \\ \to (DT', E', \langle \tau_d \Leftarrow \tau \rangle e')\end{array}}\ (\text{E\_Cast})$$

$$\frac{\tau_0 \neq \tau_2}{\begin{array}{c}(DT, E, \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v) \\ \to (DT, E, \langle\!\langle (\tau_0 \leftharpoonup \tau_2) \Leftarrow \tau_0 \rangle\!\rangle v)\end{array}}\ (\text{E\_Cast\_Merge}) \qquad \frac{\tau_0 = \tau_2}{\begin{array}{c}(DT, E, \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v) \\ \to (DT, E, v)\end{array}}\ (\text{E\_Cast\_Remove})$$

図 4.9　本研究における評価規則

しているため，E_If_True と E_If_False において，評価後の式の後に nil を追加している．

E_Def_Mth は，メソッド定義式の評価である．型付け規則において，メソッド定義式の型は Nil としているため，評価についても，nil に評価している．この時，動的テーブル DT に定義したメソッドの場所 (クラス名)，メソッド名と，メソッド本体の対応関係を追加する．

E_Ivk1，E_Ivk2，E_Ivk3 は，メソッド呼び出し $e_0.m(e_1)$ の評価である．E_Ivk1 はレシーバ e_0 を 1 ステップ評価する．E_Ivk2 は，e_0 が型注釈の有無に関わらずクラス A のインスタンス $[A]$ と判明した後に，実引数 e_1 を 1 ステップ評価する．なお，e_0 を $[A]$ としていることにより，レシーバが nil のメソッド呼び出しは実行時エラーとなる．E_Ivk3 は，e_0 と e_1 が，それぞれ値 v_0，v_1 $(SV(v_0) = [A])$ となった場合の評価であり，この場合は，メソッド本体の評価に移行する．そのため，仮定において，CT にクラス名，メソッド名と，メソッド本体の対応関係が存在していることが条件となっている．また，クラス A のメソッド m の本体においては，評価後の E の self は v_0，メソッドの引数 x は実引数 v_1 に関係づけられる．

E_Ivk4 は，G_Ivk2 によりレシーバが ? 型と型付けされた時のメソッド呼び出しである．レシーバが $[A]$ と判明し，CT においてメソッド $A.m$ が定義されていれば，不明であった m が判明したことになるので，実引数 (τ' 型) に対して，$A.m$ の引数の型である τ_1 へのキャストを挿入する．さらに，G_Ivk2 において，このメソッド呼び出し式は ? 型になるとしているため，メソッド呼び出し式に対して，$A.m$ の返り値の型 τ_2 から ? 型へのキャストを挿入する．

E_Cast は，キャストを含んだ式についての評価である．キャストを含んだ式では，キャストは評価せず，式のみを 1 ステップ評価する．したがって，式が値に評価され，$\langle \tau_d \Leftarrow \tau \rangle \xi$ となった場合，値として扱う．

E_Cast_Merge，E_Cast_Remove は，1 つの値に対して 2 つのキャストが連続して出現したときに，キャストを削減するための評価である．E_Cast_Merge は，2 つのキャストを 1 つに統合する．統合したキャストは，$\langle\!\langle (\tau_0 \leftharpoonup \tau_2) \Leftarrow \tau_0 \rangle\!\rangle$ となる．E_Cast_Remove は，2 つのキャストにおいて，元の型 τ_0 と，最終的な目的の型 τ_2 が一致しているときには，実際にはキャストを行う必要はないので，2 つのキャストを消去する．

5　型健全性の証明

本研究で定めた Ruby サブセットにおいて，次の 2 つの定理を証明することで，型健全性を証明する．型健全性とは，正しく型付けされた項において，以下の進行 (Progress) と保存 (Preservation) の両方を満たすことである．

- 進行
  $(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e')$
  $\Rightarrow (Value\ e') \vee (BadCast\ e') \vee$
  $(\forall DT\ E, (dom(MT') \subseteqq dom(DT)) \wedge$
  $(dom(\Gamma') \subseteqq dom(E)) \Rightarrow \exists e''\ DT'\ E',$
  $(DT, E, e') \rightarrow (DT', E', e''))$
  つまり，正しく型付けされ，キャストが挿入された項 e' は，値となるか，$BadCast\ e'$ となるか，1 ステップ評価を進めることができる．
- 保存
  $((MT, \Gamma, e) \rightsquigarrow (MT'', \Gamma'', \tau, e')) \wedge ((DT, E, e') \rightarrow (DT', E', e'')) \wedge$
  $MT'' \vDash DT \wedge \Gamma'' \vDash E \wedge DT' \vDash MT' \wedge E' \vDash \Gamma' \wedge \tau' <:_{TT} \tau$
  $\Rightarrow If\ (DT = DT') \wedge (E = E')\ then\ \exists e'''\ (MT, \Gamma, e'') \rightsquigarrow (MT'', \Gamma'', \tau', e''')$
  $else\ \exists e'''\ (MT', \Gamma', e'') \rightsquigarrow (MT'', \Gamma'', \tau', e''')$
  つまり，正しく型付けされた項の評価が 1 ステップ進むならば，評価後の項も正しく型付けされている．また，評価後の型と評価前の型の間には，$<:_{TT}$ 関係がある．なお，$MT \vDash DT$，$\Gamma \vDash E$ は，$\vDash$ の両辺の定義域が等しく，かつその定義域におけるメソッド・変数すべてにおいて MT と DT におけるメソッドの型・Γ と E における変数の型が等しいことを意味する．

本論文では，$(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e')$ に関する帰納法に基づき進行を証明し，$(DT, E, e) \rightarrow (DT', E', e')$ に関する帰納法に基づき保存を証明する．

また，型健全性の証明では，次の補題 1，補題 2，補題 3 を利用する．

補題 1. $(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e') \Rightarrow (dom(MT) \subseteqq dom(MT')) \wedge (dom(\Gamma) \subseteqq dom(\Gamma'))$

Proof. 型付け規則において，メソッドや変数の定義が削除されるものは存在しないため，MT，Γ で定義されているメソッドや変数は，必ず MT'，Γ' においても定義されている．　□

補題 2. $(MT, \Gamma, v) \rightsquigarrow (MT', \Gamma', \tau, v') \Rightarrow MT' = MT \wedge \Gamma' = \Gamma \wedge v' = v$

Proof. 値 v を型付けするときに用いられ得る型付け規則は，G_Object，G_Nil，G_Cast の 3 つのみである．G_Object，G_Nil では，型付け前後で MT，Γ，$[A]$ または nil に変化はないため，補題を満たす．G_Cast で型付けされる場合，つまり $v = \langle \tau \Leftarrow \tau' \rangle \xi$ の場合を考える．

$$\frac{(MT, \Gamma, \xi) \rightsquigarrow (MT', \Gamma', \tau', \xi) \quad e = \langle \tau \Leftarrow \tau' \rangle \xi}{(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e)} \quad \text{(G\_Cast)}$$

ξ の型付けは，G_Object または G_Nil による型付けのため，MT，Γ，$[A]$ または nil に変化はないことから，$MT' = MT$，$\Gamma' = \Gamma$ となる．そのため，G_Cast により，$(MT, \Gamma, \langle \tau \Leftarrow \tau' \rangle \xi) \rightsquigarrow (MT, \Gamma, \tau, \langle \tau \Leftarrow \tau' \rangle \xi)$ と型付けされる．したがって，G_Cast で v が型付けされる場合も補題を満たす．

以上より，値 v に対する型付けでは，MT，Γ，v に変化はない． □

補題 3. $(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e') \Rightarrow (MT, \Gamma, e') \rightsquigarrow (MT', \Gamma', \tau, e')$

Proof. キャストが挿入されない型付け規則に関しては自明である．

キャストが挿入される，変数への代入式，メソッド定義式，メソッド呼び出しの型付けについては，型付けに関する帰納法で証明する．なお，それぞれの型付けにおいて，キャストが挿入されない場合は自明であるため，キャストが挿入される場合のみについて証明する．最初に代入式の型付けについて証明する．G_Assign1 では，帰納法の仮定より，$(MT, \Gamma, e') \rightsquigarrow (MT', \Gamma', \tau', e')$ と型付けできる．したがって，型付け後の右辺式は，G_Cast により，$(MT, \Gamma, \langle \tau \Leftarrow \tau' \rangle e') \rightsquigarrow (MT', \Gamma', \tau, \langle \tau \Leftarrow \tau' \rangle e')$ と型付けでき，この型は $\Gamma'(x) = \tau$ と等しい．したがって，$\tau =?$ の場合は G_Assign1 により $(MT, \Gamma, x :::? = \langle \tau \Leftarrow \tau' \rangle e') \rightsquigarrow (MT', \Gamma', \tau, x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e')$，$\tau \neq?$ の場合は G_Assign2 により $(MT, \Gamma, x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e') \rightsquigarrow (MT', \Gamma', \tau, x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e')$ と型付けできるため，G_Assign1 においては，補題を満たす．同様の証明により，G_Assign2 の場合は証明できる．

次にメソッド定義式の型付けについて証明する．G_Def_Mth1 では，帰納法の仮定により $(MT, \Gamma[\text{self} \mapsto A, x \mapsto \tau_1], e') \rightsquigarrow (MT', \Gamma', \tau', e')$ と型付けでき，$\tau' \lesssim_{TT} \tau_2$，$TT(A.m) = tau_1 \to \tau_2$ が言える．したがって，G_Cast により，$(MT, \Gamma[\text{self} \mapsto A, x \mapsto \tau_1], \langle \tau_2 \Leftarrow \tau' \rangle e') \rightsquigarrow (MT', \Gamma', \tau_2, \langle \tau_2 \Leftarrow \tau' \rangle e')$ と型付けできる．ここで，$\tau_1 = \tau_2 =?$ である場合は，G_Def_Mth1 により，$(MT, \Gamma, \text{def } A.m = \lambda x.\langle \tau_2 \Leftarrow \tau' \rangle e' :::? \to?) \rightsquigarrow (MT, \Gamma, \text{Nil}, mboxdef\ A.m = \lambda x.\langle \tau_2 \Leftarrow \tau' \rangle e' ::: \tau_1 \to \tau_2)$ と型付けされる（メソッド本体の型が τ_2 のためキャストは挿入されない）．$\tau_1 \neq?$，$\tau_2 \neq?$ である場合は，メソッド m が子孫クラスに定義されているか否かで場合分けする．子孫クラスに定義されている場合は，G_Def_Mth2 により $(MT, \Gamma, \text{def } A.m =$

$\lambda x.\langle \tau_2 \Leftarrow \tau' \rangle e' ::: \tau_1 \to \tau_2) \rightsquigarrow (MT, \Gamma, \mathrm{Nil}, mboxdef\ A.m = \lambda x.\langle \tau_2 \Leftarrow \tau' \rangle e' ::: \tau_1 \to \tau_2)$ と型付けされる．子孫クラスに定義されていない場合は，G_Def_Mth4 により，型付けできる．G_Def_Mth4 において，$\tau_1' = \tau_1$，$\tau_2' = \tau_2$ となるため，$MT' = MT$，$e'' = e'$ となる．したがって，$(MT, \Gamma[\mathrm{self} \mapsto A, x \mapsto \tau_1], \langle \tau_2 \Leftarrow \tau' \rangle e') \rightsquigarrow (MT', \Gamma', \tau_2, \langle \tau_2 \Leftarrow \tau' \rangle e')$，$TT(A.m) = \tau_1 \to \tau_2$ より，G_Def_Mth4 を用いて $(MT, \Gamma, \mathrm{def}\ A.m = \lambda x.\langle \tau_2 \Leftarrow \tau' \rangle e' ::: \tau_1 \to \tau_2) \rightsquigarrow (MT, \Gamma, \mathrm{Nil}, \mathrm{def}\ A.m = \lambda x.\langle \tau_2 \Leftarrow \tau' \rangle e' ::: \tau_1 \to \tau_2)$ と型付けできる．以上より，G_Def_Mth1 においては，補題を満たす．同様の証明により，G_Def_Mth2，G_Def_Mth3，G_Def_Mth4 の場合は証明できる．

次にメソッド呼び出しの型付けについて証明する．G_Ivk1 では，帰納法の仮定により，$(MT, \Gamma, e_0') \rightsquigarrow (MT', \Gamma', A, e_0')$，$(MT', \Gamma', e_1') \rightsquigarrow (MT'', \Gamma'', \tau, e_1')$ と型付けできる．したがって，型付け後の実引数は，G_Cast により，$(MT', \Gamma', \langle (\tau \leftharpoonup \tau_1) \Leftarrow \tau \rangle e_1') \rightsquigarrow (MT'', \Gamma'', A, \langle (\tau \leftharpoonup \tau_1) \Leftarrow \tau \rangle e_1')$ と型付けできる．さらに，$\tau \lesssim_{TT} \tau_1$ であることから，$(\tau \leftharpoonup \tau_1) \lesssim_{TT} \tau_1$ が言え，かつ，$((\tau \leftharpoonup \tau_1) \leftharpoonup \tau_1) = (\tau \leftharpoonup \tau_1)$ が言える．したがって，G_Ivk1 により，$(MT, \Gamma, e_0'.m(\langle (\tau \leftharpoonup \tau_1) \Leftarrow \tau \rangle)) \rightsquigarrow (MT'', \Gamma'', \tau_2, e_0'.m(\langle (\tau \leftharpoonup \tau_1) \Leftarrow \tau \rangle))$ と型付けできるため，G_Ivk1 においては，補題を満たす．

G_Ivk2 では，帰納法の仮定により，$(MT, \Gamma, e_0') \rightsquigarrow (MT', \Gamma', ?, e_0')$，$(MT', \Gamma', e_1') \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と型付けできる．したがって，型付け後のレシーバは，G_Cast により，$(MT, \Gamma, \langle [m ::: \tau_1 \to ?] \Leftarrow ? \rangle e_0') \rightsquigarrow (MT', \Gamma', [m ::: \tau_1 \to ?], \langle [m ::: \tau_1 \to ?] \Leftarrow ? \rangle e_0')$ と型付けできる．$[m ::: \tau_1 \to ?]$ は，キャスト内の型注釈の通りの型を持つメソッド m を持つあるクラス型であるため，$TT([m ::: \tau_1 \to ?].m) = \tau_1 \to ?$ と言える．そのため，実引数 e_1 の型と要求されているメソッドの引数の型が τ_1 で一致していることから，G_Ivk1 で型付けする際に e_1 にはキャストが挿入されない．したがって，G_Ivk1 により，$(MT, \Gamma, (\langle [m ::: \tau_1 \to ?] \Leftarrow ? \rangle e_0').m(e_1')) \rightsquigarrow (MT'', \Gamma'', ?, (\langle [m ::: \tau_1 \to ?] \Leftarrow ? \rangle e_0').m(e_1'))$ と型付けできるため，G_Ivk2 においては，補題を満たす．□

5.1 進行の証明

$(MT, \Gamma, e) \rightsquigarrow (MT', \Gamma', \tau, e') \Rightarrow (Value\ e') \vee (BadCast\ e') \vee$
$(\forall DT\ E, (dom(MT') \subseteqq dom(DT)) \wedge (dom(\Gamma') \subseteqq dom(E))$
$\Rightarrow \exists e''\ DT'\ E', (DT, E, e') \to (DT', E', e''))$

G_Object，G_Nil，G_Cast の場合　$e = e' = [A], e = e' = \mathrm{nil}, e = e' = \langle \tau_d \Leftarrow \tau \rangle e_0$

G_Object と G_Nil は値についての型付けのため，進行を満たす．

G_Cast の場合は，帰納法の仮定により，$(MT, \Gamma, e_0) \rightsquigarrow (MT', \Gamma', \tau, e_0)$ と型付けされるならば，$Value\ e_0$ か，$BadCast\ e_0$ か，すべての DT，E において $(dom(MT') \subseteqq dom(DT)) \wedge$

$(dom(\Gamma') \subseteqq dom(E))$ を満たすならば，ある e_0', DT', E' が存在し，$(DT, E, e_0) \to (DT', E', e_0')$ と 1 ステップ評価が進む．$Value\ e_0$ である場合は，$Value\ e$ となる．$BadCast\ e_0$ であれば，$BadCast\ e$ となる．e_0 が 1 ステップ評価できる場合は，E_Cast により 1 ステップ評価が可能である．以上により，G_Cast において進行を満たす．

G_Self の場合　$e = e' = \text{self}$

G_Self により self の型は $\tau = \Gamma(\text{self})$ と型付けされる．型付けの前後で Γ に変化がないため，$\Gamma' = \Gamma$ である．したがって，e' が評価可能である場合，$dom(\Gamma) \subseteqq dom(E)$ となり，$\text{self} \in dom(\Gamma)$ であるならば，必ず $\text{self} \in dom(E)$ となる．したがって，評価規則 E_Self が適用できて，E において束縛されている値に評価される．以上より，e' が 1 ステップ評価を進めることができるため，進行 を満たす．

G_Variable の場合　$e = e' = x$

G_Self と同様に進行を満たす．

G_New の場合　$e = e' = A.\text{new}$

A.new は，G_New により，A と型付けされる．A.new は，E_New により，$[A]$ と評価される．したがって，e' は 1 ステップ評価を進めることができるため，進行 を満たす．

G_Sequence の場合　$e = e_1; e_2 \quad e' = e_1'; e_2'$

帰納法の仮定により，$(MT, \Gamma, e_1) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1')$ と型付けされるならば，$Value\ e_1'$ か，$BadCast\ e_1'$ か，すべての DT，E において $(dom(MT_1) \subseteqq dom(DT)) \wedge (dom(\Gamma_1) \subseteqq dom(E))$ を満たすならば，ある e_1''，DT'，E' が存在し，$(DT, E, e_1') \to (DT', E', e_1'')$ と 1 ステップ評価が進む．この 3 つの場合に分けて証明する．$Value\ e_1'$ である場合は，E_Sequence2 により，e' は e_2' に評価される．$BadCast\ e_1'$ である場合は $BadCast\ e'$ である．3 つ目の場合について考える．$dom(MT_2) \subseteqq dom(DT) \wedge dom(\Gamma_2) \subseteqq dom(E)$ を満たす任意の DT，E を仮定する．補題 5 より，$dom(MT_1) \subseteqq dom(MT_2) \wedge dom(\Gamma_1) \subseteqq dom(\Gamma_2)$ となる．したがって，$dom(MT_1) \subseteqq dom(DT) \wedge dom(\Gamma_1) \subseteqq dom(E)$ となるため，帰納法の仮定により，ある e_1''，DT'，E' が存在し，$(DT, E, e_1') \to (DT', E', e_1'')$ を満たす．そのため，e' は，E_Sequence1 により，$e_1''; e_2'$ に評価される．

G_If の場合　$e = \text{if } e_0 \text{ then } e_1 \text{ else } e_2 \qquad e' = \text{if } e_0' \text{ then } e_1' \text{ else } e_2'$

帰納法の仮定により，$(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, \tau_0, e_0')$ と型付けされるならば，$Value\ e_0'$ か，$BadCast\ e_0'$ か，すべての DT，E において $(dom(MT_0) \subseteqq dom(DT)) \wedge (dom(\Gamma_0) \subseteqq dom(E))$ を満たすならば，ある e_0''，DT'，E' が存在し，$(DT, E, e_0') \to (DT', E', e_0'')$ と 1 ステップ評価が進む．この 3 つの場合に分けて証明する．$Value\ e_0'$ の場合は，e_0' は，$[A]$，nil，$\langle \tau_d \Leftarrow \tau \rangle [A]$，$\langle \tau_d \Leftarrow \tau \rangle \text{nil}$ のいずれかになり得る．$SV(e_0') = [A]$ ならば，E_If_True により，e' は e_1' に評価が進む．$SV(e_0') = \text{nil}$ ならば，E_If_False により，e' は e_2' に評価が進む．$BadCast\ e_0'$ の

場合は，$BadCast\ e'$ である．3 つ目の場合は，G_Sequence の場合と同様に証明でき，E_If により，e' は if e_0'' then e_1' else e_2' に評価される．

G_Assign1 の場合　$e = x :::? = e_0 \quad e' = x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e_0'$

帰納法の仮定により，$(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, \tau', e_0')$ と型付けされるならば，$Value\ e_0'$ か，$BadCast\ e_0'$ か，すべての DT，E において $(dom(MT_0) \subseteqq dom(DT)) \wedge (dom(\Gamma_0) \subseteqq dom(E))$ を満たすならば，ある e_0''，DT'，E' が存在し，$(DT, E, e_0') \rightarrow (DT', E', e_0'')$ と 1 ステップ評価が進む．キャストの有無とこの 3 つの場合に分けて証明する．

最初に，$\tau \neq \tau'$ の場合について考える．この時，$e' = x ::: \tau = \langle \tau \Leftarrow \tau' \rangle\ e_0'$ となる．$Value\ e_0'$ の場合について考える．e_0' が値 ξ であれば，E_Assign2 により評価が 1 ステップ進む．e_0' が $\langle \tau'' \Leftarrow \tau''' \rangle \xi$ であれば，代入右辺はキャストが 2 つ重なった形になり，$BadCast$ を起こすか，E_Cast_Merge あるいは E_Cast_Remove で評価が進む．e_0' が $BadCast$ の場合は，$BadCast\ e'$ となる．3 つ目の場合は，G_Sequence の場合と同様に証明でき，E_Cast により代入右辺が $(DT, E, \langle \tau \Leftarrow \tau' \rangle e_0') \rightarrow (DT', E', \langle \tau \Leftarrow \tau' \rangle e_0'')$ と評価できるため，E_Assign1 により，e' は $x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0''$ に評価される．

次に，$\tau = \tau'$ の場合については，$e' = x ::: \tau = e_0'$ となる．$Value\ e_0'$ の場合はキャストの有無によらず，E_Assign2 により評価が進む．$BadCast\ e_0'$ の場合は $BadCast\ e'$ となる．3 つ目の場合は，$\tau \neq \tau'$ の場合と同様に証明でき，E_Assign1 により，e' は $x ::: \tau = e_0''$ に評価される．

G_Assign2 の場合　$e = x ::: \tau = e_0 \quad e' = x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e_0$

G_Assign1 の場合と同様である．

G_Def_Mth1 の場合　$e = \text{def}\ A.m = \lambda x.e_0 :::? \rightarrow ?$

$e' = \text{def}\ A.m = \lambda x.\langle\!\langle \tau_2 \Leftarrow \tau' \rangle\!\rangle e_0' ::: \tau_1 \rightarrow \tau_2$

帰納法の仮定により，$(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, \tau', e_0')$ と型付けされるならば，$Value\ e_0'$ か，$BadCast\ e_0'$ か，すべての DT，E において $(dom(MT_0) \subseteqq dom(DT)) \wedge (dom(\Gamma_0) \subseteqq dom(E))$ を満たすならば，ある e_0''，DT'，E' が存在し，$(DT, E, e_0') \rightarrow (DT', E', e_0'')$ と 1 ステップ評価が進む．$BadCast\ e_0'$ 以外の場合は，e' は，E_Def_Mth により，nil に評価される．$BadCast\ e_0'$ の場合は，$BadCast\ e'$ となる．

G_Def_Mth2 の場合　$e = \text{def}\ A.m = \lambda x.e_0 ::: \tau_1 \rightarrow \tau_2$

$e' = \text{def}\ A.m = \lambda x.\langle\!\langle \tau_2 \Leftarrow \tau' \rangle\!\rangle e_0' ::: \tau_1 \rightarrow \tau_2$

G_Def_Mth1 の場合と同様である．

G_Def_Mth3 の場合　$e = \text{def}\ A.m = \lambda x.e_0 ::: \tau_1 \rightarrow \tau_2$

$e' = \text{def}\ A.m = \lambda x.\langle\!\langle \tau_2 \Leftarrow \tau' \rangle\!\rangle e_0' ::: \tau_1 \rightarrow \tau_2$

G_Def_Mth1 の場合と同様である．

G_Def_Mth4 の場合　$e = \mathrm{def}\ A.m = \lambda x.e_0 ::: \tau_1 \to \tau_2$

$e' = \mathrm{def}\ A.m = \lambda x.\langle\!\langle \tau_2' \Leftarrow \tau' \rangle\!\rangle e_0' ::: \tau_1' \to \tau_2'$

G_Def_Mth1 の場合と同様である．

G_Ivk1 の場合　$e = e_0.m(e_1)$

$e' = e_0'.m(\langle\!\langle \tau' \Leftarrow \tau \rangle\!\rangle e_1')$ ただし $\tau' = \tau \leftharpoonup \tau_1$

帰納法の仮定により，$(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, A, e_0')$ と型付けされるならば，$Value\ e_0'$ か，$BadCast\ e_0'$ か，すべての DT，E において $(dom(MT_0) \subseteqq dom(DT)) \wedge (dom(\Gamma_0) \subseteqq dom(E))$ を満たすならば，ある e_0''，DT'，E' が存在し，$(DT, E, e_0') \to (DT', E', e_0'')$ と 1 ステップ評価が進む．e_1' についても同様である．また，G_Ivk1 の仮定により，e_0' の型は A で，$MT(A.m) = \tau_1 \to \tau_2$ である．キャストの有無と，帰納法の仮定の 3 つの場合に分けて証明する．最初に，$\tau \neq \tau'$ の場合について考える．この時，$e' = e_0'.m(\langle \tau' \Leftarrow \tau \rangle e_1')$ となる．e_0' が値 v である場合は，次のようになる．e_1' が値 ξ である場合，E_Ivk3 により評価が進む．e_1' が $\langle \tau \Leftarrow \tau_3 \rangle \xi$ の形であれば，G_Assign1 の時と同様にして，E_Cast_Merge あるいは E_Cast_Remove で評価が進むか $BadCast$ になるかのいずれかである．e_1' の帰納法の仮定の 3 つ目の場合は $(dom(MT'') = dom(DT)) \wedge (dom(\Gamma'') = dom(E))$ と帰納法の仮定により，ある e_1''，DT'，E' が存在し，$(DT, E, e_1') \to (DT', E', e_1'')$ を満たすことが言える．したがって E_Ivk2 により e' は $v.m(e_1'')$ に評価される．e_0' が $BadCast\ e_0'$ ならば，$BadCast\ e'$ となる．e_0' の帰納法の仮定の 3 つ目の場合は，G_Sequence の場合と同様に証明でき，e' は，E_Ivk1 により $e_0''.m(e_1)$ に評価される．

最後に，$\tau = \tau_1$ の場合，$e' = e_0'.m(e_1')$ となるため，キャストが関わる部分を除いて，$\tau \neq \tau_1$ の場合と同様である．

G_Ivk2 の場合　$e = e_0.m(e_1)$

$e' = (\langle [m ::: \tau_1 \to ?] \Leftarrow ? \rangle e_0').m(e_1')$

帰納法の仮定により，$(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, ?, e_0')$ と型付けされるならば，$Value\ e_0'$ か，$BadCast\ e_0'$ か，すべての DT，E において $(dom(MT_0) \subseteqq dom(DT)) \wedge (dom(\Gamma_0) \subseteqq dom(E))$ を満たすならば，ある e_0''，DT'，E' が存在し，$(DT, E, e_0') \to (DT', E', e_0'')$ と 1 ステップ評価が進む．e_1' についても同様である．G_Ivk1 と異なる部分は，レシーバ e_0' に必ずキャストが挿入され，キャスト先の型が $[m ::: \tau_1 \to ?]$ となる点である．G_Ivk2 の仮定により，e_0' は ? 型である．そのため，e_0' は値 ξ とはならない．e_0' が $\langle ? \Leftarrow \tau_3 \rangle \xi$ の形であり，ξ が $[A]$ であれば，E_Ivk4 により評価が進み，ξ が nil であれば，$[m ::: \tau_1 \to ?]$ 型と Nil 型に $\lesssim_{TT}$ 関係がないため $BadCast\ e'$ となる．e_0' が $BadCast\ e_0'$ の場合は，$BadCast\ e'$ となる．e_0' が e_0'' に 1 ステップ評価が進む場合は，G_Ivk1 の場合と同様である．

5.2 保存の証明

$$((MT,\Gamma,e) \rightsquigarrow (MT'',\Gamma'',\tau,e')) \wedge ((DT,E,e') \to (DT',E',e'')) \wedge$$
$$MT'' \vDash DT \wedge \Gamma'' \vDash E \wedge DT' \vDash MT' \wedge E' \vDash \Gamma' \wedge \tau' <:_{TT} \tau$$
$$\Rightarrow If\ (DT = DT') \wedge (E = E')\ then\ \exists e'''\ (MT,\Gamma,e'') \rightsquigarrow (MT'',\Gamma'',\tau',e''')$$
$$else\ \exists e'''\ (MT',\Gamma',e'') \rightsquigarrow (MT'',\Gamma'',\tau',e''')$$

E_Self の場合　$e = e' = \mathrm{self} \quad e'' = v \quad (v = E(\mathrm{self}))$

$$\frac{E(\mathrm{self}) = v}{(DT,E,e') \to (DT,E,e'')} \qquad \text{(E\_Self)}$$

$e' = \mathrm{self}$ となるような型付け規則は G_Self のみである．したがって，G_Self により，$(MT,\Gamma,\mathrm{self}) \rightsquigarrow (MT,\Gamma,\Gamma(\mathrm{self}),\mathrm{self})$ と型付けされたとする．このとき，評価と型付けにおいて，環境 MT,Γ,DT,E に変化はないため，$MT' = MT$，$\Gamma' = \Gamma$ であり，self の型 $\Gamma(\mathrm{self})$ が変化することはない．したがって，評価後の v $(v = E(\mathrm{self}))$ は $\Gamma(\mathrm{self})$ の型で型つけられる．その時の型付けは，v が $[A]$ ならば G_Object，nil ならば G_Nil，$\langle \tau \Leftarrow \tau' \rangle \xi$ ならば G_Cast で型つけられる．以上により，E_Self において保存を満たす．

E_Variable の場合　$e = e' = x \quad e'' = v \quad (v = E(x))$

$$\frac{E(\mathrm{x}) = v}{(DT,E,e') \to (DT,E,e'')} \qquad \text{(E\_Variable)}$$

E_Self と同様に保存を満たす．

E_Assign1 の場合　$e' = x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e'_0 \quad e'' = x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e''_0$

$$\frac{(DT,E,\langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e'_0) \to (DT',E',\langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e''_0)}{(DT,E,e') \to (DT',E',e'')} \qquad \text{(E\_Assign1)}$$

$e' = x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e'_0$ となるような型付け規則は，G_Assign1 または G_Assign2 である．また，型付け後の e'_0 にキャストが挿入される場合 $(\tau' \neq \tau)$ とされない場合 $(\tau' = \tau)$ があるが，どちらにおいても，帰納法の仮定の E_Assign1 による評価は可能である．キャストが挿入される場合は，E_Cast により代入式 e' の右辺が評価可能であるため，E_Assign1 による評価が可能であり，キャストが挿入されない場合は自明である．

まず，G_Assign1 で型付けされた場合について考える．

$$\frac{(MT,\Gamma,e_0) \rightsquigarrow (MT'',\Gamma'',\tau',e'_0) \quad \Gamma''(x) = \tau \quad \tau' \lesssim_{TT} \tau}{(MT,\Gamma,x :::? = e_0) \rightsquigarrow (MT'',\Gamma'',\tau,x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e'_0)} \qquad \text{(G\_Assign1)}$$

e_0' にキャストが挿入される場合における型付けついて考える．G_Assign1 の前提条件である $(MT, \Gamma, e_0) \rightsquigarrow (MT'', \Gamma'', \tau', e_0')$ と補題 3 により，$(MT, \Gamma, e_0') \rightsquigarrow (MT'', \Gamma'', \tau', e_0')$ と型付けでき，これにより G_Cast で $\langle \tau \Leftarrow \tau' \rangle e_0'$ の型付けができ，$(MT, \Gamma, \langle \tau \Leftarrow \tau' \rangle e_0') \rightsquigarrow (MT'', \Gamma'', \tau, \langle \tau \Leftarrow \tau' \rangle e_0')$ となる．これと帰納法の仮定により，DT', E' により生成される MT', Γ' において $(MT', \Gamma', \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma'', \tau, \langle \tau \Leftarrow \tau' \rangle e_0''')$ と型付けできる $\langle \tau \Leftarrow \tau' \rangle e_0'''$ が存在する．また，G_Assign1 の前提条件により，$\Gamma''(x) = \tau, \tau' \lesssim_{TT} \tau$ が言える．ここで，$\langle \tau \Leftarrow \tau' \rangle e_0''$ の型が τ であり，$\Gamma''(x) = \tau$ と等しいため，$\tau \lesssim_{TT} \tau$ となり，e'' を G_Assign1 または G_Assign2 で型付けする際にキャストが挿入されることはない．以上より，$\tau =?$ の場合は，$(MT', \Gamma', \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma'', \tau, \langle \tau \Leftarrow \tau' \rangle e_0''')$，$\Gamma''(x) = \tau$，$\tau \lesssim_{TT} \tau$ より，G_Assign1 を用いて $(MT', \Gamma', x :::? = \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma'', \tau, x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0''')$ が示せ，$\tau \neq?$ の場合は，$(MT', \Gamma', \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma'', \tau, \langle \tau \Leftarrow \tau' \rangle e_0''')$，$\tau \neq?$，$\tau \lesssim_{TT} \tau$ より G_Assign2 を用いて $(MT', \Gamma', x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma''[x \mapsto \tau], \tau, x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0''')$ が示せる．したがって，G_Assign1 で型付けされ，その際にキャストが挿入される場合において，保存を満たす．e_0' にキャストが挿入されない場合における型付けについて考える．このとき，G_Assign1 において $\tau' = \tau$ となる．G_Assign1 の前提条件である $(MT, \Gamma, e_0) \rightsquigarrow (MT'', \Gamma'', \tau, e_0')$ と帰納法の仮定により，DT', E' により生成される MT', Γ' において $(MT', \Gamma', e_0'') \rightsquigarrow (MT'', \Gamma'', \tau, e_0''')$ と型付けできる e_0''' が存在する．また，G_Assign1 の前提条件により，$\Gamma''(x) = \tau$, $\tau \lesssim_{TT} \tau$ が言える．ここで，$\tau' = \tau$ であることから，e'' を G_Assign1 または G_Assign2 で型付けする際にキャストが挿入されることはない．以上より，$\tau =?$ の場合は，$(MT', \Gamma', e_0'') \rightsquigarrow (MT'', \Gamma'', \tau', e_0''')$，$\Gamma''(x) = \tau$，$\tau \lesssim_{TT} \tau$ より，G_Assign1 を用いて $(MT', \Gamma', x :::? = e_0'') \rightsquigarrow (MT'', \Gamma'', \tau, e_0''')$ が示せ，$\tau \neq?$ の場合は，$(MT', \Gamma', e_0'') \rightsquigarrow (MT'', \Gamma'', \tau', e_0''')$，$\tau \neq?$，$\tau' \lesssim_{TT} \tau$ より G_Assign2 を用いて $(MT', \Gamma', x ::: \tau = e_0'') \rightsquigarrow (MT'', \Gamma''[x \mapsto \tau], \tau, e_0''')$ が示せる．したがって，G_Assign1 で型付けされ，その際にキャストが挿入されない場合において，保存を満たす．以上により G_Assign1 で型付けされ，E_Assign1 で評価された場合において，保存を満たす．

次に，G_Assign2 で型付けされた場合について考える．

$$\frac{\begin{array}{c}(MT, \Gamma, e_0) \rightsquigarrow (MT'', \Gamma'', \tau', e_0') \quad \tau' \lesssim_{TT} \tau \\ x \notin dom(\Gamma'') \vee \tau \neq? \quad \Gamma''' = \Gamma''[x \mapsto \tau]\end{array}}{(MT, \Gamma, x ::: \tau = e_0) \rightsquigarrow (MT'', \Gamma''', \tau, x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle e_0')} \quad \text{(G\_Assign2)}$$

e_0' にキャストが挿入される場合における型付けついて考える．G_Assign1 で型付けされた場合の証明と同様に，DT', E' により生成される MT', Γ' において $(MT', \Gamma', \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma'', \tau, \langle \tau \Leftarrow \tau' \rangle e_0''')$ と型付けできる $\langle \tau \Leftarrow \tau' \rangle e_0'''$ が存在することが言え，かつ $\langle \tau \Leftarrow \tau' \rangle e_0''$ の型が τ であることから e'' を G_Assign1 または G_Assign2 で型付けする際にキャストは挿入されない．G_Assign2 の前提条件で，$x \notin dom(\Gamma'') \vee \tau \neq?$ となっているため，それぞれにおいて場合分けして e'' の型付けを考える．$x \notin dom(\Gamma'')$ の場合は，$\tau =?$，$\tau \neq?$ のどちらにおい

ても，e'' は G_Assign2 により，$(MT', \Gamma', x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma''[x \mapsto \tau], \tau, x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0''')$ と型付けできる．$x \in dom(\Gamma'')$ の場合は，$\tau \neq ?$ となるため，やはり e'' は G_Assign2 により，$(MT', \Gamma', x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0'') \rightsquigarrow (MT'', \Gamma''[x \mapsto \tau], \tau, x ::: \tau = \langle \tau \Leftarrow \tau' \rangle e_0''')$ と型付けできる．したがって，G_Assign2 で型付けされ，その際にキャストが挿入される場合において，保存を満たす．キャストが挿入されない場合については，G_Assign1 で型付けされた場合と同様に証明できる．以上により G_Assign2 で型付けされ，E_Assign1 で評価された場合において，保存を満たす．

E_Assign2 の場合　$e' = x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle v \quad e'' = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle v$

$$\frac{E' = E[x \mapsto v]}{(DT, E, e') \to (DT, E', e'')} \qquad \text{(E\_Assign2)}$$

$e' = x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle v$ となるような型付け規則は，G_Assign1 または G_Assign2 である．また，型付け後の v にキャストが挿入される場合 $(\tau' \neq \tau)$ とされない場合 $(\tau' = \tau)$ がある．キャストが挿入される場合は，$v = \langle \tau' \Leftarrow \tau'' \rangle \xi$ であれば，代入式 e' の右辺はキャストが二重に挿入されている状態となるため，値とならず E_Cast_Merge または E_Cast_Remove により代入式 e' の右辺が評価可能であるため，右辺が値となるまで E_Assign1 による評価が進む．したがって，1 ステップ評価が E_Assign2 とならないため，v にキャストが挿入されかつ $v = \langle \tau' \Leftarrow \tau'' \rangle \xi$ となる場合は考えない．それ以外の場合は e' の右辺は値となるため，1 ステップ評価は E_Assign2 となる．

まず，G_Assign1 で型付けされた場合について考える．

$$\frac{(MT, \Gamma, v) \rightsquigarrow (MT'', \Gamma'', \tau', v') \quad \Gamma''(x) = \tau \quad \tau' \lesssim_{TT} \tau}{(MT, \Gamma, x :::? = v) \rightsquigarrow (MT'', \Gamma'', \tau, x ::: \tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle v')} \qquad \text{(G\_Assign1)}$$

その中で，v にキャストが挿入される場合における型付けついて考える．このとき，$v = \xi$ となる．ここでは，G_Assign1 において値 v に対する型付けが前提条件となっているため，補題 2 より，G_Assign1 において，$v' = v$，$MT'' = MT$，$\Gamma'' = \Gamma$ となる．したがって，$(MT, \Gamma, \xi) \rightsquigarrow (MT, \Gamma, \tau', \xi)$ となるので，G_Cast で $(MT, \Gamma, \langle \tau \Leftarrow \tau' \rangle \xi) \rightsquigarrow (MT, \Gamma, \tau, \langle \tau \Leftarrow \tau' \rangle \xi)$ と型付けできる．なお，G_Assign1 における型付けでは，定義済みの変数を？型の型注釈で再定義した場合であるため，型付けや，その後の E_Assign2 の評価により環境の定義域や型が変化することはない．したがって，DT，E' から生成される MT'，Γ' について，$MT' = MT'' = MT$，$\Gamma' = \Gamma'' = \Gamma$ となる．したがって，$(MT, \Gamma, \langle \tau \Leftarrow \tau' \rangle \xi) \rightsquigarrow (MT, \Gamma, \tau, \langle \tau \Leftarrow \tau' \rangle \xi)$ より，G_Assign1 で型付けされ，その際にキャストが挿入された場合において，保存を満たす．v にキャストが挿入されない場合における型付けについては，$\tau' = \tau$ となるため，値 v の型は τ となることから，キャストが挿入される場合と同様に証明できる．以上により G_Assign1 で型付けされ，E_Assign2 で評価された場合において，保存を満たす．

次に，G_Assign2 で型付けされた場合について考える．

$$\frac{(MT,\Gamma,v) \rightsquigarrow (MT'',\Gamma'',\tau',v') \quad \tau' \lesssim_{TT} \tau \quad x \notin dom(\Gamma'') \vee \tau \neq ? \quad \Gamma''' = \Gamma''[x \mapsto \tau]}{(MT,\Gamma,x :::\tau = v) \rightsquigarrow (MT'',\Gamma''',\tau,x :::\tau = \langle\!\langle \tau \Leftarrow \tau' \rangle\!\rangle v')} \quad \text{(G\_Assign2)}$$

その中で，v にキャストが挿入される場合における型付けついて考える．G_Assign1 で型付けされた場合の証明と同様に，$v = \xi$ となり，G_Assign2 において，$MT'' = MT$，$\Gamma'' = \Gamma$，$v' = v$ となり，$(MT,\Gamma,v) \rightsquigarrow (MT,\Gamma,\tau',v)$ となるので，G_Cast で $(MT,\Gamma,\langle \tau \Leftarrow \tau' \rangle \xi) \rightsquigarrow (MT,\Gamma,\tau,\langle \tau \Leftarrow \tau' \rangle \xi)$ と型付けできる．E_Assign2 の評価では，MT，$\Gamma[x \mapsto \tau]$ から生成された DT，E の環境で評価され，E のみが $E[x \mapsto v]$ と環境が変化しているため，環境の定義域や型が変化することはない．したがって，DT，E' から生成される MT'，Γ' について，$MT' = MT'' = MT$，$\Gamma' = \Gamma''' = \Gamma[x \mapsto \tau]$ となる．値に対する型付けは，環境の違いに関わらず同じ型がつけられることから，$(MT,\Gamma''',\langle \tau \Leftarrow \tau' \rangle \xi) \rightsquigarrow (MT,\Gamma''',\tau,\langle \tau \Leftarrow \tau' \rangle \xi)$ と型付けできる．したがって，G_Assign2 で型付けされ，その際にキャストが挿入される場合において，保存を満たす．キャストが挿入されない場合については，G_Assign1 で型付けされた場合と同様に証明できる．以上により G_Assign2 で型付けされ，E_Assign2 で評価された場合において，保存を満たす．

E_New の場合　$e' = A.\text{new}$　$e'' = [A]$

$$\frac{}{(DT,E,e') \rightarrow (DT',E',e'')} \quad \text{(E\_New)}$$

$e' = A.\text{new}$ となるような型付け規則は G_New のみである．したがって，G_New により，$(MT,\Gamma,A.new) \rightsquigarrow (MT,\Gamma,A,A.new)$ と型付けされたとする．$e'' = [A]$ のため，型環境に関わらず e'' の型は G_Object により A 型となり，評価前の型と一致する．以上より，E_New で評価された場合において，保存を満たす．

E_Sequence1 の場合　$e' = e'_1; e'_2$　$e'' = e''_1; e'_2$

$$\frac{(DT,E,e'_1) \rightarrow (DT',E',e''_1)}{(DT,E,e') \rightarrow (DT',E',e'')} \quad \text{(E\_Sequence1)}$$

$e' = e'_1; e'_2$ となるような型付け規則は G_Sequence のみである．したがって，G_Sequence により型付けされたとする．

$$\frac{(MT,\Gamma,e_1) \rightsquigarrow (MT_1,\Gamma_1,\tau_1,e'_1) \quad (MT_1,\Gamma_1,e_2) \rightsquigarrow (MT'',\Gamma'',\tau,e'_2)}{(MT,\Gamma,e_1;e_2) \rightsquigarrow (MT'',\Gamma'',\tau,e'_1;e'_2)} \quad \text{(G\_Sequence)}$$

このとき，$(MT, \Gamma, e_1) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1')$ と帰納法の仮定により，DT'，E' により生成される MT'，Γ' において，$(MT', \Gamma', e_1'') \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1''')$ となる e_1''' が存在する．また，$(MT_1, \Gamma_1, e_2) \rightsquigarrow (MT'', \Gamma'', \tau, e_2')$ と補題 3 より，$(MT_1, \Gamma_1, e_2') \rightsquigarrow (MT'', \Gamma'', \tau, e_2')$ が言える．したがって，$(MT', \Gamma', e_1'') \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1''')$，$(MT_1, \Gamma_1, e_2') \rightsquigarrow (MT'', \Gamma'', \tau, e_2')$ より，G_Sequence を用いて $(MT', \Gamma', e_1''; e_2') \rightsquigarrow (MT'', \Gamma'', \tau, e_1'''; e_2')$ と型付けできるため，E_Sequence1 で評価された場合において，保存を満たす．

E_Sequence2 の場合　$e' = v; e_2' \quad e'' = e_2'$

$$\frac{}{(DT, E, e') \to (DT, E, e'')} \qquad \text{(E\_Sequence2)}$$

$e' = e_1'; e_2'$ となるような型付け規則は G_Sequence のみである．したがって，G_Sequence により型付けされたとする．

$$\frac{(MT, \Gamma, v) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, v') \quad (MT_1, \Gamma_1, e_2) \rightsquigarrow (MT'', \Gamma'', \tau, e_2')}{(MT, \Gamma, v; e_2) \rightsquigarrow (MT'', \Gamma'', \tau, v'; e_2')} \qquad \text{(G\_Sequence)}$$

ここで, $(MT, \Gamma, v) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, v')$ は値の型付けをしているため, 補題 2 より, $MT_1 = MT$, $\Gamma_1 = \Gamma$，$v' = v$ となり，$(MT, \Gamma, v) \rightsquigarrow (MT, \Gamma, \tau_1, v)$，$(MT, \Gamma, e_2) \rightsquigarrow (MT'', \Gamma'', \tau, e_2')$ となる．また，E_Sequence1 の場合と同様に $(MT, \Gamma, e_2') \rightsquigarrow (MT'', \Gamma'', \tau, e_2')$ が言える．さらに，評価においても DT，E に変化がないため，$DT' = DT$，$E' = E$ となる．したがって，$e'' = e_2'$ であるため，$(MT, \Gamma, e_2') \rightsquigarrow (MT'', \Gamma'', \tau, e_2')$ が示せることから，E_Sequence2 で評価された場合において，保存を満たす．

E_If の場合　$e' = \text{if } e_0' \text{ then } e_1' \text{ else } e_2' \quad e'' = \text{if } e_0'' \text{ then } e_1' \text{ else } e_2'$

$$\frac{(DT, E, e_0') \to (DT', E', e_0'')}{(DT, E, e') \to (DT', E', e'')} \qquad \text{(E\_If)}$$

$e' = \text{if } e_0' \text{ then } e_1' \text{ else } e_2'$ となるような型付け規則は G_If のみである．したがって，G_If により型付けされたとする．

$$\frac{\begin{array}{c}(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, \tau_0, e_0') \\ (MT_0, \Gamma_0, e_1) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1') \qquad (MT_0, \Gamma_0, e_2) \rightsquigarrow (MT_2, \Gamma_2, \tau_2, e_2') \\ MT'' = MT_1 = MT_2 \qquad \Gamma'' = \Gamma_1 = \Gamma_2\end{array}}{(MT, \Gamma, \text{if } e_0 \text{ then } e_1 \text{ else } e_2) \rightsquigarrow (MT'', \Gamma'', \text{Nil}, \text{if } e_0' \text{ then } e_1' \text{ else } e_2')} \qquad \text{(G\_If)}$$

ここで，$(MT, \Gamma, e_0) \rightsquigarrow (MT_0, \Gamma_0, \tau_0, e_0')$ と帰納法の仮定により，DT'，E' により生成される MT'，Γ' において，$(MT', \Gamma', e_0'') \rightsquigarrow (MT_0, \Gamma_0, \tau_0, e_0''')$ となる e_0''' が存在する．ま

た，$(MT_0, \Gamma_0, e_1) \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と $(MT_0, \Gamma_0, e_2) \rightsquigarrow (MT'', \Gamma'', \tau_2, e_2')$ と補題 3 より，$(MT_0, \Gamma_0, e_1') \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と $(MT_0, \Gamma_0, e_2') \rightsquigarrow (MT'', \Gamma'', \tau_2, e_2')$ が言える．したがって，$(MT', \Gamma', e_0'') \rightsquigarrow (MT_0, \Gamma_0, \tau_0, e_0''')$，$(MT_0, \Gamma_0, e_1') \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$，$(MT_0, \Gamma_0, e_2') \rightsquigarrow (MT'', \Gamma'', \tau_2, e_2')$ より，G_If を用いて $(MT', \Gamma', \text{if } e_0'' \text{ then } e_1' \text{ else } e_2') \rightsquigarrow (MT'', \Gamma'', \text{Nil}, \text{if } e_0''' \text{ then } e_1' \text{ else } e_2')$ と型付けできるため，E_If で評価された場合において，保存を満たす．

E_If_True の場合　$e' = \text{if } v \text{ then } e_1' \text{ else } e_2' \quad e'' = e_1'; \text{nil}$

$$\frac{SV(v) = [A]}{(DT, E, e') \rightarrow (DT, E, e'')} \qquad \text{(E\_If\_True)}$$

$e' = \text{if } v \text{ then } e_1' \text{ else } e_2'$ となるような型付け規則は G_If のみである．したがって，G_If により型付けされたとする．

$$\frac{\begin{array}{c}(MT, \Gamma, v) \rightsquigarrow (MT_0, \Gamma_0, \tau_0, v') \\ (MT_0, \Gamma_0, e_1) \rightsquigarrow (MT_1, \Gamma_1, \tau_1, e_1') \qquad (MT_0, \Gamma_0, e_2) \rightsquigarrow (MT_2, \Gamma_2, \tau_2, e_2') \\ MT'' = MT_1 = MT_2 \qquad \Gamma'' = \Gamma_1 = \Gamma_2\end{array}}{(MT, \Gamma, \text{if } v' \text{ then } e_1 \text{ else } e_2) \rightsquigarrow (MT'', \Gamma'', \text{Nil}, \text{if } v' \text{ then } e_1' \text{ else } e_2')} \qquad \text{(G\_If)}$$

$(MT, \Gamma, v) \rightsquigarrow (MT_0, \Gamma_0, \tau_0, v')$ と $(MT_0, \Gamma_0, e_1) \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と補題 2 により，G_If において，$MT_0 = MT$，$\Gamma_0 = \Gamma$，$v' = v$ となり，$(MT, \Gamma, v) \rightsquigarrow (MT, \Gamma, \tau_0, v)$，$(MT, \Gamma, e_1) \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と型付けできる．また，$(MT, \Gamma, e_1) \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と補題 3 より，$(MT, \Gamma, e_1') \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と型付けできる．評価において DT，E が変化しないため，$DT' = DT$，$E' = E$ となる．さらに，型環境に関わらず nil は G_Nil により Nil 型となるため，$(MT'', \Gamma'', \text{nil}) \rightsquigarrow (MT'', \Gamma'', \text{Nil}, \text{nil})$ と型付けできる．したがって，$(MT, \Gamma, e_1') \rightsquigarrow (MT'', \Gamma'', \tau_1, e_1')$ と $(MT'', \Gamma'', \text{nil}) \rightsquigarrow (MT'', \Gamma'', \text{Nil}, \text{nil})$ により，G_Sequence を用いて，$(MT, \Gamma, e_1'; \text{nil}) \rightsquigarrow (MT'', \Gamma'', \text{Nil}, e_1'; \text{nil})$ と型付けされる．

以上より，E_If_True で評価された場合は保存を満たす．

E_If_False の場合　$e' = \text{if } v \text{ then } e_1' \text{ else } e_2' \quad e'' = e_2'; \text{nil}$

$$\frac{SV(v) = \text{nil}}{(DT, E, e') \rightarrow (DT, E, e'')} \qquad \text{(E\_If\_False)}$$

E_If_True の場合と同様に証明できる．

E_Def_Mth の場合　$e' = \text{def } A.m = \lambda x.e_0' ::: \tau_1 \rightarrow \tau_2 \quad e'' = \text{nil}$

$$\frac{DT' = DT[A.m \mapsto \lambda x.e_0' ::: \tau_1 \to \tau_2]}{(DT, E, e') \to (DT', E, e'')} \quad \text{(E\_Def\_Mth)}$$

$e' = \text{def } A.m = \lambda x.e_0' ::: \tau_1 \to \tau_2$ となるような型付けは G_Def_Mth1，G_Def_Mth2，G_Def_Mth3, G_Def_Mth4 であり，e' はどれにおいても nil と型つけられる．一方，$e'' = \text{nil}$ のため，環境に関わらず e'' も nil と型つけられる．以上より，E_Def_Mth の場合は保存を満たす．

E_Ivk1 の場合　$e' = e_0'.m(e_1')$　$e'' = e_0''.m(e_1')$

$$\frac{(DT, E, e_0') \to (DT', E', e_0'')}{(DT, E, e') \to (DT', E', e'')} \quad \text{(E\_Ivk1)}$$

$e' = e_0'.m(e_1')$ となるような型付けは，G_Ivk1 または G_Ivk2 である．また，E_Assign1 と同様に，レシーバ e_0' にキャストが挿入される場合とされない場合があるが，どちらにおいても E_Ivk1 による評価が可能である．

まず，G_Ivk1 で型付けされた場合について考える．

$$\frac{\begin{array}{c}(MT, \Gamma, e_0) \rightsquigarrow (MT_1, \Gamma_1, \tau_d, e_0') \\ (MT_1, \Gamma_1, e_1) \rightsquigarrow (MT'', \Gamma'', \tau, e_1') \\ TT(\tau_d.m) = \tau_1 \to \tau_2 \quad \tau \lesssim_{TT} \tau_1 \\ \tau_d = A \vee \tau_d = [m ::: \tau_1 \to \tau_2]\end{array}}{(MT, \Gamma, e_0.m(e_1)) \rightsquigarrow (MT'', \Gamma'', \tau_2, e_0'.m(\langle\!\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle\!\rangle e_1'))} \quad \text{(G\_Ivk1)}$$

$(MT, \Gamma, e_0) \rightsquigarrow (MT_1, \Gamma_1, \tau_d, e_0')$ と帰納法の仮定により，DT', E' により生成される MT', Γ' において $(MT', \Gamma', e_0'') \rightsquigarrow (MT_1, \Gamma_1, \tau_d, e_0''')$ と型付けできる e_0''' が存在する．また，$(MT_1, \Gamma_1, e_1) \rightsquigarrow (MT'', \Gamma'', \tau, e_1')$ と補題 3 により，$(MT_1, \Gamma_1, e_1') \rightsquigarrow (MT'', \Gamma'', \tau, e_1')$ と型付けされる．さらに，G_Ivk1 の前提条件より $TT(\tau_d.m) = \tau_1 \to \tau_2$，$\tau_d = A \vee \tau_d = [m ::: \tau_1 \to \tau_2]$ が言える．e_1' にキャストが挿入される場合 $(\tau \neq \tau_1)$ について考える．この時，$(MT_1, \Gamma_1, e_1') \rightsquigarrow (MT'', \Gamma'', \tau, e_1')$ と G_Cast により，$(MT_1, \Gamma_1, \langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e_1') \rightsquigarrow (MT'', \Gamma'', \tau \leftharpoonup \tau_1, \langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e_1')$ と型付けされる．この時，$(\tau \leftharpoonup \tau_1) \lesssim_{TT} \tau_1$ であり，$((\tau \leftharpoonup \tau_1) \sim_{TT} (\tau \leftharpoonup \tau_1)) \wedge ((\tau \leftharpoonup \tau_1) <:_{TT} \tau_1)$ であることから，$((\tau \leftharpoonup \tau_1) \leftharpoonup \tau_1) = (\tau \leftharpoonup \tau_1)$ が言える．したがって，G_Ivk1 により型付けされ，その際に引数 $\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e_1'$ にキャストは挿入されず，$(MT', \Gamma', e_0''.m(\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e_1')) \rightsquigarrow (MT'', \Gamma'', \tau_2, e_0'''.m(\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e_1'))$ と型付けされる．e_1' にキャストが挿入されない場合については，キャストが関わる部分を除いてキャストが挿入される場合と同様に証明できる．

次に，G_Ivk2 で型付けされた場合について考える．

$$\frac{\begin{array}{c}(MT,\Gamma,e_0) \rightsquigarrow (MT_1,\Gamma_1,?,e_0') \\ (MT_1,\Gamma_1,e_1) \rightsquigarrow (MT'',\Gamma'',\tau_1,e_1')\end{array}}{(MT,\Gamma,e_0.m(e_1)) \rightsquigarrow (MT'',\Gamma'',?,(\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0').m(e_1'))} \quad \text{(G\_Ivk2)}$$

e_0' には必ずキャストが挿入されるが，この場合も E_Ivk1 による評価が可能である．$(MT,\Gamma,e_0) \rightsquigarrow (MT_1,\Gamma_1,?,e_0')$ と補題 3 より，$(MT,\Gamma,e_0') \rightsquigarrow (MT_1,\Gamma_1,?,e_0')$ と型付けできる．そのため，G_Cast により，$(MT,\Gamma,\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0') \rightsquigarrow (MT_1,\Gamma_1,[m ::: \tau_1 \to ?],\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0')$ と型付けできる．したがって，G_Ivk1 で型付けされた場合と同様に，DT',E' により生成される MT',Γ' において $(MT',\Gamma',\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0'') \rightsquigarrow (MT_1,\Gamma_1,[m ::: \tau_1 \to ?],\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0''')$ と型付けできる $\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0'''$ が存在することが言える．また，$(MT_1,\Gamma_1,e_1) \rightsquigarrow (MT'',\Gamma'',\tau_1,e_1')$ と補題 3 より，$(MT_1,\Gamma_1,e_1') \rightsquigarrow (MT'',\Gamma'',\tau_1,e_1')$ と型付けできる．さらに，$TT([m ::: \tau_1 \to \tau_2].m) = \tau_1 \to \tau_2$ のため，e_1' にキャストが挿入されることはない．以上より，G_Ivk1 により $(MT',\Gamma',(\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0'').m(e_1')) \rightsquigarrow (MT_1,\Gamma_1,?,(\langle[m ::: \tau_1 \to ?] \Leftarrow ?\rangle e_0''').m(e_1'))$ と型付けできる．

以上より，E_Ivk1 により評価される場合は保存を満たす．

E_Ivk2 の場合　$e' = v.m(e_1')$　$e'' = v.m(e_1'')$

$$\frac{(DT,E,e_1') \to (DT',E',e_1'')}{(DT,E,e') \to (DT',E',e'')} \quad \text{(E\_Ivk2)}$$

$e' = v.m(e_1')$ となるような型付けは，G_Ivk1 のみである．また，E_Assign1 と同様に，引数 e_1' にキャストが挿入される場合とされない場合があるが，どちらにおいても E_Ivk1 による評価が可能である．

$$\frac{\begin{array}{c}(MT,\Gamma,v) \rightsquigarrow (MT_1,\Gamma_1,\tau_d,v') \\ (MT_1,\Gamma_1,e_1) \rightsquigarrow (MT'',\Gamma'',\tau,e_1') \\ TT(\tau_d.m) = \tau_1 \to \tau_2 \quad \tau \lesssim_{TT} \tau_1 \\ \tau_d = A \vee \tau_d = [m ::: \tau_1 \to \tau_2]\end{array}}{(MT,\Gamma,v.m(e_1)) \rightsquigarrow (MT'',\Gamma'',\tau_2,v'.m(\langle\!\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle\!\rangle e_1'))} \quad \text{(G\_Ivk1)}$$

$(MT,\Gamma,v) \rightsquigarrow (MT_1,\Gamma_1,\tau_d,v')$ と補題 2 より，$MT_1 = MT$，$\Gamma_1 = \Gamma$，$v' = v$ となる．また，値の型は型環境によらず一意に決まるため，$(MT,\Gamma,v) \rightsquigarrow (MT,\Gamma,\tau_d,v)$ より $(MT',\Gamma',v) \rightsquigarrow (MT',\Gamma',\tau_d,v)$ と型付けできる．$\tau = \tau_1$ の場合は，e_1' へのキャストは挿入されないため，$(MT,\Gamma,e_1) \rightsquigarrow (MT'',\Gamma'',\tau,e_1')$ と帰納法の仮定により，DT',E' により生成される MT',Γ' において $(MT',\Gamma',e_1'') \rightsquigarrow (MT'',\Gamma'',\tau_d,e_1''')$ と型付けできる e_1''' が存在する．したがって，G_Ivk1 により e'' は $(MT',\Gamma',v.m(e_1'')) \rightsquigarrow (MT'',\Gamma'',\tau_2,v'.m(e_1'''))$ と型付けできる．

$\tau \neq \tau_1$ の場合は，e'_1 へのキャストは挿入される．$(MT_1, \Gamma_1, e_1) \rightsquigarrow (MT'', \Gamma'', \tau, e'_1)$ と補題 3 より，$(MT_1, \Gamma_1, e'_1) \rightsquigarrow (MT'', \Gamma'', \tau, e'_1)$ が言える．したがって，G_Cast により，$(MT_1, \Gamma_1, \langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e'_1) \rightsquigarrow (MT'', \Gamma'', \tau \leftharpoonup \tau_1, \langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e'_1)$ と型付けできる．$\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e'_1$ は，E_Ivk2 で評価可能である．また，E_Ivk1 の証明と同様に，$((\tau \leftharpoonup \tau_1) \leftharpoonup \tau_1) = (\tau \leftharpoonup \tau_1)$ となることから，G_Ivk2 における e'' の型付けの際に e'_1 にキャストが挿入されることはない．したがって，G_Ivk1 により，e'' は $(MT', \Gamma', v.m(\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e''_1)) \rightsquigarrow (MT'', \Gamma'', \tau_2, v'.m(\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle e'''_1))$ と型付けできる．

以上により，E_Ivk2 で評価される場合は，保存を満たす．

E_Ivk3 の場合　$e' = v_0.m(v_1)$　$e'' = e$

$$\frac{\begin{array}{c} SV(v_0) = [A] \\ CT(A.m) = \lambda x.e ::: \tau_1 \to \tau_2 \\ E' = E[\mathrm{self} \mapsto [A], x \mapsto v_1] \end{array}}{(DT, E, e') \to (DT, E', e'')} \quad \text{(E\_Ivk3)}$$

$e' = v_0.m(v_1)$ となる型付けは，G_Ivk1 のみである．

$$\frac{\begin{array}{c} (MT, \Gamma, v_0) \rightsquigarrow (MT_1, \Gamma_1, \tau_d, v'_0) \\ (MT_1, \Gamma_1, v_1) \rightsquigarrow (MT'', \Gamma'', \tau, v'_1) \\ TT(\tau_d.m) = \tau_1 \to \tau_2 \quad \tau \lesssim_{TT} \tau_1 \\ \tau_d = A \vee \tau_d = [m ::: \tau_1 \to \tau_2] \end{array}}{(MT, \Gamma, v_0.m(v_1)) \rightsquigarrow (MT'', \Gamma'', \tau_2, v'_0.m(\langle\!\langle(\tau \leftharpoonup \tau_1) \Leftarrow \tau\rangle\!\rangle v'_1))} \quad \text{(G\_Ivk1)}$$

$(MT, \Gamma, v_0) \rightsquigarrow (MT_1, \Gamma_1, \tau_d, v'_0)$ と $(MT_1, \Gamma_1, v_1) \rightsquigarrow (MT'', \Gamma'', \tau_d, v'_1)$ と補題 2 より，$MT'' = MT_1 = MT$，$\Gamma'' = \Gamma_1 = \Gamma$，$v'_0 = v_0$，$v'_1 = v_1$ が言える．また，v_0 にはキャストが最大で 1 つしか挿入されないため，$\tau_d \neq [m ::: \tau_1 \to \tau_2]$ が言える．したがって，$\tau_d = A$ となる．また，E_Ivk3 の前提条件より $SV(v_0) = [A]$ であることから，$v_0 = [A]$ が言える．$TT(A.m) = \tau_1 \to \tau_2$，$CT(A.m) = \lambda x.e ::: \tau_1 \to \tau_2$ であり，かつ，TT は MT を参照していること，CT は DT を参照していること，さらに $MT \vDash DT \wedge \Gamma \vDash E \wedge DT \vDash MT' \wedge E' \vDash \Gamma'$ であることより，MT，Γ における v_0，v_1 の型と MT'，Γ' における v_0，v_1 の型は等しい．また，$\tau \neq \tau_1$ の場合は，v_1 にキャストが挿入され，v_1 の型は $\tau \leftharpoonup \tau_1$ 型となる．これは $(\tau \leftharpoonup \tau_1) <:_{TT} \tau_1$ であることから，型として矛盾せず，τ_1 型を必要とするメソッドの実引数に渡すことができる．したがって，MT'，Γ' において，$e'' = e$ は τ_2 型に型付けされる．その型付けにおいて，型環境はメソッド呼び出しから戻るため，MT，Γ に戻る．なお，型付け後の式 e''' は任意である．したがって，$\exists e''', (MT, \Gamma, e) \rightsquigarrow (MT, \Gamma, \tau_2, e''')$ が言えるため，E_Ivk3 により評価される場合は保存を満たす．

E_Ivk4 の場合　$e' = (\langle [m ::: \tau' \to ?] \Leftarrow ? \rangle v_0).m(v_1)$　$e'' = \langle\!\langle ? \Leftarrow \tau_2 \rangle\!\rangle([A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1))$

$$\frac{SV(v_0) = [A] \quad CT(A.m) = \lambda x.e ::: \tau_1 \to \tau_2}{(DT, E, e') \to (DT, E, e'')} \tag{E_Ivk4}$$

$e' = (\langle [m ::: \tau' \to ?] \Leftarrow ? \rangle v_0).m(v_1)$ となるような型付けは，G_Ivk2 のみである．

$$\frac{(MT, \Gamma, v_0) \rightsquigarrow (MT_1, \Gamma_1, ?, v_0') \quad (MT_1, \Gamma_1, v_1) \rightsquigarrow (MT'', \Gamma'', \tau', v_1')}{(MT, \Gamma, v_0.m(v_1)) \rightsquigarrow (MT'', \Gamma'', ?, (\langle [m ::: \tau_1 \to ?] \Leftarrow ? \rangle v_0').m(v_1'))} \tag{G_Ivk2}$$

$(MT, \Gamma, v_0) \rightsquigarrow (MT_1, \Gamma_1, \tau_d, v_0')$ と $(MT_1, \Gamma_1, v_1) \rightsquigarrow (MT'', \Gamma'', \tau_d, v_1')$ と補題 2 より，$MT'' = MT_1 = MT$，$\Gamma'' = \Gamma_1 = \Gamma$，$v_0' = v_0$，$v_1' = v_1$ が言える．E_Ivk3 の場合と同様に，かつ評価において DT と E に変化がないため，$MT' = MT$，$\Gamma' = \Gamma$ となる．また，CT が参照している DT から MT が生成されているため，$TT(A.m) = MT'(A.m) = \tau_1 \to \tau_2$ となる．さらに実引数が $\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1$ であるため，$\tau' = \tau_1$ または $\tau' \neq \tau_1$ のどちらにおいても型は τ_1 型となる．したがって，$[A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1)$ は，G_Ivk1 により，$(MT, \Gamma, [A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1)) \rightsquigarrow (MT, \Gamma, \tau_2, [A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1))$ と型付けできる．$\tau_1 = \tau_2 = ?$ の場合は，$(MT, \Gamma, [A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1)) \rightsquigarrow (MT, \Gamma, \tau_2, [A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1))$ より e'' は $\tau_2 = ?$ 型となる．$\tau_1 \neq ? \wedge \tau_2 \neq ?$ の場合は，G_Cast により，$(MT, \Gamma, \langle ? \Leftarrow \tau_2 \rangle([A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1))) \rightsquigarrow (MT, \Gamma, ?, \langle ? \Leftarrow \tau_2 \rangle([A].m(\langle\!\langle \tau_1 \Leftarrow \tau' \rangle\!\rangle v_1)))$ と型付けされる．

以上より，E_Ivk4 で評価される場合は保存を満たす．

E_Cast の場合　$e' = \langle \tau_d \Leftarrow \tau \rangle e_0'$　$e'' = \langle \tau_d \Leftarrow \tau \rangle e_0''$

$$\frac{(DT, E, e_0') \to (DT', E', e_0'')}{(DT, E, e') \to (DT, E, e'')} \tag{E_Cast}$$

$e' = \langle \tau_d \Leftarrow \tau \rangle e_0'$ となる型付けは G_Cast のみである．

$$\frac{(MT, \Gamma, e_0') \rightsquigarrow (MT'', \Gamma'', \tau, e_0')}{(MT, \Gamma, \langle \tau_d \Leftarrow \tau \rangle e_0') \rightsquigarrow (MT'', \Gamma'', \tau_d, \langle \tau_d \Leftarrow \tau \rangle e_0')} \tag{G_Cast}$$

$(MT, \Gamma, e_0') \rightsquigarrow (MT'', \Gamma'', \tau, e_0')$ と帰納法の仮定により，DT', E' により生成される MT', Γ' において $(MT', \Gamma', e_0'') \rightsquigarrow (MT'', \Gamma'', \tau_d, e_0''')$ と型付けできる e_0''' が存在する．$e_0''' = e_0''$ とすれば，G_Cast により，$(MT, \Gamma, \langle \tau_d \Leftarrow \tau \rangle e_0'') \rightsquigarrow (MT'', \Gamma'', \tau_d, \langle \tau_d \Leftarrow \tau \rangle e_0'')$ と型付けされる．以上より，E_Cast の場合は保存を満たす．

E_Cast_Merge の場合　$e' = \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v \quad e'' = \langle\!\langle (\tau_0 \leftharpoonup \tau_2) \Leftarrow \tau_0 \rangle\!\rangle v$

$$\frac{\tau_0 \neq \tau_2}{(DT, E, e') \to (DT, E, e'')} \qquad \text{(E\_Cast\_Merge)}$$

$e' = \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v$ となる型付けは，G_Cast のみである．

$$\frac{(MT, \Gamma, \langle \tau_1 \Leftarrow \tau_0 \rangle v) \rightsquigarrow (MT'', \Gamma'', \tau_1, \langle \tau_1 \Leftarrow \tau_0 \rangle v)}{(MT, \Gamma, \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v) \rightsquigarrow (MT'', \Gamma'', \tau_2, \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v)} \qquad \text{(G\_Cast)}$$

$(MT, \Gamma, \langle \tau_1 \Leftarrow \tau_0 \rangle v) \rightsquigarrow (MT'', \Gamma'', \tau_1, \langle \tau_1 \Leftarrow \tau_0 \rangle v)$ であることから，$(MT, \Gamma, v) \rightsquigarrow (MT'', \Gamma'', \tau_0, v)$ と型付けでき，さらに補題 2 より，$MT'' = MT$，$\Gamma'' = \Gamma$ が言える．このことから，$(\tau_0 \leftharpoonup \tau_2) \neq \tau_0$ の時，G_Cast により，$(MT, \Gamma, \langle (\tau_0 \leftharpoonup \tau_2) \Leftarrow \tau_0 \rangle v) \rightsquigarrow (MT'', \Gamma'', \tau_0 \leftharpoonup \tau_2, \langle (\tau_0 \leftharpoonup \tau_2) \Leftarrow \tau_0 \rangle v)$ と型付けできる．$(\tau_0 \leftharpoonup \tau_2) <:_{TT} \tau_2$ であるため，E_Cast_Merge で評価する場合は保存を満たす．

E_Cast_Remove の場合　$e' = \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v \quad e'' = v$

$$\frac{\tau_0 = \tau_2}{(DT, E, e') \to (DT, E, e'')} \qquad \text{(E\_Cast\_Merge)}$$

$e' = \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v$ となる型付けは，G_Cast のみである．

$$\frac{(MT, \Gamma, \langle \tau_1 \Leftarrow \tau_0 \rangle v) \rightsquigarrow (MT'', \Gamma'', \tau_1, \langle \tau_1 \Leftarrow \tau_0 \rangle v)}{(MT, \Gamma, \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v) \rightsquigarrow (MT'', \Gamma'', \tau_2, \langle \tau_2 \Leftarrow \tau_1 \rangle \langle \tau_1 \Leftarrow \tau_0 \rangle v)} \qquad \text{(G\_Cast)}$$

$(MT, \Gamma, \langle \tau_1 \Leftarrow \tau_0 \rangle v) \rightsquigarrow (MT'', \Gamma'', \tau_1, \langle \tau_1 \Leftarrow \tau_0 \rangle v)$ であることから，$(MT, \Gamma, v) \rightsquigarrow (MT'', \Gamma'', \tau_0, v)$ と型付けでき，さらに補題 2 より，$MT'' = MT$，$\Gamma'' = \Gamma$ が言える．また，評価において DT，E に変化がないため，$MT' = MT'' = MT$，$\Gamma' = \Gamma'' = \Gamma$ が言える．したがって，$(MT, \Gamma, v) \rightsquigarrow (MT, \Gamma, \tau_0, v)$ により E_Cast_Remove で評価する場合は保存を満たしている．

6　実装

本章では，本論文で提案した型システムの実装の一例を示す．その概観を図 6.1 に示す．

図 6.1 の左上の Ruby プログラムから下に向かっている矢印が，Gradual typing の静的型チェックに関わる処理であり，右に向かっている矢印が動的型チェックに関わる処理である．順序としては，最初に静的型チェックを行い，その後に動的型チェックを行う．

まず，静的型チェックに関わる処理について説明する．最初に，元の Ruby プログラムにユーザが型注釈を挿入する．その際，型注釈を挿入するのは，ユーザが静的型チェックを行いたい部分のみである．次に，型注釈を挿入した Ruby プログラムを元に，パーサにより抽象構文木 (AST) を生成する．その次に，生成した AST に対して，静的型チェックを行う．ここで型エラーがある場合は，プログラムまたは挿入した型注釈が間違っていることを意味するので，間違っている部分をユーザが修正し，修正したプログラムを再びパーサに渡し，AST を生成する．型エラーがなかった場合は静的型チェックが完了したことを意味する．

次に，動的型チェックに関わる処理について説明する．静的型チェックが完了した後，元の Ruby プログラムを既存の Ruby 処理系で実行して動的型チェックを行う．実行の際に型エラーを含むエラーがあれば，それは実行時エラーとなり，エラーがなければ実行結果が得られる．

以上で述べた処理を実現するために実装する必要があるのは，型注釈入り Ruby プログラムを

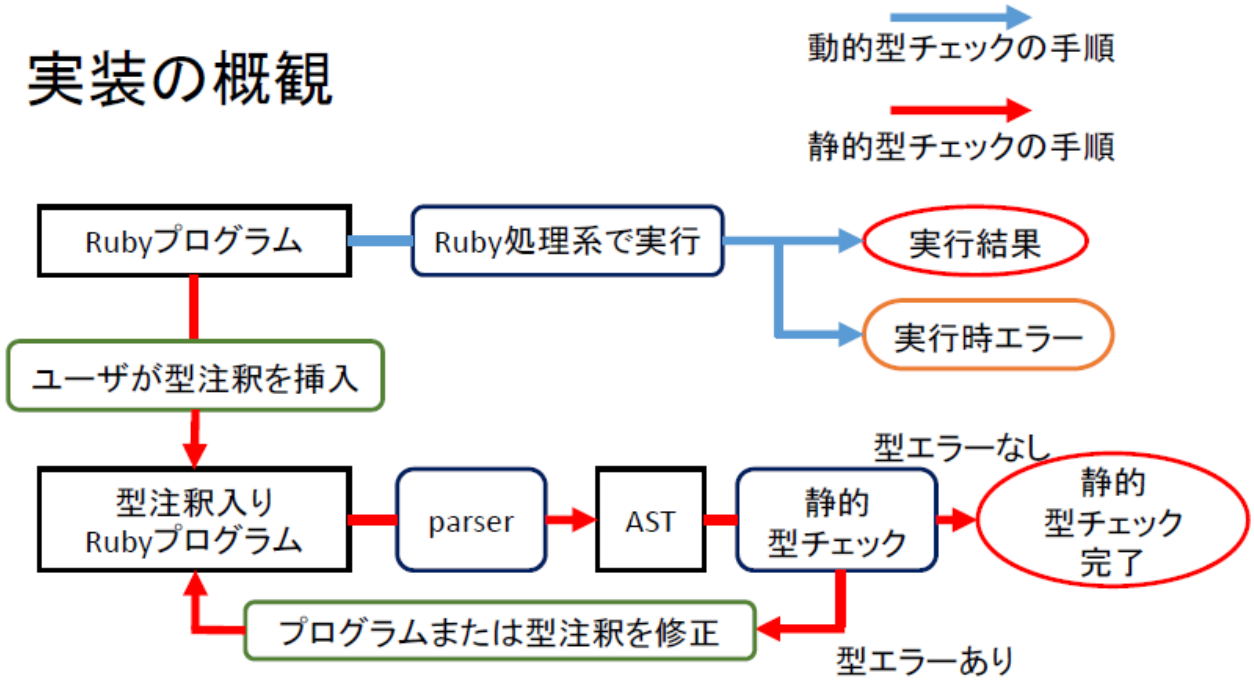

図 6.1　実装の一例

受け付けるパーサと，パーサにより生成された AST に対して静的型チェックを行うようなプログラムである．また，本論文で提案した型システムでは，動的型をキャストにより，実行時に型チェックするようにしていたが，図 6.1 の実装においてはキャストが現れない．それは提案した型システムのキャストエラーは，既存の Ruby 処理系で実行した際の型エラーと同等であるため，動的型チェックを既存の Ruby 処理系に任せているためである．

7 関連研究

Ruby Type Checker [2] は，型注釈に基づく型チェックを行う仕組みを Ruby のライブラリとして実装したものである．型注釈を挿入した部分の型チェックを，Ruby 本来の型チェックとは異なるタイミングで行う．型注釈を挿入したメソッドが呼び出されたときに型チェックを行うため，本来の Ruby よりも早いタイミングで型エラーを発見できる可能性がある．また，型注釈を挿入しなければ本来の Ruby の処理を行うので，型チェックのタイミングを早くしたい部分とそうでない部分をユーザが任意に決定できる．そのため，Gradual typing と性質が似ている．Ruby のライブラリとしての実装のため，型チェックは実行時に行われる．これに対して本研究で導入する Gradual typing は，部分的な静的型チェック・動的型チェックを行う．

Siek ら [3] は，単純型付き λ 計算に対して導入していた Gradual typing [1] を，オブジェクトを扱えるように拡張している．実装方法として，キャスト挿入を行う方法と行わない方法が述べられており，特にキャスト挿入を行う方法について詳しく説明されている．キャスト挿入を行った後のプログラムを意味する中間言語を定義している．また，提案しているオブジェクトを扱える Gradual typing の型健全性の証明も行われている．Ruby はオブジェクト指向言語なので，本研究は，[3] のオブジェクトに対する型付け規則を参考にした．本研究では，オブジェクト指向言語である Ruby に Gradual typing を導入することに加えて，代入式での変数への型注釈やメソッド定義式によるメソッドの追加が行えるように対応させているため，Siek らによるものとは異なる．

既存研究で，様々な言語に Gradual typing が導入されている [5][6][7]．Vitousek ら [5] は，Python に対して，Gradual typing を導入し実装している．キャスト挿入の型付け・評価規則を 2 つ設計しており，従来のものを含めてそれぞれ 3 つの規則に基づいて実装しており，実装したものを評価し比較している．評価した際に，本来の Python では発見できなかった型エラーによるバグをいくつか発見できたとしている．Swamy ら [6] は JavaScript に，Rastogi ら [7] は TypeScript に Gradual typing を導入している．それぞれ，セキュリティや効率を考慮した設計となっている．これら研究においては，メソッド追加と method_missing に対応していない．

8 おわりに

本論文では，Ruby に対して Gradual typing を導入するための基本的な考え方を，Ruby サブセット，型付け規則，評価規則の順に示した．さらに型健全性の証明を行った．本研究で扱う Ruby サブセットは，メソッドの更新，追加が行えるため，Ruby のオープンクラスによるメソッド追加，メソッド更新に対応することができた．

今後の課題は，まず，設計した Ruby サブセットを，本来の Ruby の構文に近づけるために拡張することである．具体的には，配列やハッシュなどの追加，while や case の構文の追加，if 式の型付けの nil ではなく then 節と else 節の型を考慮するようにすることなどが例として挙げられる．また，Ruby の特徴の 1 つである method_missing への対応も課題となる．これについては，本論文におけるメソッド定義とメソッド呼び出しの型付け規則や評価規則を拡張することで対応できると考えられる．

最後に，本論文で示した考え方，またはそれを拡張したものを元に，実装とその評価を行うことである．それにより，Ruby に対して Gradual typing を導入することの有用性を立証できる．

謝辞

本研究を行うにあたり，終始変わらぬご指導を賜りました岩崎英哉教授並びに中野圭介准教授に深く感謝いたします．また，研究を進めるにあたって熱心なご指導をいただきました岩崎研究室，中野研究室の皆様に心から御礼申し上げます．

参考文献

[1] Jeremy G. Siek and Walid Taha. Gradual Typing for Functional Languages. *Proceedings of Scheme and Functional Programming 2006* (*SFP 2006*), *pp 81–92, 2006*

[2] Brianna M. Ren, John Toman, T. Stephen Strickland and Jeffrey S. Foster. The Ruby Type Checker. *Proceedings of the 28th Annual ACM Symposium on Applied Computing 2013* (*SFP 2013*), *pp 1565–1572, 2013*

[3] Jeremy G. Siek and Walid Taha. Gradual Typing for Objects. *Proceedings of the 21st European Conference on Object-Oriented Programming 2007* (*ECOOP 2007*), *pp 2–27, 2007*

[4] Brianna M. Ren, and Jeffrey S. Foster. Just-in-Time Static Type Checking for Dynamic Languages. *Proceeding of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (*PLDI 2016*), *pp 462–476, 2016*

[5] Michael M. Vitousek, Andrew M. Kent, Jeremy G. Siek, and Jim Baker. Design and Evaluation of Gradual Typing for Python. *Proceedings of the 10th ACM Symposium on Dynamic Languages* (*DLS 2014*), *pp 45–56, 2014*

[6] N. Swamy, C. Fournet, A. Rastogi, K. Bhargavan, J. Chen, P.-Y. Strub, and G. Bierman. Gradual Typing Embedded Securely in JavaScript. *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (*POPL 2014*), *pp 425–437, 2014*

[7] A. Rastogi, N. Swamy, C. Fournet, G. Bierman, P. Vekris. Safe & Efficient Gradual Typing for TypeScript. *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (*POPL 2015*), *pp 167–180, 2015*