高階契約に対するトレース意味論の完全抽象性

井上 鉄也 [1], 中澤 巧爾 [1]

[1] 名古屋大学大学院情報学研究科

{ inoue.tetsuya@c.mbox, knak@i } .nagoya-u.ac.jp

概 要　高階契約は Findler らが提示した高階関数に対する契約検査である．高階契約は複雑であるが故に blame 割り当ての正しさの判断が難しいという問題がある．村井らはこの問題に対し，トレース意味論による解決策を提示した．村井らのトレース意味論は，あるモジュールに対し，自身とそれ以外のモジュールとの間における相互作用の列の集合（トレース集合）をその意味として与える．これにより契約検査の本質が見て取りやすいものとなっている．

本論文は，村井らの結果を基にしたトレース意味論とその完全抽象性の証明を与える．本論文の体系では，自モジュールの関数に対しては契約の検査をしないよう変更することにより，モジュールを契約主体の単位とする Racket 等の実際のプログラミング言語により近いものとしている．

1　はじめに

プログラミング言語にはプログラマがソフトウェアを作成するにあたって役に立つ様々な機能が用意されており,「ソフトウェア契約 (software contracts)」（以下単に「契約」と呼ぶ）はそのような機能の一つである．契約は関数を呼び出すモジュールが満たすべき（引数が満たすべき）制約と，関数を実装するモジュールが満たすべき（返値が満たすべき）制約の 2 つからなり，プログラマはモジュール内で実装した関数の仕様として契約を関数に与えることができる．契約の検査は実行時に行われ，引数に関する契約に違反があった場合は呼び出したモジュールを，返値に関する契約に違反があった場合は実装するモジュールを blame と呼ばれる例外アクションでプログラマに知らせる．この契約検査によってプログラマは単純な型情報だけでは分からないバグを見つけることができる．契約を実装しているプログラミング言語には Eiffel や Racket といったものがある．

契約の概念はもともと引数に関数をとったり，返値として関数を返したりする高階関数に対してのものは無かったが，Findler らによって高階関数に対する契約検査が提示された [2]．高階関数に関しては，blame 割り当ての正しさの判断が複雑であるため，その正当性を形式的に議論することが難しい．この問題を解決するために，村井らは高階契約の体系とそのトレース意味論を与えている [1]．トレース意味論とは，プログラムの意味をプログラム自身とそのプログラムが使われる文脈との間の可能な相互作用を表すイベント列（トレース）の集合として与えるものである．村井らはそれに加え，モジュール間で行われる実行時契約検査を明示的に表現することで，契約検査の本質を見て取りやすい意味論としている．しかし，この体系の完全抽象性は証明されていない．

本論文は村井らの体系を基に与えたトレース意味論の完全抽象性を証明する．このとき，Racket 等の実際のプログラミング言語で利用されている契約検査により近づけるべく，自モジュール内の関数呼び出しについては契約の監視を行わないように変更する．

2 章では契約を高階関数に対しても与えることのできる高階プログラミング言語の構文と操作的意味論を与える．3 章では各モジュールのふるまいを表すラベル付き遷移系 (LTS) を与える．4 章では LTS により得られるトレース集合によってトレース意味論を与える．5 章では与えた意味論が完全抽象性をもつことを証明する．最後 6 章で本論文のまとめと今後の課題を述べる．本論文の一部の詳細な定義や証明を [5] に与える．

2 高階契約をもつプログラミング言語

まずは高階関数に対する契約監視機構をもつプログラミング言語を与える．言語の構文を図 1 に，操作的意味論を図 2 に示す．村井ら [1] の体系との主な違いは，契約に対する責任を負う単位としてモジュールを使用し，同モジュール内の関数呼出しについては契約の監視を行っていない点である．

ここでは簡単のために，モジュールを 2 つに限定し，基底型は自然数型のみとする．モジュール m に対しその相手のモジュールを $\overline{m}$ により示し，$\overline{\overline{m}} = m$ とする．M_{all} はモジュール全体，μ はモジュールの内部変数を格納するストア（格納できるのは基底型の値のみ）である．つまりこの 2 つ（M_{all} と μ）で実行環境を示す．$e[v/x]$ は式 e の変数 x に値 v を代入したものを示す．$M + (m.f/c = i)$ は M に対し，関数 f を契約 c，実装 i で追加することを意味する．このとき追加される関数名と元から M が持つ関数名に重複は無いものとする．$FName$ は関数名，$AName$ は内部変数名，$Base$ は基底型を示す．

m	$\in$	$MName$	$::=$	$m \mid \overline{m}$	(モジュール名)
		op	$::=$	$+ \mid *$	(算術演算)
c	$\in$	$Contracts$	$::=$	$P \mid c \Rightarrow c$	(関数に付される契約)
v	$\in$	Val	$::=$	$\underline{n} \mid m.f$	(値)
e	$\in$	Exp	$::=$	$x \mid v \mid e\ op\ e \mid ee \mid a := e; e \mid !a$	(式)
				$\mid \text{case } e \text{ of } 0 \mapsto e; S(x) \mapsto e \mid [e]^c_m$	
i	$\in$	$Impl$	$::=$	$m.f \mid \lambda\vec{x}.e$	(関数の実装)
M	$\in$	$MName \longrightarrow FName \longrightarrow Contracts \times Impl$			(モジュール定義)
μ	$\in$	$MName \longrightarrow AName \longrightarrow Base$			(ストア)
E	$\in$	Env	$::=$	$[] \mid E\ op\ e \mid \underline{n}\ op\ E \mid \text{case } E \text{ of } 0 \mapsto e; S(x) \mapsto e$	(評価文脈)
				$\mid Ee \mid (m.f)E \mid a := E; e$	

図 1. 言語の構文

プログラミング言語の大まかな構成と特徴を以下より記述する．モジュールは λ 式を基調とした関数の集合である．モジュール m の関数 f を $m.f$ のように表現する．契約検査が絡む関数適用のうち引数が基底型の場合は簡約規則 (AppN) により分類する．$\text{AppN}^m_{m'}(M_m(f), \underline{n}, M_{all}, \mu)$ はモジュール m' がモジュール m の関数 f を引数 $\underline{n}$ で呼んだ結果を表す．$M_m(f)$ は関数 f の契約と実装の組である．また $[e]^c_m$ は式 e の結果得られる値が契約 c を満たすことの責任をモジュール m が負うことを意味する (Res)．なお契約を満たしていない場合は blame（$\Uparrow_m$）により即座に計算を終了する．引数が関数である関数適用の場合は簡約規則 (AppF) により分類する．AppF と AppN には引数が関数か自然数かの違いしかない．関数適用の規則 (AppN or F) が村井ら [1] の体系と比べて複雑なのは，関数の実装の違いによる分岐に加え，同モジュールどうしなら契約を無視する処理を追加したためである．このプログラミング言語の実行例は [5] に与える．

このプログラミング言語は村井ら [1] の体系と同様に型システムを持ち，型健全性を満たす（詳細は [5] を参照）．モジュール m において式 e が型 τ をもつことを

$$\langle M, \Sigma, \Gamma, S \rangle \vdash^m e : \tau$$

で表す．ここで，M はモジュール定義，Σ は契約の型情報，Γ は式 e に含まれる自由変数の型情報，S はストア内の値の型情報を示しており，

$$\Sigma \in MName \longrightarrow FName \longrightarrow Contracts \quad \text{(契約の型情報)}$$
$$S \in MName \longrightarrow AName \longrightarrow Base \quad \text{(ストアの型情報)}$$

である．M のすべての関数について正しく型付けできることを以下で表す．

$$\langle \Sigma, S \rangle \vdash \langle M, \mu \rangle : ok$$

$$\boxed{\langle e, M, \mu\rangle \rightarrow_m \langle e', M', \mu'\rangle}$$

$$\frac{n_1 \ op\ n_2 = n_3}{\langle \underline{n_1}\ op\ \underline{n_2}, M_{all}, \mu\rangle \rightarrow_m \langle \underline{n_3}, M_{all}, \mu\rangle}\ (\textsc{Arith})$$

$$\frac{}{\langle \text{case } \underline{0} \text{ of } 0 \mapsto e; S(x) \mapsto e', M_{all}, \mu\rangle \rightarrow_m \langle e, M_{all}, \mu\rangle}\ (\textsc{Case-0})$$

$$\frac{}{\langle \text{case } \underline{n+1} \text{ of } 0 \mapsto e; S(x) \mapsto e', M_{all}, \mu\rangle \rightarrow_m \langle e'[\underline{n}/x], M_{all}, \mu\rangle}\ (\textsc{Case-S})$$

$$\frac{}{\langle (m.f)\underline{n}, M_{all}, \mu\rangle \rightarrow_{m'} \text{AppN}^m_{m'}(M_m(f), \underline{n}, M_{all}, \mu)}\ (\textsc{AppN})$$

$$\text{AppN}^m_{m'}(\langle P \Rightarrow c, \lambda x\vec{y}.e\rangle, \underline{n}, M_{all}, \mu) = \langle m.g, M_{all} + (m.g/c = \lambda\vec{y}.e[\underline{n}/x]), \mu\rangle\ (m = m' \text{ or } n \in P)$$
$$\text{AppN}^m_m(\langle P \Rightarrow c, \lambda x.e\rangle, \underline{n}, M_{all}, \mu) = \langle e[\underline{n}/x], M_{all}, \mu\rangle$$
$$\text{AppN}^m_{\overline{m}}(\langle P \Rightarrow c, \lambda x.e\rangle, \underline{n}, M_{all}, \mu) = \langle [e[\underline{n}/x]]^c_m, M_{all}, \mu\rangle\ (n \in P)$$
$$\text{AppN}^m_m(\langle P \Rightarrow c, m'.g\rangle, \underline{n}, M_{all}, \mu) = \langle (m'.g)\underline{n}, M_{all}, \mu\rangle$$
$$\text{AppN}^m_{\overline{m}}(\langle P \Rightarrow c, m'.g\rangle, \underline{n}, M_{all}, \mu) = \langle [(m'.g)\underline{n}]^c_m, M_{all}, \mu\rangle\ (n \in P)$$
$$\text{AppN}^m_{\overline{m}}(\langle P \Rightarrow c, i\rangle, \underline{n}, M_{all}, \mu) = \Uparrow_{\overline{m}}\ (n \notin P)$$

$$\frac{}{\langle (m.f)(m''.h), M_{all}, \mu\rangle \rightarrow_{m'} \text{AppF}^m_{m'}(M_m(f), m''.h, M_{all}, \mu)}\ (\textsc{AppF})$$

$$\begin{aligned}
&\text{AppF}^m_{m'}(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, \lambda x\vec{y}.e\rangle, m''.h, M_{all}, \mu) = \\
&\qquad \langle m.g, M_{all} + (m.g/c_3 = \lambda\vec{y}.e[m'.h'/x]) + (m'.h'/c_1 \Rightarrow c_2 = m''.h), \mu\rangle \\
&\text{AppF}^m_m(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, \lambda x.e\rangle, m''.h, M_{all}, \mu) = \\
&\qquad \langle e[m.h'/x], M_{all} + (m.h'/c_1 \Rightarrow c_2 = m''.h), \mu\rangle \\
&\text{AppF}^m_{\overline{m}}(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, \lambda x.e\rangle, m''.h, M_{all}, \mu) = \\
&\qquad \langle [e[\overline{m}.h'/x]]^{c_3}_m, M_{all} + (\overline{m}.h'/c_1 \Rightarrow c_2 = m''.h), \mu\rangle \\
&\text{AppF}^m_m(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, m'.g\rangle, m''.h, M_{all}, \mu) = \\
&\qquad \langle (m'.g)(m.h'), M_{all} + (m.h'/c_1 \Rightarrow c_2 = m''.h), \mu\rangle \\
&\text{AppF}^m_{\overline{m}}(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, m'.g\rangle, m''.h, M_{all}, \mu) = \\
&\qquad \langle [(m'.g)(\overline{m}.h')]^{c_3}_m, M_{all} + (\overline{m}.h'/c_1 \Rightarrow c_2 = m''.h), \mu\rangle
\end{aligned}$$

$$\frac{n \in P}{\langle [\underline{n}]^P_{m'}, M_{all}, \mu\rangle \rightarrow_m \langle \underline{n}, M_{all}, \mu\rangle}\ (\textsc{ResN}) \qquad \frac{n \notin P}{\langle [\underline{n}]^P_{m'}, M_{all}, \mu\rangle \rightarrow_m \Uparrow_{m'}}\ (\textsc{ResB})$$

$$\frac{}{\langle [m''.f]^c_{m'}, M_{all}, \mu\rangle \rightarrow_m \langle m'.f', M_{all} + (m'.f'/c = m''.f), \mu\rangle}\ (\textsc{ResF})$$

$$\frac{}{\langle a := \underline{n}; e, M_{all}, \mu\rangle \rightarrow_m \langle e, M_{all}, \mu + (m.a = \underline{n})\rangle}\ (\textsc{Assg})$$

$$\frac{}{\langle !a, M_{all}, \mu\rangle \rightarrow_m \langle \mu(m.a), M_{all}, \mu\rangle}\ (\textsc{Deref})$$

$$\frac{\langle e, M_{all}, \mu\rangle \rightarrow_{m'} \langle e', M'_{all}, \mu'\rangle}{\langle [e]^c_{m'}, M_{all}, \mu\rangle \rightarrow_m \langle [e']^c_{m'}, M'_{all}, \mu'\rangle}\ (\textsc{CBra}) \qquad \frac{\langle e, M_{all}, \mu\rangle \rightarrow_{m'} \Uparrow_{m''}}{\langle [e]^c_{m'}, M_{all}, \mu\rangle \rightarrow_m \Uparrow_{m''}}\ (\textsc{CBraB})$$

$$\frac{\langle e, M_{all}, \mu\rangle \rightarrow_m \langle e', M'_{all}, \mu'\rangle}{\langle E[e], M_{all}, \mu\rangle \rightarrow_m \langle E[e'], M'_{all}, \mu'\rangle}\ (\textsc{CCon}) \qquad \frac{\langle e, M_{all}, \mu\rangle \rightarrow_m \Uparrow_{m'}}{\langle E[e], M_{all}, \mu\rangle \rightarrow_m \Uparrow_{m'}}\ (\textsc{CConB})$$

図 2. 操作的意味論

ストアの型情報 S は自然数しか無いため冗長である．しかし今後基底型を拡張した場合を考慮し，自然数でしか扱えない議論となることを回避するために与えている．

型システムの詳細は [5] に与える．

3 ラベル付遷移系 (LTS)

この章ではラベル付遷移系 (LTS) について記述する．ここで与える LTS は 2 章の言語を基にしたものである．本論文のトレース意味論は異なるモジュール間の相互作用のアクションの列（トレース）によってモジュールに意味を与えるものであり，LTS はそのトレースを取り出す．

始めに相互作用するモジュールをコンポーネントとして定義する．次に LTS の構文と遷移規則を与える．LTS における状態をコンフィギュレーションと呼ぶ．これはコンポーネントと計算中の式を合わせたものである．最後に計算中の式の分割について説明する．

3.1 コンポーネント

LTS を与えるために必要なコンポーネントを定義する．

定義 3.1 (コンポーネント)
コンポーネントは 4 つ組 $\langle M, \mu, O, I\rangle$ である．ここで，M はモジュール定義，μ はストア，O と I は関数名とその契約の組の集合である．

コンポーネントはモジュール M に他のモジュールとの相互作用に必要な情報を加えたものである．直観的には O は外部モジュールに提供する（エクスポートする）関数の契約の集まり，I は外部モジュールから提供される（インポートする）関数の契約の集まりである．

定義 3.2 (コンポーネントの整合性)
以下の条件を満たすコンポーネント $\langle M, \mu, O, I\rangle$ は整合的である．

1. ある S に対し，$\langle I, S\rangle \vdash \langle M, \mu\rangle : ok$
2. $O \subseteq \mathit{Intf}(M)$

$\mathit{Intf}(M)$ はモジュール M が定義するすべての関数の契約の集合である．

以降コンポーネントといえば整合性を満たしているものとする．

定義 3.3 (コンポーネントの結合可能性)
以下の条件を満たす 2 つのコンポーネント $\langle M_1, \mu_1, O_1, I_1\rangle, \langle M_2, \mu_2, O_2, I_2\rangle$ は結合可能である．

$$I_1 \subseteq O_2 \qquad I_2 \subseteq O_1$$

また，このとき 2 つのコンポーネントの結合を以下のように定義する．結合の結果得られるのはモジュール定義とストアの 2 つ組である．

$$\langle M_1, \mu_1, O_1, I_1\rangle \curlywedge \langle M_2, \mu_2, O_2, I_2\rangle = \langle M_1 \cup M_2, \mu_1 \cup \mu_2\rangle$$

互いに異なるモジュールを基にしたコンポーネントどうしを結合することで，モジュールが 2 つであるシステム全体の環境となる．

以降ではメタ変数 K などによってコンポーネント $\langle M, \mu, O, I\rangle$ を表現する

3.2 LTS の構文と遷移規則

LTS の構文を図 3 に，遷移規則を図 4 示す．LTS の状態をコンフィギュレーションと呼び，コンフィギュレーションはプログラムコンフィギュレーション，環境コンフィギュレーション，ブレームコンフィギュレーションの 3 種類に分けられる．プログラムコンフィギュレーションは計算されるべき式 e と，実行時スタック $\mathcal{E}$，自コンポーネントの 4 つ組の計 6 つ組からなり，自モジュール内で計算が行われている状態を示すコンフィギュレーションである．環境コンフィギュレーションは実行時スタック $\mathcal{E}$ と自コンポーネントの計 5 つ組からなり，外部コンポーネントに計算の制御が移っている状態を示す．ブレームコンフィギュレーションは名前の通り blame が発生した状態を示す．$\mathcal{E}$ は実行時スタックで，契約 c により添字つけられた評価文脈 E^c の有限列 $E^c : ... : E'^{c'}$ が収められている．E は外部コンポーネントの関数を呼び出す際に計算していた式 e の評価文脈，c は呼び出した関数の返り値が満たすべき契約である．スタックの底は右端とする．コンフィギュレーション間の遷移はラベルであるアクション (act) によって起こる（図 3）．$oact$ はプログラムコンフィギュレーションから環境コンフィギュレーションへの遷移，つまり自コンポーネントから外部コンポーネントに制御が移るアクションを意味する．ただし blame が発生する場合には環境コンフィギュレーションではなくブレームコンフィギュレーションへと遷移する (OBLAME)．$iact$ は $oact$ の逆である．こちらも同様にプログラムコンフィギュレーションではなくブレームコンフィギュレーションへと遷移する場合もある (IBLAME)．τ アクションはプログラムコンフィギュレーションからプログラムコンフィギュレーションへの遷移，つまり自コンポーネントで完結するトレース集合に含まれないアクションを意味する (TAU)．$oact, iact$ の call は関数呼び出し，return は値返しのアクションである．このモジュールをまたぐアクションが LTS において重要な役割を持つ．元となるプログラミング言語においてブラケット（$[e]^c_m$）を式に加える，または取り除く操作が，モジュールをまたぐアクションに対応している．LTS ではこのブラケットは式に対して高々1 つ，それも一番外側にしか存在しない．ブラケットを持たないのは，開始時の式，引数が 2 つ以上の関数呼びに対しての式の 2 つだけである．開始時の式は関数ではなく契約を持たないためブラケットが必要ない．引数が 2 つ以上の関数呼びについては，返す値は 2 つ目の引数を受け取るためだけの関数であるため，契約はなくブラケットが必要ない．同時に，call された時点で名前付けは完了している (ICALL の 1) ため return する際新たに名前付けする必要もないので，通常の関数値返し (ORET-F2) と区別するためにもブラケットは付けない (ORET-F1)．LTS の実行例は [5] に与える．

$$
\begin{array}{lcll}
C & ::= & \langle e, \mathcal{E}, M, \mu, O, I \rangle_m & \text{(プログラムコンフィギュレーション)} \\
 & \mid & \langle \mathcal{E}, M, \mu, O, I \rangle_m & \text{(環境コンフィギュレーション)} \\
 & \mid & \Uparrow_m & \text{(ブレームコンフィギュレーション)} \\
oact & ::= & \overline{\text{call}}\ \overline{m}.f(\underline{n}) \mid \overline{\text{call}}\ \overline{m}.f(m.g) & \\
 & \mid & \overline{\text{return}}\ \underline{n} \mid \overline{\text{return}}\ m.g \mid \overline{\text{blame}}\ m & \\
iact & ::= & \text{call}\ m.f(\underline{n}) \mid \text{call}\ m.f(\overline{m}.g) & \\
 & \mid & \text{return}\ \underline{n} \mid \text{return}\ \overline{m}.g \mid \text{blame}\ \overline{m} & \\
extact & ::= & oact \mid iact & \\
act & ::= & \tau \mid extact & \\
O & \in & MName \longrightarrow FName \longrightarrow Contracts & \\
I & \in & MName \longrightarrow FName \longrightarrow Contracts &
\end{array}
$$

図 3. コンフィギュレーションとそのアクションの構文

$$\frac{\langle e, M_m, \mu\rangle \rightarrow_m \langle e', M'_m, \mu'\rangle \qquad e \not\equiv [\underline{n}]^c_{m'}, [m''.f]^c_{m'}}{\langle e, \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\tau} \langle e', \mathcal{E}, M'_m, \mu', O, I\rangle_m} \text{(TAU)}$$

$$\frac{(\overline{m}.f/P \Rightarrow c) \in I \qquad n \in P}{\langle E[(\overline{m}.f)\underline{n}], \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\overline{\text{call}}\ \overline{m}.f(\underline{n})} \langle E^c : \mathcal{E}, M_m, \mu, O, I\rangle_m} \text{(OCALL-N)}$$

$$\frac{(\overline{m}.f/P \Rightarrow c) \in I \qquad n \notin P}{\langle E[(\overline{m}.f)\underline{n}], \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\overline{\text{blame}}\ m} \Uparrow_m} \text{(OBLAME-C)}$$

$$\frac{(\overline{m}.f/(c_1 \Rightarrow c_2) \Rightarrow c_3) \in I}{\begin{array}{l}\langle E[(\overline{m}.f)(m'.g)], \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\overline{\text{call}}\ \overline{m}.f(m.g')} \\ \qquad \langle E^{c_3} : \mathcal{E}, M_m + (m.g'/c_1 \Rightarrow c_2 = m'.g), \mu, O + (m.g'/c_1 \Rightarrow c_2), I\rangle_m\end{array}} \text{(OCALL-F)}$$

$$\frac{n \in P}{\langle [\underline{n}]^P, \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\overline{\text{return}}\ \underline{n}} \langle \mathcal{E}, M_m, \mu, O, I\rangle_m} \text{(ORET-N)}$$

$$\frac{n \notin P}{\langle [\underline{n}]^P, \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\overline{\text{blame}}\ m} \Uparrow_m} \text{(OBLAME-R)}$$

$$\frac{(m.f/c = impl) \in M_m}{\langle m.f, \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\overline{\text{return}}\ m.f} \langle \mathcal{E}, M_m, \mu, O + (m.f/c), I\rangle_m} \text{(ORET-F1)}$$

$$\frac{}{\langle [m'.f]^c, \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\overline{\text{return}}\ m.f'} \langle \mathcal{E}, M_m + (m.f'/c = m'.f), \mu, O + (m.f'/c), I\rangle_m} \text{(ORET-F2)}$$

$$\frac{(m.f/P \Rightarrow c) \in O \qquad n \in P}{\langle \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\text{call}\ m.f(\underline{n})} \text{ICallN}(M_m(f), \underline{n}, M_m, \mu, O, I)} \text{(ICALL-N)}$$

$$\text{ICallN}(\langle P \Rightarrow c, \lambda x\vec{y}.e\rangle, \underline{n}, M_m, O, I) = \langle m.g, \mathcal{E}, M_m + (m.g/c = \lambda\vec{y}.e[\underline{n}/x]), \mu, O, I\rangle_m$$
$$\text{ICallN}(\langle P \Rightarrow c, \lambda x.e\rangle, \underline{n}, M_m, O, I) = \langle [e[\underline{n}/x]]^c, \mathcal{E}, M_m, \mu, O, I\rangle_m$$
$$\text{ICallN}(\langle P \Rightarrow c, m'.g\rangle, \underline{n}, M_m, O, I) = \langle [m'.g(\underline{n})]^c, \mathcal{E}, M_m, \mu, O, I\rangle_m$$

$$\frac{(m.f/(c_1 \Rightarrow c_2) \Rightarrow c_3) \in O}{\langle \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\text{call}\ m.f(\overline{m}.h)} \text{ICallF}(M_m(f), \overline{m}.h, M_m, \mu, O, I)} \text{(ICALL-F)}$$

$$\begin{array}{l}\text{ICallF}(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, \lambda x\vec{y}.e\rangle, \overline{m}.h, M_m, O, I) = \\ \qquad \langle m.g, \mathcal{E}, M_m + (m.g/c_3 = \lambda\vec{y}.e[\overline{m}.h/x]), \mu, O, I'\rangle_m\end{array}$$
$$\text{ICallF}(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, \lambda x.e\rangle, \overline{m}.h, M_m, O, I) = \langle [e[\overline{m}.h/x]]^{c_3}, \mathcal{E}, M_m, \mu, O, I'\rangle_m$$
$$\text{ICallF}(\langle (c_1 \Rightarrow c_2) \Rightarrow c_3, m'.g\rangle, \overline{m}.h, M_m, O, I) = \langle [m'.g(\overline{m}.h)]^{c_3}, \mathcal{E}, M_m, \mu, O, I'\rangle_m$$

※ $I' = I + (\overline{m}.h/c_1 \Rightarrow c_2)$

$$\frac{n \in P}{\langle E^P : \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\text{return}\ \underline{n}} \langle E[\underline{n}], \mathcal{E}, M_m, \mu, O, I\rangle_m} \text{(IRET-N)}$$

$$\frac{}{\langle E^{c_1 \Rightarrow c_2} : \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\text{return}\ \overline{m}.f} \langle E[\overline{m}.f], \mathcal{E}, M_m, \mu, O, I + (\overline{m}.f/c_1 \Rightarrow c_2)\rangle_m} \text{(IRET-F)}$$

$$\frac{P \neq N}{\langle E^P : \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\text{blame}\ \overline{m}} \Uparrow_{\overline{m}}} \text{(IBLAME-R)} \qquad \frac{\exists (m.f/P \Rightarrow c) \in O \qquad P \neq N}{\langle \mathcal{E}, M_m, \mu, O, I\rangle_m \xrightarrow{\text{blame}\ \overline{m}} \Uparrow_{\overline{m}}} \text{(IBLAME-C)}$$

図 4. LTS の遷移規則

3.3　コンフィギュレーションの結合

コンフィギュレーション C はコンポーネント K にそれが扱う式を追加したものである．コンポーネント K の整合性を用いてコンフィギュレーション C の整合性を定義する．この定義についてスタック内の評価文脈は考慮されていないが，これは開始時スタックは空であること，開始時の式に型が付き，型健全性より開始時の型は保持されたままスタックに積まれるため結果としてスタックに積まれている評価文脈にも型が付くためである．つまり開始時コンフィギュレーション C に整合性があるなら遷移しても整合性は保たれる，

定義 3.4 (コンフィギュレーションの整合性)
コンポーネント $K = \langle M, \mu, O, I \rangle$ が整合的ならば，環境コンフィギュレーション $\langle \mathcal{E}, K \rangle_m$ は整合的である．またプログラムコンフィギュレーション $\langle e, \mathcal{E}, K \rangle_m$ については，整合的なコンポーネントにおいてあるストア型付け S に対して $\langle M, I, \emptyset, S \rangle \vdash^m e : \tau$ であれば整合的である．

コンフィギュレーション C の K 以外の部分である式とスタックの結合について定義する．プログラムコンフィギュレーションと環境コンフィギュレーションのスタックに積まれている式の数と対応する位置の契約さえ合っていれば，スタックと式は結合可能である．

定義 3.5 (スタックの結合)
環境コンフィギュレーションのスタック $\langle \mathcal{E}_2 \rangle_{m_2}$，プログラムコンフィギュレーションの式とスタックの組 $\langle e, \mathcal{E}_1 \rangle_{m_1}$ の結合 $\langle e, \mathcal{E}_1 \rangle_{m_1} \mathbin{\wedge\!\!\!\wedge} \langle \mathcal{E}_2 \rangle_{m_2}$ を以下で帰納的に定義する．

- $\langle e, [] \rangle_{m_1} \mathbin{\wedge\!\!\!\wedge} \langle [] \rangle_{m_2} = e$（$e$ は m_1 が実行する式）
- $\langle [e]^c, \mathcal{E}_1 \rangle_{m_1} \mathbin{\wedge\!\!\!\wedge} \langle E^c : \mathcal{E}_2 \rangle_{m_2} = \langle E[[e]^c_{m_1}], \mathcal{E}_2 \rangle_{m_2} \mathbin{\wedge\!\!\!\wedge} \langle \mathcal{E}_1 \rangle_{m_1}$
- $\langle m_1.f, \mathcal{E}_1 \rangle_{m_1} \mathbin{\wedge\!\!\!\wedge} \langle E^c : \mathcal{E}_2 \rangle_{m_2} = \langle E[m_1.f], \mathcal{E}_2 \rangle_{m_2} \mathbin{\wedge\!\!\!\wedge} \langle \mathcal{E}_1 \rangle_{m_1}$

ただし，両者のスタックがちょうど空になり全体が式の形となる場合にのみ $\langle e, \mathcal{E}_1 \rangle_{m_1}$ と $\langle \mathcal{E}_2 \rangle_{m_2}$ は結合可能であるという．

K の結合（定義 3.3）とスタックの結合（定義 3.5）によってコンフィギュレーション C の結合を定義する．ただしスタックの結合については結合後の式に型が付くことが求められる．

定義 3.6 (コンフィギュレーションの結合可能性)
整合的なプログラムコンフィギュレーション $C_1 = \langle e, \mathcal{E}_1, K_1 \rangle_{m_1}$ と整合的な環境コンフィギュレーション $C_2 = \langle \mathcal{E}_2, K_2 \rangle_{m_2}$ は以下の条件を満たすなら結合可能である．

1. 2 つのコンポーネント $K_1 = \langle M_1, \mu_1, O_1, I_1 \rangle$ と $K_2 = \langle M_2, \mu_2, O_2, I_2 \rangle$ が結合可能
2. $\langle e, \mathcal{E}_1 \rangle_{m_1}$ と $\langle \mathcal{E}_2 \rangle_{m_2}$ が結合可能（得られる式を実行するモジュールを m_i）で，
   $\mu_1 \cup \mu_2 = dom(S)$ とおくと

$$\langle M_1 \cup M_2, \emptyset, \emptyset, S \rangle \vdash^{m_i} \langle e, \mathcal{E}_1 \rangle_{m_1} \mathbin{\wedge\!\!\!\wedge} \langle \mathcal{E}_2 \rangle_{m_2} : \tau$$

また，このとき 2 つのコンフィギュレーションの結合を以下のように定義する．

$$C_1 \mathbin{\wedge\!\!\!\wedge} C_2 = \langle \langle e, \mathcal{E}_1 \rangle_{m_1} \mathbin{\wedge\!\!\!\wedge} \langle \mathcal{E}_2 \rangle_{m_2}, K_1 \mathbin{\wedge\!\!\!\wedge} K_2 \rangle$$

ブレームコンフィギュレーションどうしであれば，blame 対象のモジュール名が同じであれば結合可能であり，blame されたモジュール名を m とおくと以下のように定義する．

$$C_1 \mathbin{\wedge\!\!\!\wedge} C_2 = \Uparrow_m$$

4　トレース意味論

LTS より得られるトレースの集合をモジュールの意味として与えるのがトレース意味論である．以降よりトレースを取り出す方法について記述する．ここで，トレースを取り出すのに必要な情報はコンポーネントの 4 つ組（定義 3.1）であるため，実際にはコンポーネントに意味を与える．

トレースとは自身以外のモジュールとの相互作用のアクションの列である．LTS では相互作用のアクションに該当しないものを τ アクションとするので，LTS の遷移からトレースを取り出す方法は図 5 に示すように τ アクションを取り除くだけである．表現上アクションによる遷移は $\rightarrow$ で示し，トレース（複数のアクション）による遷移は $\Rightarrow$ で示す．null は空列を意味する．

$$\frac{}{C \overset{\text{null}}{\Longrightarrow} C} \qquad \frac{C \overset{s_1}{\rightarrow} C' \overset{s_2}{\rightarrow} C''}{C \overset{s_1 s_2}{\Longrightarrow} C} \qquad \frac{C \overset{\tau}{\rightarrow} C'}{C \overset{\text{null}}{\Longrightarrow} C'} \qquad \frac{C \xrightarrow{extact} C'}{C \overset{extact}{\Longrightarrow} C'}$$

図 5. 外部遷移の列（トレース）

コンポーネントのトレース集合を定義する．トレース集合に含まれるトレースはすべて環境コンフィギュレーションから始まるものとする．これはプログラムコンフィギュレーションから始まる場合，相手モジュールの関数呼びでトレースは始まるが，これ自体に自身のモジュールの性質は関係無く余分でしかないためである．

コンフィギュレーションのスタックが空になり，かつ式を持ってない状態を「計算が完全に終了した状態」と表現すると，コンフィギュレーションが「計算が完全に終了した状態」へと遷移する際のトレースを完全トレース（定義 4.1），開始時の環境コンフィギュレーションから始まるすべての完全トレースの集合をトレース集合とする（定義 4.2）．なお，新たに名前付けされたフレッシュな関数がトレース上に存在する場合，名前自体に意味はないのでその関数を名前変えしただけのトレースについては同一視する．

定義 4.1 (完全トレース)
整合的なコンフィギュレーション C, C' とトレース s について，$C \overset{s}{\Rightarrow} C'$ であるとする．このとき以下のいずれかの条件を満たすなら s は完全トレースである．

1. C' は環境コンフィギュレーションである．C のスタック $\mathcal{E}$ の長さ $|\mathcal{E}|$，トレース s に含まれる $\overline{\text{call}}, \text{return}$ アクションの個数をそれぞれ $\#(\overline{\text{call}}\ in\ s), \#(\text{return}\ in\ s)$ とすると，$|\mathcal{E}| + \#(\overline{\text{call}}\ in\ s) = \#(\text{return}\ in\ s)$
2. $C' = \Uparrow_m$

定義 4.2 (トレース集合)
整合的なコンポーネント K に対して，トレース集合を以下のように与える．

$$Traces(K) = \{s \mid \exists C. \langle [], K \rangle_m \overset{s}{\Rightarrow} C \text{ かつ } s \text{ は完全トレース } \}$$

5　完全抽象性

本章では，前章で与えたトレース意味論の完全抽象性 (full-abstraction) を扱う．

そもそも意味論における完全抽象性とは，異なるモジュールを正確に区別可能であるという性質である．このため，まずは異なるコンポーネント（モジュールに必要な情報を加えたもの）というのが如何なるものかを，等しい（文脈等価な）コンポーネントが如何なるものかを定義することで与える．

5.1　文脈等価なコンポーネント

2 つのコンポーネントについて，まず始めに互換性があるという条件を与える．

定義 5.1 (コンポーネントの互換性)
整合的な 2 つのコンポーネント K_1, K_2 が互換的であるとは，任意の整合的なコンポーネント K に対して，以下の 2 つの条件が等価な場合である．

1. K と K_1 が結合可能
2. K と K_2 が結合可能

互換的なコンポーネントについて，以下のように定義する．

定義 5.2 (文脈前順序)
互換性のある 2 つのコンポーネント K_1, K_2 が文脈前順序関係 $K_1 \lesssim K_2$ にあるとは，結合可能な任意のコンポーネント $K = \langle M, \mu, O, I\rangle$ と，$\langle M, I, \emptyset, \emptyset\rangle \vdash^m e : \tau$ を満たし $[-]^c_{m'}$ を含まない任意の式 e について以下の 2 条件が成り立つことである．ここで $m \in dom(M), m' \in dom(M \cup M_i)$

1. $\langle e, K \mathbin{⩕} K_1\rangle \downarrow_m$ ならば $\langle e, K \mathbin{⩕} K_2\rangle \downarrow_m$
2. $\langle e, K \mathbin{⩕} K_1\rangle \rightarrow^*_m \Uparrow_{m'}$ ならば $\langle e, K \mathbin{⩕} K_2\rangle \rightarrow^*_m \Uparrow_{m'}$

なお，$\downarrow_m$ は計算が終わり，最終的な値がモジュール m で得られることを示す．

$$\langle e, M, \mu\rangle \downarrow_m \quad \text{if and only if} \quad \langle e, M, \mu\rangle \rightarrow^*_m \langle v, M', \mu'\rangle$$

定義 5.3 (文脈等価なコンポーネント)
互換性のある 2 つのコンポーネント K_1, K_2 について，$K_1 \lesssim K_2$ かつ $K_1 \gtrsim K_2$ のとき文脈等価であり，これを $K_1 \simeq K_2$ と示す．

直観的に，文脈等価は任意の式と相手コンポーネントを与えられた際のふるまいが等価であることを意味する．文脈等価なコンポーネントを等しいコンポーネントであるとみなし，この場合における完全抽象性を定理 5.1 に示す．

定理 5.1 (完全抽象性)
互換的なコンポーネント K_1, K_2 について以下が成り立つ．

$$K_1 \simeq K_2 \quad \text{if and only if} \quad Traces(K_1) = Traces(K_2)$$

続く節において，本論文では定理 5.1 を健全性（トレース集合が等しいなら文脈等価）と完全性（文脈等価ならトレース集合が等しい）に分けて証明していく．

5.2 健全性

命題 5.1 (健全性)
互換的なコンポーネント K_1, K_2 について以下が成り立つ．

$$Traces(K_1) = Traces(K_2) \text{ ならば } K_1 \simeq K_2$$

命題 5.1 の証明は以下の条件を示すだけで十分である．

$$Traces(K_1) \subseteq Traces(K_2) \text{ ならば } K_1 \lesssim K_2$$

補題 5.1 (スタック結合による式の型)
結合可能なスタックのペア $\langle \mathcal{E}_2 \rangle_{m_2}, \langle e, \mathcal{E}_1 \rangle_{m_1}$（得られる式を実行するモジュールを m_i とする）と環境 $\langle M, \emptyset, \emptyset, S \rangle$ において

$$\langle M, \emptyset, \emptyset, S \rangle \vdash^{m_i} e : \tau, \quad \langle M, \emptyset, \emptyset, S \rangle \vdash^{m_i} \langle e, \mathcal{E}_1 \rangle_{m_1} \mathbin{⩕} \langle \mathcal{E}_2 \rangle_{m_2} : \tau'$$

となるならば，$\langle M, \emptyset, \emptyset, S \rangle \vdash^{m_i} e_1 : \tau$ であるような任意の式 e_1 について $\langle e_1, \mathcal{E}_1 \rangle_{m_1}$ と $\langle \mathcal{E}_2 \rangle_{m_2}$ も結合可能であり，以下が成り立つ．

$$\langle M, \emptyset, \emptyset, S \rangle \vdash^{m_i} \langle e_1, \mathcal{E}_1 \rangle_{m_1} \mathbin{⩕} \langle \mathcal{E}_2 \rangle_{m_2} : \tau'$$

ただし $e_1 = [e_1']^{c'}$ かつ $\mathcal{E}_2 = E^c : \mathcal{E}_2'$ ならば $c = c'$ である必要がある．

補題 5.1 は続く補題 5.2 に必要な補題であり，式に関する場合分けで証明する [5].
以降よりあるトレース s に対して全ての call，return アクションの上線の有無を入れかえたトレースを $\bar{s}$ とし，s と $\bar{s}$ は双対なトレースと呼ぶ.

補題 5.2 (Trace Composition)
結合可能な 2 つのコンフィギュレーション C_1 と C_2 に対し，$C_1 \overset{s}{\Rightarrow} C_1'$ かつ $C_2 \overset{\bar{s}}{\Rightarrow} C_2'$ のとき以下が成り立つ．

- C_1' と C_2' は結合可能
- $C_1 \mathbin{⩕} C_2 \rightarrow_m^* C_1' \mathbin{⩕} C_2'$

補題 5.3 (Trace Decomposition)
結合可能な 2 つのコンフィギュレーション C_1 と C_2 に対し，
$C_1 \mathbin{⩕} C_2 \rightarrow_m^* \langle e', M', \mu' \rangle$(または $\Uparrow_m$) のとき，適当な C_1', C_2', s に対して以下が成り立つ．

- $C_1 \overset{s}{\Rightarrow} C_1', C_2 \overset{\bar{s}}{\Rightarrow} C_2', C_1'$ と C_2' は結合可能
- $C_1' \mathbin{⩕} C_2' = \langle e', M', \mu' \rangle$(または $\Uparrow_m$)

2 つの補題 5.2 と 5.3 はそれぞれアクションと簡約規則に関する帰納法で証明する [5].
証明 命題 5.1
互換性のある 2 つのコンポーネント K_1, K_2 について以下を仮定する．

$$Traces(K_1) \subseteq Traces(K_2)$$

(1) $\langle e, K \mathbin{⩕} K_1 \rangle \downarrow_m$ の場合

$$\langle e, K \mathbin{⩕} K_1 \rangle \rightarrow_m^* \langle e', K' \mathbin{⩕} K_1' \rangle \quad (e' \text{は値})$$

このとき $K ⩕ K_1$ は，コンポーネント K_1 と結合可能な任意のコンポーネント $K = \langle M, \mu, O, I \rangle$ を結合したものであり，e は $\langle M, I, \emptyset, \emptyset \rangle \vdash^m e : \tau$ を満たし，$[e]^c_m$ のような部分式を含まない任意の式である．

コンフィギュレーション C, C_1 を以下のように与える．

$$C = \langle e, [], K \rangle_m, \quad C_1 = \langle [], K_1 \rangle_{m_1}, \quad C ⩕ C_1 = \langle e, K ⩕ K_1 \rangle$$

Trace Decomposition より以下のような $C', C'_1, s, \overline{s}$ が存在する．このとき $\overline{s} \in Traces(K_1)$ である．

$$C \overset{s}{\Rightarrow} C', \quad C_1 \overset{\overline{s}}{\Rightarrow} C'_1, \quad C' ⩕ C'_1 = \langle e', K' ⩕ K'_1 \rangle$$

ここで以下の環境コンフィギュレーション C_2 を考える．このとき K_1 と K_2 は互換性をもつので，C_2 と C も結合可能である．

$$C_2 = \langle [], K_2 \rangle_{m_2}, \quad C ⩕ C_2 = \langle e, K ⩕ K_2 \rangle$$

仮定より，コンフィギュレーション C_2 もトレース $\overline{s}$ により遷移し，ある状態 C'_2 となる．

$$C_2 \overset{\overline{s}}{\Rightarrow} C'_2$$

Trace Composition より C' と C'_2 は結合可能であるので，

$$C ⩕ C_2 \rightarrow^*_m C' ⩕ C'_2$$

C_2 のスタックが空かつ $\overline{s}$ が完全トレースであることより，C'_2 のスタックは空である．よって $C' ⩕ C'_2$ の式の部分は C' にのみ依存し，それは e' という値である．よって $\langle e, K ⩕ K_2 \rangle \downarrow_m$ である．

(2) $\langle e, K ⩕ K_1 \rangle \rightarrow^*_m \Uparrow_{m'}$ の場合 $(m \in dom(M), m' \in dom(M \cup M_i), i = 1 \text{ or } 2)$

blame が発生した場合，最終的な値は最後の blame アクションにのみ左右され，K_1 と同じトレースをトレース集合に含む K_2 が同じ結果を得るのは明らかである．

よって $\langle e, K ⩕ K_2 \rangle \Uparrow_{m'}$ である．

(1)，(2) より以下が成り立つ．

$$\langle e, K ⩕ K_1 \rangle \downarrow_m (\text{or } \Uparrow_{m'}) \quad \text{ならば} \quad \langle e, K ⩕ K_2 \rangle \downarrow_m (\text{or } \Uparrow_{m'})$$

□

5.3 完全性

命題 5.2 (完全性)
互換的なコンポーネント K_1, K_2 について以下が成り立つ．

$$K_1 \simeq K_2 \text{ならば} Traces(K_1) = Traces(K_2)$$

命題 5.2 の証明は以下の条件を示すだけで十分である．

$$K_1 \lesssim K_2 \text{ならば} Traces(K_1) \subseteq Traces(K_2)$$

補題 5.4 (Definability)
環境コンフィギュレーション C_1 と C_1 のトレース集合の要素 s について $C_1 \overset{s}{\Rightarrow} C'_1$ であるとき，C_1 と結合可能かつ $oact$ で終わる任意のアクション列 t について以下の *2* 条件が等価となるようなプログラムコンフィギュレーション C_2 が存在する．

1. ある C'_2 に対して $C_2 \overset{t}{\Rightarrow} C'_2$

2. $t < \overline{s}$

補題 5.4 はトレース $\bar{s}$ でしか遷移できないプログラムコンフィギュレーションが存在することを示している．ただし最後の $iact$ については考慮しない．なぜなら $iact$ は相手となるコンフィギュレーションが発するアクションであり，自身はそれを制御することができないからである．

この補題は実際に条件を満たすプログラムコンフィギュレーション C_2 を構成することによって証明する．ここで与えるモジュールは，予めストア上に用意した変数 count で現在何番目のアクションを実行しているかをカウントしておき，それを用いてトレース通りではない場合には発散するように定義されている．与えたコンフィギュレーションが補題を満たすことは，トレースの長さに関する帰納法で証明できる．

以下，このプログラムコンフィギュレーション C_2 を構成する．

まず，与えられるトレースは，以下の性質を満たすことに注意する．

- blame が起こらない場合，トレースは次の構文で与えられる．ここで関数名等は省略し，トレース集合に含まれるトレースなら MAIN が，それと双対なトレースなら INSIDE が開始記号である．

  $\mathrm{MAIN} \quad ::= \quad \epsilon \mid \mathrm{call} \quad \mathrm{INSIDE} \quad \overline{\mathrm{return}} \quad \mathrm{MAIN}$

  $\mathrm{INSIDE} \quad ::= \quad \epsilon \mid \overline{\mathrm{call}} \quad \mathrm{MAIN} \quad \mathrm{return} \quad \mathrm{INSIDE}$

  blame が起こる場合のトレースはこうしてできたトレースの一か所を blame アクションに変更し，以降のトレースを消せばよい．
- 新たに名前付けされた関数が登場するのは，相手の関数なら $\overline{\mathrm{call}}$ の引数部と return の関数値，自分の関数なら call の引数部と $\overline{\mathrm{return}}$ の関数値であり，またこの場合，新たに名前付けされた関数以外の関数は登場し得ない．
- MAIN の call で受け取った関数は続く INSIDE の中，$\overline{\mathrm{return}}$ で受け取った関数は続く MAIN の中でしか登場し得ない．
- INSIDE の $\overline{\mathrm{call}}$ で受け取った関数は続く MAIN の中，return で受け取った関数は続く INSIDE の中でしか登場し得ない．

これより C_2 を構成するアイデアを示す．

- C_2 は，式 e，空スタック，コンポーネント K の組として与える．
- 内部変数 count を用意し，やり取りの回数をカウントする．count は始めに 0 とし，以降 $oact$ の直前に +1 する．
- 内部変数 dummy を用意し，必要のない値をこれに代入することで処分する．
- 引数として渡される関数は，その時点で新たに名前が付けられているものなので，値の一致は確認せず，それ以降で関数が使用される場所で対応する関数を呼び出すようにしておく．
- call の場合は関数と引数がトレースと合致するか確認し，合致するなら続くアクションを引き起こすような式を実行する．合致しないなら発散させる．

準備

コンポーネント K が含む補助関数 inf，check_p と，いくつかの略記を与える．これらの補助関数は内部関数であるため契約が検査されることはないので，契約の記述は省略する．

- プログラムを発散させる関数 inf

  $$m.\mathrm{inf} = \lambda x.(m.\mathrm{inf}(x))$$

  inf は使用する場所によってその契約の型を変える必要があるため，実際には関数 inf は名前が被らないように複数実装する必要があるが，簡単のため，まとめて inf 1 つとする．

- 自然数定数 $p_1, \cdots, p_n$ に対し，case e of $p_1 \mapsto e_1;\ \cdots; p_n \mapsto e_n$ は，e の値が p_i のとき e_i，それ以外のときは発散する式とする．このような式は，case と inf を用いて構成できる．
- $\mathrm{count}++; e$ は，$\mathrm{count} := !\mathrm{count} + 1; e$ の略記とする．
- $_ := e; e'$ は，$\mathrm{dummy} := e; e'$ の略記とする．
- 自然数引数が定数 p と等しいことをチェックする関数 check_p

$$m.\mathrm{check}_p = \lambda x.(\mathrm{case}\ x\ \mathrm{of}\ p \mapsto 0)$$

コンポーネント K の関数

トレース $\overline{s}$ の $iact$ に予め count の値がいくつかふっておき，登場する call ごとに続く処理を行う．

count が p の call $m.f(\mathrm{arg})$ に対して，$m.f$ の定義を以下のように追加する．

$$m.f/c = \lambda t.(\mathrm{case}\ !\mathrm{count}\ \mathrm{of}\ \cdots;\ p \mapsto \mathrm{body})$$

既に $m.f$ の定義がされている場合は，$p \mapsto \mathrm{body}$ の部分のみを追加する．$m.f$ がそれよりも前のアクションにおいて新たに名前付けされた関数（実行中に O に加わる）ならば適当な名前で定義しておく．$oact$ で渡す際に名前付けされるので問題ない．

body に記述するのは呼ばれた関数の以降の動きである．blame が発生していない場合，今注目している call とそれに対応する $\overline{\mathrm{return}}\ v'$ の間は INSIDE で定義される形をしており，

$$\overline{\mathrm{call}}\ (\overline{m}.g_1)\mathrm{arg}_1 \cdots \mathrm{return}\ v'_1\ \overline{\mathrm{call}}(\overline{m}.g_2)\mathrm{arg}_2 \cdots \mathrm{return}\ v'_2 \quad \ldots \quad \overline{\mathrm{call}}\ (\overline{m}.g_j)\mathrm{arg}_j \cdots \mathrm{return}\ v'_j$$

と書ける．ここで，return v'_i の count の値を p_i とおく．さらに，呼出している関数 $\overline{m}.g_i$ は，(a) モジュール $\overline{m}$ がエクスポートしている関数，(b) はじめの call の引数 arg，(c) 途中の return で受け取った v'_l のいずれかである．以下，$\overline{m}.g_i$ が出現している箇所については，(b) のときは t で，(c) のときは t_l でそれぞれ置き換えたものとする．このとき，body を以下で定義する．

$$\mathrm{body} \equiv _ := m.\mathrm{check}_{\mathrm{arg}}(t);\ \mathrm{count}++;\ _ := (m.k^p_1)t((\overline{m}.g_1)\mathrm{arg}_1); \mathrm{count}++; v'$$

最初の $_ := \mathrm{check}_{\mathrm{arg}}(t);$ は，arg が自然数値のときのみ挿入する．さらに，以下の関数を K に追加する．これらは内部関数なので契約は省略する．

$$m.k^p_1 = \lambda tt_1.(_ := m.\mathrm{check}_{p_1}(!\mathrm{count}); _ := m.\mathrm{check}_{v'_1}(t_1); \mathrm{count}++; (m.k^p_2)tt_1((\overline{m}.g_2)\mathrm{arg}_2))$$
$$m.k^p_2 = \lambda tt_1t_2.(_ := m.\mathrm{check}_{p_2}(!\mathrm{count}); _ := m.\mathrm{check}_{v'_2}(t_2); \mathrm{count}++; (m.k^p_3)tt_1t_2((\overline{m}.g_3)\mathrm{arg}_3))$$
$$\vdots$$
$$m.k^p_j = \lambda tt_1 \cdots t_j.(_ := m.\mathrm{check}_{p_j}(!\mathrm{count}); _ := m.\mathrm{check}_{v'_j}(t_j); 0)$$

各 $; _ := m.\mathrm{check}_{v'_i}(t_i)$ は，v'_i が自然数値のときのみ挿入する．とくに，INSIDE が空の場合，body は以下のようになる．

$$_ := m.\mathrm{check}_{\mathrm{arg}}(t); \mathrm{count}++; v'$$

$\mathrm{check}_{\mathrm{arg}}$ は arg が自然数値のときのみ挿入する．

式 e の定義

$\overline{s}$ は全体として INSIDE で定義される形をしているので，式 e を以下で定義し，上の body と同様に内部関数として $m.k_1, \cdots$ を追加する．ただし引数 t は使用しないため除いておく．

$$e \equiv \mathrm{count} := 0; (m.k_1)((\overline{m}.g_1)\mathrm{arg})$$

blame を含むトレースの場合

基本的には上記と同様に定義するが，各 INSIDE の形をしたトレース断片について，l 番目の $\overline{\mathrm{call}}$ に対応する return が出現する前に blame が登場している場合を考える．

$$\overline{\mathrm{call}}\ (\overline{m}.g_1)\mathrm{arg}_1 \cdots \mathrm{return}\ v'_1 \quad \ldots \quad \overline{\mathrm{call}}\ (\overline{m}.g_l)\mathrm{arg}_l \cdots \mathrm{blame}\ m'$$

このとき，$m.k^p_{l-1}$ を以下のように定義する．

$$m.k^p_{l-1} = \lambda t t_1 t_2.(_ := m.\mathrm{check}_{v'_{l-1}}(t_{l-1}); \mathrm{count}++; ((\overline{m}.g_l)\mathrm{arg}_l); e')$$

相手モジュール $\overline{m}$ が blame されている場合は，e' は何でもよいので 0 としておく．m が blame されるのは，次のいずれかの場合である：(a) call されている $m.f$ が契約 $c \Rightarrow P$ を持ち，$P \neq N$．このとき，$n \notin P$ なる n が存在するので，e' をその n とすればよい．(b) ある $\overline{m}.g/P \Rightarrow c$ があって，$P \neq N$．このとき，$n \notin P$ なる n が存在するので，e' を $\overline{m}.g(n)$ とすればよい．

与えたプログラムコンフィギュレーションが実際に補題 5.4 を満たし得るかは，トレースに対する帰納法により証明する [5].

例:Definability により存在が示されるプログラムコンフィギュレーション

以下に示すトレースを例に実際に式とコンポーネントの関数を作成する.

$\overline{\mathrm{call}}\ \overline{m}.g1(3)$ call $m.f1(\overline{m}.g2)$ $\overline{\mathrm{call}}\ \overline{m}.g3(4)$ return 2 $\overline{\mathrm{call}}\ \overline{m}.g2(m.f2)$ return $\overline{m}.g4$ $\overline{\mathrm{return}}$ 1 call $m.f3(6)$ $\overline{\mathrm{return}}$ 1 return 10

図 6 にこのトレースの図解を示す. この図には count の様子を付け加えてある.

式を以下に与える.

$$\mathrm{count} := 0; (m.k)(\overline{m}.g1(3))$$

$$m.k = \lambda t_1.(_ := m.\mathrm{check}_4(!\mathrm{count}); _ := m.\mathrm{check}_{10}(t_1))$$

call される関数は $m.f1$ と $m.f3$ の 2 つである. $m.f1$ は INSIDE を 2 つ持つので $m.k$ が 2 つあり，$m.f3$ は INSIDE を 1 つも持たないので $m.k$ が無い.

$$m.f1/c_1 = \lambda t.(\mathrm{case}\ !\mathrm{count}\ \mathrm{of}\ \cdots ; 0 \mapsto (\mathrm{count}++; _ := (m.k^0_1)t((\overline{m}.g3)4); \mathrm{count}++; 1))$$

$$m.k^0_1 = \lambda t t_1.(_ := m.\mathrm{check}_1(!\mathrm{count}); _ := m.\mathrm{check}_2(t_1); \mathrm{count}++; (m.k^0_2)t t_1((t)m.f2))$$

$$m.k^0_2 = \lambda t t_1 t_2.(m.\mathrm{check}_2(t_2))$$

$$m.f3/c_3 = \lambda t.(\mathrm{case}\ !\mathrm{count}\ \mathrm{of}\ \cdots ; 3 \mapsto (_ := m.\mathrm{check}_6(t); \mathrm{count}++; 1))$$

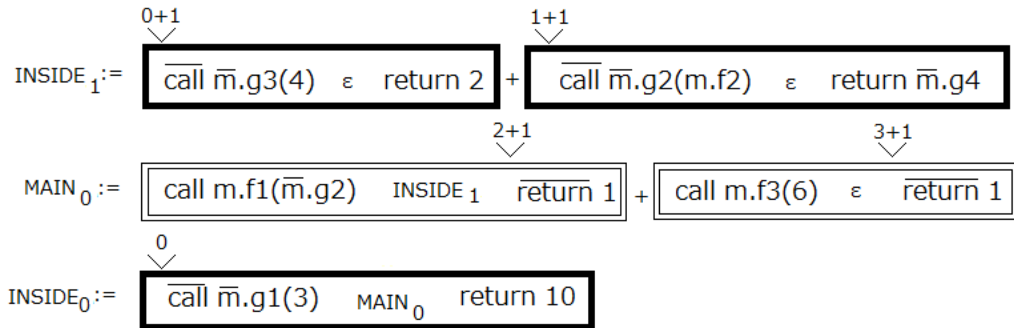

図 6. トレースの図解

証明 命題 5.2

互換性のある 2 つのコンポーネント K_1, K_2 について，

$$\langle e, K \mathbin{⩕} K_1 \rangle \downarrow_m (\mathrm{or}\ \Uparrow_{m'}) \quad ならば \quad \langle e, K \mathbin{⩕} K_2 \rangle \downarrow_m (\mathrm{or}\ \Uparrow_{m'})$$

であると仮定する. $(m \in dom(M), m' \in dom(M \cup M_i), i = 1\ \mathrm{or}\ 2)$

このとき $K \curlywedge K_1$ は, コンポーネント K_1 と結合可能な任意のコンポーネント $K = \langle M, \mu, O, I \rangle$ を結合したものであり, $\langle K \curlywedge K_2 \rangle$ も同様. e は $\langle M, I, \emptyset, \emptyset \rangle \vdash^m e : \tau$ を満たし, $[e]^c_m$ のような部分式を含まない任意の式である.

任意の完全トレース $s \in Traces(K_1)$ について以下が成り立つ.

$$C_1 = \langle [], K \curlywedge K_1 \rangle, \quad C_1 \overset{s}{\Rightarrow} C'_1$$

Definability より C_1 と結合可能なプログラムコンフィギュレーション C_{Def} が存在する.

$$C_{Def} = \langle e_{Def}, [], K_{Def} \rangle, \quad C_{Def} \overset{\overline{s}}{\Rightarrow} C'_{Def}, \quad C_{Def} \curlywedge C_1 = \langle e_{Def}, K_{Def} \curlywedge K_1 \rangle$$

Trace Composition より,

$$C_{Def} \curlywedge C_1 \downarrow_m (\text{or} \ \Uparrow_{m'})$$

ここで仮定より,

$$C_{Def} \curlywedge C_2 \downarrow_m (\text{or} \ \Uparrow_{m'})$$

Trace Decomposition より適当な $C''_{Def}, C'_2, \overline{s}'$ を用いると以下のようになる. (C''_{Def}, C'_2 は結合可能)

$$C_{Def} \overset{\overline{s}'}{\Rightarrow} C''_{Def}, \quad C_2 \overset{s'}{\Rightarrow} C'_2$$

プログラムコンフィギュレーション C_{Def} は Definability より与えられたコンフィギュレーションである. そして仮定より $\overline{s}, \overline{s}'$ の最後が $iact$ であった場合, その $iact$ は必ず等しいので,

$$\overline{s} = \overline{s}', \quad s = s'$$

よって $s \in Traces\langle M_2, \mu_2, O_2, I_2 \rangle$ であり, 以下が成り立つ.

$$s \in Traces(K_1) \quad \text{ならば} \quad s \in Traces(K_2)$$

したがって,

$$Traces(K_1) \subseteq Traces(K_2)$$

□

6　おわりに

本論文では高階契約をもつプログラミング言語を与え，さらに LTS によりモジュールのトレース集合を得て，そのトレース集合をモジュールの意味とするトレース意味論を与えた．そしてこのトレース意味論の完全抽象性を証明した．これにより村井ら [1] の提案した高階契約の体系を，現行の契約のシステムに近づけると共にその意味論が必要な性質を満たすことを証明した．

現行の契約のシステムに近づけるために，自モジュールの関数に対しては契約の監視を行わないよう変更を加えた事で，プログラミング言語そのものの意味論は少し複雑になってしまったが，LTS の遷移規則は簡単になった．自モジュールだけで完結する遷移規則 (TAU) とそれ以外の相互作用的な規則の区別を，契約の監視に関係するか否かで分けることが可能になった．またブレームの発生箇所がモジュール間のやり取りに絞られ，完全抽象性の証明も簡単になった．

その他の変更点として簡単化のためモジュール名を 2 つに絞っていることが挙げられる．現実においてモジュールが 2 つだけの場合にしか対応していないのはシステムとして問題である．しかし与えたシステムにおける 2 つのモジュールを，自モジュールとそれ以外のモジュールに分けて捉えることで，モジュールが 3 つ以上の場合にも自然な拡張で対応できる．よってモジュール名を 2 つに絞っていることによる利便性や完全抽象性の証明への影響は無いと考えられる．

内部変数は補題 5.4（Definability）の証明を簡単にしている．モジュールの内部情報が変化する可能性を付与することで，関数呼び出しに対しての結果が一意ではなくなり，存在し得るトレース

の定義が非常に簡単になっている．もし内部変数がなく，関数呼び出しに対する結果が一意である場合，トレース上のすべての同じ関数呼び出しとその結果について，不整合が起こってないか確認する必要があるため，完全抽象性（特に補題 5.4）の証明が複雑になることが予想される．

今後の課題として，契約の定義をプログラムで記述できるようにすること，この言語を実装してみること，またそれに向けて新たに名前付けされた関数については必要無くなった時点で開放し，メモリの節約を図るように変更することなどが挙げられる．

謝辞

本研究を進めるにあたり，日頃から様々な知識や助言を頂きました結縁・中澤研究室の皆様に感謝いたします.

参考文献

[1] 村井 涼, 五十嵐 淳, and 中澤 巧爾. 高階契約を持つプログラミング言語に対するトレース意味論. In 第 17 回プログラミングおよびプログラミング言語ワークショップ論文集 (PPL2015), March 2015.

[2] Robert Bruce Findler and Matthias Felleisen. Contracts for Higher-Order Functions. In *Preceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*, ICFP '02, pages 48-59, New York, NY, USA, 2002. ACM.

[3] Tim Disney, Cormac Flanagan, and Jay McCarthy. Temporal higher-order contracts. In *Preceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pages 176-188, New York, NY, USA, 2011. ACM.

[4] Benjamin C. Pierce. *Types and Programming Languages.* The MIT Press, 2002.

[5] 井上 鉄也, and 中澤 巧爾. 高階契約に対するトレース意味論の完全抽象性.
URL：http://www.sqlab.i.is.nagoya-u.ac.jp/~thesis/2018/PPL/inoue_PPL.pdf