データ抽象化に於ける
仕様とプログラムとの対応に関する
論理的アプローチの研究

北陸先端科学技術大学院大学

金 藤　栄 孝

本審査用（2004 年 7 月 22 日　組版）

データ抽象化に於ける
仕様とプログラムとの対応に関する
論理的アプローチの研究

金 藤　栄 孝

指導教官：大堀 淳 教授

情報科学研究科

北陸先端科学技術大学院大学

2004 年 9 月

A Logical Approach to Specification and Implementation of Abstract Data Types

by

HIDETAKA KONDOH

Submitted to
Japan Advanced Institute of Science and Technology
in partial fulfilment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Dr. Atsushi Ohori

School of Information Science
Japan Advanced Institute of Science and Technology

September 2004

— 目 次 —

本論文の第 1 章，第 2 章，第 3 章（特に§3.1, 3.2），第 5 章（特に§5.2, 5.3），および，第 6 章それぞれの主要な内容は，Hidetaka Kondoh: Abstract Data Types Can Have Inequations, *Formal Aspects in Computing* **14**, pp. 369–399 (2003) として報告された.

謝辞

本論文は，多くの方々の助けなしには執筆される事はあり得ませんでした．特に，企業に在籍したまま博士課程に籍を置き学位を取得する段階へと至る上で多くの方々の御指導・御理解・御尽力は絶対的な要件でありました．それらの方々全ての御名前をここで挙げ尽くす事は不可能ですが，少なくとも以下に御芳名を挙げさせて頂く少なからぬ方々の御援助なくしては本論文が形を成す事は望むべくもありませんでした．その為，この謝辞が異例な長さとなる事は予め御寛恕頂きたく存じます．

まず誰よりも主指導教官として御指導頂いた二木厚吉北陸先端科学技術大学院大学教授ならびに二木先生から指導を引き継いで下さった大堀淳同大学教授の御二人には衷心より感謝の意を表させて頂きたく存じます．両先生の御指導と御尽力なくしては本論文の執筆段階への到達が想像する事すら不可能であった事は火を見るよりも明らかです．更に，本論文審査員を引き受けて下さり本論文への貴重な様々な御批判を賜わりました片山卓也北陸先端科学技術大学院大学教授，玉井哲雄東京大学教授，小川瑞史北陸先端科学技術大学院大学特任教授，田島敬史同大学助教授の諸先生方にも深甚なる感謝を致したく存じます．とりわけ玉井先生は本論文の草稿に対して様々な間違いや誤植を指摘して下さり，本論文の質の向上にとって大きな助けとなりました事をここに明記し重ねて御礼申し上げます．残っている間違いや誤植が私一人の責任である事は改めて申すまでもありません

本論文の（第 4 章を除く部分の）主要な内容は Formal Aspects of Computing 誌の論文として発表しましたが，その論文の掲載に於きまして同誌の editorial board をなさっておられる Dines Bjørner デンマーク工科大学教授ならびに同教授が選んで下さった匿名の査読者の方には大変にお世話になりました．この査読者の方は望むべくもない程の注意深さで詳細に論文を査読して下さり数多くの間違いや議論の改善すべき点を極めて建設的かつ温かい言葉で指摘して下さり，論文をより良いものとする上で決定的な役割を果して下さいました．本論文はそれら全ての改善点を引き継げたお蔭で多少ともより良く書き上げられる事ができました．その事を御二人に心から感謝させて頂きたいと存じます．

また，本研究の前駆段階の要旨を TLCA ’95 で報告した際，同会議論文集の編集責任者であられた Mariangiola Dezani-Ciancaglini トリノ (Torino) 大学教授には論文の作成上の特段の御配慮を頂いた事を記すと共に，この場をお借りして改めて感謝の念を表させて頂きます．この TLCA ’95 の論文なくしては上記の雑誌の full paper の発表はあり得ず，従って，現在の学位論文も存在し得なかったのですから．

北陸先端科学技術大学院大学 (JAIST) での博士後期課程で要請される講義受講単位を JAIST へ通って取得する事は実際上は不可能でありました．その為，単位認定制度を活用し，入学前に（現実的に職場や拙宅より通学可能な）東京工業大学大学院の講義を受講させて頂く事により必要な単位の殆どを取得致しました．それら受講させて頂きました各講義を担当されておられました東京工業大学大学院の佐伯元司教授，米崎直樹教授，渡辺治教授，小林直樹助教授，渡部卓雄助教授，西崎真也助教授（順不同）の各先生方には大変お世話になった事を深く感謝致しております．とりわけ，佐伯先生には講義を受講させて頂いただけでなく，科目履修の為の諸手続きを御教示頂く等，履修の為に御尽力を賜わった事をこの場をお借りして改めて御礼申し上げさせて頂きたく存じます．この東京工業大学大学院での

科目履修が出来なかったならば，JAIST での課程博士としての修了が全くの不可能事であった事は自明の理であります．

Henk Barendregt ナイメヘン (Nijmegen) 大学教授には本研究の前駆段階の最初期に議論して頂く機会があり，特に構文的な面や型理論としての定式化に関して貴重な御指摘を頂戴致しました．教授の御指摘がなかったらば，本論文で報告した言語／型理論が現状とは随分と異なった見通しの悪い体系となった事は確実であり，同教授には心よりの感謝を申し述べさせて頂きたいと思います．同じく λ-計算の権威であられる J. Roger Hindley ウェールズ (Wales) 大学教授にも同教授が東京工業大学の客員教授として滞在されていた折に様々な議論を通して色々と触発して下さった事に深く感謝致します．しかし，これらの直接的な面以前に，私がお二人に負っている最も重要で決定的な事とは，修士での専攻が数理論理学でも情報科学でもなく型理論という本研究の分野に全く予備知識の無かった私にとって本研究で必要とされる殆ど全ての知識（どの様に定理を設定し補題を立てるか，どんな概念を定義するのか，証明はどの様に与えるのか，その書き方は，等々）を専ら御二人それぞれの著書（*"The Lambda Calculus"* ならびに *"Introduction to Combinators and λ-Calculus"*）のみから学ばせて頂いたという事であり，私にとって掛け替えなき知的財産を御二人それぞれの名著から得られた事は幾ら感謝しても感謝し尽くせません．

既に御名前を挙げさせて頂きました小川瑞史先生は NTT 基礎研究所に御在籍されておられた際，本研究の前駆段階の内容に関して同研究所でお話しさせて頂く機会を与えて下さった事に関しても御礼を申し述べたいと思いますが，特に，その際，同研究所の塚田恭章氏は Martin-Löf の体系に於ける型付け判別式の等式判別式に対する独立性の問題が不明である事に私の注意を喚起して下さいました．3.3 節が同氏の指摘に触発されて生まれた事は改めて申し上げるまでもありません．この事に関し塚田氏に心より御礼申し上げたいと思います．

さて，企業に所属する社会人である私が本論文へと至る迄には，周囲の多くの方々の御理解や御支援が不可欠でありました．以下は時間を遡る順にそれらの方々への御礼を申し上げさせて頂きたいと存じます．

JAIST の博士後期課程に在籍して課程博士の形で学位取得に至る上では，小泉忍（株）日立製作所システム開発研究所前第 2 部長や佐川暢俊第 2 部長の御理解と御支援とは絶対に欠く事はできませんでした．快くそれらを提供して下さいました御二人に深甚なる謝意を表させて頂きたく存じます．

私が本研究の方向へ踏み出す直接の切っ掛けは，当時，（株）日立製作所基礎研究所の主任研究員であられた野木兼六氏（現神奈川工科大学教授）が「基礎研に転勤して研究したいならば何か書いて来なさい」との一言であり，それへの答として書いたものが本研究の端緒となりました．その意味で，本論文執筆への開始点は同氏により与えられたと申せますし，また，基礎研究所在籍中も神奈川工科大学へ転出されてからも常に厳しくも建設的な御批判を賜わり続け，それらの批判に答えようと試みる事が研究の推進力となりました．これら諸々の事柄に関して心よりの御礼を申し上げます．また，システム開発研究所の主任研究員であられた大槻繁氏は，困難な状況に於いて大いなる庇護を与えて頂くと共に，常に学位取得を目指す様にと励まし続けて下さいました．その事に対して適切な感謝の言葉さえ見付かりません．本当に有難う御座いました．大槻氏の温かく継続的な励ましがなければ，元来，根性に欠け

る私は遠の昔に学位取得を諦めていた事は間違いありません．

既に触れました通り，私は学生時代に情報科学を専攻した訳ではありませんが，その私に，ソフトウェア工学研究やコンパイラ開発の業務を通し或いは輪講等の場で，プログラミング言語論，形式的意味論，関数的プログラミングといった本論文に不可欠な分野に関する勉強をする端緒を与えて頂きそれらの分野に関して様々な御指導を賜わった山野紘一氏（当時（株）日立製作所システム開発研究所主任研究員，現大阪経済大学教授）にも厚く御礼申し上げます．更に，現在の職場に就職し最初に書いた研究報告書（それは型の同値性という企業的な価値からはかけ離れた主題に関するものでしたが）を見て「君に書いて欲しかったのは正にこんな内容の報告書だよ！」と力強い励ましの言葉をかけて下さった当時システム開発研究所主任研究員であられた小林正和氏に心から感謝致しております．このポジティブで勇気付けられる一言が私のその後の研究方向を決定付けたと申し上げても過言ではありません．

最後に，私事に亙り恐縮ですが，本論文の執筆中は殆どの休日を返上し帰宅が深夜となる事も日常茶飯事であったにも拘らず, じっと忍耐し続けてくれた妻博子への「よく我慢してくれて有難う」の一言を書き添えさせて頂き異例な長さの謝辞を閉じたいと思います．

献辞

本論文を，5.1 節を執筆中の 3 月 8 日の深夜に突然の訃報が届いた今は亡き父，栄二に捧げる．

第1章

序論

— 抽象データ型のモデル化 —

本章では，ソフトウェア開発で重要なモジュール化の概念を提供する抽象データ型に対して理論に立脚したモデル化を与えようとしている二つのアプローチとして，形式的な仕様記述で広く用いられている代数的アプローチ（代数仕様）とプログラミング言語との親和性の良い論理的アプローチ（型理論）とを比較し，それら二つのアプローチ間の対応と各々による抽象データ型のモデル化に於ける問題点を観察する．

1.1 抽象データ型のモデル化に関する二つのアプローチ — 本研究の動機

ソフトウェアをモジュール化された形で開発する上で抽象データ型の概念が鍵になると一般に広く認識されている事は改めて述べるまでもない．理論的な立場からは，大別すると次の二つの見方が抽象データ型に対してなされて来た：

(1) 論理的（高階型付き λ-計算に基づく）アプローチ

抽象データ型を 2 階の命題論理の論理式で最外の構成が 2 階の存在限量子としてモデル化するアプローチで，プログラミング言語の理論（意味論）と高い親和性があるという長所を持つ；

(2) 代数的アプローチ

抽象データ型を 1 階の等式論理（およびその拡張）で規定される代数（の一定の条件を満たすクラス）としてモデル化するアプローチで，形式仕様記述の分野では仕様記述に於ける抽象データ型に対する理論的な基盤を与えるモデルとして長く認められて来た．

しかしながら，これら二つのアプローチの何れにも，抽象データ型の概念を完全には捉え切れていない欠点がある．後で例題を用いて示す通り，(1) のアプローチでは或る種の異なる抽象データ型を別の型として区別できないという不満がある．一方，(2) のアプローチの場合，代数仕様によって与えられた抽象データ型の仕様に対するプログラミング言語で記述された抽象データ型の実現が正しいとは如何なる事かに対する意味論的な基盤が充分に確立されてはいない，という問題を抱えている．

更に悪い事には，これら二つのアプローチは，お互いに関係を持つ事なく殆ど独立に研究が進められて来た． 1984 年に開催された International Workshop on Semantics of Data Types の論文集 — Kahn, MacQueen and Plotkin (eds.) (1984) — への編集者による序文で，彼らは

> The Symposium was intended to bring these somewhat disparate groups together with a view to promoting a common language ...

と述べていたが，その会議から 20 年を経ても，彼らの指摘，つまり “somewhat disparate” という点が余り改善されているとは言えない．本研究は彼らのこの序文

によって強く動機付けられた．即ち，上記の二つのアプローチの間に理論的な基礎を持った関連を与え，抽象データ型に対する代数的に記述された仕様とプログラミング言語によって記述される実現との対応の正しさに意味論的な基盤を提供すべきだ，という動機付けである．

無論，両方のアプローチを統合しようという試みが全くの皆無だった訳ではない．それら従来の関連研究については，6.1 節で纏めて概観する．

本論文では，Cardelli and Wegner (1985) の 2 階の型付き λ-計算 Fun に対して，仕様記述の為に不等式の形の表明（部分正当性を表わしている）を追加した型システムを持つ 2 階の型付き λ-計算を提案し，基本的な性質を検討する．その検討の過程で判る通り，本論文の型システムは Fun の型システムの保存的拡大となっている．この事は，Fun の型システムの持つ良い性質を全て引き継いでいる事を意味する．

本論文の体系の最大の特徴は，抽象データ型に対する実現モジュール（その抽象データ型の仕様で宣言されている当該抽象データ型の基本演算子群に対する実現の一揃い）が完全に通常のデータと同じ 1 階のデータとして自然数等と同じく一般再帰等によって自在に取り扱える点にあり，また，抽象データ型に対する仕様と実現とが共通の意味論の枠内で捉えられる事を可能とした点である．

そして，オブジェクト指向でのインスタンス変数やメソドの継承に対するモデル化として Cardelli (1984, 1988) が提案した「部分型としての継承」に関して言えば，本論文の体系で表明を付加された型の間の部分型として表わされる関係は，Bruce and Wegner (1990) が述べた意味での代数的仕様に於ける継承に正に対応している．

その意味では，本論文の体系は，抽象データ型の部分正当性に関する（代数仕様風に記述された）仕様と実現としてのプログラムとの双方を，同一の言語で記述する広帯域言語 (wide-spectrum language) の基盤を提供する体系だと言える．

■ 抽象データ型に関する用語法の纏め

以下の議論の為に，抽象データ型の概念の概略を簡単に纏め，それを通して抽象データ型に関する用語法を固定する事とする．なお，本論文で「抽象データ型」という言葉を用いるのは，（値の計算に状態を用いるのは別にして）状態でなく値の表現を隠蔽する場合に対してのみ限定する．従って，代数仕様や Standard ML の `abstype` で宣言される型等，関数的プログラミング言語での抽象データ型が本論文

の意味での「抽象データ型」の典型であるが，状態そのものを隠蔽する場合には，「クラス」，「インスタンス」，「オブジェクト」といった所謂オブジェクト指向での用語を用いて明確に区別する[1)]．

抽象データ型がそれを利用するプログラムの他の部分に対して隠蔽する情報は，大別すると次の 2 種類に分けられる：

(a) 抽象データ型に属するべき値の具体的な表現データ構造；
(b) 抽象データ型の値を操作する為に抽象データ型に伴って宣言された基本演算子の実現の一揃い．各基本演算子の実現は (a) での値の表現データ構造の型に対して正しく型付けられていなければならない．

以下，(a) での抽象データ型に属する値を表現するデータ構造の型を「**表現型**」と呼び，また，表現型を一つ選んで固定した時にその表現型に対応しての (b) での各基本演算子の実現の一揃いの成す型を「**実現型**」と呼ぶ事にする．

抽象データ型の仕様記述で広く用いられている (1) の代数的アプローチの場合，Ehrig and Mahr (1985) で述べられている様に，基本的には，抽象データ型を多ソート等式論理によってモデル化する．この場合，一つの理論は，抽象データ型の値の集合の種類を表わす一連のソートと各ソートの値を操作する為の基本演算子を表わす演算子記号の集まり，並びに，演算子記号の指す基本演算子の振舞を定める等式公理群によって規定される．一つの理論（を規定する代数仕様記述のモジュール）の**シグニチャ**とは，通常，その理論の一連のソートの集まりと演算子記号の集まりとの対を表わすが，本論文では**演算子記号の集まりだけを指す**言葉として用いる．

代数仕様によって等式理論の形で仕様が与えられた抽象データ型が等式理論により規定される一連の代数の如何なるクラスとして定義されるのか，に関しては，上で参照した Ehrig and Mahr (1985) の場合，抽象データ型は仕様としての等式理論が定める始代数全ての成すクラスとしているが，完全に合意されているとは言い難い．

1) 1983 年に Goldberg らの手になる Smalltalk-80 の言語仕様と実現に関する書籍が出版され，それに伴って我が国でもオブジェクト指向が話題となり始めた頃，随分と以前であるが，オブジェクト指向に関する非公式な集会に於いて，「オブジェクト指向とは何か，抽象データ型と同じなのか違うのか？」という質問を鈴木則久氏（当時，東京大学助教授）が受けた折，同氏は「例えばスタック s に値 i をプッシュするという事を表わすのに，“s.push(i)” と書くのがオブジェクト指向で，“push(i, s)” と書かねばならないのは抽象データ型だ」と答えておられた．即ち，最初の表記法では値 i がプッシュされたスタックは手続き push の呼出し後の状態として表わされているのに対して，後の表記法ではそれは関数 push の戻り値として表わされている．本論文での「抽象データ型」という言葉の使用の範囲は，この鈴木氏の答と全く同じである．

更に重要な問題は，代数的アプローチでは，関数型・多相型・依存型といった高階論理やそれに基づいた関数的プログラミング言語との対応を良くする為の拡張に関して余り成功していない事である．実際， Mossakowski, Haxthausen and Krieg-Brückner (2000) は，その Future Work の節に於いて，代数仕様が関数的プログラミングと意味論的により親和性を高める方向への拡張の研究の必要性を強調している．

抽象データ型の実装で用いられている論理的（或いは関数的プログラミング的）アプローチに於ける抽象データ型のモデル化の為の基本的な道具はレコード型である．このアプローチに属する様々な研究は大別して二つのカテゴリに分類できる．最初のカテゴリに属する研究は抽象データ型に於ける継承の概念に対して理論的なモデルを与える事を目的としており， Cardelli (1984) の “A Semantics of Multiple Inheritances” をこの方向への研究の流れの嚆矢とする．もう一方のカテゴリは Mitchell and Plotkin (1985) の “Abstract Data Types Have Existential Type” を源流とし，抽象データ型での情報隠蔽に対する理論的基盤を 2 階の命題論理に於ける存在限量子を用いて与えようとする一群の研究が当カテゴリに属する．

注意すべき事は， Cardelli (1984) 並びにそれに引き続く継承に関する一連の研究の多くは，抽象データ型よりもオブジェクト指向に関する研究だという事である．それらの源流である Cardelli (1984) はメソドではなくインスタンス変数の継承についての研究であった．しかし， Abadí and Cardelli (1996) に代表される彼のその後の仕事は，メソドを手続き値を持つインスタンス変数と看做す事によって，メソドの継承に対する理論的裏付けを与えている．そして，オブジェクト指向でのメソドは，メソドが操作する内部状態（インスタンス変数群）を明示的なパラメタとして引き渡す形の関数に書き直す事により抽象データ型に備えられるべき基本演算子へと対応付けられる．従って，メソドの継承に関する Cardelli らの研究は抽象データ型に於ける基本演算子の継承に関する研究とも看做して良い．

なお，「継承」という言葉のオブジェクト指向の分野での使用法はかなり混乱しており，またその表わす意味が曖昧な場合も多い．実際， Pree (1995) が指摘している様に，「継承」という言葉は，往々にして**「同一の名前を持つメソドおよび／或いはインスタンス変数を持つ事」**とか**「実現としてのソースコードの全部あるいは部分を再利用する事」**といった意味で用いられている．以上の様に「継承」という語を漠然

とした形で使うのでは，この語に対し有意な意味論的内容を与える事ができないのは明らかである．

本論文では，ここで提案する新しい部分型関係の機構に基づいて，「継承」という語をより制限された — その代わり，より豊かな意味を持つ — 概念を表わす目的で用いる．即ち，オブジェクト指向の文脈で言えば，下位のクラスのメソドは上位クラスの同名のメソドが持つのと同等以上の論理的性質を持つ，という意味で「継承」という言葉を用いる．以上に述べた形の意味論的に有意であり（少なくとも正しいプログラムを作るという観点からは）実用的にも有益な概念としての「継承」は，実用的ソフトウェア開発技法でも現実に既に採用されている．その様な開発技法の例としては Bertrand Meyer (1997) の**「契約による設計 (Design by Contract)」**を挙げる事ができる．

抽象データ型に関する文脈の場合は，上記の継承の概念の中の「メソド」を「基本演算子」に，また，「クラス」を「抽象データ型」に，各々，言い替えれば良い．その結果，抽象データ型に於ける継承とは，シグニチャと論理的性質の双方に関しての相対的な「豊かさ」に関する順序関係として捉える事が可能である．この意味論的な内容を持つ継承の概念は，オブジェクト指向の分野では， America (1991) に見られる様に，しばしば**「振舞に基づく部分型 (behavioral subtyping)」**と呼ばれる．

以下の表は， Mitchell and Plotkin (1985, 1988) および Cardelli and Wegner (1985) での論理的アプローチに於ける存在限量化されたレコード[2)]が代数的アプローチでの抽象データ型の構成概念とその実現とをどの様にモデル化するかを纏めた．

表 1.1 Fun での抽象データ型の各概念のモデル化手段

抽象データ型	存在限量化レコード型
（代数仕様での）ソート	存在限量子で束縛される型変数
実現型	レコード型
基本演算子記号	レコードのフィールドラベル
基本演算子の一揃い	レコード値
基本演算子	フィールドに割り当てられた値
継承関係	レコード型間の部分型関係

2) Mitchell and Plotkin (1985) では基本演算子群の実現型を表すのにレコード型でなく直積型を用いているが，この違いは本質的ではなく，レコード型の方が基本演算子の名前をフィールドラベルとして直接に表わせる利点があるので，本論文では Cardelli and Wegner (1985) での言語 Fun に従いレコード型を用いる．

この表を見て直ちに気付く事は，代数仕様での等式に対する行が含まれていない，という事である．この事は，抽象データ型に対する論理的 — 或いは「型付き関数的プログラミング的」と呼ぶ方が適切かもしれないが — アプローチでの本質的な欠陥を露呈している．即ち， Mitchell and Plotkin や Cardelli and Wegner での存在限量化型に基づく論理的アプローチは，抽象データ型の（代数的アプローチの意味での）代数構造を無視している，という欠陥である．この事実は， Reynolds (1983, 1985) によって指摘されている．従って， Mitchell-Plotkin 流の存在限量化型は満足と呼ぶには程遠い．言い替えれば，存在限量化されたレコード型が表わす代数のクラスとは，シグニチャのみが確定しており基本演算の振舞に関しては全く規制のない — Reynolds の言い方を借りれば *'anarchic'* な — 全ての代数の成すクラスだという事である．この問題については次節で具体例を用いて観察する．

論理的アプローチに於ける以上の問題点を解決し，また，前に述べた意味論的に内容のある継承概念を得る為に，本論文ではレコード型に不等式の表明を追加し，表明が表わす代数に類似の構造に基づく部分型関係を導入した言語とその型の体系を提案する．従って，本論文での不等式で拡張されたレコード型（を存在限量化した型）は *'non-anarchic'* な代数に準じた構造のクラスを表わす事が可能となる．

1.2 Cardelli-Mitchell-Plotkin-Wegner 流のモデル化の問題

本節では， Cardelli-Mitchell-Plotkin-Wegner による抽象データ型そその間の継承のモデル化を Cardelli and Wegner (1985) の言語 Fun で記述した具体例で示し，同時にこのモデル化での問題点を指摘する．なお， Fun には大域的定義の構文は含まれていないが，例題をコンパクトに記述する為に，大域的な定義については Standard ML の構文を借用してインフォーマルに用いる．

例題

抽象データ型としての（自然数の） LIFO (Last-In-First-Out: 後入れ先出し) スタックの型 Stack は以下の演算子を備えている： new は新しい空のスタックを生成する； isnew はスタックが空か否かを判別する； push はスタックの先頭に自然数を一つ追加する； top はスタックの先頭要素である自然数を取り出す； pop はスタックの先頭要素の自然数を捨てる．

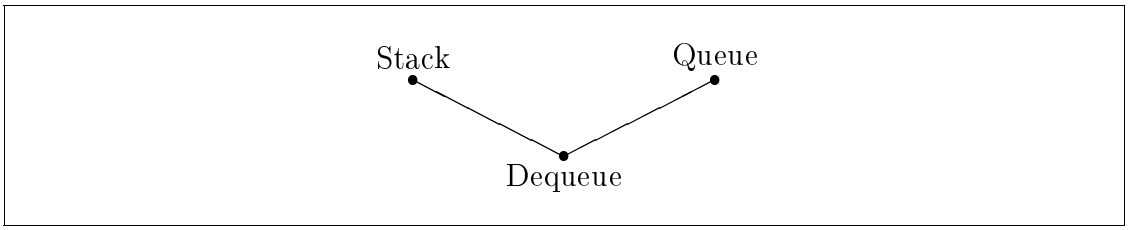


図 1.1 スタック / キュー / デキューの継承関係

```
type StackValRep = List[nat];
type QueueValRep = List[nat];
type DequeueValRep = List[nat];
```

図 1.2 スタック / キュー / デキューの表現型

一方,（自然数の） FIFO (First-In-First-Out: 先入れ先出し) キューの型 Queue は抽象データ型としては次の諸演算子により特徴付けられる： new は新しい空のキューを創る； isnew はキューが空か否かを判別する； add はキューの最後に自然数を一つ追加する； first はキューの先頭要素である自然数を取り出す； remove はキューの先頭要素の自然数を捨てる.

最後に,（自然数の）双方向デキューの型 Dequeue はスタックとキューの双方の演算を備えた抽象データ型である. この時, これら三つの抽象データ型の継承関係は図 1.1 に示す通りである.

さて, これら三つの抽象データ型 Stack, Queue, Dequeue の実現を与える為には, 各々のデータの表現構造の型である表現型 StackValRep, QueueValRep, DequeueValRep を決めねばならない. そこで, 自然数の型 nat とリスト型構成子 List は予め言語 Fun に用意されているとし, 図 1.2 に示す通り, これら三つの表現型には何れの場合も自然数のリスト List[nat] を用いる事にする.

以上の表現型 StackValRep, QueueValRep, DequeueValRep を用いると, 抽象データ型 Stack, Queue, Dequeue 各々の基本演算子群に対する実現型 StackOpImpl, QueueOpImpl, DequeueOpImpl は, それぞれ図 1.3 , 図 1.4 , 図 1.5 に示した型となる.

```
type StackOpImpl = {new: StackValRep,
                    isnew: StackValRep → bool,
                    push: nat → StackValRep → StackValRep,
                    top: StackValRep → nat,
                    pop: StackValRep → StackValRep};
```

図 1.3 Fun での LIFO スタックの実現型

```
type QueueOpImpl = {new: QueueValRep,
                    isnew: QueueValRep → bool,
                    add: nat → QueueValRep → QueueValRep,
                    first: QueueValRep → nat,
                    remove: QueueValRep → QueueValRep};
```

図 1.4 Fun での FIFO キューの実現型

```
type DequeueOpImpl = {new: DequeueValRep,
                      isnew: DequeueValRep → bool,
                      push: nat → DequeueValRep → DequeueValRep,
                      top: DequeueValRep → nat,
                      pop: DequeueValRep → DequeueValRep,
                      add: nat → DequeueValRep → DequeueValRep,
                      first: DequeueValRep → nat,
                      remove: DequeueValRep → DequeueValRep};
```

図 1.5 Fun でのデキューの実現型

これら三つの実現型の間には，Fun でのレコード型同士間の部分型関係に関する規則によって，図 1.1 と同様の順序構造が成立する．

```
val aStackOpImpl = {new = nil,
                    isnew = λ s: StackValRep. isnull(s),
                    push = λ i: Nat.λ s: StackValRep. cons(i)(s),
                    top = λ s: StackValRep. head(s),
                    pop = λ s: StackValRep. tail(s)};
```

図 1.6　Fun での LIFO スタックの或る実現

更に，抽象データ型 Stack は，前節の表 1.1 で示した通り，表現型を図 1.2 での様に具体的に与える代わりに型変数として存在限量化すれば良いので， t を型変数とすると，

$$\textbf{type}\ \mathrm{Stack} = \exists \mathrm{t}.\mathrm{StackOpImpl}[\mathrm{t}/\mathrm{StackValRep}]$$

として定義できる（ここで， “StackOpImpl[t/StackValRep]” は StackOpImpl 中の StackValRep の全てを t で置き換えた型を表わす）．他の抽象データ型 Queue, Dequeue も同様に存在限量化によって得られるが，その結果，それら三つの抽象データ型の間には，上で述べた各々の実現型の間の部分型の順序関係から， Fun での存在限量化型同士間の部分型関係に関する規則により，図 1.1 に示した順序関係が成立する．即ち，少なくともこの三つの抽象データ型のケースでは， Fun の部分型関係は抽象データ型の継承関係を捉えられるという事である．

さて，上の三つの実現型の中でスタックの基本演算群の実現型 StackOpImpl を持つ式としては，例えば図 1.6 に示すレコード式 aStackOpImpl がある．この式は，確かに「後入れ先出し」というスタックに期待されている振舞を示す．例えば新しいスタックに push を用いて 1 と 2 とをこの順に入れ， pop によって一つ要素を捨て，その時のスタックの先頭要素の値を top で観察するという事を表わす次の式

```
aStackOpImpl.top(
  aStackOpImpl.pop(
    aStackOpImpl.push(2)(
      aStackOpImpl.push(1)(new))))
```

の値は 1 となる．

```
val anotherStackOpImpl = {new = nil,
                          isnew = λ s: StackValRep. isnull(s),
                          push = λ i: Nat. λ s: StackValRep. cons(i)(s),
                          top = fix(λ t: StackValRep → Nat. λ s: StackValRep.
                                    if length(s) ≤ 1 then head(s) else t(tail(s))),
                          pop = fix(λ p: StackValRep → StackValRep. λ s: StackValRep.
                                    if length(s) ≤ 1 then nil else cons(head(s))(p(tail(s))))};
```

図 1.7 Fun での LIFO スタックの別の「実現」

一方， StackOpImpl を持つ式としては，上の aStackOpImpl だけではなく，図 1.7 に示す anotherStackOpImpl という式も存在する．そして，この式の振舞を上と同様に観察する為に，次の式の値を求めると，この場合は，

```
anotherStackOpImpl.top(
  anotherStackOpImpl.pop(
    anotherStackOpImpl.push(2)(
      anotherStackOpImpl.push(1)(new))))
```

の値は 2 となってしまう．

つまり，実現式 anotherStackOpImpl はスタックに対して期待していた「後入れ先出し」ではなく「先入れ先出し」の振舞をする．後の実現 anotherStackOpImpl はスタックではなくてキューの実現に適した（但しレコードのフィールドラベルを変更する必要があるが）式なのである．

以上の例は， Cardelli-Mitchell-Plotkin-Wegner 流の抽象データ型のモデル化に於いては，我々が「抽象データ型」という概念を使用する上で期待している基本演算の振舞を規定する為の記述手段が欠如している事を表わしている．

この抽象データ型の基本演算の振舞をも代数的アプローチでの手法によって型として把握し，それを論理的アプローチ — つまり型理論 — の枠組に組み込んだ形で記述する事こそが本論文で解決を試みる問題であり，それを可能とする言語とその型システムの性質を検討する事が本論文の主題である．

第2章

広帯域言語Funiq

— その構文と型システム —

本章では，序論で述べた Mitchell-Plotkin 流の存在限量化による抽象データ型のモデル化に於ける問題点を解決する為に，抽象データ型の実現としてのプログラムのみならず，抽象データ型の仕様に関しても，不等式の形の表明によって型構成方法の一部として記述を可能とした広帯域言語 Funiq を与え，その言語の型システムを公理と推論規則とにより定式化される型理論 FUNIQ として定義する．

なお，抽象データ型の仕様が，何故，通常の代数仕様での等式ではなくて不等式の形に制限されているのか，という事に関しては，第 5 章で本言語の表示的意味論を検討する際に明らかとなる．

しかる後に，前章で具体例を挙げて示した Cardelli-Mitchell-Plotkin-Wegner 流の抽象データ型のモデル化での問題点が Funiq の型システムでは解決可能である事を示し，代数仕様でのモジュールによる抽象データ型の仕様が Funiq で抽象型と詳細化型とを用いてどの様に表現されるかを一般形で与える．

最後に，代数仕様とは対応しない形の詳細化型の表現能力や不等式による表明の記述力の限界の一端を幾つかの例を用いて示す．

2.1　広帯域言語 Funiq の構文

本節では，広帯域言語の為の型付き λ-計算の体系 Funiq を Cardelli and Wegner の 2 階の型付き関数的プログラミング言語の為の体系 Fun の拡張として定義する．なお， Cardelli and Wegner(1985) に与えられている彼ら自身の Fun には不動点による再帰の為の **fix** e は含まれていないが，本論文では，この構文を追加したものを Fun と呼ぶ．

以下で検討の対象とする広帯域言語の為の体系 Funiq の言語としての構文を図 2.1 に示す． Funiq のどの部分がベース言語である Cardelli and Wegner の Fun への拡張なのかを明確にする為，図では Funiq 特有の構文クラスや構文規則に対し “(∗)” の印を付す．

なお，ベースの言語である Fun に於ける式の集合と型の集合とを参照する為に，部分言語 Fun を以下の様に定義しておく．

定義 2.1 （部分言語 Fun）

Funiq の部分言語 Fun は，以下の様に定義される：

- Fun の型の集合 $\mathbf{Type}_{\mathrm{Fun}} \subset \mathbf{Type}$ は， Funiq の **Type** の生成規則の中で， “(∗)” の印が付いていない規則だけで生成される集合である；
- Fun の式の集合 $\mathbf{Exp}_{\mathrm{Fun}} \subset \mathbf{Exp}$ は， Funiq の **Exp** の生成規則の中で， “(∗)” の印が付いていない規則だけで生成される集合である．

言語としての Funiq の最大の特徴は，型に含まれる値に対する制約として仕様を記述する為の構文クラスとしての **Assertion** （表明の集合）が追加されている点とそれを含んだ詳細化型の構文が集合論での内包的表記法と類似の形で **Type** の生成規則として追加されている点とである．

なお，レコード型／レコード式でのフィールドラベルの出現順序と可変型でのタグラベルの出現順序とは，何れも Funiq やその部分言語 Fun に於いては有意でない．また，詳細化型での表明の出現順序も Funiq で有意でない．従って，それらの出現順序だけが異なる型同士や式同士は同一と看做す．

	$\sigma, \tau \in \mathbf{Type}$		型：
		$\sigma ::= \mathbf{Top}$	最大型,
		$\mid \iota$	基本型,
		$\mid t$	型変数,
		$\mid \sigma_1 \to \sigma_2$	関数型,
		$\mid \{l_1 \colon \sigma_1, \ldots, l_n \colon \sigma_n\}$	レコード型 ($n \geq 0$),
		$\mid [\, l_1 \colon \sigma_1, \ldots, l_n \colon \sigma_n \,]$	可変型 ($n \geq 0$),
		$\mid \forall t{<:}\sigma.\tau$	範囲限定多相型,
		$\mid \exists t{<:}\sigma.\tau$	範囲限定抽象型,
(*)		$\mid \{r \colon \sigma \mid \gamma_1, \ldots, \gamma_m\}$	詳細化型 ($m \geq 0$).
	$e \in \mathbf{Exp}$		式：
		$e ::= c$	定数,
		$\mid x$	（通常の）変数,
(*)		$\mid r$	実現変数,
		$\mid \lambda\, x \colon \sigma.e$	関数抽象式,
		$\mid e_1 e_2$	関数適用式,
		$\mid \{l_1 = e_1, \ldots, l_n = e_n\}$	レコード式 ($n \geq 0$),
		$\mid e.l$	フィールド選択式,
		$\mid [l = e]$	タグ付け式,
		$\mid \mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n$	タグ選択式 ($n \geq 0$),
		$\mid \Lambda\, t{<:}\sigma.e$	型抽象式,
		$\mid e[\sigma]$	型適用式,
		$\mid \mathbf{pack}_{\exists t<:\sigma.\tau}\ e\ \mathbf{with}\ \rho$	データ代数式,
		$\mid \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2$	実現参照式,
		$\mid \mathbf{fix}\ e$	不動点再帰式,
		$\mid e \colon \sigma$	型制約式（明示的型付け）.
(*)	$\gamma, \delta \in \mathbf{Assertion}$		表明：
(*)		$\gamma ::= e_1 \leqslant e_2 : \sigma$	原始表明,
(*)		$\mid \mathbf{forall}\ x \colon \sigma.\gamma$	全称化表明.

［注］以下の字句カテゴリの詳細は定めない（但し互いに重なりがないとする）：

$\iota \in \mathbf{BaseType}$	基本型の有限集合,
$t \in \mathbf{TVar}$	型変数の集合,
$c \in \mathbf{Const}$	定数記号の集合,
$x \in \mathbf{Var}$	（通常）変数の集合,
(*) $r \in \mathbf{IVar}$	実現変数の集合,
$l \in \mathbf{Label}$	フィールドラベル／タグラベルの集合.

図 2.1　Funiq の構文

以下，図 2.1 で与えられた Funiq の各構文について簡単に説明する．現時点では未定義の用語 — 例えば「型の上限」 — を用いざるを得ない場合があるが，それらの用語は次節の終わり迄で定義される．

■ 型

- **最大型 Top**
  後述の範囲限定の多相型や抽象型に於いて，型変数の取り得る範囲の上限を制限しない，という事を表わす為に，「型として可能なものの中で最大の型」という概念を表わす型定数である．

- **基本型 ι**
  例えば自然数の型 nat 等であり，基本型に関しては，複合的な型でなく内部構造のない原子的な型である，という事だけを仮定している．

- **型変数 t**
  型を「値」として取る変数である．

- **関数型 $\sigma_1 \to \sigma_2$**
  関数の型であり， σ_1 は変域の型で σ_2 は値域の型とする．なお，関数は一般に部分関数であり，変域に属する全ての値に対して関数値が定義される事が保証された全域関数とは限らない．

- **レコード型 $\{l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\}$**
  Standard ML でのレコード型や C での `struct` 型に相当するラベル付き直積集合の型である．但し， Standard ML でのとは異なり，同一のフィールドラベル l_i が（全く異なるレコード型の成分フィールドとして）複数の全く異なる型を持つ事が許されている．

- **可変型 $[l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n]$**
  StandardML での **datatype** 宣言によって定義される「データ型」 $l_1{:}\sigma_1 \mid \cdots \mid l_n{:}\sigma_n$ に相当するラベル付き直和集合の型である．但し，Standard ML での **datatype** とは異なり，同一のタグラベル l_i を持つ成分が複数の異なる可変型に属する事が許されている．その場合，同一のタグラベル l_i に対する型は全く異なる型であっても構わない．

- **範囲限定多相型** $\forall t{<}{:}\sigma.\tau$

  型に関してパラメタ化された値（多相値と呼ぶ）が属する型である．型変数 t はその型パラメタを表わし，通常は，その型変数 t が本体の型 τ の中に現れる（そうでなければ，実質的にはパラメタ化されていない型になる）．なお，型 σ は型パラメタ t を置き換え得る型の上限を与える．従って，全く無制限にしたい場合には，σ として最大型 **Top** を指定する．

- **範囲限定抽象型** $\exists t{<}{:}\sigma.\tau$

  抽象データ型での抽象化の対象となる値の集まりの型としての表現型とその表現型に対応して実現される基本演算群の型（実現型）とを纏めて隠蔽したモジュールの型である．型変数 t は隠蔽された表現型を表わし，通常，その型変数 t が実現型に対応する本体の型 τ 中に現れる（さもなくば，どの様な値の集まりも抽象化していない事になる）．なお，型 σ は隠蔽された表現型としての型変数 t が取り得る具体的な表現型の上限を与える．従って，全く無制限にしたい場合には，σ として最大型 **Top** を指定する．

- **詳細化型** $\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}$

  本節の始めの方で述べた通り，集合論の内包公理での表記法と同様，各表明 γ_i には実現変数 r が出現し，型 σ に対する値の集合の中で各表明 γ_i を満たす値のみからなる部分集合を与える型である．即ち，一連の表明は，それらの論理積を取ったものとして解釈される．つまり，ここでの詳細化型の構文は集合の内包的表記法での $\{r \in \sigma \mid \gamma_1 \wedge \cdots \wedge \gamma_m\}$ に相当する．

■ 式

- **定数** c

  基本的には定数の型は基本型だけとするが，この制限は本質的ではない．

- **通常変数** x

  λ-計算での普通の変数である．

- **実現変数** r

  詳細化型の内部（中の表明中の式）でのみ使用できる．集合の内包的記法 $\{v \in S \mid P(v)\}$ に於ける v に相当する役割を持つ変数である．

- **関数抽象式 $\lambda x{:}\sigma.e$**

  通常の λ-計算と同様で，σ は変数 x を置換する引数の式が持たねばならない型を指定する．束縛変数 x として実現変数は指定できない．

- **関数適用式 $e_1 e_2$**

  通常の λ-計算と同様で，e_1 を関数として引数 e_2 に作用させる事を表わす．

- **レコード式 $\{l_1 = e_1, \ldots, l_n = e_n\}$**

  レコード型の値を構築する式である．

- **フィールド選択式 $e.l$**

  式 e が表わすレコード型の値 — ラベル付き直積集合の要素 — から指定されたフィールドラベル l に対応する直積成分の値を取り出す式である．

- **タグ付け式 $[\,l = e\,]$**

  式 e の値を指定されたタグラベル l に対する直和成分として直和集合へ入射する為の式である．

- **タグ選択式 $\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n$**

  直和集合の値 e_0 がどの直和成分 — ラベル l_i — であるかに応じて処理を振り分ける式である．なお，Standard ML での **case** 式でのパターンマッチングとは異なり，各 e_i $(1 \leq i \leq n)$ 自身が関数を表わす式でなければならない（この点に関しては後の「Funiq の構文に関する補足」の項を参照せよ）．

- **型抽象式 $\Lambda\, t{<}{:}\sigma.e$**

  型についてパラメタ化された多相値を構成する為の式である．型変数 t は型パラメタであり，σ は実引数として t に与え得る型の上限を規定する．

- **型適用式 $e[\sigma]$**

  式 e の表わす多相値を引数の具体的な型 σ により実体化する為の式である．

- **データ代数式 $\mathbf{pack}_{\exists t<:\sigma.\tau}\ e\ \mathbf{with}\ \rho$**

  抽象データ型の値の表現型 ρ とそれに基づく基本演算子群の実現の一揃い e（その型は $\tau[t := \rho]$ でなければならない）とを一纏めにして外部から隠蔽するモジュール — Mitchell and Plotkin (1985) での “data algebra” — を構成する式であり，そのモジュールの型は添字の抽象型である．

- **実現参照式　unpack e_1 as x with t in e_2**
  モジュール e_1 により隠蔽されていた抽象型の表現型と基本演算子群の実現の一揃いとを型変数 t と変数 x 各々の値として取り出し，本体式 e_2 で参照可能とする式である．

- **不動点再帰式　fix e**
  式 e は変域の型と値域の型とが同一の関数でなければならず，その時，この式は e の最小不動点を与える．即ち，この構文は Standard ML での **letrec** 式に相当し，関数の再帰的定義を可能にする．但し，後の補足にある通り，**letrec** 式とは異なり，再帰的に定義される関数名は本構文では束縛しない．

- **型制約式　e:σ**
  式 e の型を σ と指定する式であるが，後に述べる通り，この構文によって e の持っていた情報を「捨てる」事が可能となる．

■ 表明

- **原始表明　$e_1 \leqslant e_2 : \sigma$**
  直感的に言えば，左辺式 e_1 の計算が停止すれば右辺式 e_2 と同一の値を返さねばならないが左辺式の計算は停止しなくとも構わない，という左辺式 e_1 の値に対する部分正当性を表わす表明である．

- **全称化表明　forall x:σ.γ**
  型 σ の全ての値 x で表明 γ が満たされねばならない，という表明である．

■ Funiq の構文に関する補足

現実のプログラミング言語（を部分言語として含む広帯域言語）の為には，幾つかの構文で，直接に変数を束縛する — 束縛に関してはすぐ後に定義する — 形を採用した方が実際のプログラミングでは使い易く望ましい．

例えば，再帰的定義の為の **fix**　e の場合，関数の再帰的定義に用いる上での使い易さから言えば，当構文は **fix** $f{:}\sigma \to \sigma.F(f)$ の形の方が使い易い．これは，Standard ML での **letrec** 構文を用いた **letrec** $f = F(f)$ と同じ意味である．現在の **fix** 式で再帰的定義を行なうには，**fix** $(\lambda\, f{:}\sigma \to \sigma.F(f))$ と再帰的定義されるべき関数名 f で本体式 $F(f)$ を抽象化した λ-抽象を用いる必要がある．

もう一つは **case** 式である．選択肢を各タグラベル l_i のみで指定せず，これも Standard ML での **case** 式での様に **case** e_0 **of** $l_1(x_1)$ **then** $e_1, \ldots, l_n(x_n)$ **then** e_n とした方が遥かに使い易い（更に，Standard ML での様に各選択肢に変数 x_i だけでなくパターンを書ければ一層便利である）．何故ならば，現在の **case** 式では各選択肢の本体式 e_i $(1 \leq i \leq n)$ を明示的に λ-抽象化せねばならないからである．

以上の実用面での欠点があるにも拘らず現状の構文とした理由は，言語とその型システムを形式的体系として定義し性質をメタ理論的に分析する場合，束縛変数の存在は非常な邪魔となる — 定理等の証明で変数を束縛する構文に対しては細かい場合分けが必要となる事が多い — からであり，現状の構文での変数束縛は互いに独立な（他構文による束縛では代替できない）最小限の場合に限られている．

以下，本節では，今後の議論の為に，以上で概説した Funiq の構文に関する基本的概念を与える．まず，変数・型変数の出現の束縛／自由の概念を定義する．通常の λ-計算での α-変換同様，束縛変数は自在に変更可能とし，束縛変数の違いのみの式（或いは型・表明）同士は同一視する．

定義 2.2（出現の束縛／自由）

以下の場合に，型／通常／実現の各種変数は各々の型／式／表明で**束縛**されていると呼び，各々の指定部分での各変数の出現を**束縛された出現**であると言う．

(1) 範囲限定多相型 $\forall t{<:}\sigma.\tau$ の型変数 t の型 τ 中での出現；
(2) 範囲限定抽象型 $\exists t{<:}\sigma.\tau$ の型変数 t の型 τ 中での出現；
(3) 詳細化型 $\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}$ の実現変数 r の各表明 $\gamma_1, \ldots, \gamma_m$ 中での出現；
(4) λ-抽象式 $\lambda\, x{:}\sigma.e$ の通常変数 x の式 e 中での出現；
(5) 型抽象式 $\Lambda\, t{<:}\sigma.e$ の型変数 t の式 e 中での出現；
(6) 実現参照式 **unpack** e_1 **as** x **with** t **in** e_2 の通常変数 x と型変数 t との式 e_2 中での出現；
(7) 全称化表明 **forall** $x{:}\sigma.\gamma$ の通常変数 x の表明 γ 中での出現.

束縛された出現以外の各種変数の出現を**自由な出現**と呼ぶ.

以上の出現の束縛／自由の定義に基づき，Funiq の式に関する自由変数 — 通常／実現の種類の区別なく — の集合を以下の様に定義する．

定義 2.3（自由変数の集合）

e を Funiq の式とする時，e 中の**自由変数の集合** — $\mathbf{FV}(\boldsymbol{e})$ — を以下の様に定義する：

$$
\begin{aligned}
\mathrm{FV}(c) &= \varnothing,\\
\mathrm{FV}(x) &= \{x\},\\
\mathrm{FV}(r) &= \{r\},\\
\mathrm{FV}(\lambda x{:}\sigma.e) &= \mathrm{FV}(e) \setminus \{x\},\\
\mathrm{FV}(e_1\, e_2) &= \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2),\\
\mathrm{FV}(\{l_1 = e_1, \ldots, l_n = e_n\}) &= \bigcup_{i=1}^{n} \mathrm{FV}(e_i),\\
\mathrm{FV}(e.l) &= \mathrm{FV}(e),\\
\mathrm{FV}([\, l = e\,]) &= \mathrm{FV}(e),\\
\mathrm{FV}(\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n) &= \bigcup_{i=0}^{n} \mathrm{FV}(e_i),\\
\mathrm{FV}(\Lambda\, t{<:}\sigma.e) &= \mathrm{FV}(e),\\
\mathrm{FV}(e[\tau]) &= \mathrm{FV}(e),\\
\mathrm{FV}(\mathbf{pack}_{\exists t<:\rho.\sigma}\ e\ \mathbf{with}\ \tau) &= \mathrm{FV}(e),\\
\mathrm{FV}(\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2) &= \mathrm{FV}(e_1) \cup (\mathrm{FV}(e_2) \setminus \{x\}),\\
\mathrm{FV}(\mathbf{fix}\ e) &= \mathrm{FV}(e),\\
\mathrm{FV}(e{:}\sigma) &= \mathrm{FV}(e).
\end{aligned}
$$

γ を Funiq の表明とする時，γ 中の自由変数の集合 $\mathrm{FV}(\gamma)$ を以下の様に定義する：

$$
\begin{aligned}
\mathrm{FV}(e_1 \leqslant e_2 : \sigma) &= \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2),\\
\mathrm{FV}(\mathbf{forall}\ x{:}\sigma.\gamma) &= \mathrm{FV}(\gamma) \setminus \{x\}.
\end{aligned}
$$

同様に，自由型変数の集合を次の様に定義する．

定義 2.4（自由型変数の集合）

σ を Funiq の型とする時，σ 中の**自由型変数の集合** — $\mathbf{FTV}(\boldsymbol{\sigma})$ — を以下の様に定義する：

$$
\begin{aligned}
\mathrm{FTV}(\mathbf{Top}) &= \varnothing, \\
\mathrm{FTV}(\iota) &= \varnothing, \\
\mathrm{FTV}(t) &= \{t\}, \\
\mathrm{FTV}(\sigma_1 \to \sigma_2) &= \mathrm{FTV}(\sigma_1) \cup \mathrm{FTV}(\sigma_2), \\
\mathrm{FTV}(\{l_1 \colon \sigma_1, \ldots, l_n \colon \sigma_n\}) &= \bigcup_{i=1}^{n} \mathrm{FTV}(\sigma_i), \\
\mathrm{FTV}([\, l_1 \colon \sigma_1, \ldots, l_n \colon \sigma_n \,]) &= \bigcup_{i=1}^{n} \mathrm{FTV}(\sigma_i), \\
\mathrm{FTV}(\forall t{<:}\sigma.\tau) &= \mathrm{FTV}(\sigma) \cup (\mathrm{FTV}(\tau) \setminus \{t\}), \\
\mathrm{FTV}(\exists t{<:}\sigma.\tau) &= \mathrm{FTV}(\sigma) \cup (\mathrm{FTV}(\tau) \setminus \{t\}), \\
\mathrm{FTV}(\{r \colon \sigma \mid \gamma_1, \ldots, \gamma_m\}) &= \mathrm{FTV}(\sigma) \cup \bigcup_{i=1}^{m} \mathrm{FTV}(\gamma_i).
\end{aligned}
$$

ここで，表明 γ に対する $\mathrm{FTV}(\gamma)$ は以下の様に定義される：

$$
\begin{aligned}
\mathrm{FTV}(e_1 \leqslant e_2 : \sigma) &= \mathrm{FTV}(e_1) \cup \mathrm{FTV}(e_2) \cup \mathrm{FTV}(\sigma), \\
\mathrm{FTV}(\mathbf{forall}\ x \colon \sigma.\gamma) &= \mathrm{FTV}(\sigma) \cup \mathrm{FTV}(\gamma).
\end{aligned}
$$

更に，式 e に対する $\mathrm{FTV}(e)$ も同様に次の様に定義される：

$$
\begin{aligned}
\mathrm{FTV}(c) &= \varnothing, \\
\mathrm{FTV}(x) &= \varnothing, \\
\mathrm{FTV}(r) &= \varnothing, \\
\mathrm{FTV}(\lambda\, x \colon \sigma.e) &= \mathrm{FTV}(\sigma) \cup \mathrm{FTV}(e), \\
\mathrm{FTV}(e_1 e_2) &= \mathrm{FTV}(e_1) \cup \mathrm{FTV}(e_2), \\
\mathrm{FTV}(\{l_1 = e_1, \ldots, l_n = e_n\}) &= \bigcup_{i=1}^{n} \mathrm{FTV}(e_i),
\end{aligned}
$$

（次ページに続く）

定義 2.4（続き）

$$
\begin{aligned}
\mathrm{FTV}(e.l) &= \mathrm{FTV}(e),\\
\mathrm{FTV}([\,l = e\,]) &= \mathrm{FTV}(e),\\
\mathrm{FTV}(\textbf{case}\ e_0\ \textbf{of}\ l_1\ \textbf{then}\ e_1, \ldots, l_n\ \textbf{then}\ e_n) &= \bigcup_{i=0}^{n} \mathrm{FTV}(e_i),\\
\mathrm{FTV}(\Lambda\, t{<:}\sigma.e) &= (\mathrm{FTV}(e) \setminus \{t\}) \cup \mathrm{FTV}(\sigma),\\
\mathrm{FTV}(e[\tau]) &= \mathrm{FTV}(e) \cup \mathrm{FTV}(\tau),\\
\mathrm{FTV}(\textbf{pack}_{\exists t<:\rho.\sigma}\ e\ \textbf{with}\ \tau) &= \mathrm{FTV}(e) \cup \mathrm{FTV}(\tau),\\
\mathrm{FTV}(\textbf{unpack}\ e_1\ \textbf{as}\ x\ \textbf{with}\ t\ \textbf{in}\ e_2) &= \mathrm{FTV}(e_1) \cup (\mathrm{FTV}(e_2) \setminus \{t\}),\\
\mathrm{FTV}(\textbf{fix}\ e) &= \mathrm{FTV}(e),\\
\mathrm{FTV}(e{:}\,\sigma) &= \mathrm{FTV}(e) \cup \mathrm{FTV}(\sigma).
\end{aligned}
$$

Funiq に新たに追加された構文クラス **Assertion** の要素である表明が（全称化を許した）不等式の形であり代数仕様での様な等式でない理由は，Funiq に対してプログラミング言語と同様の表示的意味論を与える事を可能とする為の要請である．従って，この点に関しては第 5 章で議論する．

表明の形に関しては，上で述べた不等式である事の要請以外に，一定の構文的な制限がある．その制限を具体的に述べる為に Plotkin (1977) での活性部分式 (active subexpression) に類似した形式的厳格性という概念を導入する．

直感的には，変数 v が式 e で形式的厳格だ，とは v の値が未定義である（v の値を求めようとする計算が停止しない）ならば e 中の他の変数の値が何であろうと e 全体の値も未定義となる，という事を意味する．即ち，v が e で形式的厳格ならば，e の値を求める上で v の値への参照は不可避だという事が式 e の構造より確定している，という事である．以下の形式的厳格性の定義は，Funiq の意味論として式の値が必要となるまで式の評価を行なわない — 或る部分式の値が定義されなくとも式全体としては値が確定し得る — 名前呼び (call-by-name) 意味論を採用する場合に適した定義である．式の値の確定に全ての部分式の値の確定を必要条件とする値呼び (call-by-value) 意味論の場合，形式的厳格性の定義は変更されねばならない．以下，

基本的には名前呼び意味論に基づき考察する．故に，単に “Funiq” で参照する場合には名前呼び意味論に基づく計算体系を表わす事とし，値呼び意味論に基づく議論の場合は “$\text{Funiq}_{\perp}$” と明示する．まず，名前呼び意味論に対する形式的厳格性の定義を与える．今後，記号 “$\equiv$” を構文的な同一性を表わす記号として用いる．

定義 2.5 （形式的厳格性）

e を Funiq の式，$v \in \mathbf{Var} \cup \mathbf{IVar}$ を（通常もしくは実現）変数とする時，v が e で（或いは e が v に関して）**形式的厳格**であるとは，以下の何れかの条件が成立する事である：

(1) $e \equiv v$ である；
(2) $e \equiv \lambda\, x{:}\sigma.e_1$ の場合，$v \not\equiv x$ かつ v は e_1 で形式的厳格である；
(3) $e \equiv e_1 e_2$ の場合，v は e_1 で形式的厳格である；
(4) $e \equiv \{l_1 = e_1, \ldots, l_n = e_n\}$ の場合，v は $e_1, \ldots, e_n$ 全てで形式的厳格である；
(5) $e \equiv e_1.l$ の場合，v は e_1 で形式的厳格である；
(6) $e \equiv \mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n$ の場合，v は e_0 で形式的厳格である；
(7) $e \equiv \Lambda\, t{<:}\sigma.e_1$ の場合，v は e_1 で形式的厳格である；
(8) $e \equiv e_1[\tau]$ の場合，v は e_1 で形式的厳格である；
(9) $e \equiv \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2$ の場合，v は e_1 で形式的厳格である；
(10) $e \equiv \mathbf{fix}\ e_1$ の場合，v は e_1 で形式的厳格である；
(11) $e \equiv e_1{:}\tau$ の場合，v は e_1 で形式的厳格である．

一方，値呼びの $\text{Funiq}_{\perp}$ に対する形式的厳格性の定義は，名前呼びに対する上の定義を以下の様に変更する事で得られる．値呼びの場合，大堀 (1997) の§2.9.1 等での様に，関数抽象式（λ-抽象）内部の式は引数に適用されるまでは評価しない，として扱うのが普通である．しかし，本論文では，λ-抽象本体式の評価を遅らせるのは Abramsky (1990) や Ronchi Della Rocca and Paolini (2004) に従い**遅延 (lazy) 評**

価と称して値呼びとは独立な概念として扱い,「値呼び」とは遅延評価を前提としない評価方法を指す．以下，名前呼びの Funiq や値呼びの $\text{Funiq}_{\perp}$ に対し遅延評価を前提とする場合，各々， Funiq^{ℓ} や $\text{Funiq}^{\ell}_{\perp}$ と呼んで区別する．なお，遅延評価に対する形式的厳格性は，名前呼びの場合は定義 2.5 から (2) を除いて，また，値呼びの場合は定義 2.6 から (2) を除き (9′) は (9) のままとして，各々，定義される．

定義 2.6 （形式的厳格性 — 値呼び版）

値呼び意味論の為の形式的厳格性は，定義 2.5 の (3) と (9) との各々を

(3′) $e \equiv e_1 e_2$ の場合， v は e_1, e_2 の少なくとも一つで形式的厳格である；

(9′) $e \equiv \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2$ の場合， v は， e_1 で形式的厳格であるか，或いは， $v \not\equiv x$ かつ e_2 で形式的厳格である

と変更し，更に，以下の

(12) $e \equiv [\, l = e_1 \,]$ の場合， v は e_1 で形式的厳格である；

(13) $e \equiv \mathbf{pack}_{\exists t<:\rho.\sigma}\ e_1\ \mathbf{with}\ \tau$ の場合， v は e_1 で形式的厳格である

を追加して定義される．

より正確に言えば，定義 2.6 での形式的厳格性は値呼び評価戦略を前提とした定義なのに対して，定義 2.5 での定義は式 e に対して特定の評価戦略を前提としていない．故に，後者は e の評価に於いて変数 v の値が未定義である事に最も気付き難い名前呼び評価戦略にも対応できる，という事である．

なお，基本型に対する組み込みの基本演算子の呼出し（これも関数適用の一種）に関する形式的厳格性は次の様に考える． Funiq はベースの Fun と同様に Girard (1972, 1986) の 2 階の型付き λ-計算の体系 F を含む．故に， Böhm and Berarducci (1985) にある通り，自然数型や真理値型等の基本型を個別に追加せずとも多相型構成子 $\forall$ と関数型構成子 $\rightarrow$ とで定義でき，基本型上の基本演算（例えば自然数型の加減算）も二つの型構成子に関する式構文で定義可能である．

しかし，実用的プログラミング言語のモデル言語としては，幾つかの基本型やそれらの上の基本演算を Funiq の外側で定義し導入する事 — 実用プログラミング言語に於いて各基本型とその演算とが機械命令で直接実現される現実の反映 — に対する理論的裏付けを与える必要がある．そこで，それら個別に導入された基本演算の適用に

関しては，基本演算の厳格な引数位置に現れる変数は当該演算の呼出し式で形式的厳格と定める．例えば，nat を組み込みの自然数型とし，$+ : \mathrm{nat} \to (\mathrm{nat} \to \mathrm{nat})$ を組み込みの加法演算とする場合，nat 型の変数 v は $v + e$ や $e + v$ といった式（ここで e は nat 型の任意の式）で形式的厳格であるとして扱う．

以上で定義した形式的厳格性の概念を用いて詳細化型の中の表明の形に関する制限を記述するが，その為に必要となる詳細化型に関連する用語を定める．

定義 2.7（詳細化型に関する諸用語）

$\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}$ を任意の詳細化型とする時，

(1) 型 σ をこの詳細化型の**ベースの型**と呼ぶ．

(2) 詳細化型の “|” から “}” の間を詳細化型の**述語部**と呼ぶ．

(3) 述語部の各表明 γ_i $(1 \leq i \leq m)$ はこの詳細化型に**属する**と呼ぶ．

(3) の表明の所属に関しては，詳細化型の下位構文要素として詳細化型がある場合，下位の詳細化型に属する表明は上位のそれに属するとは**呼ばない**事を注意しておく．即ち，詳細化型に属する表明はその型の直接の構文要素の表明に限られる．

以上の準備により，詳細化型の良形性 (well-formedness) としての表明に関する制限は以下の様に規定する事ができる．

定義 2.8（詳細化型の良形性）

$\{r{:}\tau \mid \gamma_1, \ldots, \gamma_m\}$ $(m \geq 0)$ を詳細化型とする時，この詳細化型が**良形**であるとは，この詳細化型に属する各表明 γ_i $(1 \leq i \leq m)$ が以下の 2 条件を共に満たしている事である．以下，各表明 γ_i を，その一般形で

$$\gamma_i \equiv \mathbf{forall}\ x_{i,1}{:}\sigma_{i,1}.\ \ldots.\ \mathbf{forall}\ x_{i,n_i}{:}\sigma_{i,n_i}.\, e_i^{\mathrm{L}} \leqslant e_i^{\mathrm{R}} : \tau_i$$

$(n_i \geq 0)$ と表わす．

(a) 各表明 γ_i に現れる自由変数は，その表明が属する詳細化型で束縛されている実現変数 r のみである．即ち，$\mathrm{FV}(\gamma_i) = \{r\}$ である．

(b) 詳細化型で束縛された実現変数 r は，その詳細化型に直接に属する各表明 γ_i の左辺式 e_i^{L} で形式的厳格である．

以上の 2 条件の中で，(a) は証明論的な要請に基づく制限である．実際，この条件 (a) がなければ，詳細化型中の表明の左右両辺の式中に詳細化型の現れている箇所で可視な変数が自由変数として現れる事が許されてしまい，型が変数の値に依存する事が可能となる．その結果，一般の依存型を含んだ型理論の場合と同様に，型に関する推論規則は Funiq の操作的意味論を含んだ形とならざるを得ない．

一方，(b) は表示的意味論からの要請による．本論文の Funiq の場合，（最小）不動点再帰式 **fix** e により停止しない計算を許している．従って，Funiq に対する表示的意味を組み立てる素材としては，単純な集合論的な枠組でなく Scott 位相に基づく完備半順序集合を用いた領域論的な枠組を用いる必要がある．この領域論的な意味の構成の為に，条件 (b) として与えられた制限が不可欠となる．

以上の表明の形に関する制限を前提として，等式の形の表明を不等式の形の二つの表明の対を表わす簡略記法として以下の様に導入する．

記法 2.9 （等式表明）

二つの互いに対称的な（同じ形に全称化された）不等式の表明の対を（同じく全称化された）単一の等式の表明として表記する．即ち，

$$\begin{aligned}&\textbf{forall}\ x_1{:}\sigma_1.\ \ldots\ .\textbf{forall}\ x_n{:}\sigma_n.\, e = e' : \tau\\ &\qquad \overset{\text{abbrev}}{=\!=} \begin{cases}\textbf{forall}\ x_1{:}\sigma_1.\ \ldots\ .\textbf{forall}\ x_n{:}\sigma_n.\, e \leqslant e' : \tau,\\ \textbf{forall}\ x_1{:}\sigma_1.\ \ldots\ .\textbf{forall}\ x_n{:}\sigma_n.\, e' \leqslant e : \tau.\end{cases}\end{aligned}$$

本節の最後として，次節での Funiq の型システムに対する型理論の定義に於いて必要となる Funiq の構文上の注意を述べる．その注意とは，或る型 σ とそれを空に詳細化した型 $\{r{:}\sigma\ |\}$ とを同一視する，即ち，

$$\sigma \equiv \{r{:}\sigma\ |\}$$

という事である．無論，この場合，型の集合 **Type** の元として等しい — 即ち，“=” の関係にある — と謂っても等価であるが，メタ言語上で両者を全く区別しない，という事を強調する為に構文的同一性を表わす “≡” を使用する．

即ち，正確に言えば，Funiq での型の集合 **Type** とは，図 2.1 の生成規則で生成される集合ではなく，生成された集合を上記の同一性（および前に述べた束縛変数の違いを無視する事とフィールドラベル等の出現順序の違いを無視する事）という同値関係で割った商集合である．

2.2　型理論 FUNIQ — 広帯域言語 Funiq の型システム

本節では，広帯域言語 Funiq の型システムを一つの型理論の体系として公理と推論規則によって定義する．以下，Funiq の型システムに対する型理論の体系を FUNIQ と呼ぶ．なお，言語としての Funiq が Cardelli and Wegner の Fun を部分言語として含むのと同様，その型システムとしての型理論 FUNIQ も Fun の型システムを表わす型理論をその部分理論として含む．Fun の型システムに相当する部分型理論の体系は FUN で表わす．

以下，型理論 FUNIQ を定義するが，その為に必要な補助的な概念とそれに関連する記法について定義しておく．

定義 2.10（有限写像に関する記法）

X, Y を集合とし，f を X から Y への有限写像 — 即ち，f の値が定まる定義域は変域 X の有限部分集合で値域が Y なる部分写像 — とする時，

(1) f の定義域 — X の有限部分集合 — を $\mathrm{dom}(f)$ で表わす．

(2) f の像 —Y の有限部分集合 $f(\mathrm{dom}(f))$ — を $\mathrm{rng}(f)$ で表わす．

(3) $x \in X, y \in Y$ とする時，$f[\langle x, y\rangle]$ は次の有限写像を表わす：

$$f[\langle x, y\rangle](z) \triangleq \begin{cases} y, & (z = x) \\ f(z), & (z \neq x \text{ かつ } z \in \mathrm{dom}(f)) \\ \text{未定義}. & (z \neq x \text{ かつ } z \notin \mathrm{dom}(f)) \end{cases}$$

従って，その定義域は $\mathrm{dom}(f[\langle x, y\rangle]) = \{x\} \cup \mathrm{dom}(f)$ である．

(4) 空写像 — 即ち，その定義域が空集合の写像 — を $\varnothing$ で表わす．

定義 2.11（型付け基底）

(1) 通常変数に関する**型付け基底**とは，通常変数の集合 **Var** から型の集合 **Type** への有限写像である．通常変数に対する型付け基底を表わす為のメタ変数としては Γ を用いる．

(2) 実現変数に関する型付け基底も上の **Var** の代わりに **IVar** を用いる事で同様に定義され，メタ変数 Δ で表わす．

(3) Γ（或いは Δ）を通常変数（或いは実現変数）に関する任意の型付け基底とする時，有限写像 Γ（或いは Δ）を順序対の集合と看做した場合の要素対 $\langle x, \sigma\rangle \in \Gamma$（或いは $\langle r, \sigma\rangle \in \Delta$）を特に "$x : \sigma$"（或いは "$r : \sigma$"）という形で書き表わす．

(4) Γ（或いは Δ）を通常変数（或いは実現変数）に関する任意の型付け基底とする時，$\mathrm{FTV}(\Gamma)$ を以下の様に定義する（$\mathrm{FTV}(\Delta)$ も同様）：

$$\mathrm{FTV}(\Gamma) \triangleq \bigcup\{\mathrm{FTV}(\sigma) \mid \sigma \in \mathrm{rng}(\Gamma)\}.$$

定義 2.12（制約集合）

(1) 型変数に対する**制約集合**とは，型変数の集合 **TVar** の有限部分集合を定義域とし型の集合 **Type** を値域とする有限写像であり，メタ変数 C で表わす．

(2) C を型変数に対する任意の制約集合とする時，有限写像 C を順序対の集合と看做した場合の要素対 $\langle t, \sigma\rangle \in C$ を，特に "$t <: \sigma$" という形で書き表わし，型 σ を型変数 t の**上限の型**と呼ぶ．

(3) C を型変数に対する任意の制約集合とする時，$\mathrm{FTV}(C)$ を以下の様に定義する：

$$\mathrm{FTV}(C) \triangleq \mathrm{dom}(C) \cup \bigcup\{\mathrm{FTV}(\sigma) \mid \sigma \in \mathrm{rng}(C)\}.$$

なお，上の定義 2.12 (2) での「上限の型」に関しては注意が必要である．以下，その点について述べる．

Fun を始めとする多くの型付き λ-計算での部分型関係が半順序であるのに対し，Funiq の部分型関係 <: は，以下の FUNIQ の定式化から判る通り，擬順序である．例えば，型 Diag1 と Diag2 とを， $\mathrm{Diag1} \equiv \{\mathrm{p}\colon\{\mathrm{x}\colon \mathrm{nat}, \mathrm{y}\colon \mathrm{nat}\} \mid \mathrm{x} = \mathrm{y} : \mathrm{nat}\}$ と $\mathrm{Diag2} \equiv \{\mathrm{p}\colon\{\mathrm{x}\colon \mathrm{nat}, \mathrm{y}\colon \mathrm{nat}\} \mid \mathrm{x}+1 = \mathrm{y}+1 : \mathrm{nat}\}$ とする時， FUNIQ の公理と推論規則を用いれば，空の制約集合 $\varnothing$ の下で Diag1<:Diag2 と Diag2<:Diag1 とが共に成立している事が証明可能である．しかし， **Type** の元として $\mathrm{Diag1} \neq \mathrm{Diag2}$ である事は明らかである．

一般に， X を集合とし $\underset{\sim}{\sqsubseteq} \subseteq X \times X$ を X 上の擬順序とする時， $\underset{\sim}{\sqsubseteq}$ は次で定まる X 上の同値関係 $\simeq$ を誘導する：

$$\simeq \triangleq \{(x, y) \mid x, y \in X \text{ かつ } x \underset{\sim}{\sqsubseteq} y \text{ かつ } y \underset{\sim}{\sqsubseteq} x\}.$$

この同値関係による X の商集合 $X/\simeq$ を考える事により，元の X 上での擬順序 $\underset{\sim}{\sqsubseteq}$ から商集合 $X/\simeq$ 上での半順序（これも同じ $\underset{\sim}{\sqsubseteq}$ で表わす）が誘導される．

従って，定義 2.12 (2) での「上限の型」とは，正確には，擬順序としての **Type** 上の <: に関しての上限ではなく，以上の様にして誘導される商集合上の半順序としての <: についての型（の同値類）の上限という意味である．

定義 2.13 （型付け文脈）

型変数に対する制約集合 C，通常変数に対する型付け基底 Γ および実現変数に対する型付け基底 Δ からなる三つ組 C, Γ, Δ を**型付け文脈**と呼ぶ．

準備の最後として，各 $\mathcal{X}_i$ を式・型・型付け基底・制約集合の何れかとする時，$\mathrm{FTV}(\mathcal{X}_1, \ldots, \mathcal{X}_n)$ を $\mathrm{FTV}(\mathcal{X}_1) \cup \cdots \cup \mathrm{FTV}(\mathcal{X}_n)$ の略記法と定める．同様に，$\mathrm{FV}(e_1, \ldots, e_n)$ は $\mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n)$ を略したものとする．

以上で FUNIQ の公理と推論規則とを提示する上で必要な準備が揃ったので，本節の残りでそれらを示す．

$$\text{[S-TOP]}\quad \frac{}{C \ \triangleright\ \sigma <: \mathbf{Top}}$$

$$\text{[S-IDENTITY1]}\quad \frac{}{C \ \triangleright\ \iota <: \iota}$$

$$\text{[S-IDENTITY2]}\quad \frac{}{C \ \triangleright\ t <: t}$$

$$\text{[S-TVAR]}\quad \frac{}{C[t <: \sigma] \ \triangleright\ t <: \sigma}\qquad (t \notin \mathrm{FTV}(C))$$

$$\text{[S-WEAK]}^{\dagger}\quad \frac{C \ \triangleright\ \sigma <: \tau}{C[t <: \rho] \ \triangleright\ \sigma <: \tau}\qquad (t \notin \mathrm{FTV}(C))$$

$$\text{[S-TRANS]}\quad \frac{C \ \triangleright\ \sigma <: \tau \quad C \ \triangleright\ \tau <: \rho}{C \ \triangleright\ \sigma <: \rho}$$

$$\text{[S-ARROW]}\quad \frac{C \ \triangleright\ \tau_1 <: \sigma_1 \quad C \ \triangleright\ \sigma_2 <: \tau_2}{C \ \triangleright\ \sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2}$$

$$\text{[S-RECORD]}\quad \frac{C \ \triangleright\ \sigma_1 <: \tau_1 \ \ldots\ C \ \triangleright\ \sigma_n <: \tau_n}{C \ \triangleright\ \{l_1 : \sigma_1, \ldots, l_{n+m} : \sigma_{n+m}\} <: \{l_1 : \tau_1, \ldots, l_n : \tau_n\}}$$

$$\text{[S-VARIANT]}\quad \frac{C \ \triangleright\ \sigma_1 <: \tau_1 \ \ldots\ C \ \triangleright\ \sigma_n <: \tau_n}{C \ \triangleright\ [\, l_1 : \sigma_1, \ldots, l_n : \sigma_n \,] <: [\, l_1 : \tau_1, \ldots, l_{n+m} : \tau_{n+m} \,]}$$

$$\text{[S-ALL]}\quad \frac{C[t <: \rho] \ \triangleright\ \sigma <: \tau}{C \ \triangleright\ \forall t <: \rho.\sigma <: \forall t <: \rho.\tau}\qquad (t \notin \mathrm{FTV}(C))$$

$$\text{[S-EXIST]}\quad \frac{C[t <: \rho] \ \triangleright\ \sigma <: \tau}{C \ \triangleright\ \exists t <: \rho.\sigma <: \exists t <: \rho.\tau}\qquad (t \notin \mathrm{FTV}(C))$$

$$(*)\quad \text{[S-REFINE]}\quad \frac{C \ \triangleright\ \sigma <: \tau \qquad \left\{ \begin{array}{l} C \ \triangleright\ \sigma <: \tau \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_1 \\ \ldots \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_m \end{array} \right\} \vdash_{\mathrm{FUNIQ}} \bigwedge_{j=1}^{n} C, \varnothing, \{r : \tau\} \ \triangleright\ \delta_j}{C \ \triangleright\ \{r : \sigma \mid \gamma_1, \ldots, \gamma_m\} <: \{r : \tau \mid \delta_1, \ldots, \delta_n\}}$$

ここで，単一の前提の形の

$$\left\{ \begin{array}{l} C \ \triangleright\ \sigma <: \tau \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_1 \\ \ldots \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_m \end{array} \right\} \vdash_{\mathrm{FUNIQ}} \bigwedge_{j=1}^{n} C, \varnothing, \{r : \tau\} \ \triangleright\ \delta_j$$

は，実際には $j = 1, \ldots, n$ の各々について

$$\left\{ \begin{array}{l} C \ \triangleright\ \sigma <: \tau \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_1 \\ \ldots \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_m \end{array} \right\} \vdash_{\mathrm{FUNIQ}} C, \varnothing, \{r : \tau\} \ \triangleright\ \delta_j$$

という形の合計 n 個の前提の並びを表わす.

図 2.2　Funiq の部分型関係に関する公理と推論規則

形式化された理論としての型理論は（その型理論の公理と推論規則とを用いて導出可能な）判別式の集合であるが，型理論 FUNIQ の場合，その判別式は以下の三つの異なるクラスに分類される：

$$\begin{array}{rl} C \ \triangleright\ \sigma <: \tau & \text{部分型判別式；} \\ C, \Gamma, \Delta \ \triangleright\ e : \sigma & \text{型付け判別式；} \\ C, \Gamma, \Delta \ \triangleright\ \gamma & \text{表明判別式.} \end{array}$$

なお，型付け判別式 $C, \Gamma, \Delta \ \triangleright\ e : \sigma$ での式 e を型付け判別式の**主部**と呼ぶ.

以下，各々のクラス毎に，そのクラスの判別式を結論として導く公理・推論規則を示す.

まず，部分型判別式に関する公理・推論規則を図 2.2 に示す．図中で “†” を付けた [S-TWEAK] 則に関しては 3.1 節で議論する．また，図 2.1 での構文の場合と同様，FUNIQ 特有の公理や推論規則には “(∗)” の印を付した.

この図の部分型関係の公理・推論規則の中で FUN には存在せず FUNIQ 特有の推論規則は [S-REFINE] 則である．この規則が述べている直感的な意味は，或る詳細化型 ρ_1 のベースの型 σ が他方の詳細化型 ρ_2 のベースの型 τ の部分型であり，しかも ρ_1 の表明 $\gamma_1, \ldots, \gamma_m$ から ρ_2 の各表明 δ_i $(1 \leq j \leq n)$ を導く事ができる（即ち，$\gamma_1, \ldots, \gamma_m$ は $\delta_1, \ldots, \delta_n$ よりも論理的に強い）ならば，ρ_1 は ρ_2 の部分型だ，という事である．言い替えれば，**[S-REFINE] 則は表明の論理的な強度を部分型関係に反映する推論規則である.**

ここで，次章での議論の為に，[S-REFINE] 則に関連する用語を一つ定義する.

定義 2.14（表明強度の前提）

[S-REFINE] 則（およびその適用）で，表明間の導出可能性を求めている

$$\left\{ \begin{array}{l} C \ \triangleright\ \sigma <: \tau \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_1 \\ \cdots \\ C, \varnothing, \{r : \sigma\} \ \triangleright\ \gamma_m \end{array} \right\} \vdash_{\mathrm{FUNIQ}} C, \varnothing, \{r : \tau\} \ \triangleright\ \delta_j$$

という形の前提 $(1 \leq j \leq n)$ を**表明強度の前提**と呼ぶ.

$$\text{[T-CONST]}\quad \frac{}{C, \Gamma, \Delta \ \triangleright\ c_{ij} : \iota_i}$$

$$\text{[T-VAR]}\quad \frac{}{C, \Gamma[x : \sigma], \Delta \ \triangleright\ x : \sigma}$$

$$(*)\quad \text{[T-IVAR]}\quad \frac{}{C, \Gamma, \Delta[r : \sigma] \ \triangleright\ r : \sigma}$$

$$\text{[T-WEAK]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma}{C, \Gamma[x : \tau], \Delta \ \triangleright\ e : \sigma} \qquad (x \notin \mathrm{FV}(e))$$

$$(*)\quad \text{[T-IWEAK]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma}{C, \Gamma, \Delta[r : \tau] \ \triangleright\ e : \sigma} \qquad (r \notin \mathrm{FV}(e))$$

$$\text{[T-TWEAK]}^{\ddagger}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma}{C[t <: \tau], \Gamma, \Delta \ \triangleright\ e : \sigma} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma, \tau))$$

$$\text{[T-SUBTYPE]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma \quad C \ \triangleright\ \sigma <: \tau}{C, \Gamma, \Delta \ \triangleright\ e : \tau}$$

$$\text{[T-ABS]}\quad \frac{C, \Gamma[x : \sigma], \Delta \ \triangleright\ e : \tau}{C, \Gamma, \Delta \ \triangleright\ (\lambda\, x{:}\sigma.e) : \sigma \to \tau} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{[T-APP]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 : \sigma \to \tau \quad C, \Gamma, \Delta \ \triangleright\ e_2 : \sigma}{C, \Gamma, \Delta \ \triangleright\ (e_1 e_2) : \tau}$$

$$\text{[T-RECORD]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 : \sigma_1 \ \ldots\ C, \Gamma, \Delta \ \triangleright\ e_n : \sigma_n}{C, \Gamma, \Delta \ \triangleright\ \{l_1 = e_1, \ldots, l_n = e_n\} : \{l_1 : \sigma_1, \ldots, l_n : \sigma_n\}}$$

$$\text{[T-SELECT]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \{l_1 : \sigma_1, \ldots, l_n : \sigma_n\}}{C, \Gamma, \Delta \ \triangleright\ e.l_i : \sigma_i} \qquad (1 \le i \le n)$$

$$\text{[T-VARIANT]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma}{C, \Gamma, \Delta \ \triangleright\ [\, l = e\,] : [\, l{:}\sigma\,]}$$

$$\text{[T-CASE]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_0 : [\, l_1 : \sigma_1, \ldots, l_n : \sigma_n\,] \quad C, \Gamma, \Delta \ \triangleright\ e_1 : \sigma_1 \to \tau \ \ldots\ C, \Gamma, \Delta \ \triangleright\ e_n : \sigma_n \to \tau}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n) : \tau}$$

$$\text{[T-TABS]}\quad \frac{C[t <: \sigma], \Gamma, \Delta \ \triangleright\ e : \tau}{C, \Gamma, \Delta \ \triangleright\ (\Lambda\, t{<:}\sigma.e) : \forall t{<:}\sigma.\tau} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma))$$

$$\text{[T-TAPP]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \forall t{<:}\rho.\tau \quad C \ \triangleright\ \sigma <: \rho}{C, \Gamma, \Delta \ \triangleright\ e[\sigma] : \tau[t := \sigma]}$$

$$\text{[T-PACK]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \tau[t := \rho] \quad C \ \triangleright\ \rho <: \sigma}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{pack}_{\exists t <: \sigma.\tau}\ e\ \mathbf{with}\ \rho) : \exists t{<:}\sigma.\tau} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma, \rho))$$

$$\text{[T-UNPACK]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 : \exists t{<:}\sigma.\tau \quad C[t <: \sigma], \Gamma[x : \tau], \Delta \ \triangleright\ e_2 : \rho}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2) : \rho} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma, \rho))$$

$$\text{[T-FIX]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma \to \sigma}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{fix}\ e) : \sigma}$$

$$\text{[T-RESTRICT]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma \quad C \ \triangleright\ \sigma <: \tau}{C, \Gamma, \Delta \ \triangleright\ (e{:}\tau) : \tau}$$

$$(*)\quad \text{[T-REFINE]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma \quad C, \Gamma, \Delta \ \triangleright\ \gamma_1[r := e] \ \ldots\ C, \Gamma, \Delta \ \triangleright\ \gamma_m[r := e]}{C, \Gamma, \Delta \ \triangleright\ e : \{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}}$$

図 2.3　Funiq の型付けに関する公理と推論規則

次に，判別式の第二のクラスの型付け判別式に関する公理・推論規則を図 2.3 に示す．図中で “‡” を付けた [T-TWEAK] 則に関しては 3.1 節で議論する．また，図 2.2 と同様， FUNIQ 特有の公理や推論規則には “(∗)” の印を付した．

図 2.3 の型付け判別式に関する公理と推論規則で FUNIQ 特有のものとして “(∗)” が付されているのは [T-IVAR] 則, [T-IWEAK] 則, [T-REFINE] 則の三つである．この中で [T-IVAR] 則と [T-IWEAK] 則とは， Funiq には Fun に存在しない変数のクラスとしての実現変数が存在する為であり，これら二つの規則は，各々，通常変数に関する [T-VAR] 則と [T-WEAK] 則とに対応している．故に， [T-IVAR] 則と [T-IWEAK] 則は FUNIQ 特有ではあるが内容的に特別な意義を持つものではない．

一方， [T-REFINE] 則は Funiq が Fun にはなかった表明を型に導入した事により必要となった規則であるので，内容的に本質的な意義を持つ．この [T-REFINE] 則が表わしている直感的な内容は以下の通りである．即ち，型付け文脈 C, Γ, Δ の下で式 e の型が σ で，しかも実現変数 r を唯一の自由変数とする表明 γ_i $(1 \leq i \leq m)$ 中の当該自由変数 r を式 e で置換した表明 $\gamma_i[r := e]$ の各々が同じ型付け文脈下で導出可能ならば，式 e は元の型 σ をそれらの表明 $\gamma_1, \ldots, \gamma_m$ により詳細化された型をも持つ，という事である．言い替えると， **[T-REFINE] 則は表明を満たすという式に関する論理的な性質を型の世界に反映する推論規則である．**

最後に第三のクラスの判別式である表明判別式を結論として与える公理・推論規則を図 2.4 に示す．なお，このクラスの判別式は FUN には存在しない（Fun には表明がないから）ので，この図の全ての公理や推論規則は FUNIQ 特有である．従って，同図では FUNIQ 特有である事を示す印の “(∗)” は省いている．

図中の表明判別式に関する公理や推論規則の多くは図 2.3 に示した型付け判別式に関するものと対応するものである．即ち，規則名の頭の “T-” と “A-” を除いて同じ名前の規則は対応している．また， “[A-β_{xxx}]” という名前の推論規則は Funiq での操作的意味論での簡約規則（つまり値の計算規則）に対応するものである．これら “[A-β_{xxx}]” 則とは向きが逆の不等式の表明判別式を帰結部とする推論規則を許さない理由は FUNIQ に関する証明論にある．従って，この事は次章で議論する．

$$\text{[A-CONST]}\quad \frac{}{C, \Gamma, \Delta \ \triangleright\ c_{ij} \leqslant c_{ij} : \iota_i}$$

$$\text{[A-VAR]}\quad \frac{}{C, \Gamma[x:\sigma], \Delta \ \triangleright\ x \leqslant x : \sigma}$$

$$\text{[A-IVAR]}\quad \frac{}{C, \Gamma, \Delta[r:\sigma] \ \triangleright\ r \leqslant r : \sigma}$$

$$\text{[A-TRANS]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_2 : \sigma \quad C, \Gamma, \Delta \ \triangleright\ e_2 \leqslant e_3 : \sigma}{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_3 : \sigma}$$

$$\text{[A-WEAK]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_2 : \sigma}{C, \Gamma[x:\tau], \Delta \ \triangleright\ e_1 \leqslant e_2 : \sigma} \qquad (x \notin \mathrm{FV}(e_1, e_2))$$

$$\text{[A-IWEAK]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_2 : \sigma}{C, \Gamma, \Delta[r:\tau] \ \triangleright\ e_1 \leqslant e_2 : \sigma} \qquad (x \notin \mathrm{FV}(e_1, e_2))$$

$$\text{[A-TWEAK]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_2 : \sigma}{C[t <: \tau], \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_2 : \sigma} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma, \tau))$$

$$\text{[A-SUBTYPE]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_2 : \sigma \quad C \ \triangleright\ \sigma <: \tau}{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_2 : \tau}$$

$$\text{[A-}\mathbf{forall}\text{-I]}\quad \frac{C, \Gamma[x:\sigma], \Delta \ \triangleright\ \gamma}{C, \Gamma, \Delta \ \triangleright\ \mathbf{forall}\ x{:}\,\sigma.\,\gamma} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{[A-}\mathbf{forall}\text{-E]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ \mathbf{forall}\ x{:}\,\sigma.\,\gamma \quad C, \Gamma, \Delta \ \triangleright\ e : \sigma}{C, \Gamma, \Delta \ \triangleright\ \gamma[x := e]}$$

$$\text{[A-}\beta_{\to}\text{]}\quad \frac{C, \Gamma[x:\sigma], \Delta \ \triangleright\ e_1 : \tau \quad C, \Gamma, \Delta \ \triangleright\ e_2 : \sigma}{C, \Gamma, \Delta \ \triangleright\ (\lambda\, x{:}\,\sigma.e_1)e_2 \leqslant e_1[x := e_2] : \tau} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{[A-ABS]}\quad \frac{C, \Gamma[x:\sigma], \Delta \ \triangleright\ e \leqslant e' : \tau}{C, \Gamma, \Delta \ \triangleright\ (\lambda\, x{:}\,\sigma.e) \leqslant (\lambda\, x{:}\,\sigma.e') : \sigma \to \tau} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{[A-APP]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_1' : \sigma \to \tau \quad C, \Gamma, \Delta \ \triangleright\ e_2 \leqslant e_2' : \sigma}{C, \Gamma, \Delta \ \triangleright\ (e_1 e_2) \leqslant (e_1' e_2') : \tau}$$

$$\text{[A-}\beta_{\{\ldots\}}\text{]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 : \sigma_1 \ \ldots\ C, \Gamma, \Delta \ \triangleright\ e_n : \sigma_n}{C, \Gamma, \Delta \ \triangleright\ \{l_1 = e_1, \ldots, l_n = e_n\}.l_i \leqslant e_i : \sigma_i} \qquad (1 \le i \le n)$$

$$\text{[A-RECORD]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_1' : \sigma_1 \ \ldots\ C, \Gamma, \Delta \ \triangleright\ e_n \leqslant e_n' : \sigma_n}{C, \Gamma, \Delta \ \triangleright\ \{l_1 = e_1, \ldots, l_n = e_n\} \leqslant \{l_1 = e_1', \ldots, l_n = e_n'\} : \{l_1 : \sigma_1, \ldots, l_n : \sigma_n\}}$$

$$\text{[A-SELECT]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \{l_1 : \sigma_1, \ldots, l_n : \sigma_n\}}{C, \Gamma, \Delta \ \triangleright\ e.l_i \leqslant e'.l_i : \sigma_i} \qquad (1 \le i \le n)$$

$$\text{[A-}\beta_{[\ldots]}\text{]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma_i \quad C, \Gamma, \Delta \ \triangleright\ e_1 : \sigma_1 \to \tau \ \ldots\ C, \Gamma, \Delta \ \triangleright\ e_n : \sigma_n \to \tau}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{case}\,[\,l_i = e\,]\,\mathbf{of}\,l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n) \leqslant (e_i e) : \tau} \qquad (1 \le i \le n)$$

$$\text{[A-VARIANT]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \sigma}{C, \Gamma, \Delta \ \triangleright\ [\,l = e\,] \leqslant [\,l = e'\,] : [\,l{:}\,\sigma\,]}$$

$$\text{[A-CASE]}\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_0 \leqslant e_0' : [\,l_1 : \sigma_1, \ldots, l_n : \sigma_n\,] \quad C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_1' : \sigma_1 \to \tau \ \ldots\ C, \Gamma, \Delta \ \triangleright\ e_n \leqslant e_n' : \sigma_n \to \tau}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n) \leqslant (\mathbf{case}\ e_0'\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1', \ldots, l_n\ \mathbf{then}\ e_n') : \tau}$$

図 2.4　Funiq の表明に関する公理と推論規則 (1/2)

$$[\text{A-}\beta_\forall]\quad \frac{C[t <: \sigma], \Gamma, \Delta \ \triangleright\ e : \tau \quad C \ \triangleright\ \rho <: \sigma}{C, \Gamma, \Delta \ \triangleright\ (\Lambda\, t{<:}\sigma.e)[\rho] \leqslant e[t := \rho] : \tau[t := \rho]} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta))$$

$$[\text{A-TABS}]\quad \frac{C[t <: \sigma], \Gamma, \Delta \ \triangleright\ e \leqslant e' : \tau}{C, \Gamma, \Delta \ \triangleright\ (\Lambda\, t{<:}\sigma.e) \leqslant (\Lambda\, t{<:}\sigma.e') : \forall t{<:}\sigma.\tau} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma))$$

$$[\text{A-TAPP}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \forall t{<:}\rho.\tau \quad C \ \triangleright\ \sigma <: \rho}{C, \Gamma, \Delta \ \triangleright\ e[\sigma] \leqslant e'[\sigma] : \tau[t := \sigma]}$$

$$[\text{A-}\beta_\exists]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 : \tau_1[t := \rho] \quad C, \Gamma[x : \tau_1[t := \rho]], \Delta \ \triangleright\ e_2 : \tau_2 \quad C \ \triangleright\ \rho <: \sigma}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{unpack}\ (\mathbf{pack}_{\exists t<:\sigma.\tau_1}\ e_1\ \mathbf{with}\ \rho)\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2) \leqslant e_2[t := \rho][x := e_1] : \tau_2} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma, \rho))$$

$$[\text{A-PACK}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \tau[t := \rho] \quad C \ \triangleright\ \rho <: \sigma}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{pack}_{\exists t<:\sigma.\tau}\ e\ \mathbf{with}\ \rho) \leqslant (\mathbf{pack}_{\exists t<:\sigma.\tau}\ e'\ \mathbf{with}\ \rho) : \exists t{<:}\sigma.\tau} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma, \rho))$$

$$[\text{A-UNPACK}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e_1 \leqslant e_1' : \exists t{<:}\sigma.\tau \quad C[t <: \sigma], \Gamma[x : \tau], \Delta \ \triangleright\ e_2 \leqslant e_2' : \rho}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2) \leqslant (\mathbf{unpack}\ e_1'\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2') : \rho} \qquad (t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma, \rho))$$

$$[\text{A-}\beta_{\mathrm{fix}}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma \to \sigma}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{fix}\ e) \leqslant e(\mathbf{fix}\ e) : \sigma}$$

$$[\text{A-FIX}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \sigma \to \sigma}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{fix}\ e) \leqslant (\mathbf{fix}\ e') : \sigma}$$

$$[\text{A-LEAST}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \sigma}{C, \Gamma, \Delta \ \triangleright\ (\mathbf{fix}\ \lambda\, x{:}\sigma.x) \leqslant e : \sigma}$$

$$[\text{A-RESTRICT}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \sigma \quad C \ \triangleright\ \sigma <: \tau}{C, \Gamma, \Delta \ \triangleright\ (e{:}\,\tau) \leqslant (e'{:}\,\tau) : \tau}$$

$$[\text{A-ASSERT}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e : \{r{:}\,\sigma \mid \gamma_1, \ldots, \gamma_m\}}{C, \Gamma, \Delta \ \triangleright\ \gamma_i[r := e]} \qquad (1 \leq i \leq m)$$

$$[\text{A-REFINE}]\quad \frac{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \sigma \quad \begin{array}{ccc} C, \Gamma, \Delta \ \triangleright\ \gamma_1[r := e] & & C, \Gamma, \Delta \ \triangleright\ \gamma_m[r := e] \\ C, \Gamma, \Delta \ \triangleright\ \gamma_1[r := e'] & \cdots & C, \Gamma, \Delta \ \triangleright\ \gamma_m[r := e'] \end{array}}{C, \Gamma, \Delta \ \triangleright\ e \leqslant e' : \{r{:}\,\sigma \mid \gamma_1, \ldots, \gamma_m\}}$$

図 2.4　Funiq の表明に関する公理と推論規則 (2/2)

[A-**forall**-I] 則と [A-**forall**-E] 則とは表明に対する **forall**- 全称化の導入と除去であり，通常の述語論理に対する自然演繹の体系での ∀- 導入と ∀- 除去に相当する常識的な推論規則である．

以上の何れのカテゴリにも属さない推論規則として残っているのは [A-ASSERT] 則である．この推論規則が表わしている直感的な内容は，型付け文脈 C, Γ, Δ の下で

式 e の型が表明 $\gamma_1, \ldots, \gamma_m$ を述語部に持つ詳細化型（その実現変数は r）である時，式 e は各表明 γ_i $(1 \leq i \leq m)$ を満たす — 即ち，同じ型付け文脈下で $\gamma_i[r := e]$ という表明を導出して良い — という事である．言い替えると，**[A-ASSERT] 則は表明を満たすという式に関する論理的な性質を式の型から抽出する推論規則であり，[T-REFINE] 則とは逆の役割を果たす．**

なお，次章の議論で用いる為に，FUNIQ 特有の推論規則である [S-REFINE] 則と [T-REFINE] 則とに関連した用語を定義しておく．

定義 2.15（主前提）

(1) [S-REFINE] 則（の適用）に於いて，$C \triangleright \sigma <: \tau$ に相当する部分型判別式の形の前提を本推論規則（の適用）の**主前提**と呼ぶ．

(2) [T-REFINE] 則（の適用）に於いて，$C, \Gamma, \Delta \triangleright e : \sigma$ に相当する型付け判別式の形の前提を本推論規則（の適用）の**主前提**と呼ぶ．

以上で型理論 FUNIQ の公理と推論規則とによる定式化が完了した．部分型理論としての FUN は，図 2.2 と図 2.3 の中の “(*)” が付されていない公理・推論規則のみとして定式化されるものである．

型理論 FUNIQ の体系がその部分型理論 FUN の体系と決定的に異なる点は，各種の判別式の導出の為の規則群が**相互再帰的に依存している**点である．即ち，Fun の型システムである部分型理論 FUN では型付け判別式の導出は [T-SUBTYPE] 則により部分型判別式に関する図 2.2 の規則群に依存しているが，部分型判別式の導出は図 2.3 に示した型付け判別式に関する規則群には依存していない．従って，FUN の場合，その公理と推論規則とを図 2.2 と図 2.3 に相当する形で分割し提示する事には論理的な意味がある．

一方，本論文の型理論 FUNIQ の場合，部分型判別式の導出は [S-REFINE] 則の存在の為に表明判別式に関する図 2.4 の規則群に依存している．また，型付け判別式の導出は，FUN 同様に部分型判別式に関する図 2.2 の規則群に依存しているだけでなく，[T-REFINE] 則により図 2.4 の表明判別式に関する規則群にも依存する．更に，表明判別式の導出は，[A-SUBTYPE] 則と [A-ASSERT] 則とにより，各々，

図 2.2 の部分型判別式に関する規則群と図 2.3 の型付け判別式に関する規則群とに依存している．この様に，FUNIQ では，各々のクラスの判別式の導出は他のクラスの判別式の導出に相互依存している．従って，FUNIQ の場合は，各クラスの判別式に関する規則群を図 2.2 ～ 図 2.4 に分けて与える事には FUN の場合の様な論理的な意味はなく読み易さの為という全く便宜的な意義しか持たず，**図 2.2 ～ 図 2.4 に分割して示した規則群は相互に依存し合う分割不可能な単一の体系を成している．**

後の使用の為，これら二つの型理論での導出可能性を表わす記法を導入する．

記法 2.16 （導出可能性の記号）

(1) 図 2.2 ～ 図 2.4 で示した規則群による導出可能性を “$\vdash_{\mathrm{FUNIQ}}$” により表わす．

(2) 図 2.2 と図 2.3 の規則群の中の “(∗)” を持たない規則群だけでの導出可能性を “$\vdash_{\mathrm{FUN}}$” により表わす．

(3) 前後の文脈から何れの体系での導出可能性なのかが明らかな場合は，体系を示す添字を省略して単に “$\vdash$” と書く．

以上の記法を用いると，部分型理論 FUN は導出可能な判別式の集合として次の様に定義される．

定義 2.17 （部分型理論 FUN）

$$\begin{aligned}\mathrm{FUN} \triangleq\ & \{C \triangleright \sigma <: \tau \mid \vdash_{\mathrm{FUN}} C \triangleright \sigma <: \tau \text{ かつ} \\ & \qquad \sigma, \tau \in \mathbf{Type}_{\mathrm{Fun}} \text{ かつ } \mathrm{rng}(C) \subseteq \mathbf{Type}_{\mathrm{Fun}}\} \\ & \cup \{C, \Gamma, \varnothing \triangleright e : \sigma \mid \vdash_{\mathrm{FUN}} C, \Gamma, \varnothing \triangleright e : \sigma \text{ かつ } e \in \mathbf{Exp}_{\mathrm{Fun}} \text{ かつ} \\ & \qquad \sigma \in \mathbf{Type}_{\mathrm{Fun}} \text{ かつ } \mathrm{rng}(\Gamma) \subseteq \mathbf{Type}_{\mathrm{Fun}}\}.\end{aligned}$$

前に述べた通り，Funiq での部分型関係 $<:$ は擬順序である．そこで，それから誘導される同値関係の為の記号を導入して，本節を終わる．

記法 2.18（同等な型）

$\sigma <: \tau$ かつ $\tau <: \sigma$ である事を $\sigma \simeq \tau$ で表わす．例えば，$\vdash C \triangleright \sigma \simeq \tau$ とは $\vdash C \triangleright \sigma <: \tau$ かつ $\vdash C \triangleright \tau <: \sigma$ を意味する．

2.3　Funiq による抽象データ型の記述 — Fun での問題の解決

本節では，Funiq で抽象データ型がどの様な形で記述されるかを例題を中心に示し，その過程で，1.3 節で述べた Fun での — 即ち，Cardelli-Mitchell-Plotkin-Wegner のアプローチによる — 抽象データ型のモデル化の問題点として挙げた例，つまり FIFO キューと LIFO スタックとの区別が型の違いとして表わせないという課題が不等式の形での表明を追加された Funiq でどの様に解決されるかを示す．最後に，代数仕様での最も基本的な形のモジュールが Funiq での型としてどの様に記述されるかを一般形として与える．

まず，以下の例題の記述を読み易くする為の記法を約束しておく．

記法 2.19（フィールド選択式の略記法）

ベースの型がレコード型である詳細化型 $\{r: \{l_1: \sigma_1, \ldots, l_m: \sigma_m\} \mid \gamma_1, \ldots, \gamma_n\}$ $(m \geq 1, n \geq 1)$ の各表明 γ_i $(1 \leq i \leq n)$ 中では，実現変数 r の各フィールド l_j $(1 \leq j \leq m)$ に対するフィールド選択式 $r.l_j$ の代わりに，何ら修飾されていない単なるフィールドラベル l_j そのものを用いる事を許す．

以上の略記法の下では，或る詳細化型がレコード型の詳細化である場合，何ら修飾されていない（ベースの型としてのレコード型の）フィールドラベル l が式が現れるべき箇所に出現しているならば，この式 l はその詳細化型の実現変数について形式的厳格である事を注意しておく．

```
type StackOpImpl = {aStackOpImpl : {new: StackValRep,
                                    isnew: StackValRep → bool,
                                    push: nat → StackValRep → StackValRep,
                                    top: StackValRep → nat,
                                    pop: StackValRep → StackValRep}
                   | forall i: nat. forall s: StackValRep. top(push(i)(s)) ⩽ i : nat,
(*push-pop*)         forall i: nat. forall s: StackValRep.
                       pop(push(i)(s)) ⩽ s : StackValRep,
                     forall i: nat. forall s: StackValRep. isnew(push(i)(s)) ⩽ false : bool,
                     isnew(new) ⩽ true : bool};
```

図 2.5　Funiq での LIFO スタックの実現型

本論文の Funiq で記述したスタックの実現型を図 2.5 に示す．図中の詳細化型 StackOpImpl の各表明が定義 2.8 の両条件 (a), (b) を共に満たしている事を確認しておく．例えば，最後の表明

$$\mathrm{isnew(new)} \leqslant \mathrm{true} : \mathrm{bool}$$

について考えると，これは，記法 2.19 での約束により，省略しない本来の形は，

$$\mathrm{aStackOpImpl.isnew(aStackOpImpl.new)} \leqslant \mathrm{true} : \mathrm{bool}$$

である．この不等式の表明全体に含まれている自由変数は aStackOpImpl のみであるが，これは詳細化型 StackOpImpl の実現変数そのものである．故に，この表明は条件 (a) を満たしている．それと同時に，この表明の不等式の左辺式 aStackOpImpl.isnew(aStackOpImpl.new) で実現変数 aStackOpImpl は形式的に厳格である．より詳細に述べれば，aStackOpImpl は定義 2.5 の (1) よりそれ自身に関して形式的に厳格である．故に，同じく (5) より aStackOpImpl は aStackOpImpl.isnew で形式的に厳格である．従って，(3) により aStackOpImpl は

左辺式全体でも形式的に厳格である．以上から，当表明は条件 (b) を満たしている．同様にして，他の表明に関しても条件 (a), (b) を共に満足している事が判る．

記法 2.19 で許された省略を行なっている場合，代数仕様のモジュールに対応する — その対応は本節の後半で一般形として示す — 形の詳細化型に対して，一般に次の命題が成立する．

命題 2.20

任意に与えられた詳細化型 σ が以下の 3 条件

(1) σ のベースの型 ρ はレコード型である；
(2) σ の各不等式表明の左辺は関数適用の形である；
(3) それら関数適用の形の左辺式それぞれの関数部分はベースの型 ρ のフィールドラベルである

を満たしているとする．この時，その詳細化型 σ は定義 2.8 の条件 (b) を満たす．

証明

上の例での場合と全く同様に，形式的厳格性の定義 2.5 での (1), (3), (5) を用いれば良い．

証明終

実際，「フィールドラベル」と「レコード型」とを，各々，「演算子記号」と「シグニチャの演算子部分」と読み替えれば， Ehrig and Mahr (1985) の教科書での全ての例での各等式（の等号を適当な — 殆どの場合は $\leqslant$ の — 向きの不等号に弱めた不等式）が上の条件 (2) と (3) とを満たしている事を確認できる．

次に図 2.6 に示す型 StackoidOpImpl （スタックもどきの実現型）を考える．元のスタックの実現型 StackOpImpl での表明 ($*$push-pop$*$) は明らかに新たな型 StackoidOpImpl での対応する表明 ($*\text{push}^2\text{-pop}^2*$) よりも強い．従って，任意のスタックの実現はスタックもどきの実現として用いる事ができる．言い替えれば，型 StackOpImpl は型 StackoidOpImpl の構造を継承していると言える．

$$
\begin{array}{ll}
\textbf{type}\ \mathrm{StackoidOpImpl} = \{\mathrm{aStackoidOpImpl} : \{\mathrm{new}\colon \mathrm{StackValRep}, & \\
\qquad \mathrm{isnew}\colon \mathrm{StackValRep} \to \mathrm{bool}, & \\
\qquad \mathrm{push}\colon \mathrm{nat} \to \mathrm{StackValRep} \to \mathrm{StackValRep}, & \\
\qquad \mathrm{top}\colon \mathrm{StackValRep} \to \mathrm{nat}, & \\
\qquad \mathrm{pop}\colon \mathrm{StackValRep} \to \mathrm{StackValRep}\} & \\
\quad | \ \textbf{forall}\ \mathrm{i}\colon \mathrm{nat}.\ \textbf{forall}\ \mathrm{s}\colon \mathrm{StackValRep}.\, \mathrm{top}(\mathrm{push}(\mathrm{i})(\mathrm{s})) \leqslant \mathrm{i} : \mathrm{nat}, & \\
(*\mathrm{push}^2\text{-}\mathrm{pop}^2*) \quad \textbf{forall}\ \mathrm{i}\colon \mathrm{nat}.\ \textbf{forall}\ \mathrm{j}\colon \mathrm{nat}.\ \textbf{forall}\ \mathrm{s}\colon \mathrm{StackValRep}. & \\
\qquad \mathrm{pop}(\mathrm{pop}(\mathrm{push}(\mathrm{i})(\mathrm{push}(\mathrm{j})(\mathrm{s})))) \leqslant \mathrm{s} : \mathrm{StackValRep}, & \\
\quad \textbf{forall}\ \mathrm{i}\colon \mathrm{nat}.\ \textbf{forall}\ \mathrm{s}\colon \mathrm{StackValRep}.\, \mathrm{isnew}(\mathrm{push}(\mathrm{i})(\mathrm{s})) \leqslant \mathrm{false} : \mathrm{bool}, & \\
\quad \mathrm{isnew}(\mathrm{new}) \leqslant \mathrm{true} : \mathrm{bool}\}; &
\end{array}
$$

図 2.6　Funiq でのスタックもどきの実現型

この事実は，FUNIQ に於いては $\varnothing \ \triangleright\ \mathrm{StackOpImpl} <: \mathrm{StackoidOpImpl}$ という部分型判別式が証明可能である事により表現され，この判別式が表わす部分型関係の証明には [S-REFINE] 則が不可欠である．

以下，1.2 節で指摘した Fun ではスタックの実現型とキューの実現型とを区別できないという問題が Funiq の型システム FUNIQ でどの様に解決されるかを示す．

なお，以下の例題記述では **if** 式を用いる．そこで，bool 型を可変型として定義し，**if** 式を bool 型の為の拡張記法として与えておく（なお，FUNIQ では $\varnothing, \varnothing, \varnothing \ \triangleright\ \{\} : \{\}$ が証明可能 — 即ち，$\vdash_{\mathrm{FUNIQ}} \varnothing, \varnothing, \varnothing \ \triangleright\ \{\} : \{\}$ — であるので，次の定義 2.21 で **if** 式の定義を与えている右辺の式は正しく型付け可能である事を予め断わっておく）．

定義 2.21（bool 型と関連構文）

$$
\begin{array}{rl}
\mathrm{bool} & \triangleq [\,\mathrm{trueTag}\colon \{\}, \mathrm{falseTag}\colon \{\}\,], \\
\mathrm{true} & \triangleq [\,\mathrm{trueTag} = \{\}\,], \\
\mathrm{false} & \triangleq [\,\mathrm{falseTag} = \{\}\,],
\end{array}
$$

$$
\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2 \triangleq \textbf{case}\ e_0\ \textbf{of}\ \mathrm{trueTag}\ \textbf{then}\ \lambda\, x\colon \{\}.e_1, \mathrm{falseTag}\ \textbf{then}\ \lambda\, x\colon \{\}.e_2.
$$

```
type QueueOpImpl = {aQueueOpImpl : {new: QueueValRep,
                                    isnew: QueueValRep → bool,
                                    push: nat → QueueValRep → QueueValRep,
                                    top: QueueValRep → nat,
                                    pop: QueueValRep → QueueValRep}
                   | forall i: nat. forall s: QueueValRep.
                       top(push(i)(s)) ⩽ if isnew(s) then i else top(s) : nat,
                     forall i: nat. forall s: QueueValRep.
                       pop(push(i)(s)) ⩽ if isnew(s) then s
                                         else push(i)(pop(s)) : QueueValRep,
                     forall i: nat. forall s: QueueValRep. isnew(push(i)(s)) ⩽ false : bool,
                     isnew(new) ⩽ true : bool};
```

図 2.7 Funiq での FIFO キューの実現型

図 2.7 に示した型 QueueOpImpl の定義を見ると，シグニチャ（つまりベースの型としてのレコード型のフィールドラベルとその型）に関しては前の StackOpImpl と全く同一（図 1.2 より QueueValRep = List[nat] = StackValRep と両者に対する表現型は同一である事に注意）であり，両者は表明に関して違うだけである．明らかに，

$$\nvdash_{\mathrm{FUNIQ}} \varnothing \triangleright \mathrm{StackOpImpl} <: \mathrm{QueueOpImpl}$$

かつ

$$\nvdash_{\mathrm{FUNIQ}} \varnothing \triangleright \mathrm{QueueOpImpl} <: \mathrm{StackOpImpl}$$

であるが，更に， 1.2 節の図 1.6 のスタックの正しい実現の式に関しては，

$$\vdash_{\mathrm{FUNIQ}} \varnothing, \varnothing, \varnothing \triangleright \mathrm{aStackOpImpl} : \mathrm{StackOpImpl}$$

である．一方，図 1.7 のキューの実現となっている FIFO に振舞う式に対しては，

$$\vdash_{\mathrm{FUNIQ}} \varnothing, \varnothing, \varnothing \triangleright \mathrm{anotherStackOpImpl} : \mathrm{QueueOpImpl}$$

である．この事は Cardelli-Mitchell-Plotkin-Wegner による抽象データ型のモデル化での問題が FUNIQ に於いて解決されている事を示している．

```
type QueueOpImpl2 = {aQueueOpImpl : {new: QueueValRep,
                                     isnew: QueueValRep → bool,
                                     push: Nat → QueueValRep → QueueValRep,
                                     top: QueueValRep → Nat,
                                     pop: QueueValRep → QueueValRep}
                   | forall i: Nat. forall s: QueueValRep.
                       top(push(i)(s)) = if isnew(s) then i else top(s) : Nat,
                     forall i: Nat. forall s: QueueValRep.
                       pop(push(i)(s)) = if isnew(s) then s
                                         else push(i)(pop(s)) : QueueValRep,
                     forall i: Nat. forall s: QueueValRep. isnew(push(i)(s)) ⩽ false : bool,
                     isnew(new) ⩽ true : bool};
```

図 2.8　Funiq での FIFO キューの実現型（一部の表明を等式に強めたもの）

なお，上の QueueOpImpl に関して補足すると，その表明の中の二つ（最初のと 2 番目のと）は，実は，記法 2.9 の意味での等式表明に強める事が可能である．それら二つの表明を等式に強めたものを図 2.8 に示す．

最初の表明が等式となる事が許される理由は次の通りである．右辺式は **if** 式で，その条件式は isnew(s) である．記法 2.19 によれば，この式 “isnew(s)” の本来の形は “aQueueOpImpl.isnew(s)” である．故に定義 2.5 (5) より，この式は詳細化型の実現変数 aQueueOpImpl に関して形式的厳格である．さて，定義 2.21 での **if** 式の定義から **if** 式での条件式は **case** 式での値がテストされる式（図 2.1 での “e_0”）であるので，実現変数 aQueueOpImpl は定義 2.5 (6) より **if** 式全体 — 右辺式 — で形式的に厳格である．故に，最初の表明の右辺式は定義 2.8 の条件 (b) を満たす．従って，不等式の左右両辺を入れ替えた逆向きの表明も良形である．

さて，記法 2.9 によれば，各等式表明は二つの互いに逆向きの不等式表明の対であったので， QueueOpImpl2 は QueueOpImpl の表明と全ての演算とを継承した型である．従って， [S-REFINE] により，

$$\varnothing \ \triangleright\ \text{QueueOpImpl2} <: \text{QueueOpImpl}$$

を証明する事ができる．

```
val aPartialQueueOpImpl = {new = nil,
                           isnew = λ s: QueueValRep. isnull(s),
                           push = λ i: nat. λ s: QueueValRep. cons(i)(s),
                           top = fix(λ t: QueueValRep → nat. t),
                           pop = fix(λ p: QueueValRep → QueueValRep. p)};
```

図 2.9 FIFO キューの演算の不完全な実現

図 1.7 に示した FIFO キューの演算の正しい実現の式 anotherStackOpImpl は，当然ながら，この強められた型をも持つ．即ち，

$$\vdash_{\text{FUNIQ}} \varnothing, \varnothing, \varnothing \triangleright \text{anotherStackOpImpl} : \text{QueueOpImpl2}.$$

一方，弱い方の型 QueueOpImpl を持つ式の中には強い方の型 QueueOpImpl2 を持つ事ができない式も存在する．その実例を図 2.9 に示す．型理論 FUNIQ では，

$$\vdash_{\text{FUNIQ}} \varnothing, \varnothing, \varnothing \triangleright \text{aPartialQueueOpImpl} : \text{QueueOpImpl}$$

である．その理由は pop フィールドの式 $\mathbf{fix}(\lambda\, \text{p: QueueValRep} \to \text{QueueValRep. p})$ は [A-LEAST] 則から StackValRep → StackValRep 型のどんな式 — とりわけ anotherStackOpImpl での pop フィールドの式 — よりも $\leqslant$ の意味で小さいからである． top フィールドの式に関しても同様である．その結果， [A-APP] 則，[A-TRANS] 則，および幾つか他の推論規則を用いて上の判別式の証明を行なう事ができる．

一方，この式 aPartialQueueOpImpl が等式表明を用いた強い方の型 QueueOpImpl2 を持つ事は示せない．即ち，

$$\nvdash_{\text{FUNIQ}} \varnothing, \varnothing, \varnothing \triangleright \text{aPartialQueueOpImpl} : \text{QueueOpImpl2}$$

である．**形式化された型理論 FUNIQ は否定の概念を含まないので，特定の判別式が成立しない事を FUNIQ の内部で示す事は不可能である**が，メタ理論での分析から上の型付け判別式が導出不能である事は容易に示す事ができる．

```
type Stack = ∃stackValRep<:Top.{aStackOpImpl : {new: stackValRep,
                                                isnew: stackValRep → bool,
                                                push: nat → stackValRep → stackValRep,
                                                top: stackValRep → nat,
                                                pop: stackValRep → stackValRep}
                              | forall i: nat. forall s: stackValRep. top(push(i)(s)) ⩽ i : nat,
                                forall i: nat. forall s: stackValRep.
                                    pop(push(i)(s)) ⩽ s : stackValRep,
                                forall i: nat. forall s: stackValRep. isnew(push(i)(s)) ⩽ false : bool,
                                isnew(new) ⩽ true : bool};
```

図 2.10 抽象データ型としての LIFO スタックの Funiq での型

詳細化型に基づいて抽象データ型を構成するには，値の集合を表わしていた表現型を型変数に置き換え，その型変数について存在限量化を行なえば良い．例えば，LIFO スタックの場合，抽象データ型としての型は図 2.10 に示した通りである．ここで，これまでは表現型であった StackValRep が型変数 stackValRep となり ∃ により束縛されている事がポイントである．なお，上の Stack 型を持つ式は，例えば，$\mathbf{pack}_{\mathrm{Stack}}$ aStackOpImpl **with** StackValRep である．

以上の LIFO スタックの抽象データ型の例から判る通り，代数仕様のモジュールで宣言されたソートは Funiq では抽象型（∃ 型）での束縛型変数になる．その対応を含め，代数仕様の最も基本的な形（幾つかのソートと演算子記号を宣言してそれらに関する等式公理群を与えた形）は次ページの図 2.11 に示す一般的な形で Funiq での型として表現できる．ここで，“$s(\_,\_)$”，“$t(\_)$”，“$v(\_)$”（次に示すベクトル風の表現を展開すると本来は “$v(\_,\_)$” という 2 引数），“$ol(\_)$”，“$or(\_)$” は番号の付け直し付けを表わす（s, t, v の値域は $\{1,\ldots,m\}$, ol と or の値域は $\{1,\ldots,n\}$）．また，“$\overline{x_i}$” は通常変数の列 “$x_{i,1},\ldots,x_{i,q_i}$” を表わし，“$\mathbf{forall}\,\overline{x_i : S_{v(i)}}$” は “$\mathbf{forall}\,x_{i,1} : S_{v(i,1)}.\ \ldots\ .\mathbf{forall}\,x_{i,q_i} : S_{v(i,q_i)}$” という全称束縛列である．“$F_{i,j}[\overline{x_i}]$” は高々 $\{\overline{x_i}\}$ のみを自由変数として含む式を表わす．最後に “$E_i[\overline{x_i}]$” も同様に高々 $\{\overline{x_i}\}$ のみを自由変数として含む式を表わすが，最外演算子記号がモジュールで宣言されている演算子記号 $Op_1,\ldots,Op_n$ の何れでもないという条件が付いている．

module M **is**
sorts
$S_1, \ldots, S_m$;
operations
$Op_1 : S_{s(1,1)}, \ldots, S_{s(1,p_1)} \to S_{t(1)}$,
$\vdots$
$Op_n : S_{s(n,1)}, \ldots, S_{s(n,p_n)} \to S_{t(n)}$;
axioms
$\forall \overline{x_1 : S_{v(1)}}.\ Op_{ol(1)}(F_{1,1}[\overline{x_1}], \ldots, F_{1,p_{ol(1)}}[\overline{x_1}]) = Op_{or(1)}(F_{1,1}[\overline{x_1}], \ldots, F_{1,p_{or(1)}}[\overline{x_1}])$,
$\vdots$
$\forall \overline{x_k : S_{v(k)}}.\ Op_{ol(k)}(F_{k,1}[\overline{x_k}], \ldots, F_{k,p_{ol(k)}}[\overline{x_k}]) = Op_{or(k)}(F_{k,1}[\overline{x_k}], \ldots, F_{k,p_{or(k)}}[\overline{x_k}])$,
$\forall \overline{x_{k+1} : S_{v(k+1)}}.\ Op_{ol(k+1)}(F_{k+1,1}[\overline{x_{k+1}}], \ldots, F_{k+1,p_{ol(k+1)}}[\overline{x_{k+1}}]) = E_{k+1}[\overline{x_{k+1}}]$,
$\vdots$
$\forall \overline{x_{k+l} : S_{v(k+l)}}.\ Op_{ol(k+l)}(F_{k+l,1}(\overline{x_{k+l}}), \ldots, F_{k+l,p_{ol(k+l)}}[\overline{x_{k+l}}]) = E_{k+l}[\overline{x_{k+l}}]$
end module;

$\Downarrow$

type $M =$
$\exists S_1 <: \mathbf{Top}. \ldots \exists S_m <: \mathbf{Top}.$
$\{r : \{ Op_1 : S_{s(1,1)} \cdots S_{s(1,p_1)} \to S_{t(1)},$
$\vdots$
$Op_n : S_{s(n,1)} \cdots S_{s(n,p_n)} \to S_{t(n)}$
$\}$
$\mid$ **forall** $\overline{x_1 : S_{v(1)}}.\ Op_{ol(1)}(F_{1,1}[\overline{x_1}]) \ldots (F_{1,p_{ol(1)}}[\overline{x_1}]) \leqslant Op_{or(1)}(F_{1,1}[\overline{x_1}]) \ldots (F_{1,p_{or(1)}}[\overline{x_1}]) : S_{t(ol(1))}$,
forall $\overline{x_1 : S_{v(1)}}.\ Op_{or(1)}(F_{1,1}[\overline{x_1}]) \ldots (F_{1,p_{or(1)}}[\overline{x_1}]) \leqslant Op_{ol(1)}(F_{1,1}[\overline{x_1}]) \ldots (F_{1,p_{ol(1)}}[\overline{x_1}]) : S_{t(ol(1))}$,
$\vdots$
forall $\overline{x_k : S_{v(k)}}.\ Op_{ol(k)}(F_{k,1}[\overline{x_k}]) \ldots (F_{k,p_{ol(k)}}[\overline{x_k}]) \leqslant Op_{or(k)}(F_{k,1}[\overline{x_k}]) \ldots (F_{k,p_{or(k)}}[\overline{x_k}]) : S_{t(ol(k))}$,
forall $\overline{x_k : S_{v(k)}}.\ Op_{or(k)}(F_{k,1}[\overline{x_k}]) \ldots (F_{k,p_{or(1)}}[\overline{x_k}]) \leqslant Op_{ol(k)}(F_{k,1}[\overline{x_k}]) \ldots (F_{k,p_{ol(k)}}[\overline{x_k}]) : S_{t(ol(k))}$,
forall $\overline{x_{k+1} : S_{v(k+1)}}.\ Op_{ol(k+1)}(F_{k+1,1}[\overline{x_{k+1}}]) \ldots (F_{k,p_{ol(k+1)}}[\overline{x_{k+1}}]) \leqslant E_{k+1}[\overline{x_{k+1}}] : S_{t(ol(k+1))}$,
$\vdots$
forall $\overline{x_{k+l} : S_{v(k+l)}}.\ Op_{ol(k+l)}(F_{k+l,1}[\overline{x_{k+l}}]) \ldots (F_{k,p_{ol(k+l)}}[\overline{x_{k+l}}]) \leqslant E_{k+l}[\overline{x_{k+l}}] : S_{t(ol(k+l))}$
$\}$;

図 2.11　基本的な形の代数仕様モジュールの Funiq での表現の一般形式

```
module Stack is
  import
    Bool, Nat;
  sorts
    stack;
  operations
    new : stack,
    isnew : stack → bool,
    push : stack nat → stack,
    top : stack → nat,
    pop : stack → stack;
  axioms
    ∀ i: nat. ∀ s: stack. pop(push(i)(s)) = s,
    ∀ i: nat. ∀ s: stack. top(push(i)(s)) = i,
    ∀ i: nat. ∀ s: stack. isnew(push(i)(s)) = false,
    isnew(new) = true
end module;
```

図 2.12　抽象データ型としての LIFO スタックの代数仕様

図 2.11 での一般形の対応で注目すべきは，代数仕様での最初の k 個の等式は右辺もそのモジュールで宣言された演算子記号 — $Op_{or(i)}$ $(1 \leq i \leq k)$ — が最外演算子記号となっている式なので，それらの等式には Funiq の型で二つの逆向きの不等式表明の対が対応しているのに対して，$k+1$ 番目以降の等式の場合，それらの右辺は当該モジュールで宣言された演算子記号を最外演算子記号として含まない式 — $E_i[\overline{x_i}]$ $(k+1 \leq i \leq k+l)$ — であるので，Funiq の型で単一の不等式表明に対応付けられている事である．

なお，図 2.11 に於ける代数仕様モジュールと Funiq の抽象型との間の対応の一般形の提示は，代数仕様における他モジュールの輸入とか限定された名前（ソート名や演算子記号）に限っての輸出とかを含まない最も基本的なモジュール・スキーマに関してである．

代数仕様の場合，組み込みのソートとか演算子という概念はない．無論，実際の代数仕様の処理系には最低限必要な一連のモジュールが最初から組み込まれているが，

それらを仕様記述で用いる場合，ユーザが記述したモジュールと同様に明示的に輸入する必要がある．従って，現実の代数仕様では，殆ど常に真理値に関するソート名と基本演算子記号を提供する Bool とか自然数に関する Nat といったモジュールを輸入する必要がある．

他モジュールの輸入を含んだ場合の代数仕様モジュールの Funiq での抽象型との対応を一般形として扱うには図 2.11 よりも遥かに錯綜とした形となる．しかし，その場合にも，両者の対応のポイントは，上で述べた通り，代数仕様の等式公理で左右両辺共に最外演算子記号が当該モジュールで宣言されている場合は Funiq での等式表明（逆向きの不等式表明の対）に対応し，そうでない場合には単一の不等式表明に対応する，という事である．

以上の輸入の問題が省略されている事を考慮して抽象データ型としての（要素が自然数の）スタックの代数仕様を記述している図 2.12 を前に示した図 2.10 の Funiq の抽象型としてのスタックの記述と見比べると，両者の対応が図 2.11 で示した対応の一般形に従っている事が理解できる．

本節の最後として，型制約式 $e{:}\sigma$ の機能について補足しておく．代数仕様の場合にシグニチャや等式公理をモジュール外部から参照可能とするか否かの輸出制御は，仕様 — つまり Funiq での型 — のレベルでのみ行なわれる． Funiq の場合，型レベルでの輸出制御は，レコード型の間の部分型関係に関する [S-RECORD] 則と詳細化型の間の部分型関係に関する [S-REFINE] 則とによって可能である．

更に， Funiq の場合，輸出制御は型＝仕様レベルのみならず，実現レベルでも可能となっている．それを許しているのが，型制約式 $e{:}\sigma$ である．この式中の e がレコード型をベースの型とする詳細化型を持つ式である場合，この式の型付けに関する [T-RESTRICT] 則と上で述べた [S-RECORD] 則ならびに [S-REFINE] 則により，$e{:}\sigma$ という構成は e のシグニチャや表明の一部分を捨てる事を可能とする．

即ち，非常に単純な構成である型制約式 $e{:}\sigma$ は，**実現のレベルでの輸出制御**を達成する目的に使用できる．

2.4 Funiq の詳細型の例

前節の様々な例では，代数仕様と詳細化型の対応を議論する為，専らレコード型をベースの型とする詳細化型のみを扱って来た．しかしながら， Funiq の詳細化型は

もっと一般的な型構成法である．本章の最後として，この事を具体例によって示し，同時に，詳細化型での不等式表明が部分正当性しか表現できないという限界に起因する注意点にも触れておく．

例えば，直積型構成子 $\times$ とその二つの射影演算 "_.1", "_.2" が用意されており，基本型として real が存在とした時，

$$\{\text{plainPoint}\colon \text{real} \times \text{real} \mid \text{plainPoint.1} = \text{plainPoint.2} : \text{real}\}$$

は x-y 平面上の右上がりの対角線 — $y = x$ — 上の点の集合を表わす型であり，

$$\{\text{plainPoint}\colon \text{real} \times \text{real} \mid \text{plainPoint.2} = (\text{plainPoint.1})^2 : \text{real}\}$$

は x-y 平面上で言えば $y = x^2$ のグラフを表わす型である．より一般には，$f(x)$ を x の多項式とする時，

$$\{\text{plainPoint}\colon \text{real} \times \text{real} \mid \text{plainPoint.2} = f[(\text{plainPoint.1})/x] : \text{real}\}$$

は x-y 平面上での $y = f(x)$ のグラフの点の集合を表わす型である（$f(x)$ が x の有理式でも同様）．ここで，"$f[(\text{plainPoint.1})/x]$" は，$x$ に関する多項式 f の中の変数 x の全てを plainPoint.1 という式で置換した式である．これらの表明が等式であるのは，実数の四則演算や自然数冪が引数の実数に関して厳格であるので，2.1 節の定義 2.6 の直後での組み込み演算の呼出しに対する形式的厳格性の扱いに関しての説明通り，それら実数演算の呼出しを形式的に厳格として取り扱っている為である．

以上の直積の例は前節でのレコード型に対する詳細化型としても表現できるので，より毛色の異なったものとして関数型をベースの型とする詳細化型の例を挙げよう．関数の合成演算子を $\circ : \forall t{<:}\mathbf{Top}.(t \to t) \to (t \to t) \to (t \to t)$ で表わし，それを型 σ に具現化した $\circ[\sigma]$ を内置記法の 2 項演算子に相応しい様に "$\circ_\sigma$" と書くと，関数の合成演算子 $\circ_\sigma$ は左側の引数に関して厳格であるので，

$$\{\text{f}\colon \text{nat} \to \text{nat} \mid \text{f} \circ_{\text{nat}} \text{f} = \text{f} : \text{nat} \to \text{nat}\}$$

と，等式表明の形にする事が可能で，上の型は自然数上の冪等な関数の型を表わす．

次も関数型をベースとする詳細化型の例である．

$$\{\mathrm{f}\colon \mathrm{nat} \to \mathrm{nat} \mid \textbf{forall}\ \mathrm{m}\colon \mathrm{nat}.\ \textbf{forall}\ \mathrm{n}\colon \mathrm{nat}.\ \mathrm{f}(\mathrm{m}) \leq \mathrm{f}(\mathrm{m}+\mathrm{n}) \leqslant \mathrm{true} : \mathrm{bool}\}$$

は自然数型 nat の上の単調な部分関数 — 即ち，定義されている自然数の上では単調に振舞う関数 — の型である．但し，この型の代わりに

$$\{\mathrm{f}\colon \mathrm{nat} \to \mathrm{nat} \mid \textbf{forall}\ \mathrm{m}\colon \mathrm{nat}.\ \mathrm{f}(\mathrm{m}) \leq \mathrm{f}(\mathrm{m}+1) \leqslant \mathrm{true} : \mathrm{bool}\}$$

と書くと，これは全く異なる別の型になってしまう．実際，後者の型だと定義区間 — 即ち，全ての点で関数が定義されている区間 — 毎には局所的に単調だが大域的には単調とは限らない関数を許してしまう．例えば，f が後者の型を持つ関数だとし，$1, 3 \in \mathrm{dom}(\mathrm{f})$ であるが $2 \notin \mathrm{dom}(\mathrm{f})$ である場合，$\mathrm{f}(1) > \mathrm{f}(3)$ となる事が許される．その理由は，$\mathrm{f}(1) \leq \mathrm{f}(2)$ も $\mathrm{f}(2) \leq \mathrm{f}(3)$ も共に未定義となるので両方の bool 型の不等式共に true よりも $\leqslant$ の意味で小さくなり，その結果，後者の型の表明を満たしてしまう事による．

以上の奇妙さは Funiq での不等式表明が部分正当性しか表わせない事からの帰結である．その為，単調性の様な**大域的な性質を詳細化型の表明として表現する時には充分な注意が必要である．**

最後に，Fun で非詳細化型を多相型や抽象型での型変数に対する上限の型として用いる事ができるのと同様，Funiq では詳細化型をもそれらの上限の型として使用できる事を注意しておく．この事は，例えば，束縛型変数の上限が詳細化型である様な多相型を持つ式に対しては，型変数の表わす型が一定の表明を満たす事を利用して式の性質を証明しても構わない，という事を意味する．つまり，詳細化型を上限とする型変数を用いる事によって，代数仕様での部分仕様の輸入に相当する効果を達成できるという事である．

第3章

型理論FUNIQの証明論的性質

— 保存的拡大性を中心に —

本章では，広帯域言語 Funiq の型システムの形式化として 2.2 節で定義した型理論 FUNIQ の証明論的な性質を検討する．

まず， Mitchell and Plotkin による抽象型のモデル化でのポイントである $\exists$ が多相型構成子 $\forall$ と関数型構成子 $\rightarrow$ とでコード化できる事を示す．この事は Prawitz により 2 階の命題論理での論理演算子の表現可能性として古くから知られている事実の応用である．しかし，限量化の上限に制約のある場合に関しては完全な証明が与えられた事はない．事実，上限に制約のある抽象型を上記の二つの型構成子でコード化する為には，コード化に伴い導入される新たな型変数を扱う必要上，従来，知られていなかった推論規則を追加する必要のある事が判明する．

次は本章の中心課題であり，ベース言語である Fun の型理論 FUN に対して FUNIQ が保存的拡大となっている事を示す．この性質は，本論文での型システム FUNIQ が FUN の良い性質を乱す事なく素直に引き継いでいる事を保証する．

第三に， FUNIQ の型付け判別式が，何故，独立な種類の判別式として扱われねばならないか，つまり，型付け判別式を表明判別式の特殊な形として扱うのは何故いけないのか，という理由を検討する．この検討は，同時に， FUNIQ の証明論的性質に関して今後の検討課題として残されているものを明らかにする．

最後に，本論文での詳細化型の導入方法が充分な一般性を持つ方法であるか否かを検討する．

3.1　範囲限定抽象型の多相型と関数型とによるコード化

前章では範囲限定抽象型 $\exists t{<:}\_.\_$ を独立した型構成方法として扱った．実際，実用的な観点からは，抽象型は情報隠蔽という他の型構成方法で作られる型とは全く異なる用途を持っているので，それを独立した型構成方法として取り扱うのは極めて妥当である．しかし，理論的な分析に於いては，例えば或る言明を型の構文に関する構造的帰納法での場合分けによって証明する状況を考えれば判る通り，独立した型構成方法が少なければ少ない程，分析は簡潔に行なえる．そこで，本節では，範囲限定抽象型 $\exists t{<:}\_.\_$ の範囲限定多相型 $\forall t{<:}\_.\_$ と関数型 $\_ \to \_$ とによるコード化に関して検討する．

このコード化そのものは古くから良く知られた事実である．実際， 2 階の命題論理に於いての 2 階の存在限量子 $\exists$ の 2 階の全称限量子 $\forall$ と含意 $\supset$ とによるコード化については，例えば，自然演繹法に関する代表的文献である Prawitz (1965) で具体的にコード化方法が与えられている．また，プログラミング言語論の世界でも例えば Reynolds (1983) は抽象型が多相型と関数型とでコード化可能な事を示した．

しかしながら，抽象型に対応する式構文（**pack** 式と **unpack** 式）の側のコード化が多相型と関数型とに対応する式構文によって具体的にどの様に行なわれ，それが型の側のコード化に基づく推論規則の組み立てに対して健全となるか，という事は，調べ得た限りでは具体的に証明された事は皆無である．更に， Fun の様な束縛型変数の取り得る型の範囲限定を許す場合，抽象型構成子を多相型構成子と関数型構成子とで具体的にコード化できる事を示した論文も存在していない．

それ以上に重要な事は，限量化での範囲限定を許す場合，範囲限定のない場合に知られている上記のコード化を実際に適用すると，コード化の為に導入される新しい型変数を扱う為に，従来は知られていない推論規則が不可欠である事が判明する．

そこで，本節では，範囲限定を許した抽象型 $\exists$ と関連する式構文を多相型 $\forall$ と関数型 $\to$ （やそれらに対応する式構文）とによって具体的にコード化を行ない，そのコード化が前章で示した部分型関係や型付けに関する推論規則に関して整合的である事を示し，更に，その証明の中で，従来は知られていなかった新たな推論規則が必要となる事を示す．

定義 3.1（抽象型の多相型と関数型とによるコード化）

抽象型 $\exists t{<:}\sigma.\tau$ と関連する式構文である **pack** 式および **unpack** 式を以下の様に定義する：

$$\exists t{<:}\sigma.\tau \triangleq \forall s{<:}\mathbf{Top}.((\forall t{<:}\sigma.(\tau \to s)) \to s),$$

$$\mathbf{pack}_{\exists t{<:}\sigma.\tau}\ e\ \mathbf{with}\ \rho \triangleq \Lambda\, s{<:}\mathbf{Top}.\lambda\, x{:}\forall t{<:}\sigma.(\tau \to s).x[\rho]e,$$

$$\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2 \triangleq e_1[\rho](\Lambda\, t{<:}\sigma.\lambda\, x{:}\tau.e_2).$$

ここで，最後の式の中の型 ρ は式 e_2 の型であり，型 τ は式 e_1 の型 $\exists t{<:}\sigma.\tau$ 中のそれである．

抽象型の多相型と関数型とによるコード化を定義 3.1 に示す．以下，前章で与えた抽象型に関する様々な推論規則がこのコード化に対して矛盾していない事を示す．

定理 3.2（抽象型のコード化の整合性）

前章での抽象型に関する推論規則 [S-EXIST]，[T-PACK]，[T-UNPACK]，[A-$\beta_\exists$]，[A-PACK]，[A-UNPACK] の何れもは，上のコード化の下で他の公理や推論規則によって導出可能である．

証明

各々の推論規則に対する導出を具体的に与える事によって証明する．なお，導出木中の “†” と “‡” とは，各々，[S-WEAK] 則と [T-TWEAK] 則との適用である（これらの推論規則の適用に関しては，本証明の後の議論を参照されたい）．

[S-EXIST] 則の導出：

$$\dfrac{\dfrac{\dfrac{C[t<:\sigma] \triangleright \tau <: \tau'}{C[s<:\mathbf{Top}][t<:\sigma] \triangleright \tau <: \tau'}(\dagger) \quad \dfrac{}{C[s<:\mathbf{Top}][t<:\sigma] \triangleright s <: s}}{C[s<:\mathbf{Top}][t<:\sigma] \triangleright \tau' \to s <: \tau \to s} \quad \dfrac{}{C[s<:\mathbf{Top}] \triangleright s <: s}}{\dfrac{C[s<:\mathbf{Top}] \triangleright \forall t<:\sigma.(\tau' \to s) <: \forall t<:\sigma.(\tau \to s)}{C[s<:\mathbf{Top}] \triangleright ((\forall t<:\sigma.(\tau \to s)) \to s) <: ((\forall t<:\sigma.(\tau' \to s)) \to s)}}{C \triangleright \forall s<:\mathbf{Top}.((\forall t<:\sigma.(\tau \to s)) \to s) <: \forall s<:\mathbf{Top}.((\forall t<:\sigma.(\tau' \to s)) \to s)}$$

[T-PACK] 則の導出：

$$\dfrac{\dfrac{\dfrac{}{C[s<:\mathbf{Top}],\Gamma[x:\forall t<:\sigma.(\tau\to s)],\Delta\triangleright x:\forall t<:\sigma.(\tau\to s)} \quad \dfrac{C\triangleright\rho<:\sigma}{C[s<:\mathbf{Top}]\triangleright\rho<:\sigma}(\dagger)}{C[s<:\mathbf{Top}],\Gamma[x:\forall t<:\sigma.(\tau\to s)],\Delta\triangleright x[\rho]:\tau[t:=\rho]\to s} \quad \dfrac{C,\Gamma,\Delta\triangleright e:\tau[t:=\rho]}{C[s<:\mathbf{Top}],\Gamma,\Delta\triangleright e:\tau[t:=\rho]}(\ddagger)}{\dfrac{C[s<:\mathbf{Top}],\Gamma[x:\forall t<:\sigma.(\tau\to s)],\Delta\triangleright x[\rho]e:s}{\dfrac{C[s<:\mathbf{Top}],\Gamma,\Delta\triangleright(\lambda x:\forall t<:\sigma.(\tau\to s).x[\rho]e):(\forall t<:\sigma.(\tau\to s))\to s}{C,\Gamma,\Delta\triangleright(\Lambda s<:\mathbf{Top}.\lambda x:\forall t<:\sigma.(\tau\to s).x[\rho]e):\forall s<:\mathbf{Top}.((\forall t<:\sigma.(\tau\to s))\to s)}}}$$

[T-UNPACK] 則の導出：

$$\dfrac{\dfrac{C,\Gamma,\Delta\triangleright e_1:\forall s<:\mathbf{Top}.((\forall t<:\sigma.(\tau\to s))\to s) \quad \dfrac{}{C\triangleright\rho<:\mathbf{Top}}}{C,\Gamma,\Delta\triangleright e_1[\rho]:(\forall t<:\sigma.(\tau\to\rho))\to\rho} \quad \dfrac{\dfrac{C[t<:\sigma],\Gamma[x:\tau],\Delta\triangleright e_2:\rho}{C[t<:\sigma],\Gamma,\Delta\triangleright(\lambda x:\tau.e_2):\tau\to\rho}}{C,\Gamma,\Delta\triangleright(\Lambda t<:\sigma.\lambda x:\tau.e_2):\forall t<:\sigma.(\tau\to\rho)}}{C,\Gamma,\Delta\triangleright e_1[\rho](\Lambda t<:\sigma.\lambda x:\tau.e_2):\rho}$$

[A-PACK] 則の導出：

[T-PACK] 則の導出と同様.

[A-UNPACK] 則の導出：

[T-UNPACK] 則の導出と同様.

[A-$\beta_\exists$] 則の導出：

紙幅の関係で，帰結部の不等式の導出に関してのみ以下に示す：

$$
\begin{array}{ll}
\quad \textbf{unpack}\ (\textbf{pack}_{\exists t<:\sigma.\tau_1}\ e_1\ \textbf{with}\ \rho)\ \textbf{as}\ x\ \textbf{with}\ t\ \textbf{in}\ e_2 & \\
\equiv (\Lambda\, s<:\textbf{Top}.(\lambda\, x{:}\,\forall t<:\sigma.(\tau_1 \to s).x[\rho]e_1))[\tau_2](\Lambda\, t<:\sigma.\lambda\, x{:}\,\tau_1.e_2) & \text{定義 3.1 でのコード化による} \\
\leqslant (\lambda\, x{:}\,\forall t<:\sigma.(\tau_1 \to \tau_2).x[\rho]e_1)(\Lambda\, t<:\sigma.\lambda\, x{:}\,\tau_1.e_2) & [\text{A-}\beta_\forall]\ \text{による} \\
\leqslant (\Lambda\, t<:\sigma.\lambda\, x{:}\,\tau_1.e_2)[\rho]e_1 & [\text{A-}\beta_\to]\ \text{による} \\
\leqslant (\lambda\, x{:}\,\tau_1[t := \rho].e_2[t := \rho])e_1 & [\text{A-}\beta_\forall]\ \text{による} \\
\leqslant e_2[t := \rho][x := e_1] & [\text{A-}\beta_\to]\ \text{による}.
\end{array}
$$

完全な形の導出木は，以上の不等式の導出の各ステップでの推論規則の適用に適当な型付け文脈を補い，更に前後のステップを [A-TRANS] 則の適用によって結び付ける事で容易に構成できる．

証明終

部分型関係に関する図 2.2 の規則群で “(∗)” の付いていない公理と推論規則の中で “(†)” が付けられている [S-WEAK] 則は Cardelli and Wegner (1985) に示された本来の Fun の規則群には存在しない．この推論規則は抽象型 ∃ を固有の型構成子として他の型構成子とは独立に導入している — 本来の Fun ではそうしている — 場合には不要な規則である．しかし，定理 3.2 の証明での [S-EXIST] 則の導出や [T-PACK] 則の導出での (†) が付いているステップで見た通り，抽象型を独立な型構成子として扱わずに多相型と関数型とでコード化する場合には，コード化で導入される新しい型変数の処理の為に [S-WEAK] 則が不可欠である．

同様に型付けに関する図 2.3 の規則群で “(‡)” が付された [T-TWEAK] 則も本来の Fun には存在しないが，上の証明での [T-PACK] 則の導出での (‡) が付いているステップでコード化で導入される型変数の処理の為に不可欠である．

注目すべき事は，抽象型を多相型と関数型とでコード化する場合の抽象型に関する規則群の導出での [S-EXIST] 則と [T-TWEAK] 則との適用は，常に「自明な」適用だという事である．即ち，これら二つの規則の適用で型変数の制約集合に追加される新しい型変数に対する上限の型は常に最大の型 **Top** である，という事実に注意する必要がある．

2 階命題論理での全称限量子 ∀ と含意 ⊃ とによる存在限量子 ∃ のコード化以来，命題＝型 の立場から抽象型も多相型と関数型とでコード化可能と考えられ，また，

型のみならず対応する式構文に関しても正しく扱えると信じられて来た．しかし，束縛型変数に対する範囲限定を許す場合には，コード化が全くの従来通りでは通用せず新たに [S-EXIST] 則と [T-T-WEAK] 則とを導入する必要がある，という事が，上の定理の証明より判る．

基本的には以下に於いても抽象型は独立なものとして扱うが，抽象型を独立に扱うと繁雑になる場合には，以上の結果を用いて，抽象型は多相型と関数型とでコード化されており抽象型に関する構文や推論規則は存在しないと看做し，抽象型を固有に扱う事を省く場合がある．

3.2　保存的拡大性と関連する性質

本節では，本論文の型理論 FUNIQ が FUN に対して保存的拡大となっている事を示す．この事は，直感的に言えば，本論文の型理論 FUNIQ は FUN の良い性質を素直に引き継いでいる，という事を意味している．そこで，この保存的拡大という概念をきちんと定義しておく．

定義 3.3（保存的拡大）

二つの体系 T_1 と T_2 とについて，T_1 が T_2 の**保存的拡大**である，とは，以下の 2 条件が共に成り立っている場合かつその場合のみである：

(1) T_1 が T_2 の拡大 — 即ち，$T_1 \supseteq T_2$ — である；

(2) T_2 で許される形の文 Σ に関しては，Σ が T_1 の帰結であるのは，Σ が T_2 の帰結である場合に限られる．即ち，

$$\vdash_{T_1} \Sigma \iff \vdash_{T_2} \Sigma.$$

例えば，古典命題論理の体系 NK は直観主義命題論理の体系 NJ の拡大ではあるが保存的拡大ではない．何故ならば，任意の命題変数 A に対して $A \vee \neg A$ は NK で証明可能だが NJ では証明可能でない．一方，肯定含意論理（論理結合子として含意のみを許す命題論理の体系）に対して NJ は保存的拡大である．

FUNIQ と FUN とは共に判別式の導出に関する体系であるので，上の定義での「文」は今の場合は判別式を表わすが，FUNIQ には FUN では許されない形の判別式のクラスとして表明判別式が含まれている．更に，両方の体系で許される判別式のクラスである部分型判別式と型付け判別式に関しても，それらの構成要素である型や式として許される構文は異なっている．例えば，式の場合，言語としての Funiq には Fun には存在しない変数のクラスとして実現変数が含まれる．型に関しては，Funiq には Fun では許されない詳細化型という型構成法が含まれる．

そこで，FUNIQ の FUN に対する保存的拡大性を正確な言明として述べるには，Fun で許される形の式や型，或いは，そういった式や型のみを要素とする — 即ち，FUN で許される形の — 判別式のクラスを予め定義しておかねばならない．以下，順を追ってそれらのクラスを定義する．

定義 3.4 （実現変数無し）

(1) 言語 Funiq の式 e が**実現変数無し**である，とは，e の自由変数には実現変数が含まれない事である．即ち，$\mathrm{FV}(e) \cap \mathbf{IVar} = \varnothing$.

(2) Funiq の表明 γ が**実現変数無し**である，とは，γ の自由変数には実現変数が含まれない事である．即ち，$\mathrm{FV}(\gamma) \cap \mathbf{IVar} = \varnothing$.

(3) 理論 FUNIQ の型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ が**実現変数無し**である，とは，型付け対象となっている式 e が実現変数無しという事である．

(4) FUNIQ の表明判別式 $C, \Gamma, \Delta \triangleright \gamma$ が**実現変数無し**である，とは，表明 γ が実現変数無しという事である．

以上の定義で注意すべき点が幾つかある．第一は，式 e が実現変数無しだからと言って式 e は Fun の式である — 即ち，$e \in \mathbf{Exp}_{\mathrm{Fun}}$ — とは限らない，という点である．その理由は，上の定義 (1) では e 中の構成要素としての型 — 例えば e が λ-抽象 $\lambda x{:}\sigma.e'$ の形の場合の σ — が詳細化型である事は許されるからである（詳細化型は実現変数を含むが，その実現変数は詳細化型自身で束縛されているので e の自由変数にはならない）．第二の注意点として，(3) での型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ が実現変数無しであっても実現変数に関する型付け基底 Δ は空基底 $\varnothing$ だとは限らない，

という点である．この注意点は (4) での表明判別式に関しても同様である．

次に「表明無し」という概念を定義する．この概念は上の「実現変数無し」の概念を用いずに定義されるので,「実現変数無し」という概念は中途半端に映る可能性がある．しかし，本節での保存的拡大性の証明を見れば判る通り，その証明の中で実現変数無しの概念は実際に必要となる．

定義 3.5 （表明無し）

(1) 言語 Funiq の型 σ が**表明無し**である，とは， $\sigma \in \mathbf{Type}_{\mathrm{Fun}}$ という事である．

(2) Funiq の式 e が**表明無し**である，とは， $e \in \mathbf{Exp}_{\mathrm{Fun}}$ という事である．

(3) 型変数に対する制約集合 C が**表明無し**である，とは，その定義域に属する各型変数に対する上界の型が表明無しという事である．即ち，

$$\forall (t <: \sigma) \in C. \big[\sigma \in \mathbf{Type}_{\mathrm{Fun}}\big].$$

(4) 通常変数に対する型付け基底 Γ が**表明無し**である，とは，その定義域に属する各通常変数に対する型が表明無しという事である．即ち，

$$\forall (x : \sigma) \in \Gamma. \big[\sigma \in \mathbf{Type}_{\mathrm{Fun}}\big].$$

(5) 理論 FUNIQ の部分型判別式 $C \triangleright \sigma <: \tau$ が**表明無し**である，とは，C, σ, τ の何れもが表明無しという事である．

(6) FUNIQ の型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ が**表明無し**である，とは，実現変数に対する型付け基底 Δ が空写像 $\varnothing$ であり，更に C, Γ, e, σ の何れもが表明無しという事である．

次に，与えられた型の中の詳細化型を全て消去する写像を定義する．

定義 3.6 （詳細化型消去写像 $(\cdot)^*$）

詳細化型消去写像 $(\cdot)^* : \mathbf{Type} \to \mathbf{Type}_{\mathrm{Fun}}$ を型の構造に関する帰納法により以下の様に定義する：

$$
\begin{aligned}
\mathbf{Top}^* &= \mathbf{Top}, \\
\iota^* &= \iota, \\
t^* &= t, \\
(\sigma_1 \to \sigma_2)^* &= {\sigma_1}^* \to {\sigma_2}^*, \\
\{l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\}^* &= \{l_1{:}{\sigma_1}^*, \ldots, l_n{:}{\sigma_n}^*\}, \\
[\, l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n \,]^* &= [\, l_1{:}{\sigma_1}^*, \ldots, l_n{:}{\sigma_n}^* \,], \\
(\forall t{<:}\sigma.\tau)^* &= \forall t{<:}\sigma^*.\tau^*, \\
(\exists t{<:}\sigma.\tau)^* &= \exists t{<:}\sigma^*.\tau^*, \\
\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}^* &= \sigma^*.
\end{aligned}
$$

この写像 $(\cdot)^*$ は，以下に示す通り，式の集合 **Exp**，表明の集合 **Assertion**，型変数に対する制約集合（の集合），型付け基底（の集合），及び，FUNIQ の各種の判別式（の集合）の各々の上の写像へと拡張される．それらの様々な集合上に拡張された詳細化型消去写像も $(\cdot)^*$ で表わすが曖昧さは生じない．

定義 3.7 （型付け文脈に対する詳細化型消去写像）

(1) C を型変数に対する制約集合とする時，C^* を以下の様に定義する：

$$C^* \triangleq \{(t <: \sigma^*) \mid (t <: \sigma) \in C\}.$$

(2) Γ を通常変数の型付け基底とする時，Γ^* を以下の様に定義する：

$$\Gamma^* \triangleq \{(x : \sigma^*) \mid (x : \sigma) \in \Gamma\}.$$

(3) 実現変数に対する型付け基底 Δ に関しても，Δ^* は (2) と同様に定義される．

式の上への拡張を次に定義する.

定義 3.8（式に対する詳細化型消去写像）

式の集合上への詳細化型消去写像の拡張 $(\cdot)^* : \mathbf{Exp} \to \mathbf{Exp}$（値域が $\mathbf{Exp}_{\mathrm{Fun}}$ でない事に注意）を式の構造に関する帰納法によって以下の様に定義する：

$$
\begin{aligned}
c^* &= c, \\
x^* &= x, \\
r^* &= r, \\
(\lambda\, x{:}\sigma.e)^* &= \lambda\, x{:}\sigma^*.e^*, \\
(e_1 e_2)^* &= {e_1}^* {e_2}^*, \\
\{l_1 = e_1, \ldots, l_n = e_n\}^* &= \{l_1 = {e_1}^*, \ldots, l_n = {e_n}^*\}, \\
(e.l)^* &= e^*.l, \\
[\,l = e\,]^* &= [\,l = e^*\,], \\
(\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n)^* &= \mathbf{case}\ e_0^*\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1^*, \ldots, l_n\ \mathbf{then}\ e_n^*, \\
(\Lambda\, t{<:}\sigma.e)^* &= \Lambda\, t{<:}\sigma^*.e^*, \\
(e[\sigma])^* &= e^*[\sigma^*], \\
(\mathbf{pack}_{\exists t<:\rho.\sigma}\ e\ \mathbf{with}\ \tau)^* &= \mathbf{pack}_{\exists t<:\rho^*.\sigma^*}\ e^*\ \mathbf{with}\ \tau^*, \\
(\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2)^* &= \mathbf{unpack}\ {e_1}^*\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ {e_2}^*, \\
(\mathbf{fix}\ e)^* &= \mathbf{fix}\ e^*, \\
(e{:}\sigma)^* &= e^*{:}\sigma^*.
\end{aligned}
$$

以上の式に対する拡張を用いて，表明に対する拡張を次の様に定義する.

定義 3.9（表明に対する詳細化型消去写像）

表明の集合上への詳細化型消去写像の拡張 $(\cdot)^* : \mathbf{Assertion} \to \mathbf{Assertion}$ を表明の構造に関する帰納法によって以下の様に定義する：

$$
\begin{aligned}
(e_1 \leqslant e_2 : \sigma)^* &= {e_1}^* \leqslant {e_2}^* : \sigma^*, \\
(\mathbf{forall}\ x{:}\sigma.\,\gamma)^* &= \mathbf{forall}\ x{:}\sigma^*.\,\gamma^*.
\end{aligned}
$$

最後に，各種の判別式に対する拡張を与える．

定義 3.10 （判別式に対する詳細化型消去写像）

Σ を FUNIQ の判別式とする時，判別式 Σ の種類に応じて Σ^* を以下の様に定義する：

(1) 部分型判別式 $\Sigma \equiv C \ \triangleright\ \sigma <: \tau$ の場合：

$$\Sigma^* \triangleq C^* \ \triangleright\ \sigma^* <: \tau^*;$$

(2) 型付け判別式 $\Sigma \equiv C, \Gamma, \Delta \ \triangleright\ e : \sigma$ の場合：

$$\Sigma^* \triangleq C^*, \Gamma^*, \Delta^* \ \triangleright\ e^* : \sigma^*;$$

(3) 表明判別式 $\Sigma \equiv C, \Gamma, \Delta \ \triangleright\ \gamma$ の場合：

$$\Sigma^* \triangleq C^*, \Gamma^*, \Delta^* \ \triangleright\ \gamma^*.$$

以上で定義された詳細化型消去写像は，表明無しの対象に対しては恒等写像として振舞う．即ち，

補題 3.11

Σ を FUNIQ の表明無し（部分型もしくは型付け）判別式とする．この時，$\Sigma^* \equiv \Sigma$.

証明

詳細化型消去写像の定義（定義 3.6 ∼ 定義 3.10）より明らか.

証明終

次に， FUNIQ の公理や推論規則が以上で定義された詳細化型消去写像と両立するとは如何なる事かを定義する．

定義 3.12 （∗-両立性）

(1) FUNIQ の或る公理が **∗-両立的**である，とは，任意の判別式 Σ が当該公理のインスタンスであるならば Σ^* も同じ公理のインスタンスだ，という事である．

(2) FUNIQ の或る推論規則が **∗-両立的**である，とは，任意の判別式 $\Sigma_0, \ldots, \Sigma_n$ について判別式 Σ_0 が $\Sigma_1, \ldots, \Sigma_n$ を前提として当該推論規則の適用により帰結として得られるならば ${\Sigma_0}^*$ も ${\Sigma_1}^*, \ldots, {\Sigma_n}^*$ を前提として同じ推論規則の適用によって得られる，という事である．

型理論 FUNIQ の殆ど全ての公理と推論規則とは $*$-両立的である．即ち，

補題 3.13

[A-ASSERT] 則以外の FUNIQ の公理と推論規則の全ては $*$-両立的である．

証明

定義 3.12 に従って図 2.2 ∼ 図 2.4 の各公理／推論規則をチェックすれば良い．

証明終

上の補題の証明に於いて，幾つかの推論規則に関しては注意が必要である．以下，それらについて述べる．

まず [S-REFINE] 則に関してであるが，この推論規則は表明強度の前提を持つ．しかし，この規則の適用 — 即ち，規則の適用での全ての前提と帰結 — に詳細化型消去写像 $(\,\cdot\,)^*$ を作用させると，帰結部の判別式は $C^* \triangleright \sigma^* <: \tau^*$ という表明無しの形になる．故に，その [S-REFINE] 則の適用での表明強度の前提は冗長となるので捨てる（無視する）事が許される．すると，[S-REFINE] 則の各適用は $(\,\cdot\,)^*$ の作用により

$$\frac{C^* \triangleright \sigma^* <: \tau^*}{C^* \triangleright \sigma^* <: \tau^*}$$

という形の自明な — 即ち，前提と帰結とが同一の判別式である — 導出ステップとなる．よって，[S-REFINE] 則は $*$-両立的だと看做して良い．

同様に，[T-REFINE] 則の場合も，その適用に $(\,\cdot\,)^*$ を作用させる事により，

$$\frac{C^*,\Gamma^*,\Delta^* \;\triangleright\; e^* : \sigma^* \quad C^*,\Gamma^*,\Delta^* \;\triangleright\; (\gamma_1[r := e])^* \;\ldots\; C^*,\Gamma^*,\Delta^* \;\triangleright\; (\gamma_m[r := e])^*}{C^*,\Gamma^*,\Delta^* \;\triangleright\; e^* : \sigma^*}$$

という形になるが，この前提部の各々の表明判別式 $C^*,\Gamma^*,\Delta^* \;\triangleright\; (\gamma_i[r := e])^*$ は，最早，冗長な前提であるので，それを捨てると，$(\,\cdot\,)^*$ を作用させた [T-REFINE] 則の適用は，

$$\frac{C^*,\Gamma^*,\Delta^* \;\triangleright\; e^* : \sigma^*}{C^*,\Gamma^*,\Delta^* \;\triangleright\; e^* : \sigma^*}$$

という自明な導出ステップとなる．故に，[T-REFINE] 則も $*$-両立的な推論規則として扱える．

次の補題は，λ-計算の簡約での「β-簡約では λ-式中の自由変数は増えない」という良く知られた事実に相当するものである．

補題 3.14（自由変数の非増加性）

$$\vdash_{\mathrm{FUNIQ}} C,\Gamma,\Delta \;\triangleright\; e_1 \leqslant e_2 : \sigma \quad \Longrightarrow \quad \mathrm{FV}(e_1) \supseteq \mathrm{FV}(e_2).$$

証明

図 2.4 の各公理が主張の右辺を満たし，各推論規則が主張を保存する事を，一つずつ確認すれば良い．

証明終

2.2 節の図 2.4 で表明判別式に関する規則群を提示した直後で，式の簡約に対応する各 [A-β_{xxx}] 則に関して逆向きの不等式を許さないと述べたが，その理由は，この補題の為である．厳密に言えば，上の補題の為には [A-$\beta_{\to}$]，[A-$\beta_{\{\ldots\}}$]，[A-$\beta_{[\ldots]}$] および [A-$\beta_{\exists}$] の各規則が現在の向きの不等式であれば良く，[A-$\beta_{\forall}$] 則と [A-β_{fix}] 則とは等式でも構わない — 現在の不等式と逆向きの不等式の規則を追加しても良い — のであるが，体系の一貫性を重視し，後の二つの推論規則に関しても現在の形の不等式だけを許す．

次の補題は，実現変数無しの（型付けあるいは表明）判別式が証明できるならば，その判別式の実現変数に対する型付け基底は必要ない（ので空基底 $\varnothing$ にできる），という事を表わしている．

補題 3.15

Σ を実現変数無しの型付け判別式もしくは表明判別式とする．この時，

$$\vdash_{\mathrm{FUNIQ}} \Sigma \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} \Sigma^{\flat}.$$

ここで，

$$\Sigma^{\flat} \triangleq \begin{cases} C, \Gamma, \varnothing \triangleright e : \sigma, & \Sigma \equiv C, \Gamma, \Delta \triangleright e : \sigma \text{ の場合；} \\ C, \Gamma, \varnothing \triangleright e_1 \leqslant e_2 : \sigma, & \Sigma \equiv C, \Gamma, \Delta \triangleright e_1 \leqslant e_2 : \sigma \text{ の場合．} \end{cases}$$

証明

Σ の導出に関する帰納法による．

(1) Σ が公理のインスタンスの場合：

その公理は [T-IVAR], [A-IVAR] ではあり得ない．何故なら，これら二つの公理は実現変数無しではあり得ないからである．さて，[T-IVAR], [A-IVAR] 以外の公理の場合，実現変数に対する型付け基底が何の付帯条件も付いていない単純なメタ変数 Δ となっている．よって，Σ がそれらの公理のインスタンスの場合，$\Sigma^{\flat}$ も同一の公理のインスタンスである．従って，右辺の導出可能性が成立する．

(2) Σ の導出の最後のステップが [A-TRANS] 則以外の規則の適用の場合：

[A-TRANS] 則以外の推論規則 $\mathcal{R}$ に於いては規則の前提部の判別式に現れる全ての式は帰結部の判別式にも現れるので，それらの式は実現変数無しである．従って，帰納法の仮定が適用でき，推論規則 $\mathcal{R}$ の適用の前提部の各（型付けもしくは表明）判別式 Σ_i に対し，$\vdash_{\mathrm{FUNIQ}} \Sigma_i^{\flat}$ だとして良い．

(a) $\mathcal{R} \neq$ [T-IWEAK], [A-IWEAK] の場合,

これら二つの規則以外では，前提部に現れる判別式の実現変数に対する型付け基底は帰結部の判別式のそれと同じであるので，帰納法の仮定で導出可能性が保証される ${\Sigma_i}^\flat$ を前提として $\mathcal{R}$ を適用すれば， $\Sigma^\flat$ の導出を得る．

(b) $\mathcal{R} =$ [T-IWEAK] の場合,

この場合， $\Sigma \equiv C, \Gamma, \Delta[r:\tau] \triangleright e:\sigma$ で，前提は $\Sigma_1 \equiv C, \Gamma, \Delta \triangleright e:\sigma$ である．従って， $\Sigma^\flat \equiv (C, \Gamma, \Delta[r:\tau] \triangleright e:\sigma)^\flat \equiv C, \Gamma, \varnothing \triangleright e:\sigma \equiv (C, \Gamma, \Delta \triangleright e:\sigma)^\flat \equiv {\Sigma_1}^\flat$ である．一方，帰納法の仮定により $\vdash_{\mathrm{FUNIQ}} {\Sigma_1}^\flat$ であるので $\vdash_{\mathrm{FUNIQ}} \Sigma^\flat$ である．

(c) $\mathcal{R} =$ [A-IWEAK] の場合,

(b) と同様.

(3) Σ の導出の最後のステップが [A-TRANS] 則の適用の場合：

補題 3.14 より， [A-TRANS] 則の帰結部に現れない中間式 — 前提部での e_2 — も実現変数無しである．従って，帰納法の仮定を前提部の二つの表明判別式に適用し，それで導出可能性が保証される各々の $(\cdot)^\flat$-版の判別式に当規則を適用すれば， $\Sigma^\flat$ の導出を得る．

証明終

Σ を FUNIQ の部分型判別式もしくは型付け判別式で，かつ，実現変数無しとする時，一般に $(\Sigma^\flat)^*$ が FUN で許される形の同種の判別式である事は明らかである．一方，補題 3.13 より [A-ASSERT] 則以外の FUNIQ の公理や推論規則は詳細化型消去写像に対して素直に振舞う事が判る．だから， FUNIQ での Σ の導出と FUN での $(\Sigma^\flat)^*$ の導出とを関係付ける為には「Σ の導出木の中で [A-ASSERT] 則を含み得ない部分」という概念を導入して用いると便利である．そこで，以下の一連の定義でこの概念を与える．

定義 3.16（導出木の主部分）

Σ を $\vdash_{\mathrm{FUNIQ}} \Sigma$ である様な部分型あるいは型付け判別式とし，Π を Σ の導出木とする．この時，Π の**主部分** $\tilde{\Pi}$ を Π の構造（つまり Σ の導出）に関する帰納法によって以下の様に定義する．

(a) Π が（部分型関係もしくは型付けの公理）の適用である場合，$\tilde{\Pi} \triangleq \Pi$ とする．

(b) Π の最終ステップが [S-REFINE] 則でも [T-REFINE] 則でもない推論規則の適用の場合，$\tilde{\Pi}$ を以下の様に定義する：

- $\tilde{\Pi}$ の帰結部の判別式は Π の帰結部の判別式 Σ_c 自身とする；
- $\tilde{\Pi}$ の直接の部分木は $\tilde{\Pi}_1, \ldots, \tilde{\Pi}_n$ とする．ここで，$\Pi_1, \ldots, \Pi_n$ は Π の直接の部分木である．

(c) Π の最終ステップが [S-REFINE] 則の適用の場合，$\tilde{\Pi}$ を以下の様に定義する：

- $\tilde{\Pi}$ の帰結部の判別式は Π の帰結部の判別式 Σ_c 自身とする；
- $\tilde{\Pi}$ は只一つの直接の部分木 $\tilde{\Pi}_p$ を持つ．ここで，Π_p は [S-REFINE] 則の主前提に対応する Π の直接の部分木である（故に，[S-REFINE] 則の適用での表明強度の前提を帰結部とするその他の Π の直接の部分木は $\tilde{\Pi}$ の構成には使用されない）．

(d) Π の最終ステップが [T-REFINE] 則の適用の場合，$\tilde{\Pi}$ を以下の様に定義する：

- $\tilde{\Pi}$ の帰結部の判別式は Π の帰結部の判別式 Σ_c 自身とする；
- $\tilde{\Pi}$ は只一つの直接の部分木 $\tilde{\Pi}_p$ を持つ．ここで，Π_p は [T-REFINE] 則の主前提に対応する Π の直接の部分木である（従って，[T-REFINE] 則の適用での表明判別式 $C, \Gamma, \Delta \triangleright \gamma_j[r := e]$ $(j = 1, \ldots, m)$ を帰結部とするその他の Π の直接の部分木は $\tilde{\Pi}$ の構成には使用されない）．

定義 3.17（導出木の主経路）

Σ を $\vdash_{\mathrm{FUNIQ}} \Sigma$ である様な部分型あるいは型付け判別式とし，Π を Σ の導出木とする．この時，元の導出木 Π 中の経路の中で主部分 $\tilde{\Pi}$ に対応する経路が残っているものを，Π の**主経路**と呼ぶ．

定義 3.18（導出木の主骨格）

Σ を $\vdash_{\mathrm{FUNIQ}} \Sigma$ である様な部分型あるいは型付け判別式とし，Π を Σ の導出木とする．この時，Π の**主骨格** $\tilde{\Pi}^*$ とは主部分 $\tilde{\Pi}$ に詳細化型消去写像 $(\ \cdot\ )^*$ を適用した木である．即ち，$\tilde{\Pi}$ 中の各判別式 Σ を Σ^* で置き換え，[S-REFINE] 則や [T-REFINE] 則の適用結果としての自明な推論ステップを除いたものである．より形式的には，$\tilde{\Pi}^*$ は導出木 Π の構造に関する帰納法によって以下の様に定義される．

(a) Π が（部分型関係もしくは型付けの公理）の適用である場合，判別式 Σ はその公理のインスタンスであるが，補題 3.13 により Σ^* も当該公理のインスタンスである．従って，Π^* は同じ公理の適用であり，$\tilde{\Pi}^* \triangleq \Pi^*$ とする．

(b) Π の最終ステップが [S-REFINE] 則でも [T-REFINE] 則でもない推論規則の適用の場合，

- $\tilde{\Pi}^*$ の帰結部の判別式は ${\Sigma_c}^*$ とする．ここで，Σ_c は Π の帰結部の判別式である；
- $\tilde{\Pi}^*$ の直接の部分木は $\tilde{\Pi}^*_1, \ldots, \tilde{\Pi}^*_n$ とする．ここで，$\Pi_1, \ldots, \Pi_n$ は Π の直接の部分木である．

(c) Π の最終ステップが [S-REFINE] 則の適用の場合，$\tilde{\Pi}^* \equiv \tilde{\Pi}^*_p$ とする．ここで，Π_p は [S-REFINE] 則の主前提に対応する Π の直接の部分木である．

(d) Π の最終ステップが [T-REFINE] 則の適用の場合，$\tilde{\Pi}^* \equiv \tilde{\Pi}^*_p$ とする．ここで，Π_p は [T-REFINE] 則の主前提に対応する Π の直接の部分木である．

以上で定義された導出木の主骨格には，単に [A-ASSERT] 則が出現し得ないだけでなく，どの様な表明判別式も全く現れ得ない．即ち，

補題 3.19

(1) Σ を FUNIQ の部分型判別式とする時，Σ の任意の導出木の主骨格は部分型判別式のみから構成されている．

(2) Σ を FUNIQ の型付け判別式とする時，Σ の任意の導出木の主骨格は部分型判別式と型付け判別式とのみから構成されている．

証明

(1), (2) 共に主骨格の構成に関する帰納法で示せば良いが，それは定義 3.18 の主骨格の構築方法から明らかである．

証明終

次が本節の最も重要な補題である．この補題は，**或る判別式の導出木の主骨格それ自身が実は（元の判別式に対応する FUN での判別式に非常に近い形の判別式の）導出木になっている**，という強い内容を表わしている．即ち，主骨格は「骨格」という呼び方から類推される論理的な中身に乏しい骨組だけというイメージとは裏腹に型理論的な内容が詰まった木である，という事が次の補題から判る．

補題 3.20 （主補題）

Σ を FUNIQ の部分型判別式もしくは型付け判別式とし，Π を Σ の導出木とする．この時，Π の主骨格 $\tilde{\Pi}^*$ は Σ^* の導出木である．

証明

補題 3.13 および補題 3.19 による．

証明終

以上で必要な準備が全て整ったので，表明無し判別式の導出は表明無し判別式の集合の中で閉じさせる事が可能である事を証明する．

定理 3.21

表明無し判別式 Σ が FUNIQ で導出可能とする．この時，Σ の導出木で表明無し判別式のみで構成されたものが存在する．

証明

判別式 Σ は表明無しであるので，Σ は部分型判別式か型付け判別式かのどちらかでなければならない．そこで各々の場合に分けて証明する．

(1) Σ が部分型判別式の場合：

Σ は表明無しであるので補題 3.11 より $\Sigma \equiv \Sigma^*$ である．故に Π を Σ の導出木の一つとする時，主補題 3.20 より $\tilde{\Pi}^*$ は Σ^* の導出木であるが，$\Sigma \equiv \Sigma^*$ であるので，それは Σ 自身の導出木でもある．

この Σ の導出木としての主骨格 $\tilde{\Pi}^*$ は補題 3.19 (1) より部分型判別式のみから構成されており，しかも主骨格の定義より，$\tilde{\Pi}^*$ 中の判別式は全て詳細化型消去写像 $(\cdot)^*$ が作用した形であるので表明無しである．

従って，Σ は表明無し判別式のみで構成された導出木 $\tilde{\Pi}^*$ を持つ．

(2) Σ が型付け判別式の場合：

Σ は表明無しなので，$\Sigma \equiv C, \Gamma, \varnothing \ \triangleright \ e : \sigma$ という形でなければならず，しかも $e \in \mathbf{Exp}_{\mathrm{Fun}}$ であるから e は実現変数を含まないという事を注意しておく．

故に Π を Σ の導出木の一つとする時，e は実現変数無しなので，補題 3.15 を用いれば Π の主骨格 $\tilde{\Pi}$ 中の全ての型付け判別式は上の形 — 実現変数に対する型付け基底が空 $\varnothing$ の形 — である（この点を厳密に示すには，主経路の長さに関する帰納法を用いれば良いが，簡単に言えば，図 2.2 ～ 図 2.4 の推論規則で前提の判別式の方が帰結のそれよりも実現変数の型付け基底に関して大きいのは [S-REFINE] 則での表明強度の前提だけだという事の確認で済む）．

補題 3.11 より $\Sigma \equiv \Sigma^*$ であり，主補題 3.20 より $\tilde{\Pi}^*$ は Σ^* の導出木であるが，$\Sigma \equiv \Sigma^*$ であるので，それは Σ 自身の導出木でもある．

この Σ の導出木としての主骨格 $\tilde{\Pi}^*$ は補題 3.19 (2) より型付け判別式と部分型判別式とのみから構成されており，しかも主骨格の定義より，$\tilde{\Pi}^*$ 中の判別式は全て詳細化型消去写像 $(\cdot)^*$ が作用した形であるので，(1) と同様に $\tilde{\Pi}^*$ 中の全ての部分型判別式は表明無しである．

$\tilde{\Pi}^*$ 中の型付け判別式に関しても，$(\cdot)^*$ が作用した形でありしかも最初の注意の通り実現変数に関する型付け基底が空である事より，表明無しである．

従って，Σ は表明無し判別式のみで構成された導出木 $\tilde{\Pi}^*$ を持つ．

証明終

本節の最初に予告した通り，本論文の型理論 FUNIQ は Cardelli and Wegner (1985) の体系 FUN の保存的拡大となっている．即ち，

系 3.22 （保存的拡大定理）

型理論 FUNIQ はその部分型理論 FUN の保存的拡大である．即ち， FUNIQ の任意の表明無し判別式に対して，

$$\vdash_{\mathrm{FUNIQ}} \Sigma \quad \Longleftrightarrow \quad \vdash_{\mathrm{FUN}} \Sigma.$$

証明

$(\Leftarrow)$ に関しては， FUN の全ての公理と推論規則とは FUNIQ に含まれている事から自明である． $(\Rightarrow)$ に関しては，直前の定理から従う．

証明終

FUN は型付き関数的プログラミング言語 Fun の型システムとして決定可能な体系である．一方，本論文の広帯域言語 Funiq の型システムとしての FUNIQ は，関数的プログラムに対する部分正当性を表現する記述力を持っているので，明らかに決定不能な体系である．しかしながら， FUNIQ でも全ての表明を忘れる事によって決定可能性を回復できる．言い替えれば， FUNIQ は仕様記述や検証の為の型システムであり， FUN はプログラムの最低限の整合性をコンパイル時に検査する為の型システムだと言う事ができる．次の定理は**「全ての（部分正当性の意味で）正しいプログラムはコンパイル時の型検査を合格する」** — 言い替えれば，コンパイラの型検査はプログラムの部分正当性に対して忠実である — という事を保証している．

定理 3.23（型検査の忠実性）

(1) 任意の C, σ および τ に対して，

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma <: \tau \quad \Longrightarrow \quad \vdash_{\mathrm{FUN}} C^* \triangleright \sigma^* <: \tau^*.$$

(2) 任意の $C, \Gamma, \Delta, \sigma$ および実現変数無しの任意の e に対して，

$$\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e : \sigma \quad \Longrightarrow \quad \vdash_{\mathrm{FUN}} C^*, \Gamma^*, \varnothing \triangleright e^* : \sigma^*.$$

証明

(1) について

主補題 3.20 より直ちに成立する．

(2) について

Σ を左辺の判別式とする．この時，補題 3.15 より $\Sigma^\flat$ が FUNIQ で導出可能であるので，$\Sigma^\flat$ の導出木が少なくとも一つは存在する．そこで，$\Sigma^\flat$ の導出木を一つ選び Π で表わすとする．前の定理 3.21 の証明での (2) の場合と同様，この導出木 Π の主骨格 $\tilde{\Pi}$ 中の全ての判別式に於ける実現変数に対する型付け基底は $\varnothing$ として良い．

従って，補題 3.20 より，右辺の判別式 $(\Sigma^\flat)^*$ は $\tilde{\Pi}^*$ という導出木を持つ．

証明終

3.3　型付け判別式の独立性

本論文の型理論 FUNIQ では，一般に以下が成立する．

命題 3.24

任意の C, Γ, Δ, e および σ に対して，

$$\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e : \sigma \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e \leqslant e : \sigma.$$

証明

左辺の導出に関する帰納法による．その概略を言えば，左辺の導出木中の [T-xxx] という型付けに関する公理／推論規則の各々を対応する表明に関する規則 [A-xxx] に置き換えれば，右辺の導出木が得られる．

証明終

そこで，型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ を不等式の左右両辺が同じ式という特別な形の表明判別式 $C, \Gamma, \Delta \triangleright e \leqslant e : \sigma$ の略記法と看做し，図 2.3 の各公理／推論規則を除いた形で Funiq の型システムを定式化する事が考えられる[1)]．

しかしながら，型付け判別式を不等式表明判別式の特殊な場合として取り扱う上の方法は上手く行かない．その理由は前節での保存的拡大定理の証明に必要な補題群の証明とそれで用いている定義から判る．

前節の証明は，型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ を表明判別式 $C, \Gamma, \Delta \triangleright e \leqslant e : \sigma$ とは異なるクラスの判別式として扱う事により，部分型／型付け判別式の導出中で部分型／型付け判別式に関する導出と表明判別式に関する導出とを分離する方法を用いて達成した．そのポイントは定義 3.16 で導出木の主部分を定義する事にある．

しかしながら，もしも型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ を $C, \Gamma, \Delta \triangleright e \leqslant e : \sigma$ という制限された形の表明判別式として扱っていたならば，導出木の主部分を定義する事は不可能であった．何故ならば，[A-TRANS] 則 — 極めて基本的で不可欠な規則なので除く事はできない — が存在する為に，$C, \Gamma, \Delta \triangleright e \leqslant e : \sigma$ の導出に於いて一般の形の不等式を持つ表明判別式 — $C, \Gamma, \Delta \triangleright e_1 \leqslant e_2 : \sigma$ で $e_1 \not\equiv e_2$ — が現れる可能性を防ぐ事はできない．従って，FUNIQ の定式化では — 本論文での扱い通り — **型付け判別式を表明判別式とは独立なクラスの判別式として導入する必要がある．**

本論文での型付け判別式と表明判別式とに相当する 2 種類の判別式を持つ型理論は他にも存在する．その中でも最も広く知られている型理論は Martin-Löf (1984) の直観主義的型理論 (ITT) の体系である．ITT では，型付け判別式 $e \in \sigma$ が等式判別式 $e = e \in \sigma$ とは独立に導入されている．しかしながら，何故，これら二つの判別式

1) 実際，本論文で報告する研究の初期段階に Henk Barendregt と議論する機会があり，その際，彼から「君の体系では命題 3.24 は成立するのか？」と尋ねられ「成立する」と答えたところ，「それならば，型付け判別式とその為の規則群は除き型付け判別式は表明判別式の特殊な形として扱った方が良い」とアドバイスされた事がある．その後，一時期は彼のアドバイスに従う方向で検討したが，最終的に本節で述べる事実が判明し，現在の形の定式化となった．

を別々に導入する必要があるのか，という理由は，彼自身の論文 (1975, 1982, 1998) やモノグラフ (1984) にも，また，計算機科学向けに書かれた ITT の優れた解説書である Nordström, Petersson and Smith (1990) にも全く触れられていない[2)]．

一方，本論文の体系 FUNIQ の場合，型付け判別式と表明判別式とを区別せねばならない事は，少なくともメタ理論的（証明論的）な理由が存在する事が判明した．この事は，FUNIQ という型理論を理解する上でも重要なポイントである．

さて，以上の 2 種類の判別式の区別が FUNIQ の証明論から必要であったという事は，逆に言うと，本節冒頭の命題の逆が成立するのかしないのかが不明である，という意味である．そこで，それに関する予想を与えて，本節の纏めとする．

予想 3.25 （FUNIQ の型付けに関する基本予想）
任意の $C, \Gamma, \Delta, e \in \mathbf{Exp}$ および $\sigma \in \mathbf{Type}$ に対して，

$$\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \rhd e : \sigma \quad \Longleftrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \rhd e \leqslant e : \sigma.$$

無論，左から右への含意は冒頭の命題として証明済であるので，予想として証明（もしくは反例）を待っているのは，右から左への含意である．しかしながら，この予想の証明が可能だとしても，その証明は FUNIQ の導出に関する非常に詳細でかつ注意深い場合分けを用いた分析が必要であると思われる．

3.4　詳細化型の導入の一般性

本論文では，関数的プログラムに関して或る種の部分正当性を表現できる不等式の形の表明を仕様記述の手段として許す詳細化型を Cardelli and Wegner の Fun に導入する事で，仕様とプログラムとを同一の言語で記述する為の広帯域言語 Funiq を構成し，その性質を様々な側面から検討している．

2) 2004 年 3 月 29 日に慶應大学へ来られた Martin-Löf に，ITT で型付け判別式を等式判別式と独立に扱っている理由が体系の形式的性質に基づくものか否かをお尋ねした所，「純形式的には必要ないが，等式判別式 $e_1 = e_2 : \sigma$ の概念は左右両辺の式に対する型付け $e_1 : \sigma$, $e_2 : \sigma$ を前提としている．従って，概念的あるいは meaning-theoretical には型付けの方が等式よりも基本的であるので，型付け判別式を等式判別式とは独立なものとして導入した」との答を頂いた．

$$[\text{S-ALL}_{\text{F}_\leq}] \quad \frac{C \triangleright \tau_1 <: \sigma_1 \quad C[t <: \tau_1] \triangleright \sigma_2 <: \tau_2}{C \triangleright \forall t{<:}\sigma_1.\sigma_2 <: \forall t{<:}\tau_1.\tau_2} \qquad (t \notin \mathrm{FTV}(C))$$

$$[\text{S-EXIST}_{\text{F}_\leq}] \quad \frac{C \triangleright \sigma_1 <: \tau_1 \quad C[t <: \sigma_1] \triangleright \sigma_2 <: \tau_2}{C \triangleright \exists t{<:}\sigma_1.\sigma_2 <: \exists t{<:}\tau_1.\tau_2} \qquad (t \notin \mathrm{FTV}(C))$$

図 3.1　$\mathrm{F}_\leq$ 固有の部分型関係に関する推論規則

その場合，基本的な問題として浮かび上がるのは，Fun という型付き関数的プログラミング言語をベース言語として不等式表明に基づく詳細化型を導入し広帯域言語を得る，という本論文での広帯域言語の構成方法が様々なベース言語に適用できる一般的な方法なのか，それとも，ベース言語が Fun という特定の言語のみでしか通用しない普遍性に欠ける方法に過ぎないのか，という事である．

本節では，3.2 節での保存性拡大定理の証明を再検討し，本論文での詳細化型の導入による広帯域言語の構成方法がベースの型付き関数的プログラミング言語を選ばない一般性の高い方法だという事を幾つかの例を用いて示す．

最初の具体例として，一見しただけでは Fun と非常に良く似ているが型の体系が持つ性質が Fun とは全く異なる型理論に対して，本論文での詳細化型による拡張を試みる．その体系とは Curien and Ghelli (1991, 1992) での $\mathrm{F}_\leq$ （“F sub” と読む）という型理論である（以下，その言語も $\mathrm{F}_\leq$ と呼ぶ）． Cardelli, Martini, Mitchell and Scedrov (1991) での類似の体系 $\mathrm{F}_{<:}$ も本節の議論の範囲では $\mathrm{F}_\leq$ と等価である．

言語としての $\mathrm{F}_\leq$ は Fun と全く同じである．即ち，図 2.1 で “(*)” が付いていない生成規則のみを用いて生成される言語である（オリジナルの $\mathrm{F}_\leq$ には，オリジナルの Fun と同様，不動点再帰式は存在せず，またレコード型や可変型も含まないが，ここでは追加されているものとして扱う．但し，それらの追加は本節の議論には全く影響しないので，それらの構文がないと考えても一向に差し支えない）．

さて，その型システムを定式化した型理論としての $\mathrm{F}_\leq$ は判別式の種類と形式に関しては FUN と全く異ならない．即ち，判別式に関しては部分型と型付けとの 2 種類があり，型付け判別式での実現変数に対する型付け基底は空でなければならない．

また，型付けに関する公理と推論規則に関しても $\mathrm{F}_\leq$ は FUN と全く同一である．即ち，図 2.3 で “(*)” が付いていないものである．

$F_{\leq}$ と FUN との違いは，部分型関係に関する公理と推論規則とに関してであり，FUN のそれら — 図 2.3 で “(*)” が付いていないもの — の中から多相型と抽象型とに関する [S-ALL] と [S-EXIST] を除き，代わりに，上の図 3.1 に示した二つの推論規則 [S-$ALL_{F_{\leq}}$] と [S-$EXIST_{F_{\leq}}$] とを加えたのが $F_{\leq}$ の部分型関係に関する規則群である．[S-$ALL_{F_{\leq}}$] と [S-ALL] との違いは，多相型同士の部分型関係に於いて束縛型変数の上限の型の間に部分型関係を許すか上限の型は同一な場合に限るか，という点である．[S-$EXIST_{F_{\leq}}$] と [S-EXIST] との違いも同様である（但し，前提での上限の型 σ_1 と τ_1 との間の部分型関係の向きが [S-$ALL_{F_{\leq}}$] とは逆）．

以上の $F_{\leq}$ と FUN との違いは，一見しただけでは非常に些細な差に過ぎない．しかも部分型に関する新しい推論規則の [S-$ALL_{F_{\leq}}$] と [S-$EXIST_{F_{\leq}}$] とでの上限の型 σ_1, τ_1 間の部分型関係を許す事は，その直感的意味からは極めて自然である．

しかし，Pierce (1992) が証明した通り，**型理論 $F_{\leq}$ は決定不能な体系である．** Howard (1969) の「命題 = 型」的観点からは，$F_{\leq}$ が FUNIQ の詳細化型の如き述語論理レベル相当の内容を一切含まず内容的には命題論理レベルの型の体系に過ぎぬ事を考えれば驚くべき性質である．この事は，言語としては $F_{\leq}$ は Fun と同一であるにも拘らず，型理論としての $F_{\leq}$ は FUN とは全く異質である事を示している．

そこで，FUN とは非常に性質の違う $F_{\leq}$ に詳細化型を導入した型理論 — FUN に対する FUNIQ 相当のもの — を考え，$F^{IQ}_{\leq}$ と呼ぶ．言語としての $F^{iq}_{\leq}$ は Funiq と同一である．型理論としての $F^{IQ}_{\leq}$ は，判別式の種類と形式に関しては FUNIQ と同一であり，公理／推論規則については図 2.2 ～ 図 2.4 での FUNIQ の規則の中で，[S-ALL] と [S-EXIST] とを，各々，[S-$ALL_{F_{\leq}}$] と [S-$EXIST_{F_{\leq}}$] とに差し替えて得られる．この時，次の定理が成立する．

定理 3.26 （保存的拡大定理 — $F^{IQ}_{\leq}$ 版）

型理論 $F^{IQ}_{\leq}$ はその部分型理論 $F_{\leq}$ の保存的拡大である．即ち，$F^{IQ}_{\leq}$ の任意の表明無し判別式 Σ に対して，

$$\vdash_{F^{IQ}_{\leq}} \Sigma \quad \Longleftrightarrow \quad \vdash_{F_{\leq}} \Sigma.$$

$$[\text{T-TABS}_{\text{FB}}]\quad \frac{C[t<:\sigma],\Gamma,\Delta \triangleright e:\tau}{C,\Gamma,\Delta \triangleright (\Lambda t<:\sigma.e):\forall t<:\sigma.\tau}\qquad (t \notin \text{FTV}(C,\Gamma,\Delta))$$

$$[\text{A-TABS}_{\text{FB}}]\quad \frac{C[t<:\sigma],\Gamma,\Delta \triangleright e \leqslant e':\tau}{C,\Gamma,\Delta \triangleright (\Lambda t<:\sigma.e) \leqslant (\Lambda t<:\sigma.e'):\forall t<:\sigma.\tau}\qquad (t \notin \text{FTV}(C,\Gamma,\Delta))$$

図 3.2　*F*-限定 Fun 特有の型付け規則と *F*-限定 Funiq 特有の表明規則

この証明は，3.2 節での系 3.22 の証明と全く同一である．つまり，FUN とは全く性質の異なる型理論 $\text{F}_{\leq}$ に対しても FUN に対する拡張と同じ様に詳細化型を導入できる事が，この定理から理解できる．

以上の $\text{F}_{\leq}$ に対する詳細化型の導入が保存的拡大性を保つ事は，型理論としての $\text{F}_{\leq}$ は FUN とは全く性質が異なるとは言えどもベースの言語が Fun と同一であったので当然と感じるかも知れない．そこで，次はベースの言語自体が Fun とは異なる場合を取り上げる．

新しいベース言語は F-限定 Fun という言語[3]である．この言語は Fun の多相型での全称限量化 $\forall$ の上限型に Canning, Cook, Hill, Orthoff and Mitchell (1989) で提案されている F-限定という形の拡張を導入した[4] 2 階の型付き λ-計算の体系である．

この拡張された F-限定 Fun という言語は，BNF の生成規則として表現可能な文脈自由言語としては図 2.1 で定まる Fun と同一であるが，実際に良形と認められるストリングの集合としての言語としては Fun とは異なっている．その違いは図 3.2 に示した型付け規則で表わされている．同図の $[\text{T-TABS}_{\text{FB}}]$ が図 2.3 での Fun オリジナルの [T-TABS] と異なる点は，束縛対象となる型変数 t に関する制約条件である．オリジナルの [T-TABS] では束縛対象の型変数 t はそれに対する上限型 σ 中に自由に現れる事が許されないのに対して，$[\text{T-TABS}_{\text{FB}}]$ では $t \in \text{FTV}(\sigma)$ である事が許

3) これも Fun のバリエーションに過ぎないではないか，と思われるかも知れないが，Fun が 2 階の型付き λ-計算の体系として充分に一般的で豊かな言語機能を提供しているので，2 階の型付き λ-計算の体系への詳細化型の導入に関して議論しようとする以上，Fun やそのバリエーションがベース言語となるのは必然的である．

4) 全くの形式的な立場からは抽象型での存在限量化 $\exists$ の方も同じ形で拡張して良いと思われるが，存在限量化での F-限定の計算機科学的な意義は全く不明である — Canning 達の論文でも $\exists$ での上限型に対する F-限定化は全く検討されていない — ので，ここでは，Canning 達の原論文通り，$\forall$ の上限型に対してのみ F-限定による拡張を考える．

されている．そこで，F-限定 Fun では，上限型を単に σ で表わさずに型変数 t が自由に出現可能だという事を強調する為に，$F[t]$ — ここで，$F[t]$ は t を含み得る型の表式 — という形で書き表わす．これが “F-限定” という名称の由来である．

この型変数の出現に関する違いは，実用的に見ると非常に大きな差をもたらす．実際，上の Canning 達の論文では，Fun では充分には成功しなかったオブジェクト指向での継承を 2 階の型付き λ-計算での部分型関係として捉える事に関して，この F-限定による拡張だけで相当な成功を収めた．

そこで，図 3.2 の [T-TABS$_{\mathrm{FB}}$] 則で [T-TABS] 則を置き換えて定まる F-限定 Fun の為の型理論を F-限定 FUN と呼ぶ事とし，詳細化型を導入した言語を F-限定 Funiq，その型理論として，FUNIQ の [A-TABS] 則を図 3.2 の [A-TABS$_{\mathrm{FB}}$] 則に変更したものを F-限定 FUNIQ と呼ぶ．やはりこの場合も，次の保存的拡大定理が成立する．

定理 3.27 （保存的拡大定理 — F-限定 FUNIQ 版）

型理論 F-限定 FUNIQ はその部分型理論 F-限定 FUN の保存的拡大である．

即ち，F-限定 FUNIQ の任意の表明無し判別式 Σ に対して，

$$\vdash_{F\text{-限定 FUNIQ}} \Sigma \quad \Longleftrightarrow \quad \vdash_{F\text{-限定 FUN}} \Sigma.$$

この証明も系 3.22 の証明と全く同一である．

上で参照した Canning 達の論文にもある通り，実は，F-限定という言語拡張は，少なくとも直感的な意味では再帰型の概念を隠れた形で言語の中に導入した事に相当する．そこで，詳細化型の導入の具体的な事例の最後として，再帰型をコッソリとではなく堂々とベース言語に導入する[5)]場合を考える．

5) 但し，意味論的には詳細化型を含む再帰型は大いなる問題を生じ，第 5 章で述べる通り，詳細化型を含む再帰型に対して表示的意味を与える事は未解決である．本節では，あくまでも詳細化型の証明論的な側面のみを議論している．

$$\sigma ::= \ldots \mid \mu t.\sigma$$ 再帰型.

図 3.3　FunnY の再帰型の構文

$$\text{[T-FOLD]}\quad \frac{C,\Gamma,\Delta \triangleright e : \sigma[t := \mu t.\sigma]}{C,\Gamma,\Delta \triangleright e : \mu t.\sigma}$$

$$\text{[T-UNFOLD]}\quad \frac{C,\Gamma,\Delta \triangleright e : \mu t.\sigma}{C,\Gamma,\Delta \triangleright e : \sigma[t := \mu t.\sigma]}$$

$$\text{[A-FOLD]}\quad \frac{C,\Gamma,\Delta \triangleright e \leqslant e' : \sigma[t := \mu t.\sigma]}{C,\Gamma,\Delta \triangleright e \leqslant e' : \mu t.\sigma}$$

$$\text{[A-UNFOLD]}\quad \frac{C,\Gamma,\Delta \triangleright e \leqslant e' : \mu t.\sigma}{C,\Gamma,\Delta \triangleright e \leqslant e' : \sigma[t := \mu t.\sigma]}$$

図 3.4　FUNNY 特有の型付け規則と FUNIQY 特有の表明規則

型に関する再帰的定義を許す言語 FunnY は図 3.3 に示す型に関する不動点演算子の形[6] として再帰型を Fun に追加したものである．この言語の型理論 FUNNY は，FUN に図 3.4 に示した [T-FOLD] 則と [T-UNFOLD] 則とを追加したものとする．なお，再帰型に関しては部分型関係に関する推論規則は存在しない．

なお， Fun や同系統の 2 階の型付き λ-計算の体系に再帰型を導入する場合の推論規則に関しては幾つかの選択肢が存在する．そこで，再帰型の為の推論規則の代表的な選択肢に関して，簡単に纏めておく．

まず第一の選択肢としては，再帰型に対する型付け規則の選び方である．上の図 3.4 に示した規則は Cardone (1989, 1991) で与えられたものであるが，これらの代わりに，再帰型の定式化に於いて，式に **fold** e と **unfold** e という二つの構文を追

6) 但し，型に関しては式での関数抽象に相当する型変数を束縛する構文がないので，再帰型の構文自身で不動点としての再帰型を表わす型変数 t を束縛せねばならない．

加し，[T-FOLD] 則と [T-UNFOLD] 則とを，上図の形ではなく，

$$[\text{T-FOLD}'] \quad \frac{C, \Gamma, \Delta \ \triangleright \ e : \sigma[t := \mu t.\sigma]}{C, \Gamma, \Delta \ \triangleright \ (\mathbf{fold}\ e) : \mu t.\sigma}$$

$$[\text{T-UNFOLD}'] \quad \frac{C, \Gamma, \Delta \ \triangleright \ e : \mu t.\sigma}{C, \Gamma, \Delta \ \triangleright \ (\mathbf{unfold}\ e) : \sigma[t := \mu t.\sigma]}$$

という — 例えば Amadio (1991) や Pierce (2002, p. 276) 等で採用されている — 形とする選択肢がある．後の型付け推論規則群は：

- 再帰型の導入と除去とが型付け判別式の主部にある式の構文として明示されており他の型構成方法と同様に扱える；
- 自明な無限の導出を許さない — 例えば，図 3.4 に示した規則群では [T-FOLD] 則と [T-UNFOLD] 則との適用を交互に反復する事で無限の導出が可能だが，後の規則群では各規則の前提の判別式での式の大きさは結論のそれよりも必ず小さいので無限の交互反復は不可能である

という長所がある．但し，この場合，この形式の再帰型を導入した言語に対する操作的意味論を（次章で述べる様な）簡約関係の形で与える上では，簡約概念として

$$\boldsymbol{\beta_\mu} \triangleq \{\langle \mathbf{fold}\ (\mathbf{unfold}\ e), e\rangle \mid e \in \mathbf{Exp}\} \cup \{\langle \mathbf{unfold}\ (\mathbf{fold}\ e), e\rangle \mid e \in \mathbf{Exp}\}$$

を与え，更に，この簡約概念に対応して表明推論規則として二つの [A-β] 則（各々，上の和集合の各成分集合に対応）を追加する必要がある．

本節の趣旨を外れるが再帰型に関して少し議論しておくと，図 3.4 で示した型付け規則と **fold**/**unfold** を導入した後の規則とは再帰型に所属する「値」に対する考え方が異なっている．図に示した規則の立場では，型 $\mu t.\sigma$ と型 $\sigma[t := \mu t.\sigma]$ とは全く同一の「値の集まり」を表わしていると考えるのに対して，後者の **fold**/**unfold** を用いる立場では，型 $\mu t.\sigma$ と型 $\sigma[t := \mu t.\sigma]$ とが表わす「値の集まり」は集合としては 1 : 1 対応を持ち同型ではあるが異なっており，これら二つの集合の間の同型射を与えるのが **fold** $(\cdot)$ と **unfold** $(\cdot)$ という基本操作だと考える．

その結果，何れの型付け規則対を選ぶかによって型検査の実装上等での様々な違いが生ずる事は Pierce (2002) が指摘している通りである．しかし，これらの点は本節での議論には無関係なので，図 3.4 に示した簡単な形の型付け規則を採用する．

また，再帰型を導入すると，式の上の不動点再帰 **fix** を含まなくとも計算が停止しない — しかも正しく型付け可能な — 式が書けてしまう点に注意する必要がある．実際，図 3.4 に示した型付け規則により再帰型を導入した場合，

$$\Omega \triangleq (\lambda\,\mathrm{x}{:}\,\mu\mathrm{t}.(\mathrm{t}\to\mathrm{t}).\mathrm{xx})(\lambda\,\mathrm{x}{:}\,\mu\mathrm{t}.(\mathrm{t}\to\mathrm{t}).\mathrm{xx})$$

とすれば，$\vdash \varnothing,\varnothing,\varnothing \triangleright \Omega : \mu\mathrm{t}.(\mathrm{t}\to\mathrm{t})$ である．後の **fold**/**unfold** を用いる型付け規則によって再帰型を導入した場合も

$$\Omega' \triangleq (\lambda\,\mathrm{x}{:}\,\mu\mathrm{t}.(\mathrm{t}\to\mathrm{t}).(\mathbf{unfold}\ \mathrm{x})\mathrm{x})(\mathbf{fold}\ (\lambda\,\mathrm{x}{:}\,\mu\mathrm{t}.(\mathrm{t}\to\mathrm{t}).(\mathbf{unfold}\ \mathrm{x})\mathrm{x}))$$

とすれば，やはり $\vdash \varnothing,\varnothing,\varnothing \triangleright \Omega' : \mu\mathrm{t}.(\mathrm{t}\to\mathrm{t})$ である．

また，再帰型同士の間に部分型関係を認める選択肢も存在する．再帰型の間の部分型関係に関する推論規則としての一つの候補として提案されているのは，Cardelli (1988) が彼の Amber という言語の為に導入した

$$\text{[S-RECTYPE]}\quad \frac{C[s <: t] \triangleright \sigma <: \tau}{C \triangleright \mu s.\sigma <: \mu t.\tau} \qquad (s \notin \mathrm{FTV}(\tau),\ t \notin \mathrm{FTV}(\sigma))$$

という規則である（Cardone (1991) では [Amber] という名前で呼ばれている）．この規則は型変数 s を含む型の表式 σ や t を含む τ が表わす「(型の上の）関数」が部分型関係に関して単調である場合に限り両者の間に部分型関係を認めるものである．この規則の存否も本節の議論には影響しないので，やはり省略する．

今までと同様，言語 FunnY に詳細型を導入し，それで得られる言語を FuniqY と呼び，その型理論として FUNNY に図 3.4 に示した表明に関する二つの推論規則を追加したものを FUNIQY と呼ぶ事にすると，やはり以下を示す事ができる．

定理 3.28 （保存的拡大定理 — FUNIQY 版）

型理論 FUNIQY はその部分型理論 FUNNY の保存的拡大である．即ち，FUNIQY の任意の表明無し判別式 Σ に対して，

$$\vdash_{\mathrm{FUNIQY}} \Sigma \quad\Longleftrightarrow\quad \vdash_{\mathrm{FUNNY}} \Sigma.$$

実は，系 3.22 の証明やその証明の為の補題群の証明，更には，その証明の為の補助概念の定義に於いて，ベースの型理論 (FUN, $\mathrm{F}_{\leq}$, F-限定 FUN, FUNNY) の公理や推論規則に依存している箇所は全く存在しない．その事は 3.2 節の内容を詳細に点検すれば理解でき，その証明は本論文での FUN よりも複雑な型理論，例えば任意の高階の型の上の関数（型構成子）に関する $\forall$-限量化を許す Girard (1972) の各体系 — $\mathrm{F}_2\,(=\mathrm{F}), \mathrm{F}_3, \ldots, \mathrm{F}_\omega$ — や F_ω に共通集合型を追加した Compagnoni (1995) の型理論 $\mathrm{F}^{\omega}_{\wedge}$ 等の様々な高階型付き λ-計算の為の型理論に適用可能である．即ち，各々の型理論に対してその詳細化型による拡張型理論は保存的拡大である，という命題が常に成立し，それらの命題の証明は，定理 3.26 ～ 定理 3.28 と同様に 3.2 節での系 3.22 の証明と全く同一となる．

そして， 3.2 節での証明を振り返ると，可能な型構成法としては，レコード型・可変型・多相型等の通常の構成法に限らず， Girard (1987) での命題線型論理の論理演算子に対応するものでも保存的拡大性に関する証明は影響を受けないと考えられる．即ち， 3.2 節での保存的拡大性の証明は，ベースの型理論が Coquand and Huet (1985, 1988) の Calculus of Constructions， Martin-Löf (1984) の ITT， Luo (1994) の ECC 等の様な型が式の値に依存する「依存型」を含む型理論でなければ，オブジェクト指向の為の Abadí and Cardelli (1996) の ς-計算の型付き版や各種の型付き並行プロセス計算等，型付き λ-計算とは大幅に異なる型理論をベースの型理論とする場合でも証明は通用し，その結果，次ページに示す極めて一般的な主張[7]が成立すると予想される（なお，予想中の簡約に関する諸用語は次章を参照せよ）．

従って，**本論文での不等式表明に基づく詳細化型の導入方法は，少なくとも証明論的な側面だけに限定して言えば，充分に一般性がある**と考えられる．

7) この予想が主張している事を端的に述べれば，**「どんなプログラミング言語の型システムでも持っていらっしゃい．元の型システムの性質を保ったまま詳細化型を加えて広帯域言語化して差し上げましょう」**である．

予想 3.29（詳細化型の一般性に関する予想）

L を型付き計算の言語，$\mathcal{T}_L$ をその型理論とし，

(1) L の型は構文要素として式（型付け対象の構文クラス）を含まない．

(2) L の操作的意味を与える簡約概念 $\boldsymbol{R}_L$ の定義では，縮約に伴って自由変数が新たに現れる事はない．即ち，

$$\langle e_1, e_2 \rangle \in \boldsymbol{R}_L \quad \Longrightarrow \quad \mathrm{FV}(e_1) \supseteq \mathrm{FV}(e_2).$$

(3) L での簡約関係は主部簡約定理を成立させる．

(4) 型理論 $\mathcal{T}_L$ は図 2.2 と同様の形で形式化可能な部分型の概念を含み，型付けに関しては [T-SUBTYPE] 則に相当する推論規則を含む．

という全条件を満たすとする．この型付き言語 L を本論文の方法，即ち，式として実現変数を追加し，構文クラスとして表明を追加し，型として本論文と同じ形の詳細化型を追加して得られる拡張言語を L^{IQ} とし，その型理論 $\mathcal{T}_L^{\mathrm{IQ}}$ をベース言語 L の型理論 $\mathcal{T}_L$ に以下の拡張を施したものとする．

(a) 部分型に関する推論規則として [S-REFINE] 則を加える．

(b) 簡約概念の全ての対（但し，型付け可能な式同士の対）は不等式表明の公理のインスタンスとする．即ち，

$$\langle e_1, e_2 \rangle \in \boldsymbol{R}_L \quad \Longrightarrow \quad e_1 \leqslant e_2 : \sigma.$$

（σ は e_1 の型付け可能性と主部簡約定理とから存在が保証される型）

(c) 型付け推論規則として式の表明に関する充足性を型付けに反映させる [T-REFINE] 則を追加する．同様に，表明に関しても [A-REFINE] 則を与える．

(d) 表明に関する推論規則として，通常変数に関する全称化の導入の為の [A-**forall**-I] 則と消去の為の [A-**forall**-E] 則とを与える．

(e) 表明に関する推論規則として，型付け文脈に関する推論規則ならびに構文との両立性を満たす為に必要な規則を型付けに関する推論規則と並行な形で与える．

この時，$\mathcal{T}_L^{\mathrm{IQ}}$ は $\mathcal{T}_L$ の保存的拡大である．

第4章

Funiqの簡約論的性質

— 主部簡約性を中心に —

本章では広帯域言語 Funiq の簡約（操作的意味論）に関する基本的な性質を検討する．その為に，まず Funiq の式に対する操作的意味を簡約概念／簡約関係の形で定義する．また， 3.1 節の定義 3.1 で示した抽象型のコード化が簡約に関しても適切に振舞う事を示す．

次に，簡約に関して式の型が保存される事を保証する主部簡約性を示し，その証明を与える．この性質の場合，簡約関係と基本的には相似な形の表明を含む詳細化型の存在が証明を複雑にし，従って， Funiq での主部簡約性の証明では， Fun での同性質の証明と比べて遥かに繁雑にならざるを得ない，という事が観察される．

第三に， Funiq から不動点再帰式 **fix** e を除いた部分言語の場合，簡約 — つまり式の計算 — は必ず停止する，という強正規化性について述べる．この性質の場合，Funiq 独自に証明する必要はなく，詳細化型を含まない Fun （**fix** e を追加していないオリジナル版）に対する強正規化性の証明に帰着させる事ができる．

更に，式の中に簡約可能な箇所が複数ある場合，どの簡約から先に行なっても簡約結果は一致する，という合流性の証明を与える．この証明は， Fun や型無し λ-計算での合流性に帰着させる事はできず，独立に証明する必要があるが，証明自体はそれらの体系での合流性の証明と全く同様の方法で行なう事ができる．

最後の二つの節で，形式的厳格な変数の簡約での振舞を検討する．最初は通常の β-簡約に基づいて，また，最後の節では $\text{Funiq}_{\perp}$ に対する値呼びの為の β-簡約（β^v-簡約）を定義して，形式的厳格性の簡約論的特徴付けを行なう．

4.1 Funiq に於ける簡約関係の定義

まず，Funiq での計算としての操作的意味を表わす簡約の概念を特定の形を持つ式の間の 2 項関係として定義する．

定義 4.1 （簡約概念 β）

簡約概念 (notion of reduction) $\boldsymbol{\beta}$ とその部分簡約概念 $\boldsymbol{\beta}_{\to}$, $\boldsymbol{\beta}_{\{\cdots\}}$, $\boldsymbol{\beta}_{[\cdots]}$, $\boldsymbol{\beta}_{\forall}$, $\boldsymbol{\beta}_{\exists}$, $\boldsymbol{\beta}_{\mathbf{fix}}$ を，各々，以下に示す形の式の 2 項対の集合として定義する：

$$\boldsymbol{\beta}_{\to} \triangleq \{\langle (\lambda\, x{:}\,\sigma.e_1)e_2, e_1[x := e_2]\rangle \mid e_1, e_2 \in \mathbf{Exp}\};$$

$$\boldsymbol{\beta}_{\{\cdots\}} \triangleq \{\langle \{l_1 = e_1, \ldots, l_n = e_n\}.l_i, e_i\rangle \mid e_1, \ldots, e_n \in \mathbf{Exp} \text{ and } 1 \le i \le n\};$$

$$\boldsymbol{\beta}_{[\cdots]} \triangleq \{\langle \mathbf{case}\ [\, l_i = e_0\,]\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n, e_i e_0\rangle \mid e_0, \ldots, e_n \in \mathbf{Exp} \text{ and } 1 \le i \le n\};$$

$$\boldsymbol{\beta}_{\forall} \triangleq \{\langle (\Lambda\, t{<:}\sigma.e)[\rho], e[t := \rho]\rangle \mid e \in \mathbf{Exp} \text{ and } \sigma, \rho \in \mathbf{Type}\};$$

$$\boldsymbol{\beta}_{\exists} \triangleq \{\langle \mathbf{unpack}\ (\mathbf{pack}_{\exists t<:\rho.\sigma}\ e_1\ \mathbf{with}\ \tau)\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2, e_2[t := \tau][x := e_1]\rangle \mid e_1, e_2 \in \mathbf{Exp} \text{ and } \sigma, \tau, \rho \in \mathbf{Type}\};$$

$$\boldsymbol{\beta}_{\mathbf{fix}} \triangleq \{\langle \mathbf{fix}\ e, e(\mathbf{fix}\ e)\rangle \mid e \in \mathbf{Exp}\};$$

$$\boldsymbol{\beta} \triangleq \boldsymbol{\beta}_{\to} \cup \boldsymbol{\beta}_{\{\cdots\}} \cup \boldsymbol{\beta}_{[\cdots]} \cup \boldsymbol{\beta}_{\forall} \cup \boldsymbol{\beta}_{\exists} \cup \boldsymbol{\beta}_{\mathbf{fix}}.$$

次に簡約概念から導かれる簡約関係（単に「簡約」とも呼ぶ）を以下の様に定義するが，その為には，「両立可能な関係」という言葉を必要とする．更に，式上の 2 項関係の両立可能性を定義するには，文脈の概念が必要である．そこで，この二つを順を追って定義する．

定義 4.2 （文脈）

(1) **文脈** $D[\ ]$ とは，構文上で式 —**Exp** の元 — が出現できる箇所に 0 個以上の穴の空いた式を表わす.

(2) $D[e]$ は $D[\ ]$ 中の全ての穴を一斉に式 e で置換した式を表わす.

$D[e]$ の場合， $D[\ ]$ の穴を置換する式 e での自由変数が $D[e]$ 中で束縛出現となり得る事に注意する必要がある．この点が “$e[x := e']$” と全く異なる.

定義 4.3 （両立可能性）

式の上の 2 項関係 $S \subseteq \mathbf{Exp} \times \mathbf{Exp}$ が簡約概念 $\boldsymbol{R}$ によって生成される**両立可能**な関係である，とは，任意の文脈 $D[\ ]$ に対して， $\langle e_1, e_2 \rangle \in \boldsymbol{R}$ ならば常に $\langle D[e_1], D[e_2] \rangle \in S$ を満たす事である.

必要な準備が整ったので，簡約概念から生成される簡約関係ならびに関連する用語の定義を以下に与える.

定義 4.4 （簡約関係）

$\boldsymbol{R}$ を簡約概念とする時， $\boldsymbol{R}$ から導かれる簡約関係を **Exp** 上の 2 項関係として以下の様に定義する：

$$\longrightarrow_R \triangleq \boldsymbol{R} \text{ で生成される両立可能関係；}$$

$$\longrightarrow\!\!\!\!\!+_R \triangleq \longrightarrow_R \text{ の推移的閉包；}$$

$$\longrightarrow\!\!\!\!\!\!\rightarrow_R \triangleq \longrightarrow_R \text{ の反射推移的閉包；}$$

$$=_R \triangleq \longrightarrow_R \text{ の反射対称推移的閉包.}$$

なお， $\longrightarrow_R$, $\longrightarrow\!\!\!\!\!\!\rightarrow_R$, $=_R$ を，各々，**単一ステップ** $\boldsymbol{R}$**-簡約**， $\boldsymbol{R}$**-簡約**， $\boldsymbol{R}$**-変換**と呼ぶ.

定義 4.5（簡約基・縮約結果）

$\boldsymbol{R}$ を簡約概念とし $\langle e_1, e_2\rangle \in \boldsymbol{R}$ である時，e_1 を **$\boldsymbol{R}$-簡約基**，e_2 を **$\boldsymbol{R}$-縮約結果**と呼ぶ．また，$\longrightarrow\!\!\!\!\!\rightarrow_R$ を R-簡約とする時，$e_1 \longrightarrow\!\!\!\!\!\rightarrow_R e_2$ ならば e_2 を e_1 の **$\boldsymbol{R}$-簡約結果**と呼ぶ．最後に，R-簡約基を含まない式を **$\boldsymbol{R}$-正規形**と呼び，$\boldsymbol{R}$ が文脈から明らかな場合，単に**正規形**と呼ぶ．

以上で言語 Funiq に対する β-簡約の定義を終わる．第 2 章で述べた通り，Funiq は Cardelli and Wegner の Fun を部分言語として含んでいる．そこで，以上で定義した簡約概念 $\boldsymbol{\beta}$（および各部分簡約概念）を $\mathbf{Exp}_{\mathrm{Fun}}$ 上に制限し，その制限された簡約概念から上と同様に定義される関係を Fun での β-簡約と定める．

本節の最後として，3.1 節の定義 3.1 で与えた抽象型の多相型と関数型とによるコード化が簡約に関しても以下で述べる意味で適切に振舞う事を示す．

定理 4.6（$\boldsymbol{\beta_\exists}$ の $\boldsymbol{\beta_\forall}$ と $\boldsymbol{\beta_\rightarrow}$ とによる模倣可能性）

$\mathrm{Funiq}^{-\{\exists\}}$ を Funiq から $\exists$ 型と **pack**/**unpack** 式を除いた部分言語とする．この部分言語の上の簡約概念を Funiq の簡約概念 $\boldsymbol{\beta}$ の定義から $\boldsymbol{\beta_\exists}$ を除いたものとして定義し $\boldsymbol{\beta}^{-\{\exists\}}$ で表わす．

e を Funiq の任意の式とする時，$\|e\|$ を e 中の $\exists$ 型，**pack** 式，**unpack** 式の全てを定義 3.1 でのそれらのコード化に従って展開した $\mathrm{Funiq}^{-\{\exists\}}$ の式とする．この時，

$$\forall e' \in \mathbf{Exp}.\left[e \longrightarrow\!\!\!\!\!\rightarrow_\beta e' \quad\Longrightarrow\quad \|e\| \longrightarrow\!\!\!\!\!\rightarrow_{\beta^{-\{\exists\}}} \|e'\|\right].$$

証明

$e \longrightarrow\!\!\!\!\!\rightarrow_\beta e'$ の簡約の長さ s に関する帰納法で証明する．

(1) $s = 0$ の場合

$e' \equiv e$ であるので，$\|e'\| \equiv \|e\|$ であるから明らかに成立．

(2) $s = k+1$ の場合

この場合，$e \longrightarrow\!\!\!\!\!\rightarrow_\beta e'' \longrightarrow_\beta e'$ であり，帰納法の仮定から $\|e\| \longrightarrow\!\!\!\!\!\rightarrow_{\beta^-\{\exists\}} \|e''\|$ が成り立っている．最後のステップが e'' 中のどの種類の簡約基に対する縮約であったかによって場合分けする．

● $\beta_\exists$-簡約基以外の場合

$\|e''\|$ の対応する簡約基を縮約すれば，その簡約結果の式が $\|e'\|$ である事は明らかである．

● $\beta_\exists$-簡約基の場合

$\beta_\exists$-簡約基の形の式のコード化結果についての次の Funiq$^{-\{\exists\}}$ での簡約列を見よ：

$$
\begin{array}{ll}
 & \|\mathbf{unpack}\ (\mathbf{pack}_{\exists t<:\sigma.\tau_1}\ e_1\ \mathbf{with}\ \rho)\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2\| \\
\equiv & (\Lambda\, s<:\mathbf{Top}.(\lambda\, x{:}\,\forall t<:\sigma.(\tau_1 \to s).x[\rho]\|e_1\|))[\tau_2](\Lambda\, t<:\sigma.\lambda\, x{:}\,\tau_1.\|e_2\|) \\
\longrightarrow_{\beta_\forall} & (\lambda\, x{:}\,\forall t<:\sigma.(\tau_1 \to \tau_2).x[\rho]\|e_1\|)(\Lambda\, t<:\sigma.\lambda\, x{:}\,\tau_1.\|e_2\|) \\
\longrightarrow_{\beta_\to} & (\Lambda\, t<:\sigma.\lambda\, x{:}\,\tau_1.\|e_2\|)[\rho]\|e_1\| \\
\longrightarrow_{\beta_\forall} & (\lambda\, x{:}\,\tau_1[t := \rho].\|e_2\|[t := \rho])\|e_1\| \\
\longrightarrow_{\beta_\to} & \|e_2\|[t := \rho][x := \|e_1\|] \\
\equiv & \|e_2[t := \rho][x := e_1]\|.
\end{array}
$$

以上から，Funiq での $\beta_\exists$-簡約は，Funiq$^{-\{\exists\}}$ では 2 ステップの $\beta_\forall$-簡約と 2 ステップの $\beta_\to$-簡約との計 4 ステップの簡約によって表わす事ができる．

従って，最後のステップが e'' 中の $\beta_\exists$-簡約基の縮約であった場合，コード化により対応する $\|e''\|$ での $\beta_\forall$-簡約基の縮約から始まる上記の 4 ステップの簡約を行なえば良い．

証明終

上の命題は，簡約に関する或る種の性質の証明に於いては，$\beta_\exists$-簡約を $\beta_\forall$-簡約と $\beta_\to$-簡約とで代用する事により簡約概念 $\boldsymbol{\beta_\exists}$ を除く事が許され，証明での場合分けを削減できる事を意味している．その様な証明の簡略化が可能な性質としては，後で述べる強正規化性を挙げる事ができる．一方，合流性の証明の場合は以上の簡略化は許されない．これらの性質の証明で，何故，定義 3.1 の抽象型のコード化に基づいて $\beta_\exists$-簡約を除く事が許される／許されないのか，という理由に関しては，各々の性質の証明のところで説明する．

4.2　主部簡約定理の証明

まず，定理の証明で用いる補題とそれに必要な定義を与える．

定義 4.7

Funiq の型の集合 **Type** 上の 2 項関係 $\lesssim$ を以下の各条件から生成される推移的閉包として定義する．

- $\sigma \lesssim \mathbf{Top}$,
- $\iota \lesssim \iota$,
- $t \lesssim t$,
- $\tau_1 \lesssim \sigma_1$ かつ $\sigma_2 \lesssim \tau_2 \quad \Longrightarrow \quad \sigma_1 \to \sigma_2 \lesssim \tau_1 \to \tau_2$,
- 各 $1 \leq i \leq n$ について $\sigma_i \lesssim \tau_i$
  $\Longrightarrow \quad \{l_1 : \sigma_1, \ldots, l_{n+k} : \sigma_{n+k}\} \lesssim \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$ $(n, k \geq 0)$,
- 各 $1 \leq i \leq n$ について $\sigma_i \lesssim \tau_i$
  $\Longrightarrow \quad [\, l_1 : \sigma_1, \ldots, l_n : \sigma_n \,] \lesssim [\, l_1 : \tau_1, \ldots, l_{n+k} : \tau_{n+k} \,]$ $(n, k \geq 0)$,
- $\sigma \lesssim \tau \quad \Longrightarrow \quad \forall t{<:}\rho.\sigma \lesssim \forall t{<:}\rho.\tau$,
- $\sigma \lesssim \tau \quad \Longrightarrow \quad \exists t{<:}\rho.\sigma \lesssim \exists t{<:}\rho.\tau$,
- $\{r : \sigma \mid \gamma_1, \ldots, \gamma_m\} \lesssim \sigma$,
- $\sigma \lesssim \tau \quad \Longrightarrow \quad \{r : \sigma \mid \gamma_1, \ldots, \gamma_m\} \lesssim \{r : \tau \mid \delta_1, \ldots, \delta_l\}$.

ここで定義した型の上の関係 $\lesssim$ の直感的な意味は次の通りである．$\sigma \lesssim \tau$ であるとは，表明 — γ や δ — の内容やその論理的な強弱を無視すれば，型変数 — t 等 — に対する制約集合 C に依存せず，σ は τ の部分型であると σ と τ の構文の形から断言できる，という事である．

次の補題は，Funiq の型システム FUNIQ での部分型関係の導出が型の構文（関数型なのかレコード型なのか等々）を尊重する，という事を表わす．この補題は，Funiq のベースとなる体系である Cardelli and Wegner の Fun では自明であるが，我々の体系の場合，[S-REFINE] 則によって詳細化型同士が表明の論理的強弱に応じて部分型の関係となる事を許しており，また，[S-TRANS] 則によって部分型関係の推移性も許しているので，全く自明という訳ではない．

補題 4.8

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma <: \tau \quad \Longrightarrow \quad \sigma \lesssim \tau.$$

証明

部分型判別式 $C \triangleright \sigma <: \tau$ の導出木の高さ h についての数学的帰納法による.

(1) $h = 0$ の場合：

図 2.2 の部分型に関する各公理が言明を満たしている事は明らか.

(2) $h = k + 1$ の場合：

部分型に関する [S-TRANS] 則以外の各推論規則が言明を保存する事は明らか. [S-TRANS] 則が言明を保存する事は 2 項関係 $\lesssim$ が推移性に関して閉じている事から従う.

証明終

この補題の直感的な意味は次の通りである． Funiq での証明論的に定義された部分型関係 $<:$ は表明の論理的強弱に応じて部分型となる事を許した詳細化型（部分集合型）を含み複雑に見えるが，適当な型変数への制約集合 C の下で $\sigma <: \tau$ という関係 — つまり， $C \triangleright \sigma <: \tau$ という部分型判別式 — が導出できるのは，実は，型の構文に従い単純に定義される $\sigma \lesssim \tau$ が成り立つ場合に限られる，という事である（但し，後者は表明の強弱を無視しているので，逆は一般には成立しない）.

次の補題の直感的な内容は，部分型判別式の導出に沿って帰結から前提へと逆に辿る事ができる，という事である.

補題 4.9

(1) $\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2$
$\Longrightarrow$ $\vdash_{\mathrm{FUNIQ}} C \triangleright \tau_1 <: \sigma_1$ かつ $\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma_2 <: \tau_2$;

(2) $\vdash_{\mathrm{FUNIQ}} C \triangleright \{l_1 : \sigma_1, \ldots, l_m : \sigma_m\} <: \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$
$\Longrightarrow$ $m \geq n$ かつ 各 $1 \leq i \leq n$ について $\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma_i <: \tau_i$;

(3) $\vdash_{\mathrm{FUNIQ}} C \triangleright [\, l_1 : \sigma_1, \ldots, l_m : \sigma_m \,] <: [\, l_1 : \tau_1, \ldots, l_n : \tau_n \,]$
$\Longrightarrow$ $m \leq n$ かつ 各 $1 \leq i \leq m$ について $\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma_i <: \tau_i$;

(4) $\vdash_{\mathrm{FUNIQ}} C \triangleright \forall t{<:}\sigma_1.\sigma_2 <: \forall t{<:}\tau_1.\tau_2$
$\Longrightarrow$ $\sigma_1 \equiv \tau_1$ かつ $\vdash_{\mathrm{FUNIQ}} C[t <: \sigma_1] \triangleright \sigma_2 <: \tau_2$;

(5) $\vdash_{\mathrm{FUNIQ}} C \triangleright \exists t{<:}\sigma_1.\sigma_2 <: \exists t{<:}\tau_1.\tau_2$
$\Longrightarrow$ $\sigma_1 \equiv \tau_1$ かつ $\vdash_{\mathrm{FUNIQ}} C[t <: \sigma_1] \triangleright \sigma_2 <: \tau_2$.

証明

(1) について

左辺の部分型判別式の導出での [S-TRANS] 則の適用回数 l に関する数学的帰納法による.

(a) $l = 0$ の時

左辺の導出の最終ステップは [S-ARROW] 則以外にはあり得ず，この場合，右辺の主張は明らかに成立する.

(b) $l = k + 1$ の時

導出の最終ステップが [S-TRANS] 則の場合のみを考えれば良い（最終ステップとして他に可能性があるのは [S-ARROW] 則だけでその場合は (a) と同じになる）.

適当な型 ρ が存在して,

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma_1 \to \sigma_2 <: \rho, \tag{i}$$

かつ

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \rho <: \tau_1 \to \tau_2 \tag{ii}$$

で，各々の導出が共に高々 k 回の [S-TRANS] 則の適用で行なえる.

(i) に補題 4.8 を適用すれば $\sigma_1 \to \sigma_2 \lesssim \rho$ を得る．関係 $\lesssim$ の定義から ρ は **Top** か $\rho_1 \to \rho_2$ という形でなければならない．しかし，同時に (ii) に補題 4.8 を適用すると $\rho \lesssim \tau_1 \to \tau_2$ であるので ρ は **Top** ではあり得ない．従って，型 ρ は $\rho_1 \to \rho_2$ という形でなければならない．

故に，帰納法の仮定から

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \rho_1 <: \sigma_1, \tag{iii}$$

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma_2 <: \rho_2, \tag{iv}$$

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \tau_1 <: \rho_1, \tag{v}$$

$$\vdash_{\mathrm{FUNIQ}} C \triangleright \rho_2 <: \tau_2 \tag{vi}$$

であるが，(iii) と (v) とに [S-TRANS] 則を適用すれば $\vdash_{\mathrm{FUNIQ}} C \triangleright \tau_1 <: \sigma_1$ を，また，(iv) と (vi) とに [S-TRANS] 則を適用すれば $\vdash_{\mathrm{FUNIQ}} C \triangleright \sigma_2 <: \tau_2$ を得る．

(2)∼(5) についても同様.

証明終

次の補題は，複合型（関数型，レコード型等々）を導入する式（λ-抽象，レコード式等々）の場合，それらの式に対する型付け判別式の導出を帰結から各々の部分式の型付けに関する前提へと逆向きに辿れる，という事を表わす．

補題 4.10

(1) $\vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright (\lambda x{:}\sigma.e) : \sigma \to \tau$
$\quad\Longrightarrow\quad \vdash_{\text{FUNIQ}} C,\Gamma[x{:}\sigma],\Delta \triangleright e : \tau;$

(2) $\vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright \{l_1 = e_1,\ldots,l_n = e_n\} : \{l_1{:}\sigma_1,\ldots,l_n{:}\sigma_n\}$
$\quad\Longrightarrow\quad \vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright e_i : \sigma_i\ (1 \le i \le n);$

(3) $\vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright [l_i = e] : [l_1{:}\sigma_1,\ldots,l_n{:}\sigma_n]$
$\quad\Longrightarrow\quad \vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright e : \sigma_i\ (1 \le i \le n);$

(4) $\vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright \Lambda\, t{<:}\sigma.e : \forall t{<:}\sigma.\tau$
$\quad\Longrightarrow\quad \vdash_{\text{FUNIQ}} \Delta, C[t <: \sigma],\Gamma \triangleright e : \tau;$

(5) $\vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright \mathbf{pack}_{\exists t<:\sigma.\tau}\ \rho\ \mathbf{with}\ e : \exists t{<:}\sigma.\tau$
$\quad\Longrightarrow\quad \vdash_{\text{FUNIQ}} C,\Gamma,\Delta \triangleright e : \tau[t := \rho].$

証明

左辺の型付け判別式の導出中の [T-SUBTYPE] 則の適用回数 m に関する数学的帰納法によって行なう.

(a) $m = 0$ の場合

この場合, [T-SUBTYPE] 則の適用はあり得ないので, 各々の左辺の判別式の主部とそれに対する型の形から, 左辺の判別式の導出の最終ステップは

(i) 各々の主部の構文に対応する型付け推論規則 ([T-ABS], [T-RECORD], [T-TAG], [T-TABS], [T-PACK])

であるか,

(ii) 型付け文脈の成分への要素対の追加を行なう型付け推論規則 ([T-TWEAK], [T-WEAK], [T-IWEAK])

の何れかでなければならない.

(ii) の型付け文脈の成分に要素対の追加を行なう規則の場合, その規則の前提部の判別式に立ち戻ると C, Γ, Δ の何れかは真に小さな集合となるので, 最終的には (i) の主部の構文に対応する規則に帰着される. そして, (i) の場合, 各々の結論の判別式の成立は明らかである.

(b) $m = k + 1$ の場合

左辺の判別式の導出の最終ステップが [T-SUBTYPE] 則以外の場合は (a) と同一の議論になるので，最終ステップが [T-SUBTYPE] 則の適用の場合のみを考える．

(1) について

左辺の判別式の最終ステップが [T-SUBTYPE] 則なので，適当な型 ρ が存在し，

$$\vdash_{\text{FUNIQ}} C \ \triangleright\ \rho <: \sigma \to \tau \tag{i}$$

かつ

$$\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \ \triangleright\ (\lambda\, x{:}\sigma.e) : \rho \tag{ii}$$

であるが，(i) に補題 4.8 を適用すると $\rho \lesssim \sigma \to \tau$ となるので，適当な型 ρ_1, ρ_2 で $\sigma \lesssim \rho_1$ および $\rho_2 \lesssim \tau$ なるものを用いると，ρ は，

$$\rho \equiv \{r_1 : \{\ldots\{r_j : \rho_1 \to \rho_2 \mid \gamma_{j,1}, \ldots, \gamma_{j,m_j}\} \mid \ldots\} \mid \gamma_{1,1}, \ldots, \gamma_{1,m_1}\}$$

と，$\rho_1 \to \rho_2$ に j 重 $(j \geq 0)$ の詳細化を施した型として表わせる．

以下，型付け判別式 $C, \Gamma[x{:}\sigma], \Delta \ \triangleright\ e : \tau$ が導出可能である事を，詳細化型の深さ j に関する数学的帰納法で証明する．以下の議論で用いる為に，以上の多重の詳細化を持つ型の為の記法を導入しておく．即ち，

$$\begin{aligned} \rho^{(0)} &\triangleq \rho_1 \to \rho_2, \\ \rho^{(i+1)} &\triangleq \{r_{j-i} : \rho^{(i)} \mid \gamma_{j-i,1}, \ldots, \gamma_{j-i,m_{j-i}}\} \qquad (0 \leq i \leq j-1) \end{aligned}$$

とする．無論，$\rho \equiv \rho^{(j)}$ である．また，各 $0 \leq i \leq j-1$ について $C \ \triangleright\ \rho^{(i+1)} <: \rho^{(i)}$ が導出可能である事に注意しておく．

(イ) $j = 0$ の時

$\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \ \triangleright\ (\lambda\, x{:}\sigma.e) : \rho_1 \to \rho_2$ であるので，[T-SUBTYPE] 則の適用回数 m に関する帰納法の仮定より，

$$\vdash_{\text{FUNIQ}} C, \Gamma[x{:}\sigma], \Delta \ \triangleright\ e : \rho_2 . \tag{iii}$$

また，$\vdash_{\mathrm{FUNIQ}} C \rhd \rho_1 \to \rho_2 <: \sigma \to \tau$ なので，これに補題 4.9 (1) を適用すると，

$$\vdash_{\mathrm{FUNIQ}} C \rhd \rho_2 <: \tau \tag{iv}$$

が得られる．(iii) と (iv) それぞれの導出木に [T-SUBTYPE] 則を適用すると，$C, \Gamma[x{:}\sigma], \Delta \rhd e : \tau$ を結論とする導出木が得られる．

(ロ) $j = k + 1$ の時

$\rho \equiv \rho^{(j)} \equiv \rho^{(k+1)}$ である事に注意すれば，$\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \rhd (\lambda\, x{:}\sigma.e) : \rho^{(k+1)}$ であり，また，$\rho^{(i)}$ の記法の導入時に注意した通り，$\vdash_{\mathrm{FUNIQ}} C \rhd \rho^{(k+1)} <: \rho^{(k)}$ であるので，これら二つの導出木に [T-SUBTYPE] 則を適用すると，$C, \Gamma, \Delta \rhd (\lambda\, x{:}\sigma.e) : \rho^{(k)}$ を結論とする導出木が得られる．

この結論の型付け判別式の型 $\rho^{(k)}$ は k 重の詳細化しか持たないので，この判別式の導出可能性に詳細化の深さ j についての帰納法の仮定を適用すると，求める結論が得られる．

(2)〜(5) について

(1) と同様．それぞれ補題 4.9 (2)〜(5) を用いれば良い．

証明終

以上で，必要な準備が整ったので，本節の目標である主部簡約定理を与える主補題の証明を行なう．

なお，Fun 等の通常の型付き λ-計算の場合，主部簡約定理は型付け判別式の導出可能性が簡約で保存される事を示せば良いが，我々の Funiq では詳細化型の中の表明の不等式 $e_1 \leqslant e_2$ の両辺として式が現れるので，それらの式の簡約に関しても表明の導出可能性が保存される事を示す必要がある点を注意しておく．

補題 4.11（主部縮約補題）

式 e は β-簡約基で e' がその縮約結果である — 即ち，$\langle e, e' \rangle \in \boldsymbol{\beta}$ — とする．

この時，

(1) $\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright e : \sigma \implies \vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright e' : \sigma.$

(2) $\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[x := e] \implies \vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[x := e'].$

(3) $\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[r := e] \implies \vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[r := e'].$

証明

(1), (2), (3) を同時に証明する．証明は，各々の前提の導出木の高さ h に関する帰納法によって行なう．なお，$\langle e, e' \rangle \in \boldsymbol{\beta}$ であるので，e は簡約基の形の式であり，e から e' へは e の最外簡約基についての簡約で至る事を注意しておく．

(a) $h = 0$ の場合

(1) について

上の注意より e は複合的な — 即ち，単一の変数や定数でない — 式であるので，型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ は図 2.3 に示した型付けに関する公理のインスタンスではあり得ない．従って，$h = 0$ で $\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright e : \sigma$ となる事はあり得ず，言明の前提は偽となる．よって，この場合，(1) の主張は自明に成立する．

(2) について

高さ $h = 0$ であるので，表明判別式 $C, \Gamma, \Delta \triangleright \gamma[x := e]$ の導出木は図 2.4 中の公理 [A-CONST], [A-VAR], [A-IVAR] の何れかでなければならない．

(イ) [A-CONST] 則の場合

$\gamma \equiv c \leqslant c : \iota$ である．この時，$\gamma[x := e] \equiv c \leqslant c : \iota \equiv \gamma[x := e']$ であるので，前提の表明 $\gamma[x := e]$ の導出可能性から結論の表明 $\gamma[x := e']$ の導出可能性は直ちに導かれる．

(ロ) [A-VAR] 則の場合

或る通常変数 y について $\gamma \equiv y \leqslant y : \sigma$ である． y が e の代入先の変数 x と同じ変数か否かで場合分けする．

• $y \equiv x$ の場合

$\gamma[x := e] \equiv \gamma[y := e] \equiv e \leqslant e : \sigma$ である．(1) での指摘通り，式 e は複合的なので $C, \Gamma, \Delta \triangleright e \leqslant e : \sigma$ は表明に関する公理のインスタンスではあり得ず，故に導出木の高さは $h = 0$ でない．従って言明の前提が偽となり，言明は自明に成立する．

• $y \not\equiv x$ の場合

$\gamma[x := e] \equiv y \leqslant y : \sigma \equiv \gamma[x := e']$ であるので，(イ) の場合と同様に自明に成立．

(ハ) [A-IVAR] 則の場合

(イ) の場合と同様に自明に成立．

(3) について

(2) と同様．

(b) $h = k + 1$ の場合

(1), (2), (3) それぞれの前提の判別式の導出木の高さが k 以下の時に (1), (2), (3) の何れもが成立していると仮定する．

(1) について

e が簡約基の形の式であるので，型付け判別式 $C, \Gamma, \Delta \triangleright e : \sigma$ の導出の最終ステップで適用した規則は，

- 型付け文脈の成分へ要素対の追加を行なう推論規則 ([T-TWEAK], [T-WEAK], [T-IWEAK]) か，
- より大きな型へと型付けを変更する [T-SUBTYPE] 則か，
- 詳細化型の表明の導入に関する [T-REFINE] 則か，
- 簡約基の種類 ($\beta_{\rightarrow}$-, $\beta_{\{\ldots\}}$-, $\beta_{[\ldots]}$-, $\beta_{\forall}$-, $\beta_{\exists}$-, β_{fix}-) に応じた簡約基の構文に対する規則 ([T-APP], [T-SELECT], [T-CASE], [T-TAPP], [T-UNPACK], [T-FIX]) か

の何れかである．以下，各々の場合に分けて示す．

(イ) [T-TWEAK], [T-WEAK], [T-IWEAK], [T-SUBTYPE] 則の何れかの場合

これらの規則では，その規則の前提部の — 高さ k 以下で導出可能な — 型付け判別式は結論部と同じ式 e に関するものであるので，前提部の型付け判別式に帰納法の

仮定を適用し，その結果として得られた e' に関する型付け判別式に元の規則を適用すれば良い．

(ロ) [T-REFINE] 則の場合

前提部の e に関する型付け判別式に関しては，上と同様，帰納法の仮定を適用し，また，各表明 $\gamma_i[r := e]$ $(1 \leq i \leq m)$ に関する表明判別式も高さ k 以下で導出可能なので，各々の表明判別式に帰納法の仮定（(3) についての）を適用し，それらの結果に [T-REFINE] 則を適用すれば e' に対する型付け判別式の導出を得る．

(ハ) 簡約基の構文に応じた規則の場合

各規則に応じて場合分けする．

(i) [T-APP] 則の場合

この場合は，e が $\beta_{\rightarrow}$-簡約基でなければならないので，e と e' とは，各々，$e \equiv (\lambda\, x{:}\tau.e_1)e_2$, $e' \equiv e_1[x := e_2]$ という形である．

この時，$C, \Gamma, \Delta \rhd (\lambda\, x{:}\tau.e_1) : \tau \to \sigma$ の導出 Π_1 と $C, \Gamma, \Delta \rhd e_2 : \tau$ の導出 Π_2 とが存在する．そこで，導出 Π_1 に補題 4.10 (1) を適用すれば $C, \Gamma[x{:}\tau], \Delta \rhd e_1 : \sigma$ の導出 Π_1' を得る．

従って，Π_1' と Π_2 とを前提として，単純型付き λ-計算に於いて良く知られている次の補題

補題 A

$\vdash C, \Gamma[x{:}\tau], \Delta \rhd e_a : \sigma$ かつ $\vdash C, \Gamma, \Delta \rhd e_b : \tau$ ならば

$\vdash C, \Gamma, \Delta \rhd e_a[x := e_b] : \sigma$.

証明

式 e_a の構造に関する帰納法による（直感的には，前者の導出木中で変数 x に関する公理 [T-VAR] の適用のそれぞれを後者の導出木で置換すれば良い．但し，適当に [T-WEAK] 則を用いて型付け基底を適切に整える必要がある）．

証明終

を適用すれば，$C, \Gamma, \Delta \rhd e_1[x := e_2] : \sigma$ の導出可能性を得る．

(ii) [T-SELECT] 則の場合

この場合，e が $\beta_{\{\cdots\}}$-簡約基でなければならない．従って，e，e' は，各々，$e \equiv \{l_1 = e_1, \ldots, l_n = e_n\}.l_i$，$e' \equiv e_i$ という形で，しかも $\sigma \equiv \sigma_i$ であり，更に，高さ k 以下の $C, \Gamma, \Delta \triangleright \{l_1 = e_1, \ldots, l_n = e_n\} : \{l_1 : \sigma_1, \ldots, l_n : \sigma_n\}$ に対する導出が存在する．

この最後の型付け判別式の導出可能性に補題 4.10 (2) を適用し，$C, \Gamma, \Delta \triangleright e_i : \sigma_i$ の導出可能性を得る．

(iii) [T-CASE] 則の場合

(ii) の場合と同様．補題 4.10 (3) を用いる．

(iv) [T-TAPP] 則の場合

(ii) の場合と同様．補題 4.10 (4) を用いる．

(v) [T-UNPACK] 則の場合

(ii) の場合と同様．補題 4.10 (5) を用いる．

(vi) [T-FIX] 則の場合

この場合，$e \equiv \mathbf{fix}\ e_1$，$e' \equiv e_1(\mathbf{fix}\ e_1)$ という形である．この規則の前提部の判別式 $C, \Gamma, \Delta \triangleright e_1 : \sigma \to \sigma$ が導出可能であるので，この判別式の導出木と e 自身の導出木を前提として [T-APP] 則を適用すれば，$C, \Gamma, \Delta \triangleright e_1(\mathbf{fix}\ e_1) : \sigma$ の導出を得る．

(2) について

$C, \Gamma, \Delta \triangleright \gamma[x := e]$ の導出の最終ステップが各 [A-β_{xxx}] 則（ここで xxx は $\to$, $\{\cdots\}$, $[\cdots]$, $\forall$, $\exists$, fix のいずれか），[A-**forall**-E], [A-LEAST], [A-ASSERT] 以外の推論規則の適用の場合，帰納法の仮定を自然な形で適用すれば証明できる．

従って，以下では，導出の最終ステップが各 [A-β_{xxx}], [A-**forall**-E], [A-LEAST], [A-ASSERT] の各規則の場合のみを考える．なお，表明 $\gamma[x := e]$ の一般形を

$$\mathbf{forall}\ y_1 : \sigma_1.\ \ldots\ .\mathbf{forall}\ y_p : \sigma_p.\, e_1[x := e] \leqslant e_2[x := e] : \tau$$

と表わす ($p \geq 0$).

(i) [A-$\beta_{\to}$] 則の場合

本規則の帰結部の表明の形から，$p = 0$ で

$$e_1[x := e] \equiv (\lambda z{:}\,\rho.e_{11})e_{12},$$

$$e_2[x := e] \equiv e_{11}[z := e_{12}]$$

という形でなければならない．

(イ) $x \notin \mathrm{FV}(e_1)$ の場合

この場合，$e_1[x := e'] \equiv e_1 \equiv e_1[x := e] \equiv (\lambda z{:}\,\rho.e_{11})e_{12}$ であり，また，$x \notin \mathrm{FV}(e_1) = \mathrm{FV}((\lambda z{:}\,\rho.e_{11})e_{12}) = \mathrm{FV}(e_{11}) \setminus \{z\} \cup \mathrm{FV}(e_{12}) = \mathrm{FV}(e_{11}[z := e_{12}])$ であるので，$x \notin \mathrm{FV}(e_2)$ となる．よって，$e_2[x := e'] \equiv e_2 \equiv e_2[x := e]$ であるので，$\gamma[x := e] \equiv \gamma[x := e']$ であり，$C, \Gamma, \Delta \triangleright \gamma[x := e']$ の導出は $C, \Gamma, \Delta \triangleright \gamma[x := e]$ の導出そのもので良い．

(ロ) $x \in \mathrm{FV}(e_1)$ の場合

e_1 中の変数 x の出現箇所を全て穴にした文脈を $D[\ \ ]$ とする — $e_1 \equiv D[x]$ — と，$D[e] \equiv e_1[x := e] \equiv (\lambda z{:}\,\rho.e_{11})e_{12}$ である事と e が $\beta_{\to}$-簡約基の形の項である事を考慮すると，$D[\ \ ]$ の形は以下の (I) か (II) のいずれかに限られる．

(I) $D[\ \ ] \equiv \Box$（つまり $e_1 \equiv x$）の場合

この場合，$e \equiv x[x := e] \equiv e_1[x := e] \equiv (\lambda z{:}\,\rho.e_{11})e_{12}$ である．無論，この場合，$x \notin \mathrm{FV}(e) = \mathrm{FV}(e_{11}) \setminus \{z\} \cup \mathrm{FV}(e_{12})$ と考えて良い．故に，$x \notin \mathrm{FV}(e_2)$ なので，$e_2[x := e'] \equiv e_2 \equiv e_2[x := e] \equiv e_{11}[z := e_{12}]$ である事に注意する．

さて，$\langle e, e' \rangle \in \boldsymbol{\beta_{\to}}$ である，つまり $\langle (\lambda z{:}\,\rho.e_{11})e_{12}, e' \rangle \in \boldsymbol{\beta_{\to}}$ であるので $e' \equiv e_{11}[z := e_{12}]$ でなければならない．即ち，$e' \equiv e_2[x := e']$ である．

よって，

$$\begin{aligned}\gamma[x := e'] \equiv e_1[x := e'] &\leqslant e_2[x := e'] : \tau \\ &\equiv x[x := e'] \leqslant e_2[x := e'] : \tau \\ &\equiv e' \leqslant e' : \tau \\ &\equiv e_{11}[z := e_{12}] \leqslant e_{11}[z := e_{12}] : \tau\end{aligned}$$

と不等式の両辺が同一の式 $e_{11}[z := e_{12}]$ となるので，公理と構文との両立性を与える推論規則とをその式の構造に従って組み合わせれば $C,\Gamma,\Delta \triangleright \gamma[x := e']$ の導出を得る事ができる．

注：なお，細かい事を言うと，ここでは簡約で自由変数が増加しない事実 — 3.2 節の補題 3.14 に相当する事実 — を使用している．即ち，この最後の表明判別式の C,Γ,Δ は $\gamma[x := e']$ 中の自由な型変数や変数に関する情報を与えているが，それが簡約前の式 e に対するもので充分である，という事を保証するには，e から e' への簡約によって自由（型）変数が増加しない事が必要十分である．

(II) $D[\ \ ] \equiv (\lambda z{:}\rho.D_1[\ \ ])D_2[\ \ ]$ の場合

この時，$e_1[x := e'] \equiv D[e'] \equiv (\lambda z{:}\rho.D_1[e'])D_2[e']$ であり，また，$e_2[x := e'] \equiv D_1[e'][z := D_2[e']]$ であるので，$\langle e_1[x := e'], e_2[x := e']\rangle \in \boldsymbol{\beta}_{\rightarrow}$ である．

また，前の補題 A より $C,\Gamma[z{:}\rho],\Delta \triangleright D_1[e'] : \sigma$ と $C,\Gamma,\Delta \triangleright D_2[e'] : \rho$ とが共に導出可能である事も与えられる．

よって，この二つの型付け判別式の導出木に [A-$\beta_{\rightarrow}$] 則を適用すれば $C,\Gamma,\Delta \triangleright \gamma[x := e']$ の導出木が得られる．

(ii) [A-$\beta_{\{\cdots\}}$] 則の場合

(iii) [A-$\beta_{[\cdots]}$] 則の場合

(iv) [A-$\beta_{\forall}$] 則の場合

(v) [A-$\beta_{\exists}$] 則の場合

(vi) [A-β_{fix}] 則の場合

いずれも (i) と同様，代入される簡約基 e が表明 γ 中のどこに代入され得るかの場合分けによって示す事ができる．

(vii) [A-**forall**-E] 則の場合

本規則の前提部の $C,\Gamma,\Delta \triangleright e : \sigma$ が高さ k 以下の導出を持つ事に注意すると，(1) についての帰納法の仮定を用いる事ができ，それで存在が保証される $C,\Gamma,\Delta \triangleright e' : \sigma$ の導出木に [A-**forall**-E] 則を適用すれば，$\gamma[x := e']$ に対する導出を得る．

(viii) [A-LEAST] 則の場合

(vi) の [A-β_{fix}] 則の場合と同様に示せば良い．

(ix) [A-ASSERT] 則の場合

この場合，代入 $[x := e]$ の対象となる γ は詳細化型の表明に直接に由来する．従って，定義 2.8 で示した詳細化型の表明に関する構文的制約 (a) より，$x \notin \mathrm{FV}(\gamma)$ である．よって，この場合は自明に成立．

(3) について

基本的には (2) と同様．但し，実現変数に関しては **forall** による束縛は存在しないので，[A-**forall**-E] 則は分けて扱う必要はない．なお，[A-ASSERT] 則の適用については次の様に示せば良い．即ち，[A-ASSERT] 則前提部の型付け判別式が k 以下の高さで導出されるので，これに (1) についての帰納法の仮定を適用すると $C, \Gamma, \Delta \triangleright \gamma_i[r := e']$ についての導出が得られる．

証明終

補題 4.12（単一ステップ主部簡約補題）

$e \longrightarrow_\beta e'$ とする．この時，

(1) $\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e : \sigma \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e' : \sigma.$

(2) $\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[x := e] \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[x := e'].$

(3) $\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[r := e] \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[r := e'].$

証明

(1) について

$e \longrightarrow_\beta e'$ であるので，この単一ステップ簡約で縮約される簡約基を e_1，縮約結果を e_2 とする時，適当な文脈 $D[\ \ ]$ により $e \equiv D[e_1]$, $e' \equiv D[e_2]$ と表わせる．

e は型付け可能なので，e の任意の部分式も型付け可能である．特に，簡約基 e_1（は e 中に複数出現し得るが，e_1 のそれらの出現の中で 1 穴の文脈 $D[\ \ ]$ によって特定される出現）は型付け可能である．従って，(1) の導出可能性を与える導出木 Π 中に e_1 を主部とする型付け判別式が少なくとも一つは現れている — [T-REFINE] 則と [A-ASSERT] 則とが存在する為，e_1 の特定の出現に対応する型付け判別式は Π 中に複数回出現し得る — 筈である．その中で導出木 Π の根に最も近いものを選び，$C', \Gamma', \Delta' \triangleright e_1 : \sigma'$ で表わす．また，この判別式を根（結論）とする Π の部分導出

木を Π_1 で表わす．即ち，$\vdash_{\mathrm{FUNIQ}} C', \Gamma', \Delta' \triangleright e_1 : \sigma'$ である．

$e_1 \longrightarrow_\beta e_2$ であるので，帰納法の仮定から $\vdash_{\mathrm{FUNIQ}} C', \Gamma', \Delta' \triangleright e_2 : \sigma'$ である．そこで，これに対する導出木が存在するので，それを Π_2 で表わす．

すると，導出木 Π 中の部分木 Π_1 を Π_2 で置き換え，更に，全体の導出木 Π で以上の置換より下（導出木の根の方向）の簡約基 e_1 の当該出現を簡約結果 e_2 で置き換えた導出木を $\Pi[e_2/e_1]$ と書く事にすれば，この導出木の構成方法から $\Pi[e_2/e_1]$ の結論の判別式は，その導出木の結論の型付け判別式が

$$\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e[e_2/e_1] : \sigma$$

とでも表現すべき形となるが，そもそも “$[e_2/e_1]$” とは $D[\ \ ]$ の唯一の穴に入れられている e_1 を e_2 で置き換える事であったので，$e[e_2/e_1] \equiv e'$ である．従って，

$$\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e' : \sigma$$

となり，求める結論を得た．

(2), (3) について

(1) を用いれば良い．

証明終

以上から，本節の主題であった次の主部簡約定理が得られる．

定理 4.13（主部簡約定理）

$e \longrightarrow\!\!\!\!\!\rightarrow_\beta e'$ とする．この時，

(1) $\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e : \sigma \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright e' : \sigma.$

(2) $\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[x := e] \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[x := e'].$

(3) $\vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[r := e] \quad \Longrightarrow \quad \vdash_{\mathrm{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[r := e'].$

証明

補題 4.12 を用いて簡約 $e \longrightarrow\!\!\!\!\!\rightarrow_\beta e'$ のステップ数 s に関する数学的帰納法で示せば良い．

(a) $s = 0$ の場合

$e \equiv e'$ であるので自明に成立.

(b) $s = k + 1$ の場合

この場合，$e \longrightarrow\!\!\!\!\!\rightarrow_\beta e'' \longrightarrow_\beta e'$ である．簡約 $e \longrightarrow\!\!\!\!\!\rightarrow_\beta e''$ のステップ数は k なので帰納法の仮定を適用すると，(1), (2), (3) の各々に対して

$(1')$ $\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright e'' : \sigma$,

$(2')$ $\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[x := e'']$,

$(3')$ $\vdash_{\text{FUNIQ}} C, \Gamma, \Delta \triangleright \gamma[r := e'']$

を得る．$e'' \longrightarrow_\beta e'$ と $(1') \sim (3')$ の各々を前提として補題 4.12 の (1) $\sim$ (3) をそれぞれ適用すれば，求める結論を得る．

証明終

以上の主部簡約定理の証明，特に証明の中心である主部縮約補題の証明は，上で観察した通り，既存の型付き λ- 計算の諸体系（Girard の F や F_ω，Cardelli and Wegner の Fun）に対する証明と比べて遥かに錯綜せざるを得なかった．Funiq に対する主部縮約補題の証明がこれほどに長くならざるを得なかったのは，Funiq が実用的な構文（レコード式，タグ付け式，データ抽象化の為の **pack**/**unpack** 式等）を含み構文の種類が多い，という事 — それは見掛け上の原因とはなり得るけれども — が本当の原因ではない．

主部縮約補題の証明が繁雑にならざるを得なかった本当の原因は，Funiq が型の構文要素として式を含み得る（詳細化型の中の表明は二つの式の間の不等式である）という事，そして，型の要素としての表明の不等式は或る意味では式の上の β-簡約関係を形式化したものだという事にある．

型の構文が下位構文として式を含む事から，式の上の簡約が型に対しても影響する為に，式と型（の中の表明）の主部縮約性は相互依存せざるを得ない．その結果，両者を並行して示す必要があり，しかも，表明の不等式と簡約関係とは殆ど相似的であるので，表明の不等式と簡約関係との間で或る種の干渉とも呼ぶべき状況が生じ，その解決の為に証明での場合分けが複雑となった．この事が主部縮約補題の証明を長く繁雑にした真の原因である．

4.3　強正規化性定理の証明

本節では，Funiq から不動点再帰式 **fix** を除いた部分言語 $\text{Funiq}^{-\{\text{fix}\}}$ での簡約が常に停止する事 — 即ち，強正規化性 — を示す．

但し，Cardelli and Wegner (1995) でのオリジナルの Fun（本論文での Fun から不動点再帰式 **fix** を除いた言語）や簡約の停止性に関しては，それと等価な Girard の F の強正規化性に対して既に多くの証明が存在する．例えば，Girard, Lafont and Taylor (1989) の第 6 章や Krivin (1993) の第 VIII 章には F の強正規化性定理の証明が与えられている．また，Stenlund (1972) の§5.6 には直観主義的 2 階述語論理に相当する F の拡張体系に対する同定理の証明が示されている．

以上の様な理由から，本論文では上記の Funiq の部分言語に対する強正規化性をオリジナルの Fun の強正規化性へと帰着させる方法を示し，Girard の簡約候補 (candidates of reduction) の概念等を用いた直接的な証明は与えない．

定義 4.14

Funiq から抽象型関連の構文と不動点再帰式 **fix**（と型制約式）の構文を除いた Funiq の部分言語を $\text{Funiq}^{-\{\exists,\text{fix}\}}$ で表わすとする．この時，$\text{Funiq}^{-\{\exists,\text{fix}\}}$ の式から Girard の 2 階の型付き λ-計算の体系 F（にレコード型と可変型 — これらは計算の停止性には影響しない — を追加したもの）への写像 $\|\!|\cdot|\!\| : \mathbf{Exp}_{\text{Funiq}^{-\{\exists,\text{fix}\}}} \to \mathbf{Exp}_{\text{F}}$ を式の構造に関する帰納法によって以下の様に定義する：

$$
\begin{aligned}
\|\!|c|\!\| &= c,\\
\|\!|x|\!\| &= x,\\
\|\!|r|\!\| &= r,\\
\|\!|\lambda\, x{:}\,\sigma.e|\!\| &= \lambda\, x{:}\,\sigma^*.\|\!|e|\!\|,\\
\|\!|e_1 e_2|\!\| &= \|\!|e_1|\!\|\|\!|e_2|\!\|,\\
\|\!|\{l_1 = e_1, \ldots, l_n = e_n\}|\!\| &= \{l_1 = \|\!|e_1|\!\|, \ldots, l_n = \|\!|e_n|\!\|\},\\
\|\!|e.l|\!\| &= \|\!|e|\!\|.l,\\
\|\!|[l = e]|\!\| &= [l = \|\!|e|\!\|],\\
\|\!|\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n|\!\| &= \mathbf{case}\ \|\!|e_0|\!\|\ \mathbf{of}\ l_1\ \mathbf{then}\ \|\!|e_1|\!\|, \ldots, l_n\ \mathbf{then}\ \|\!|e_n|\!\|,\\
\|\!|\Lambda\, t{<:}\sigma.e|\!\| &= \Lambda\, t{<:}\mathbf{Top}.\|\!|e|\!\|,\\
\|\!|e[\sigma]|\!\| &= \|\!|e|\!\|[\sigma^*],\\
\|\!|e{:}\,\sigma|\!\| &= \|\!|e|\!\|{:}\,\sigma^*.
\end{aligned}
$$

ここで，“$(\cdot)^*$” は定義 3.6 で定義した詳細化型の消去写像である．

本節冒頭で述べた通り，Girard の F に関しては次の事実が知られており，その証明は上で挙げた諸文献に与えられている．

事実 4.15（F の強正規化性定理）

Girard の 2 階の型付き λ-計算の体系 F（にレコード型と可変型を追加した体系）の任意の型付け可能な式 $e \in \mathbf{Exp}_{\mathrm{F}}$（つまり或る通常変数への型付け基底 Γ と型 $\sigma \in \mathbf{Type}_{\mathrm{F}}$ とが存在し，$\vdash \varnothing, \Gamma, \varnothing \triangleright e : \sigma$ なる e）から始まる全ての β^{F}-簡約列は有限長である．ここで，β^{F}-簡約とは，簡約概念 $\boldsymbol{\beta}_{\rightarrow} \cup \boldsymbol{\beta}_{\{\ldots\}} \cup \boldsymbol{\beta}_{[\ldots]} \cup \boldsymbol{\beta}_{\forall}$ により生成されるものである．

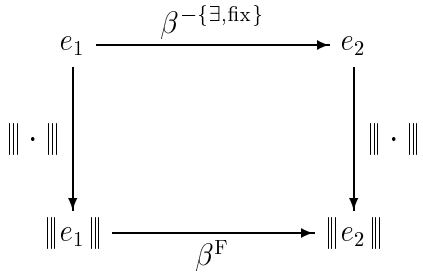
この事実を用いれば，次に示す通り $\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}$ の $\beta^{-\{\exists,\mathrm{fix}\}}$-簡約の強正規化性が直ちに導かれる．

補題 4.16（$\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}$ の強正規化性）

$\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}$ の任意の型付け可能な式から始まる全ての $\beta^{-\{\exists,\mathrm{fix}\}}$-簡約列は有限である．ここで，$\beta^{-\{\exists,\mathrm{fix}\}}$-簡約とは，簡約概念 $\boldsymbol{\beta}_{\rightarrow} \cup \boldsymbol{\beta}_{\{\ldots\}} \cup \boldsymbol{\beta}_{[\ldots]} \cup \boldsymbol{\beta}_{\forall}$ により生成される簡約関係である．

証明

上の事実と次の図の可換性，即ち，$\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}$ での簡約の 1 ステップは F での単一ステップの簡約と正確に対応する事，という事より明らか．



なお，型付け可能な式から始まる簡約列に現れる何れの式も型付け可能である事は，主部簡約定理 4.13 — そこでは Funiq に対してのみ示しているがこの定理の証明は任意の部分言語に関して閉じているので $\mathrm{Funiq}^{-\{\exists\}}$ や $\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}$ に対しても証明されていると考えて良い — が保証している．

証明終

以上より Funiq から **fix** を除いた $\mathrm{Funiq}^{-\{\mathrm{fix}\}}$ の強正規化性が導かれる．

定理 4.17（強正規化性定理）

(1) $\mathrm{Funiq}^{-\{\mathrm{fix}\}}$ の任意の型付け可能な式から始まる全ての $\beta^{-\{\mathrm{fix}\}}$-簡約列は有限長である．ここで，$\beta^{-\{\mathrm{fix}\}}$-簡約とは，$\boldsymbol{\beta}$ の定義から $\boldsymbol{\beta}_{\mathbf{fix}}$ を除いた簡約概念で生成される簡約関係である．

(2) Funiq で **fix** を含まない型付け可能な式 e から始まる無限の簡約列は存在しない．

証明

(1) について

$\mathrm{Funiq}^{-\{\mathrm{fix}\}}$ の任意の型付け可能な式 e について，対応する $\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}$ の式 $\|e\|$ は定理 3.2 により $\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}$ で型付け可能である．

既に見た通り，$\mathrm{Funiq}^{-\{\mathrm{fix}\}}$ での $\beta_\exists$-簡約は $\beta_\forall$, $\beta_\to$, $\beta_\forall$, $\beta_\to$ の 4 ステップの簡約で表わされる．従って，もしも或る $e \in \mathbf{Exp}_{\mathrm{Funiq}^{-\{\mathrm{fix}\}}}$ から始まる無限 $\beta^{-\{\mathrm{fix}\}}$-簡約列

$$e \equiv e_0 \longrightarrow_{R_0} e_1 \longrightarrow_{R_1} \cdots \longrightarrow_{R_{i-1}} e_i \longrightarrow_{R_i} e_{i+1} \longrightarrow_{R_{i+1}} \cdots$$

が存在すれば，$\|e\| \in \mathbf{Exp}_{\mathrm{Funiq}^{-\{\exists,\mathrm{fix}\}}}$ から始まる次の無限 $\beta^{-\{\exists,\mathrm{fix}\}}$-簡約列を構成できる：

$$\|e\| \equiv \|e_0\| \longrightarrow\!\!\!\!\rightarrow^{+}_{R'_0} \|e_1\| \longrightarrow\!\!\!\!\rightarrow^{+}_{R'_1} \cdots \longrightarrow\!\!\!\!\rightarrow^{+}_{R'_{i-1}} \|e_i\| \longrightarrow\!\!\!\!\rightarrow^{+}_{R'_i} \|e_{i+1}\| \longrightarrow\!\!\!\!\rightarrow^{+}_{R'_{i+1}} \cdots.$$

ここで，各 $j \geq 0$ について

$$\longrightarrow\!\!\!\!\rightarrow^{+}_{R'_j} \triangleq \begin{cases} \longrightarrow_{\beta_\forall} \cdot \longrightarrow_{\beta_\to} \cdot \longrightarrow_{\beta_\forall} \cdot \longrightarrow_{\beta_\to}, & (R_j = \beta_\exists \text{ の時}) \\ \longrightarrow_{R_j}, & (R_j \neq \beta_\exists \text{ の時}) \end{cases}$$

である．これは補題 4.16 に矛盾する．従って，$\mathrm{Funiq}^{-\{\mathrm{fix}\}}$ の型付け可能な式から始まる任意の簡約列は有限である．

(2) について

定義 4.1 での Funiq の簡約概念の定義を見れば判る通り，— $\beta_{\rightarrow}$-簡約や $\beta_{\exists}$-簡約で簡約前の式に含まれていたのが複数個コピーされる場合はあるが — β_{fix}-簡約基が簡約により新たに生成される事はないので，簡約前の式に不動点再帰式 **fix** が含まれていなければ (1) より直ちに従う．

証明終

4.4　合流性定理の証明

本節では，Funiq の式の値は式を計算する順序に依らない事，即ち，式の中に複数の簡約基が含まれる場合に何れの簡約基を縮約しても共通の式に到達できる事を保証する合流性（Church-Rosser 性）を Funiq の β-簡約が満たしている事を示す．

なお，本節での合流性の証明方針は，型無し λ-計算の合流性定理に対する並行簡約を用いた高橋 (1991) による証明法を踏襲する．そこで，まず，Funiq の並行簡約関係 $\Longrightarrow_{\beta}$ を **Exp** 上の 2 項関係として図 4.1 に示す形で定義する．

補題 4.18

任意の $e \in \mathbf{Exp}$ に対して，$e \Longrightarrow_{\beta} e$.

証明

式 e の構造に関する帰納法で示せば良いが，図 4.1 の $\Longrightarrow_{\beta}$ の定義は，原子的な構文（定数，通常および実現変数）の各々に対して結論の形の公理を含み，また，複合的な構文の各々に対して構文との両立性を与える推論規則（名前が [P-β_{xxx}] でない規則）を含んでいるので，言明の成立は明らか．

証明終

$$[\text{P-CONST}] \quad \frac{}{c \Longrightarrow_\beta c}$$

$$[\text{P-VAR}] \quad \frac{}{x \Longrightarrow_\beta x}$$

$$[\text{P-IVAR}] \quad \frac{}{r \Longrightarrow_\beta r}$$

$$[\text{P-}\beta_{\to}] \quad \frac{e_1 \Longrightarrow_\beta e_1' \quad e_2 \Longrightarrow_\beta e_2'}{(\lambda\, x{:}\,\sigma.e_1)e_2 \Longrightarrow_\beta e_1'[x := e_2']}$$

$$[\text{P-ABS}] \quad \frac{e \Longrightarrow_\beta e'}{(\lambda\, x{:}\,\sigma.e) \Longrightarrow_\beta (\lambda\, x{:}\,\sigma.e')}$$

$$[\text{P-APP}] \quad \frac{e_1 \Longrightarrow_\beta e_1' \quad e_2 \Longrightarrow_\beta e_2'}{(e_1 e_2) \Longrightarrow_\beta (e_1' e_2')}$$

$$[\text{P-}\beta_{\{\dots\}}] \quad \frac{e_i \Longrightarrow_\beta e_i'}{\{l_1 = e_1, \dots, l_n = e_n\}.l_i \Longrightarrow_\beta e_i'} \qquad (1 \le i \le n)$$

$$[\text{P-RECORD}] \quad \frac{e_1 \Longrightarrow_\beta e_1' \ \dots \ e_n \Longrightarrow_\beta e_n'}{\{l_1 = e_1, \dots, l_n = e_n\} \Longrightarrow_\beta \{l_1 = e_1', \dots, l_n = e_n'\}}$$

$$[\text{P-SELECT}] \quad \frac{e \Longrightarrow_\beta e'}{e.l \Longrightarrow_\beta e'.l}$$

$$[\text{P-}\beta_{[\dots]}] \quad \frac{e \Longrightarrow_\beta e' \quad e_i \Longrightarrow_\beta e_i'}{(\mathbf{case}\,[\,l_i = e\,]\,\mathbf{of}\,l_1\,\mathbf{then}\,e_1, \dots, l_n\,\mathbf{then}\,e_n) \Longrightarrow_\beta (e_i' e')} \qquad (1 \le i \le n)$$

$$[\text{P-VARIANT}] \quad \frac{e \Longrightarrow_\beta e'}{[\,l = e\,] \Longrightarrow_\beta [\,l = e'\,]}$$

$$[\text{P-CASE}] \quad \frac{e_0 \Longrightarrow_\beta e_0' \quad e_1 \Longrightarrow_\beta e_1' \ \dots \ e_n \Longrightarrow_\beta e_n'}{(\mathbf{case}\,e_0\,\mathbf{of}\,l_1\,\mathbf{then}\,e_1, \dots, l_n\,\mathbf{then}\,e_n) \Longrightarrow_\beta (\mathbf{case}\,e_0'\,\mathbf{of}\,l_1\,\mathbf{then}\,e_1', \dots, l_n\,\mathbf{then}\,e_n')}$$

$$[\text{P-}\beta_{\forall}] \quad \frac{e \Longrightarrow_\beta e'}{(\Lambda\, t{<:}\sigma.e)[\rho] \Longrightarrow_\beta e'[t := \rho]}$$

$$[\text{P-TABS}] \quad \frac{e \Longrightarrow_\beta e'}{(\Lambda\, t{<:}\sigma.e) \Longrightarrow_\beta (\Lambda\, t{<:}\sigma.e')}$$

$$[\text{P-TAPP}] \quad \frac{e \Longrightarrow_\beta e'}{e[\sigma] \Longrightarrow_\beta e'[\sigma]}$$

$$[\text{P-}\beta_{\exists}] \quad \frac{e_1 \Longrightarrow_\beta e_1' \quad e_2 \Longrightarrow_\beta e_2'}{(\mathbf{unpack}\,(\mathbf{pack}_{\exists t<:\sigma.\tau_1}\,e_1\,\mathbf{with}\,\rho)\,\mathbf{as}\,x\,\mathbf{with}\,t\,\mathbf{in}\,e_2) \Longrightarrow_\beta e_2'[t := \rho][x := e_1']}$$

$$[\text{P-PACK}] \quad \frac{e \Longrightarrow_\beta e'}{(\mathbf{pack}_{\exists t<:\sigma.\tau}\,e\,\mathbf{with}\,\rho) \Longrightarrow_\beta (\mathbf{pack}_{\exists t<:\sigma.\tau}\,e'\,\mathbf{with}\,\rho)}$$

$$[\text{P-UNPACK}] \quad \frac{e_1 \Longrightarrow_\beta e_1' \quad e_2 \Longrightarrow_\beta e_2'}{(\mathbf{unpack}\,e_1\,\mathbf{as}\,x\,\mathbf{with}\,t\,\mathbf{in}\,e_2) \Longrightarrow_\beta (\mathbf{unpack}\,e_1'\,\mathbf{as}\,x\,\mathbf{with}\,t\,\mathbf{in}\,e_2')}$$

$$[\text{P-}\beta_{\mathbf{fix}}] \quad \frac{e \Longrightarrow_\beta e'}{(\mathbf{fix}\,e) \Longrightarrow_\beta e'(\mathbf{fix}\,e')}$$

$$[\text{P-FIX}] \quad \frac{e \Longrightarrow_\beta e'}{(\mathbf{fix}\,e) \Longrightarrow_\beta (\mathbf{fix}\,e')}$$

$$[\text{P-RESTRICT}] \quad \frac{e \Longrightarrow_\beta e'}{(e{:}\,\tau) \Longrightarrow_\beta (e'{:}\,\tau)}$$

図 4.1 Funiq の並行簡約関係

補題 4.19

$e_{11} \Longrightarrow_\beta e_{21}$ かつ $e_{12} \Longrightarrow_\beta e_{22}$ とする．この時，

$$e_{11}[x := e_{12}] \Longrightarrow_\beta e_{21}[x := e_{22}].$$

証明

式 e_{11} の構造に関する帰納法による．

(1) $e_{11} \equiv c$ の場合

この場合，$e_{11} \Longrightarrow_\beta e_{21}$ を導いたのは [P-CONST] 則以外にあり得ず，その為には $e_{21} \equiv c$ でなければならない．故に $e_{11}[x := e_{12}] \equiv c \Longrightarrow_\beta c \equiv e_{21}[x := e_{22}]$ である．

(2) $e_{11} \equiv y$ の場合

この場合，$e_{11} \Longrightarrow_\beta e_{21}$ を導いたのは [P-VAR] 則以外にはあり得ず，その為には $e_{21} \equiv y$ でなければならない．変数 y が代入先変数 x と同一か否かで場合分けする．

(2a) $y \equiv x$ の場合

$e_{21} \equiv x$ なので，仮定より $e_{11}[x := e_{12}] \equiv e_{12} \Longrightarrow_\beta e_{22} \equiv e_{21}[x := e_{22}]$ である．

(2b) $y \not\equiv x$ の場合

$e_{21} \equiv y \not\equiv x$ なので，仮定より $e_{11}[x := e_{12}] \equiv y \Longrightarrow_\beta y \equiv e_{21}[x := e_{22}]$ である．

(3) $e_{11} \equiv r$ の場合

(1) や (2b) と同様.

(4) $e_{11} \equiv \lambda y{:}\sigma.e_{111}$ の場合

この場合，$e_{11} \Longrightarrow_\beta e_{21}$ を導いたのは [P-ABS] 則以外にはあり得ず，その為には $e_{21} \equiv \lambda y{:}\sigma.e_{211}$ かつ $e_{111} \Longrightarrow_\beta e_{211}$ でなければならない．束縛変数 y が代入先の変数 x と同じか否かで場合分けする．

(4a) $y \equiv x$ の場合

この場合，$e_{11}[x := e_{12}] \equiv (\lambda x{:}\sigma.e_{111})[x := e_{12}] \equiv \lambda x{:}\sigma.e_{111} \equiv e_{11}$ かつ $e_{21}[x := e_{22}] \equiv (\lambda x{:}\sigma.e_{211})[x := e_{22}] \equiv \lambda x{:}\sigma.e_{211} \equiv e_{21}$ であるので，仮定から求める結論を得る．

(4b) $y \not\equiv x$ の場合

この場合，$e_{11}[x := e_{12}] \equiv (\lambda x{:}\sigma.e_{111}) \equiv \lambda x{:}\sigma.e_{111}[x := e_{12}]$ かつ $e_{21}[x := e_{22}] \equiv (\lambda x{:}\sigma.e_{211})[x := e_{22}] \equiv \lambda x{:}\sigma.e_{211}[x := e_{22}]$ となるが，帰納法の仮定を適用すると $e_{111}[x := e_{12}] \Longrightarrow_\beta e_{211}[x := e_{22}]$ なので，これに [P-ABS] 則を適用し求める結論を得る．

(5) $e_{11} \equiv e_{111}e_{112}$ の場合

$e_{11} \Longrightarrow_\beta e_{21}$ を導く規則によって場合分けする．

(5a) [P-APP] 則の場合

$e_{21} \equiv e_{211}e_{212}$ という形で $e_{111} \Longrightarrow_\beta e_{211}$，$e_{112} \Longrightarrow_\beta e_{212}$ でなければならない．これらに帰納法の仮定を適用すれば

- $e_{111}[x := e_{12}] \Longrightarrow_\beta e_{211}[x := e_{22}]$,
- $e_{112}[x := e_{12}] \Longrightarrow_\beta e_{212}[x := e_{22}]$

が共に成立している事が判る．これらを前提として [P-APP] 則を適用すれば求める結論が得られる．

(5b) [P-$\beta_\to$] 則の場合

$e_{111} \equiv \lambda y{:}\sigma.e_{1111}$ および $e_{21} \equiv e_{1111}[y := e_{112}]$ という形でなければならない．そこで束縛変数 y が代入先の変数 x と同じか否かで場合分けする．

(5b1) $y \equiv x$ の場合

補題 4.18 より $e_{112} \Longrightarrow_\beta e_{112}$ である．これに帰納法の仮定を適用し，$e_{112}[x := e_{12}] \Longrightarrow_\beta e_{112}[x := e_{22}]$ を得る．この時,

$$
\begin{aligned}
& e_{11}[x := e_{12}] \\
\equiv\ & ((\lambda x{:}\sigma.e_{1111})e_{112})[x := e_{12}] \\
\equiv\ & ((\lambda x{:}\sigma.e_{1111})[x := e_{12}])(e_{112}[x := e_{12}]) \\
\equiv\ & (\lambda x{:}\sigma.e_{1111})(e_{112}[x := e_{12}]) \\
\Longrightarrow_\beta\ & (\lambda x{:}\sigma.e_{1111})(e_{112}[x := e_{22}]) \\
\Longrightarrow_\beta\ & e_{1111}[x := e_{112}[x := e_{22}]] \\
\Longrightarrow_\beta\ & (e_{1111}[x := e_{112}])[x := e_{22}] \\
\equiv\ & e_{21}[x := e_{22}]
\end{aligned}
$$

となり，求める結論が得られた．

(5b2) $y \not\equiv x$ の場合

補題 4.18 より $e_{112} \Longrightarrow_\beta e_{112}$ である．これに帰納法の仮定を適用し，$e_{112}[x := e_{12}] \Longrightarrow_\beta e_{112}[x := e_{22}]$ を得る．同様に，$e_{1111} \Longrightarrow_\beta e_{1111}$ であるので帰納法の仮定から $e_{1111}[x := e_{12}] \Longrightarrow_\beta e_{1111}[x := e_{22}]$ も成り立つ．この時，

$$
\begin{array}{ll}
 & e_{11}[x := e_{12}] \\
\equiv & ((\lambda\, y{:}\,\sigma.e_{1111})e_{112})[x := e_{12}] \\
\equiv & ((\lambda\, y{:}\,\sigma.e_{1111})[x := e_{12}])(e_{112}[x := e_{12}]) \\
\equiv & (\lambda\, y{:}\,\sigma.e_{1111}[x := e_{12}])(e_{112}[x := e_{12}]) \\
\equiv & (\lambda\, y{:}\,\sigma.e_{1111}[x := e_{22}])(e_{112}[x := e_{12}]) \\
\Longrightarrow_\beta & (\lambda\, y{:}\,\sigma.e_{1111}[x := e_{22}])(e_{112}[x := e_{22}]) \\
\Longrightarrow_\beta & (e_{1111}[x := e_{22}])[y := e_{112}[x := e_{22}]] \\
\equiv & (e_{1111}[y := e_{112}])[x := e_{22}] \\
\equiv & e_{21}[x := e_{22}]
\end{array}
$$

となり，求める結論が得られた．

(6) e_{11} が他の構文の場合

(4) の関数抽象式や (5) の関数適用式の場合と同様にして証明できる．それらの場合に共通するポイントは以下の二つである．

- e_{11} が通常変数を束縛する構文（以上の他には **unpack** 式）の場合，束縛変数と代入先変数 x とが同一か否かで場合分けして扱う．
- e_{11} が複合型を分解する構文（型付け推論規則で前提部の型付け判別式に複合型が現れる構文）の場合，$e_{11} \Longrightarrow_\beta e_{21}$ を導いた推論規則がその構文との両立性を与える規則か e_{11} が β-簡約基の場合の規則かで場合分けする．

証明終

定義 4.20

与えられた式に対し，その中の全ての β-簡約基を縮約した結果を与える写像 $(\,\cdot\,)^\sharp : \mathbf{Exp} \to \mathbf{Exp}$ を式の構造に関する帰納法により以下の様に定義する：

$$
\begin{aligned}
c^\sharp &= c, \\
x^\sharp &= x, \\
r^\sharp &= r, \\
(\lambda\, x{:}\,\sigma.e)^\sharp &= \lambda\, x{:}\,\sigma.e^\sharp, \\
(e_1 e_2)^\sharp &= e_1{}^\sharp e_2{}^\sharp, \qquad (e_1 \not\equiv \lambda\, x{:}\,\sigma.e_1') \\
((\lambda\, x{:}\,\sigma.e_1)e_2)^\sharp &= e_1{}^\sharp[x := e_2{}^\sharp], \\
\{l_1 = e_1, \ldots, l_n = e_n\}^\sharp &= \{l_1 = e_1{}^\sharp, \ldots, l_n = e_n{}^\sharp\}, \\
(e.l)^\sharp &= e^\sharp.l, \qquad (e \not\equiv \{l_1 = e_1, \ldots, l_n = e_n\}) \\
(\{l_1 = e_1, \ldots, l_n = e_n\}.l_i)^\sharp &= e_i{}^\sharp, \\
[\,l = e\,]^\sharp &= [\,l = e^\sharp\,], \\
(\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n)^\sharp &= \mathbf{case}\ e_0^\sharp\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1^\sharp, \ldots, l_n\ \mathbf{then}\ e_n^\sharp, \\
&\qquad (e_0 \not\equiv [\,l_i = e_0'\,]) \\
(\mathbf{case}\ [\,l_i = e\,]\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n)^\sharp &= e_i{}^\sharp e^\sharp, \\
(\Lambda\, t{<:}\sigma.e)^\sharp &= \Lambda\, t{<:}\sigma.e^\sharp, \\
(e[\sigma])^\sharp &= e^\sharp[\sigma], \qquad (e \not\equiv \Lambda\, t{<:}\sigma.e) \\
((\Lambda\, t{<:}\sigma.e)[\tau])^\sharp &= e^\sharp[t := \tau], \\
(\mathbf{pack}_{\exists t<:\rho.\sigma}\ e\ \mathbf{with}\ \tau)^\sharp &= \mathbf{pack}_{\exists t<:\rho.\sigma}\ e^\sharp\ \mathbf{with}\ \tau, \\
(\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2)^\sharp &= \mathbf{unpack}\ e_1{}^\sharp\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2{}^\sharp, \\
&\qquad (e_1 \not\equiv \mathbf{pack}_{\exists t<:\rho.\sigma}\ e_1'\ \mathbf{with}\ \tau) \\
(\mathbf{unpack}\ (\mathbf{pack}_{\exists t<:\rho.\sigma}\ e_1\ \mathbf{with}\ \tau)\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2)^\sharp &= e_2{}^\sharp[t := \tau][x := e_1{}^\sharp], \\
(\mathbf{fix}\ e)^\sharp &= e^\sharp(\mathbf{fix}\ e^\sharp), \\
(e{:}\,\sigma)^\sharp &= e^\sharp{:}\,\sigma.
\end{aligned}
$$

以上の写像 $(\,\cdot\,)^\sharp$ に関して次の性質が成り立つ.

補題 4.21

任意の $e \in \mathbf{Exp}$ に対して,

$$\forall e' \in \mathbf{Exp}.\big[e \Longrightarrow_\beta e' \quad \Longrightarrow \quad e' \Longrightarrow_\beta e^\sharp\big].$$

証明

式 e の構造に関する帰納法による.

(1) $e \equiv c$ の場合

$e' \equiv c \equiv e^\sharp$ より成立.

(2) $e \equiv x$ の場合

(1) と同様.

(3) $e \equiv r$ の場合

(1) と同様.

(4) $e \equiv \lambda x{:}\sigma.e_1$ の場合

$e' \equiv \lambda x{:}\sigma.e_1'$ かつ $e_1 \Longrightarrow_\beta e_1'$ でなければならない. 帰納法の仮定より $e_1' \Longrightarrow_\beta {e_1}^\sharp$. これに [P-ABS] 則を適用し $e' \equiv \lambda x{:}\sigma.e_1' \Longrightarrow_\beta \lambda x{:}\sigma.{e_1}^\sharp \equiv e^\sharp$ を得る.

(5) $e \equiv e_1 e_2$ の場合

$e \Longrightarrow_\beta e'$ の導出での最後の規則により場合分けする.

(5a) [P-APP] 則の場合

この場合, $e' \equiv e_1' e_2'$ かつ $e_1 \Longrightarrow_\beta e_1'$, $e_2 \Longrightarrow_\beta e_2'$ でなければならない. $e^\sharp$ は e_1 が関数抽象式か否かで異なるので, e_1 の形について場合分けして扱う.

(5a1) $e_1 \equiv \lambda x{:}\sigma.e_{11}$ の場合

$e_1 \Longrightarrow_\beta e_1'$ より, $e_1' \equiv \lambda x{:}\sigma.e_{11}'$ かつ $e_{11} \Longrightarrow_\beta e_{11}'$ でなければならない. この時,

$$
\begin{array}{lll}
e' \equiv & (\lambda x{:}\sigma.e_{11}')e_2' & \\
\Longrightarrow_\beta & {e_{11}}^\sharp[x := {e_2}^\sharp] & e_{11}', e_2' \text{ の各々の帰納法の仮定と [P-}\beta_\to\text{] 則とを適用} \\
\equiv & ((\lambda x{:}\sigma.e_{11})e_2)^\sharp & \\
\equiv & e^\sharp &
\end{array}
$$

となり, 求める結論が得られた.

(5a2) $e_1 \not\equiv \lambda x{:}\sigma.e_{11}$ の場合

この場合, $e^\sharp \equiv {e_1}^\sharp {e_2}^\sharp$ である. 帰納法の仮定より, $e_1' \Longrightarrow_\beta {e_1}^\sharp$ かつ $e_2' \Longrightarrow_\beta {e_2}^\sharp$ である. これらを前提として [P-APP] 則を適用すれば $e' \Longrightarrow_\beta e^\sharp$ を得る.

(5b) [P-$\beta_\to$] 則の場合

この場合, $e_1 \equiv \lambda x{:}\sigma.e_{11}$ という形で, $e' \equiv e_{11}'[x := e_2']$ かつ $e_{11} \Longrightarrow_\beta e_{11}'$, $e_2 \Longrightarrow_\beta e_2'$ でなければならない.

e の形から $e^\sharp \equiv ((\lambda x{:}\sigma.e_{11})e_2)^\sharp \equiv {e_{11}}^\sharp[x := {e_2}^\sharp]$ であり，帰納法の仮定から $e'_{11} \Longrightarrow_\beta {e_{11}}^\sharp$ および $e'_2 \Longrightarrow_\beta {e_2}^\sharp$ である．従って，補題 4.19 を用いれば $e' \equiv e'_{11}[x := e'_2] \Longrightarrow_\beta {e_{11}}^\sharp[x := {e_2}^\sharp] \equiv e^\sharp$ を得る．

(6) e が他の構文の場合

以上の (4), (5) と同様の論法と場合分けによって証明できるので省略する．

証明終

上の補題は，与えられた式 e から並行簡約 $\Longrightarrow_\beta$ で至れる何れの式からも e 中の β-簡約基全てを簡約した結果へ $\Longrightarrow_\beta$ で到達できる，という事を表わしている．この事から，並行簡約が菱形性を持つ事が直ちに導かれる．即ち，

補題 4.22 （並行簡約の菱形性）

$e \Longrightarrow_\beta e_1$ かつ $e \Longrightarrow_\beta e_2$ とする．この時，

$$\exists e_3 \in \mathbf{Exp}.\big[e_1 \Longrightarrow_\beta e_3 \text{ かつ } e_2 \Longrightarrow_\beta e_3\big].$$

証明

前の補題 4.21 より e_3 として $e^\sharp$ を選べば求める条件が満たされる．

証明終

以上から並行簡約 $\Longrightarrow_\beta$ の推移的閉包は合流性を持つ事が判った．そこで，β-簡約を並行簡約と関係付ける事で β-簡約の合流性を示す事ができる．その関係付けは以下の二つの補題で与えられる．

補題 4.23

任意の $e_1, e_2 \in \mathbf{Exp}$ に対して，

$$e_1 \longrightarrow_\beta e_2 \quad \Longrightarrow \quad e_1 \Longrightarrow_\beta e_2.$$

証明

式 e_1 の構造に関する帰納法によって示す． e_1 は β-簡約基を含むので複合的な構文に限られる．

(1) $e_1 \equiv \lambda\, x{:}\sigma.e_{11}$ の場合

この場合， $e_2 \equiv \lambda\, x{:}\sigma.e_{21}$ という形で $e_{11} \longrightarrow_\beta e_{12}$ が成立している筈である．従って，帰納法の仮定から $e_{11} \Longrightarrow_\beta e_{12}$ であるので， [P-ABS] 則を適用し求める結論を得る．

(2) $e_1 \equiv e_{11}e_{12}$ の場合

以下の三つの場合が可能であるので，各々の場合に分けて示す．

(2a) $e_2 \equiv e_{21}e_{12}$ かつ $e_{11} \longrightarrow_\beta e_{21}$ の場合

帰納法の仮定より $e_{11} \Longrightarrow_\beta e_{21}$ である．また，補題 4.18 より $e_{12} \Longrightarrow_\beta e_{12}$ であるので，以上の二つを前提として [P-APP] 則を適用し $e_1 \Longrightarrow_\beta e_2$ を得る．

(2b) $e_2 \equiv e_{11}e_{22}$ かつ $e_{12} \longrightarrow_\beta e_{22}$ の場合

(2a) と同様．

(2c) $e_{11} \equiv \lambda\, x{:}\sigma.e_{111}$ かつ $e_2 \equiv e_{111}[x := e_{12}]$ の場合

補題 4.18 より $e_{111} \Longrightarrow_\beta e_{111}$， $e_{12} \Longrightarrow_\beta e_{12}$ であるので，これらを前提として [P-$\beta_\rightarrow$] 則を適用し $e_1 \Longrightarrow_\beta e_2$ を得る．

(3) $e_1 \equiv \{l_1 = e_{11}, \ldots, l_n = e_{1n}\}$ の場合

この場合， $e_2 \equiv \{l_1 = e_{21}, \ldots, l_n = e_{2n}\}$ で，或る $1 \leq i \leq n$ が存在して，$e_{1i} \longrightarrow_\beta e_{2i}$，かつ，全ての $j \neq i$ について $e_{1j} \equiv e_{2j}$ である．

帰納法の仮定より $e_{1i} \Longrightarrow_\beta e_{2i}$ であり，補題 4.18 より各 $j \neq i$ について $e_{1j} \Longrightarrow_\beta e_{2j}$ であるので，これらを前提として [P-RECORD] 則を適用すれば求める結論を得る．

(4) $e_1 \equiv e_{11}.l$ の場合

以下の二つの場合が可能であるので，各々に分けて示す．

(4a) $e_2 \equiv e_{21}.l$ かつ $e_{11} \longrightarrow_\beta e_{21}$ の場合

帰納法の仮定より $e_{11} \Longrightarrow_\beta e_{21}$ なので，これを前提として [P-SELECT] 則を適用すれば求める結論を得る．

(4b) $e_{11} \equiv \{l_1 = e_{111}, \ldots, l_n = e_{11n}\}$ かつ $l \equiv l_i$ $(1 \leq i \leq n)$ かつ $e_2 \equiv e_{11i}$ の場合

補題 4.18 より $e_{11i} \Longrightarrow_\beta e_{11i}$ である．これを前提として [P-$\beta_{\{\ldots\}}$] を適用すれば求める結論を得る．

(5) e が他の構文の場合

以上と同様の形で証明できるので省略する．それらの場合の証明でのポイントは以下の通りである：

(i) 式 e_1 の形が

(i-a) 複合型を分解する構文か **fix** 式の場合，

- e_1 自身が β_{xxx}-簡約基の場合，[P-β_{xxx}] 則を用いる；
- それ以外の場合，構文との両立性を与える規則（規則名の頭の “P-” と “T-” とを除けば図 2.3 中に同じ名前の型付け規則が存在する規則）を用いる；

(i-b) それ以外の構文の場合，構文との両立性を与える規則を用いる；

(ii) それら規則の適用での各前提としては，その前提が

(ii-a) β-簡約基を含む部分式に対応する場合，その部分式に帰納法の仮定を適用して得られる並行簡約関係を用いる；

(ii-b) それ以外の場合，その部分式に補題 4.18 を適用して得られる並行簡約関係を用いる．

証明終

補題 4.24

任意の $e_1, e_2 \in \mathbf{Exp}$ に対して，

$$e_1 \Longrightarrow_\beta e_2 \quad \Longrightarrow \quad e_1 \longrightarrow\!\!\!\!\!\rightarrow_\beta e_2 .$$

証明

式 e_1 の構造に関する帰納法による．

(1) $e_1 \equiv c$ の場合

この場合，$e_1 \Longrightarrow_\beta e_2$ を導けるのは [P-CONST] 則だけである．故に $e_2 \equiv c$ であるので，$e_1 \equiv c \longrightarrow\!\!\!\!\!\rightarrow_\beta c \equiv e_2$.

(2) $e_1 \equiv x$ の場合

(1) と同様.

(3) $e_1 \equiv r$ の場合

(1) と同様.

(4) $e_1 \equiv \lambda x{:}\sigma.e_{11}$ の場合

この場合，$e_1 \Longrightarrow_\beta e_2$ を導けるのは [P-ABS] 則だけである．故に $e_2 \equiv \lambda x{:}\sigma.e_{21}$ かつ $e_{11} \Longrightarrow_\beta e_{21}$ でなければならない．帰納法の仮定より $e_{11} \longrightarrow\!\!\!\!\!\rightarrow_\beta e_{21}$ である．故に $e_1 \longrightarrow\!\!\!\!\!\rightarrow_\beta e_2$.

(5) $e_1 \equiv e_{11}e_{12}$ の場合

この場合，$e_1 \Longrightarrow_\beta e_2$ を導くのに用いる事ができる規則としては二つの規則の可能性がある．そこで，各規則の場合に分けて示す.

(5a) [P-APP] 則の場合

この場合，$e_2 \equiv e_{21}e_{22}$ かつ $e_{11} \Longrightarrow_\beta e_{21}$，$e_{12} \Longrightarrow_\beta e_{22}$ でなければならない．これらに帰納法の仮定を適用すると $e_{11} \longrightarrow\!\!\!\!\!\rightarrow_\beta e_{21}$，$e_{12} \longrightarrow\!\!\!\!\!\rightarrow_\beta e_{22}$ となる．従って，$e_1 \equiv e_{11}e_{12} \longrightarrow\!\!\!\!\!\rightarrow_\beta e_{21}e_{12} \longrightarrow\!\!\!\!\!\rightarrow_\beta e_{21}e_{22} \equiv e_2$ である.

(5b) [P-$\beta_\rightarrow$] 則の場合

この場合，$e_{11} \equiv \lambda x{:}\sigma.e_{111}$ および $e_2 \equiv e_{211}[x := e_{22}]$ という形で，しかも $e_{111} \Longrightarrow_\beta e_{211}$, $e_{12} \Longrightarrow_\beta e_{22}$ が成立している筈である.

故に帰納法の仮定より $e_{111} \longrightarrow\!\!\!\!\!\rightarrow_\beta e_{211}$, $e_{12} \longrightarrow\!\!\!\!\!\rightarrow_\beta e_{22}$ である．従って，

$$\begin{aligned} e_1 \equiv (\lambda x{:}\sigma.e_{111})e_{12} &\longrightarrow\!\!\!\!\!\rightarrow_\beta \ (\lambda x{:}\sigma.e_{111})e_{22} \\ &\longrightarrow\!\!\!\!\!\rightarrow_\beta \ (\lambda x{:}\sigma.e_{211})e_{22} \\ &\longrightarrow_\beta \ e_{211}[x := e_{22}] \equiv e_2 \end{aligned}$$

を得る.

(6) e が他の構文の場合

以上の (4), (5) と同様の場合分けで示す事ができるので省略する．それらの場合の証明でのポイントは，式 e_1 の形が複合型を分解する構文か **fix** 式の場合，$e_1 \Longrightarrow_\beta e_2$ の導出が

(i) e_1 自身が β_{xxx}-簡約基で [P-β_{xxx}] 則の適用によるのか，

(ii) 構文との両立性を与える規則によるのか，

の 2 通りに場合分けして扱う事である．

証明終

系 4.25

並行簡約関係 $\Longrightarrow_\beta$ の推移的閉包を $\Longrightarrow_\beta^*$ で表わす事にする．この時，**Exp** 上の 2 項関係として

$$\Longrightarrow_\beta^* = \longrightarrow\!\!\!\!\!\rightarrow_\beta .$$

証明

補題 4.23 より $\longrightarrow_\beta \subseteq \Longrightarrow_\beta$ であるので，両辺の推移的閉包を取れば $\longrightarrow\!\!\!\!\!\rightarrow_\beta \subseteq \Longrightarrow_\beta^*$ を得る．

他方，補題 4.24 より $\Longrightarrow_\beta \subseteq \longrightarrow\!\!\!\!\!\rightarrow_\beta$ であるので，両辺の推移的閉包（$\longrightarrow\!\!\!\!\!\rightarrow_\beta$ は推移的閉包であるので，その推移的閉包は自分自身）を取れば $\Longrightarrow_\beta^* \subseteq \longrightarrow\!\!\!\!\!\rightarrow_\beta$ を得る．

証明終

以上の準備により Funiq での β-簡約の合流性を証明する事ができる．

定理 4.26 （合流性定理）

$e \longrightarrow\!\!\!\!\!\rightarrow_\beta e_1$ かつ $e \longrightarrow\!\!\!\!\!\rightarrow_\beta e_2$ とする．この時，

$$\exists e_3 \in \mathbf{Exp}.\left[e_1 \longrightarrow\!\!\!\!\!\rightarrow_\beta e_3 \text{ かつ } e_1 \longrightarrow\!\!\!\!\!\rightarrow_\beta e_3\right]$$

証明

系 4.25 と補題 4.22 とから直ちに成立する．

証明終

以上から Funiq での β-簡約は合流的であり，与えられた式が正規形を持つ場合，式中の何れの簡約基から簡約するかの選択に依存せず正規形に辿り着く事ができる．故に操作的意味での計算結果の一意性が保証されている．

以上の β-簡約の合流性を，何故， 4.1 節の定理 4.6 を利用し先ず Funiq$^{-\{\exists\}}$ 上の $\beta^{-\{\exists\}}$ について示しそれから Funiq の β の合流性を導くという道筋を採らず， Funiq に関して直接に証明したのか，という理由は， Funiq での β-簡約の合流性は $\exists$ 型をコード化で除いた Funiq$^{-\{\exists\}}$ 上の $\beta^{-\{\exists\}}$-簡約の合流性から単純には導かれないからである．

この問題を少し詳しく述べると以下の通りである．補題 4.24 に於いて e_1 から並行簡約 $\Longrightarrow_\beta$ で e_2 に至るならば適当なステップ数の β-簡約列でも e_2 に辿り着ける事を示した．本補題は Funiq$^{-\{\exists\}}$ 上の $\beta^{-\{\exists\}}$-簡約に関しても当然成立する．その場合，本補題が保証するのは e_2 への $\beta^{-\{\exists\}}$-簡約列の存在である．

Funiq の $\beta_\exists$-簡約は Funiq$^{-\{\exists\}}$ では $\beta_\forall$, $\beta_\to$, $\beta_\forall$, $\beta_\to$ の合計 4 ステップの連続した簡約として表わされる．補題 4.24 が存在を保証する Funiq$^{-\{\exists\}}$ での $\|e_1\|$ から $\|e_2\|$ への $\beta^{-\{\exists\}}$-簡約列の中で $\beta_\exists$ 対応の上記 4 ステップの簡約が正しく連続して実行される保証はない． 4 ステップの途中で他の簡約基の縮約が行なわれたり 4 ステップの中の最初の幾つかが実行されるだけ，という可能性を排除できない．即ち，Funiq$^{-\{\exists\}}$ に関する補題 4.24 が保証する $\|e_1\|$ から $\|e_2\|$ への $\beta^{-\{\exists\}}$-簡約列は元の e_1 から e_2 への（Funiq での） β-簡約列に対応しないものである可能性が残る．故に Funiq$^{-\{\exists\}}$ の $\beta^{-\{\exists\}}$-簡約と Funiq の β-簡約との関係の詳細な分析がない限り，

$$\|e_1\| \longrightarrow\!\!\!\!\!\rightarrow_{\beta^{-\{\exists\}}} \|e_2\| \quad \not\Longrightarrow \quad e_1 \longrightarrow\!\!\!\!\!\rightarrow_\beta e_2$$

であるので，合流性の証明は Funiq に対して直接に示す必要があった．

4.5　形式的厳格性の簡約論的特徴付け

本節では，形式的厳格性を簡約論的に特徴付ける為に，形式的厳格な変数が β-簡約に於いてどの様に振舞うかを調べる．

まず最初に，変数が或る式で形式的厳格であれば，その式の中の別の変数への構文的な代入では影響を受けない事を注意しておく．即ち，

補題 4.27

$v \in \mathbf{Var} \cup \mathbf{IVar}$, $e \in \mathbf{Exp}$ とし，v は e で形式的厳格とする．この時，任意の $v' \not\equiv v$ と $e' \in \mathbf{Exp}$ とに対して，v は $e[v' := e']$ で形式的厳格である．

証明

式 e の構造に関する帰納法で示せば良い．

証明終

以上の補題を用いると，形式的厳格な変数が β-簡約で保存される事が示される．即ち，

定理 4.28 （形式的厳格変数の β-簡約保存性）

$v \in \mathbf{Var} \cup \mathbf{IVar}$ は $e \in \mathbf{Exp}$ で形式的厳格とする．この時，

$$\forall e' \in \mathbf{Exp}.\big[e \longrightarrow\!\!\!\!\!\rightarrow_\beta e' \quad \Longrightarrow \quad v \text{ は } e' \text{ で形式的厳格}\big].$$

証明

以下，単一ステップ簡約の場合 — $e \longrightarrow_\beta e'$ — のみを示す．一般の場合は簡約 $e \longrightarrow\!\!\!\!\!\rightarrow_\beta e'$ のステップ数に関する数学的帰納法を用いれば良い．

単一ステップ簡約の証明は e の構造に関する帰納法によって行なうが，v が e で形式的厳格であるので，式 e の構造として可能なのは以下に示す各場合のみである．

(1) $e \equiv v$ の場合

$e \longrightarrow_\beta e'$ であるので e は簡約基を含んでいなければならず，従って，この場合はあり得ない．

(2) $e \equiv \lambda x{:}\sigma.e_1$ かつ $v \not\equiv x$ かつ v は e_1 で形式的厳格な場合

$e \longrightarrow_\beta e'$ であるので $e' \equiv \lambda x{:}\sigma.e_1'$ かつ $e_1 \longrightarrow_\beta e_1'$ である．帰納法の仮定より，v は e_1' で形式的厳格であるので e' でも形式的厳格である．

(3) $e \equiv e_1 e_2$ かつ v は e_1 で形式的厳格な場合

$e \longrightarrow_\beta e'$ で縮約される β-簡約基の所在により以下の三つの場合に分けられる．

(3a) $e \longrightarrow_\beta e'$ で縮約される β-簡約基が e 自身である場合

この場合，$e_1 \equiv \lambda x{:}\sigma.e_{11}$ で $e' \equiv e_{11}[x := e_2]$ である．e に於ける v の形式的厳格性から $v \not\equiv x$ かつ v は e_{11} で形式的厳格でなければならず，従って補題 4.27 より，v は $e_{11}[x := e_2]$ つまり e' で形式的厳格である．

(3b) $e \longrightarrow_\beta e'$ で縮約される β-簡約基が e_1 中にある場合

この時，$e' \equiv e_1' e_2$ という形でかつ $e_1 \longrightarrow_\beta e_1'$ である．帰納法の仮定より v は e_1' で形式的厳格であるので e' で形式的厳格である．

(3c) $e \longrightarrow_\beta e'$ で縮約される β-簡約基が e_2 中にある場合

この時，$e' \equiv e_1 e_2'$ という形でかつ $e_2 \longrightarrow_\beta e_2'$ である．v は e_1 で形式的厳格であったので e' でも形式的厳格である．

式 e がその他の形の場合も (2), (3) と同様にして証明できる．

証明終

以上より，式の中の形式的厳格な変数は β-簡約によって単に自由変数として失われないだけでなく形式的厳格性も保存される事が判った．この事ならびに形式的厳格性の直感的な意味が「その変数の値が未定義ならば式全体の値も未定義になる」という事は，以下で定める新しい簡約概念 $\perp$ の導入とそれが導く定理とにより，一層，印象的な形で示される．以下で定義する $\perp$-簡約は，Barendregt (1984) が Definition 14.3.1 で型無し λ-計算に対して与えたものを参考にしている．なお，次の簡約概念 $\perp$ の定義では，定義 4.1 での様に簡約概念を式の 2 項対の集合として書き下

すのは繁雑なので，集合としての簡約概念の要素となるべき 2 項対を成す形の式を “$\longmapsto$” の記号で区切った形で示す.

定義 4.29 （$\bot$-簡約）

集合 $\mathbf{Exp}\bot$ を新たな定数記号として $\bot$ を追加して図 2.1 に示した Funiq の構文規則で生成される式の集合とし，この集合上で以下の一連の簡約概念を定義する：

$$
\begin{aligned}
\bot_{\to\mathbf{I}} &: \lambda x{:}\sigma.\bot \longmapsto \bot,\\
\bot_{\to\mathbf{E}} &: \bot e \longmapsto \bot,\\
\bot_{\{\cdots\}\mathbf{I}} &: \{l_1 = \bot, \ldots, l_n = \bot\} \longmapsto \bot,\\
\bot_{\{\cdots\}\mathbf{E}} &: \bot.l \longmapsto \bot,\\
\bot_{[\cdots]\mathbf{E}} &: \mathbf{case}\ \bot\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n \longmapsto \bot,\\
\bot_{\forall\mathbf{I}} &: \Lambda\, t{<:}\sigma.\bot \longmapsto \bot,\\
\bot_{\forall\mathbf{E}} &: \bot[\sigma] \longmapsto \bot,\\
\bot_{\exists\mathbf{E}} &: \mathbf{unpack}\ \bot\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e \longmapsto \bot,\\
\bot_{\mathbf{fix}} &: \mathbf{fix}\ \bot \longmapsto \bot,\\
\bot_{:} &: \bot{:}\sigma \longmapsto \bot;
\end{aligned}
$$

$$
\begin{aligned}
\bot \triangleq\ & \bot_{\to\mathbf{I}} \cup \bot_{\to\mathbf{E}} \cup \bot_{\{\cdots\}\mathbf{I}} \cup \bot_{\{\cdots\}\mathbf{E}} \cup \bot_{[\cdots]\mathbf{E}}\\
& \cup \bot_{\forall\mathbf{I}} \cup \bot_{\forall\mathbf{E}} \cup \bot_{\exists\mathbf{E}} \cup \bot_{\mathbf{fix}} \cup \bot_{:};
\end{aligned}
$$

$$
\boldsymbol{\beta}\bot \triangleq \boldsymbol{\beta} \cup \bot.
$$

以上で定義した $\bot$-簡約は以下の性質を満たしている.

命題 4.30

(1) $\beta\bot$-簡約は合流的である.

(2) 任意の $e \in \mathbf{Exp}\bot$ は，丁度 1 個の $\bot$-正規形を持つ.

証明

(1) について

省略．前節での β-簡約の合流性の証明で定義した並行簡約を $\beta\bot$-簡約に拡張して証明しても良いし，Barendregt (1984) の Proposition 14.3.2(1) での様にラベル付き簡約の概念を定義して，それを用いて証明する事もできる．

(2) について

$\bot$-簡約は必ず式の長さを短くするので強正規化性を持つ．故に任意の $e \in \mathbf{Exp}\bot$ は 1 個以上の $\bot$-正規形を持つ．丁度 1 個である事は高々 1 ステップの $\bot$-簡約が菱形性を満たす（ので推移的閉包の $\longrightarrow\!\!\!\!\!\rightarrow_{\bot}$ も菱形性を満たす）事より従う．

証明終

形式的厳格な変数は，$\bot$-簡約に関して次の際立った性質を持っている．

定理 4.31 （簡約論的厳格性）

$e \in \mathbf{Exp}$ を任意の式とし $v \in \mathbf{Var} \cup \mathbf{IVar}$ は e で形式的厳格とする．この時，

$$e[v := \bot] \longrightarrow\!\!\!\!\!\rightarrow_{\bot} \bot.$$

証明

e の構造に関する帰納法により証明する． v が e で形式的厳格であるので，式 e の構造として可能なのは以下に示す各場合のみである．

(1) $e \equiv v$ の場合

$v[v := \bot] \equiv \bot$ より成立．

(2) $e \equiv \lambda x{:}\sigma.e_1$ の場合

定義 2.5 (2) より v は e_1 で形式的厳格かつ $v \not\equiv x$ である．従って，帰納法の仮定から $e_1[v := \bot] \longrightarrow\!\!\!\!\!\rightarrow_{\beta\bot} \bot$．故に

$$\begin{array}{rll} e[v := \bot] \equiv & & (\lambda x{:}\sigma.e_1)[v := \bot] \\ \equiv & & \lambda x{:}\sigma.(e_1[v := \bot]) \\ \longrightarrow\!\!\!\!\!\rightarrow_{\bot} & & \lambda x{:}\sigma.\bot \\ \longrightarrow_{\bot\to\mathrm{I}} & & \bot. \end{array}$$

(3) $e \equiv e_1 e_2$ の場合

定義 2.5 (3) より v は e_1 で形式的厳格なので，(2) と同様の形で，帰納法の仮定と $\bot_{\to \mathrm{E}}$-簡約とで示す事ができる．

(4) $e \equiv \{l_1 = e_1, \ldots, l_n = e_n\}$ の場合

定義 2.5 (4) より v は $e_1, \ldots, e_n$ の各々で形式的厳格なので，帰納法の仮定と $\bot_{\{\cdots\}\mathrm{I}}$-簡約とで示す事ができる．

(5) $e \equiv e_1.l$ の場合

定義 2.5 (5) より v は e_1 で形式的厳格なので，帰納法の仮定と $\bot_{\{\cdots\}\mathrm{E}}$-簡約とで示す事ができる．

(6) $e \equiv \mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n$ の場合

定義 2.5 (6) より v は e_0 で形式的厳格なので，帰納法の仮定と $\bot_{[\cdots]\mathrm{E}}$-簡約とで示す事ができる．

(7) $e \equiv \Lambda\, t{<:}\sigma.e_1$ の場合

定義 2.5 (7) より v は e_1 で形式的厳格なので，帰納法の仮定と $\bot_{\forall \mathrm{I}}$-簡約とで示す事ができる．

(8) $e \equiv e_1[\sigma]$ の場合

定義 2.5 (8) より v は e_1 で形式的厳格なので，帰納法の仮定と $\bot_{\forall \mathrm{E}}$-簡約とで示す事ができる．

(9) $e \equiv \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2$ の場合

定義 2.5 (9) より v は e_1 で形式的厳格なので，帰納法の仮定と $\bot_{\exists \mathrm{E}}$-簡約とで示す事ができる．

(10) $e \equiv \mathbf{fix}\ e_1$ の場合

定義 2.5 (10) より v は e_1 で形式的厳格なので，帰納法の仮定と $\bot_{\mathrm{fix}}$-簡約とで示す事ができる．

(11) $e \equiv e_1{:}\sigma$ の場合

定義 2.5 (11) より v は e_1 で形式的厳格なので，帰納法の仮定と $\bot_{:}$-簡約とで示す事ができる．

証明終

無論，$\perp$-簡約で導入した定数記号 $\perp$ 自身には「未定義」という意味はない．しかしながら，簡約概念 $\perp$ の各成分簡約概念の定義での簡約基で定数記号 $\perp$ が占める位置の部分式の値が未定義であれば，その簡約基全体の値も未定義となる事は直感的に納得できる．

本論文の Funiq には型無し λ-計算での $\Omega \equiv (\lambda\,\mathrm{x.\,xx})(\lambda\,\mathrm{x.\,xx})$ の様なそれ自身が簡約基でありながら幾ら簡約しても姿を変えない式は存在しない．しかし，Ω の近似として，$\Omega_{\mathrm{Funiq}} \equiv \mathbf{fix}\ (\lambda\,\mathrm{x}{:}\sigma.\mathrm{x})$（型 σ は任意）は，それから始まる簡約列中に現れる任意の式はこの式へ β-簡約で戻す事が可能だという意味と Ω と同様に直感的な意味で値が定義されないという意味との両方の意味で型無し λ-計算に於ける Ω 相当の式と看做す事が許される．

上の定理が述べている事は，式 e 中の形式的厳格な変数を Ω_{Funiq} で置換してから β-簡約を行なえば直感的な値を持たない式 Ω_{Funiq} と（値を持たないという意味で）同等な式に辿り着く，という事である．即ち，上の定理は形式的厳格性の定義の際に述べた「変数が式の中で形式的厳格であるとは，その変数の値が未定義であれば式全体の値も未定義になる」という直感的説明を簡約論の立場から裏付けるものである．

4.6　値呼びに基づく Funiq$_\perp$ に関する簡約論

本節では，値呼び意味論に基づく Funiq$_\perp$ の式の評価に関する簡約関係を与え，この場合にも前節で検討した形式的厳格性が簡約で保存される事を示す．

値呼びの場合，関数適用の評価に先立ち引数式を値まで評価せねばならない．そこで Funiq$_\perp$ に於ける値の概念を予め定義する必要がある．通常は Felleisen, Friedman, Kohlbecker and Duba (1987) やそれに従う Amadio and Curien (1998), 大堀 (1997) 等にある通り，プログラミング言語の立場からは値は自由変数を含まない閉じた式に対してだけ定義されれば良い．一方，本論文では，Funiq/Funiq$_\perp$ を，雛型プログラミング言語としてではなく，そのベースとなる calculus（Barendregt (1984) の λ-calculus の意味で）の立場から自由変数を含み得る一般の式を検討対象としている．また，以上の諸文献では，値呼びでの未適用の λ-抽象内部の簡約基は評価しないとしている．この点に関しては，形式的厳格性の定義 2.5 の直後で述べた通り，本論文では遅延評価という名前呼び／値呼びとは独立な概念として扱う．以上の理由から，値呼びの為の値の集合 **Val** としては次の定義を用いる．

定義 4.32（簡約論的な値の集合）

簡約論的な意味での**値の集合 Val ⊆ Exp** を以下で定義する：

$$V \in \mathbf{Val} \triangleq \beta\text{-正規形の集合}.$$

値呼びの場合，関数適用に先立ち引数式を値まで評価する，という条件がある．故に式の中に複数の簡約基が存在する場合，その評価順序に制限があるので，Felleisen, Friedman, Kohlbecker and Duba (1987) での評価文脈を用いて簡約関係を定義する．

定義 4.33（評価文脈）

評価文脈 $\boldsymbol{E}[\,\cdot\,]$ を以下によって定義する：

$$\begin{aligned}
E ::= {} & [\,\cdot\,] \\
\mid {} & \lambda x{:}\sigma.E \\
\mid {} & Ee \\
\mid {} & VE \\
\mid {} & \{l_1 = V_1, \ldots, l_{i-1} = V_{i-1}, l_i = E, l_{i+1} = e_{i+1}, \ldots, l_n = e_n\} \\
\mid {} & E.l \\
\mid {} & [\,l = E\,] \\
\mid {} & \mathbf{case}\ E\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n \\
\mid {} & \Lambda\, t{<:}\sigma.E \\
\mid {} & E[\sigma] \\
\mid {} & \mathbf{pack}_{\exists t<:\sigma.\tau}\ E\ \mathbf{with}\ \rho \\
\mid {} & \mathbf{unpack}\ E\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e \\
\mid {} & \mathbf{unpack}\ V\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ E \\
\mid {} & \mathbf{fix}\ E \\
\mid {} & E{:}\sigma.
\end{aligned}$$

なお値呼びで遅延評価を採用する Funiq$^\ell_\perp$ に対する評価文脈 E は，上の定義から $\lambda x{:}\sigma.E$ の場合を除いて定義される．また，その場合，値 V としては，β-正規形だけでなく関数抽象式の形の全ての式を許さねばならない．

以上で定義された評価文脈を用いて Funiq$_\perp$ の式の集合上に簡約概念を定義する．

定義 4.34 （簡約概念 $\boldsymbol{\beta^v}$）

Funiq$_\perp$ の式の集合 **Exp** 上に簡約概念 $\boldsymbol{\beta^v}$ を以下の様に定義する：

$$
\begin{aligned}
\boldsymbol{\beta^v_{\to}} &: E[(\lambda x{:}\sigma.V_1)V_2] \longmapsto E[V_1[x := V_2]],\\
\boldsymbol{\beta^v_{\{\cdots\}}} &: E[\{l_1 = V_1, \ldots, l_n = V_n\}.l_i] \longmapsto V_i, \qquad (1 \le i \le n)\\
\boldsymbol{\beta^v_{[\cdots]}} &: E[\mathbf{case}\ [l_i = V\,]\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n] \longmapsto E[e_i V],\\
&\qquad\qquad\qquad\qquad\qquad\qquad (1 \le i \le n)\\
\boldsymbol{\beta^v_{\forall}} &: E[(\Lambda\, t{<:}\sigma.V)[\tau]] \longmapsto E[V[t := \tau]],\\
\boldsymbol{\beta^v_{\exists}} &: E[\mathbf{unpack}\ (\mathbf{pack}_{\exists t<:\sigma.\tau}\ V_1\ \mathbf{with}\ \rho)\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ V_2]\\
&\qquad \longmapsto E[V_2[t := \rho][x := V_1]],\\
\boldsymbol{\beta^v_{\mathbf{fix}}} &: E[\mathbf{fix}\ V] \longmapsto E[V(\mathbf{fix}\ V)];
\end{aligned}
$$

$$
\boldsymbol{\beta^v} \triangleq \boldsymbol{\beta^v_{\to}} \cup \boldsymbol{\beta^v_{\{\cdots\}}} \cup \boldsymbol{\beta^v_{[\cdots]}} \cup \boldsymbol{\beta^v_{\forall}} \cup \boldsymbol{\beta^v_{\exists}} \cup \boldsymbol{\beta^v_{\mathbf{fix}}}.
$$

以上の簡約概念 $\boldsymbol{\beta^v}$ を用いて，大堀 (1997), p. 86 にある通り， Funiq$_\perp$ の式に対する β-簡約を定義する．

定義 4.35 （$\boldsymbol{\beta^v}$-簡約）

Funiq$_\perp$ の式に対する簡約関係 — β^v-簡約と呼ぶ — を次で定義する：

$$
\begin{aligned}
&e_1 \longrightarrow_{\beta^v} e_2 \overset{\text{def}}{\iff}\\
&\qquad \exists E.\, \exists e_1', e_2' \in \mathbf{Exp}.\, \left[e_1 \equiv e_1' \text{ かつ } e_2 \equiv e_2' \text{ かつ } \langle E[e_1'], E[e_2']\rangle \in \boldsymbol{\beta^v}\right].
\end{aligned}
$$

更に， $\twoheadrightarrow_{\beta^v}$ は $\longrightarrow_{\beta^v}$ の反射推移閉包とする．

以上の単一ステップ β^v-簡約に関しては次の性質が成り立っている．

命題 4.36 （β^v-簡約の決定性）

β^v-簡約は決定性を持つ．即ち，$e, e' \in \mathbf{Exp}$ で $e \longrightarrow_{\beta^v} e'$ とする時，$e \equiv E[e_1]$, $e' \equiv E[e'_1]$, $\langle e_1, e'_1 \rangle \in \boldsymbol{\beta^v}$ を満たす二つの式 e_1, e'_1 ならびに評価文脈 E は一意に定まる．従って，以下では

- e_1 を **e の β^v-簡約基**,
- e'_1 を e_1 の **β^v-縮約結果**,
- E を **e の評価文脈**

と呼ぶ（4.1 節での β-簡約の場合と異なり，β^v-簡約の場合は**「どの式の」**という事抜きでは簡約基の概念を定義できない事に注意する必要がある）．

証明

e の構造に関する帰納法で示せば良い．

証明終

以上の値呼びの β^v-簡約に対しては，しかしながら，定理 4.28 に相当する次の主張は成り立たない．

非定理 4.37 （形式的厳格変数の β^v-簡約保存性）

$v \in \mathbf{Var} \cup \mathbf{IVar}$ は $e \in \mathbf{Exp}$ で形式的厳格とする．この時，

$$\forall e' \in \mathbf{Exp}.\big[e \longrightarrow\!\!\!\!\!\rightarrow_{\beta^v} e' \quad \Longrightarrow \quad v \text{ は } e' \text{ で形式的厳格}\big].$$

反証

次の v, e, e' の組合せが反例を与える． $v \equiv \mathrm{z}$, $e \equiv (\lambda \mathrm{x}{:}\{\}.\mathrm{y})\mathrm{z}$, $e' \equiv \mathrm{y}$ と選ぶと，確かに $e \longrightarrow\!\!\!\!\!\rightarrow_{\beta^v} e'$ （$\because \langle e, e' \rangle \in \boldsymbol{\beta^v}$）であり，変数 z は e 中で定義 2.6 $(3')$ より形式的厳格であるが， $\mathrm{z} \notin \mathrm{FV}(e')$ であるので z は e' で形式的厳格でない．

反証終

従って，値呼び意味論での形式的厳格性を β^v-簡約で保存される変数として特徴付ける事はできないが，値呼びの場合も，それに応じた $\bot$-簡約を定義する事で，定理 4.31 に準ずる形で形式的厳格性を特徴付けられる．

定義 4.38（$\perp^v$-簡約）

$\perp$ で拡大された式に対する値の集合 $\mathbf{Val}\perp \subseteq \mathbf{Exp}\perp$ を

$$V \in \mathbf{Val}\perp \triangleq \perp\text{-正規形の集合}$$

と定義し（ここで $\perp$-簡約は定義 4.29 で定義されたもの），定義 4.33 の評価文脈の定義から VE と **unpack** V **as** x **with** t **in** E との二つの場合を除いた上でその定義を式の集合 $\mathbf{Exp}\perp$ と値の集合 $\mathbf{Val}\perp$ との上に拡大し，以下の一連の簡約概念を定義する：

$$\begin{aligned}
\perp^{\boldsymbol{v}}_{\boldsymbol{\to}\mathbf{I}} &: E[\lambda x{:}\sigma.\perp] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\to}\mathbf{E1}} &: E[\perp e] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\to}\mathbf{E2}} &: E[V\perp] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\{\cdots\}}\mathbf{I}} &: E[\{l_1 = \perp, \ldots, l_n = \perp\}] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\{\cdots\}}\mathbf{E}} &: E[\perp.l] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{[\cdots]}\mathbf{I}} &: E[[\,l = \perp\,]] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{[\cdots]}\mathbf{E}} &: E[\mathbf{case}\ \perp\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\forall}\mathbf{I}} &: E[\Lambda\, t{<:}\sigma.\perp] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\forall}\mathbf{E}} &: E[\perp[\sigma]] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\exists}\mathbf{I}} &: E[\mathbf{pack}_{\exists t<:\sigma.\tau}\ \perp\ \mathbf{with}\ \rho] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\exists}\mathbf{E1}} &: E[\mathbf{unpack}\ \perp\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{\exists}\mathbf{E2}} &: E[\mathbf{unpack}\ V\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ \perp] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\mathbf{fix}} &: E[\mathbf{fix}\ \perp] \longmapsto \perp,\\
\perp^{\boldsymbol{v}}_{\boldsymbol{:}} &: E[\perp{:}\sigma] \longmapsto \perp;
\end{aligned}$$

$$\begin{aligned}
\perp^{\boldsymbol{v}} \triangleq\ & \perp^{\boldsymbol{v}}_{\boldsymbol{\to}\mathbf{I}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\to}\mathbf{E1}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\to}\mathbf{E2}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\{\cdots\}}\mathbf{I}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\{\cdots\}}\mathbf{E}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{[\cdots]}\mathbf{I}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{[\cdots]}\mathbf{E}}\\
& \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\forall}\mathbf{I}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\forall}\mathbf{E}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\exists}\mathbf{I}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\exists}\mathbf{E1}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{\exists}\mathbf{E2}} \cup \perp^{\boldsymbol{v}}_{\mathbf{fix}} \cup \perp^{\boldsymbol{v}}_{\boldsymbol{:}}.
\end{aligned}$$

以上の簡約概念 $\perp^{\boldsymbol{v}}$ を元に，単一ステップならびに一般の $\perp^v$-簡約 $\longrightarrow_{\perp^v}$ と $\twoheadrightarrow_{\perp^v}$ とは，定義 4.35 での $\boldsymbol{\beta^v}$ に対するものと同様にして定義される．

以上の $\perp^v$-簡約を用いると，値呼び意味論での形式的厳格性は次の定理で特徴付けられる．

定理 4.39（簡約論的厳格性 — 値呼び版）

$e \in \mathbf{Exp}$ を任意の式とし $v \in \mathbf{Var} \cup \mathbf{IVar}$ は e で形式的厳格とする．この時，

$$e[v := \bot] \longrightarrow\!\!\!\!\!\rightarrow_{\bot^v} \bot.$$

証明

まず，$\bot^v$-簡約は，式の長さ（文字数）を確実に減少させるので，強正規化性を満たす事を注意しておく．

証明は，式 e の構造に関する帰納法によって行なう．何れの構造の場合も同様の論証で行なえるので，e が関数適用式の場合のみを示す．

$e \equiv e_1 e_2$ である場合

値呼びの場合の形式的厳格性の定義定義 2.6 $(3')$ より，変数 v は e_1, e_2 の少なくとも一方で形式的厳格である．

(a) v が e_1 で形式的厳格である場合

帰納法の仮定より $e_1[v := \bot] \longrightarrow\!\!\!\!\!\rightarrow_{\bot^v} \bot$ である．従って，

$$e[v := \bot] \equiv (e_1[v := \bot])(e_2[v := \bot]) \longrightarrow\!\!\!\!\!\rightarrow_{\bot^v} \bot(e_2[v := \bot]) \longrightarrow_{\bot^v_{\rightarrow \mathrm{E1}}} \bot$$

となり，この場合は定理の言明が成立する．

(b) v が e_1 では形式的厳格でなく e_2 で形式的厳格である場合

帰納法の仮定より $e_2[v := \bot] \longrightarrow\!\!\!\!\!\rightarrow_{\bot^v} \bot$ である．この場合，v は e_1 に自由変数として含まれ得るが形式的厳格ではないので，$e_1[v := \bot]$ には $\bot$ は形式的厳格でない位置にしか現れ得ない．従って，$e_1[v := \bot]$ 中に $\bot$ が存在しても，それは $\bot^v$-簡約の原因にはなり得ない（その事は，$\bot^v$-簡約概念の定義 4.38 と定義 2.6 とを対照すれば判る）．

故に，$e[v := \bot] \equiv (e_1[v := \bot])(e_2[v := \bot]) \longrightarrow\!\!\!\!\!\rightarrow_{\bot^v} (e_1 e[v := \bot])\bot \longrightarrow_{\bot^v_{\rightarrow \mathrm{E2}}} \bot$ となり，この場合も定理の言明が成立する．

証明終

第5章

Funiqの表示的意味論

— 健全性を中心に —

本章では，広帯域言語 Funiq の Scott 理論（領域論）に基づく表示的意味論の定義を与え，関連する性質に関して検討する．

まず， 5.1 節では本論文での Funiq の表示的意味を与える上で前提とされる内容および後の議論で用いる為の必要最小限の範囲の Scott 流の領域論の要約として，完備半順序集合等に関する基本的な概念／用語／記号を準備する．更に， 2 階の型付き λ-計算に対する領域論的な意味論の従来研究の概観を研究略史の形で与える．それ故，本節は — 分量としては少なくないが — 本論文独自の内容を含まない．

次に，上の領域論で用意された道具に基づき Funiq の式／型／表明の表示的意味を与え，型の意味関数 $\mathcal{T}$ が整定 (well-defined) である事を確認する．この確認の過程で， 2.1 節の定義 2.8 での条件 (b) が型の意味関数 $\mathcal{T}$ の整定性を保証する — 特に詳細化型の意味を与える — 上で本質的である事が判る．

第三に，上で与えられた表示的意味に対し，型理論 FUNIQ が健全である事，つまり判別式 Σ が導出可能ならば表示的意味に基づき定義される妥当性の意味で Σ は妥当である事を証明する．次いで β-簡約での式の評価が式の表示を保つという簡約の健全性も示す．これらの健全性は，値呼びに基づく $\text{Funiq}_{\perp}$ に対しても成立する．

なお，本章では， 3.1 節での定義 3.1 で与えた多相型と関数型による抽象型のコード化と定理 3.2 でのそれの正しさとに基づき，表示的意味を与える対象からは抽象型と関連する式を除いて扱う．

5.1 領域論からの準備と多相型の領域論的意味の研究史

本論文での Funiq は，従来，数理論理学的関心から研究されて来た型付き λ-計算の様々な体系 — 例えば， Girard (1971, 1972) と Reynolds (1974) の F， Cardelli and Wegner (1985) のオリジナルの Fun， Coquand and Huet (1985, 1988) の Calculus of Construction， Luo (1993) の ECC 等 — とは異なり，不動点による再帰の為の構成 **fix** e を含み停止しない計算を表わし得る．その為，上記の各体系に対する表示的意味を単純な集合論に基づいて構築できるのと対照的に， Funiq の表示的意味は，通常のプログラミング言語に対する表示的意味と同様，一定の性質を満たす半順序集合 — 即ち， Scott の領域 — を用いて与えられねばならない．

領域論の必要性に関しては若干の注意が必要であるので，本論からは外れるがより正確に述べておく．多相型を許す λ-計算の体系である F やそれを部分言語として含む Cardelli and Wegner オリジナルの Fun 等，非可述的 (impredicative)[1)] な多相型を許す 2 階の型付き λ-計算の場合，不動点再帰等の停止しない計算を許さずとも，1 階の単純型付き λ-計算の体系 $\lambda^{\rightarrow}$ （不動点演算を含まず）の場合とは対照的に単純な — 型を値の集合として解釈する意味での — 集合論的モデルを与える事が不可能

1) 非可述性の肯定形である可述性は， Frege との文通の中で Russell が発見した彼のパラドックス — 自分自身を含まない集合全ての成す集合 R は R 自身の要素か否か？ — の根本的原因である悪循環 (vicious circle) を防ぐ為に数学での定義が満たすべき要件として導入した概念である．

Girard の体系 F 等での多相型について言えば，多相型 $\forall t.\sigma$ の意味を考える場合，「全ての型 t について云々」での t として，この型自身も許される点が非可述的と言われる結縁である．

多相型の非可述性は意味論の構築やそれに関する定理の証明で様々な障害を引き起こす．実際，多相型に対して表示的意味を与えようとした初期の研究の一つである Donahue (1979) では，以上に述べた多相型の非可述性を見落としている為に，その意味関数が well-defined でなくなってしまうというミスを犯している．また， Wadler (1989) の “Theorems for Free!” では， Parametricity Theorem は一見しただけでは構文的（証明論的）に証明を与えられそうにも拘らず，多相型の非可述性の為に意味論的に証明せねばならなくなっている．簡約に関する強正規化性定理の証明に関しても，多相型のない単純型付き λ-計算 $\lambda^{\rightarrow}$ の場合と比べ多相型を許す F の場合は遥かに複雑になる．

一方では，多相型の持つ非可述性は， Fortune, Leivant and O'Donnell (1983) で示された通り，F の計算能力を $\lambda^{\rightarrow}$ と比べて遥かに強力にしている．実際， $\lambda^{\rightarrow}$ の計算能力は自然数上の原始再帰関数の範囲にも届かないが， F では Ackermann 関数が表現可能なので， elementary function （時間計算量で任意の有限の高さの指数冪を許す全域関数のクラス． NP や NEXP も極めて小さな真部分集合として含む）の全てを表現可能である．この事は，計算の停止性を保証する強正規化性の証明が F の場合に $\lambda^{\rightarrow}$ に比べて遥かに困難である事の裏返しでもある．

故に，時間計算量論的に言えば，停止する事が保証されている自然数上の関数については，実用的に有用な関数だけでなく単なる理論的興味だけの関数も含め，それらの殆ど全ては F で表現可能である，との Girard (1986) の主張は間違っていない．

但し，一般再帰的な形で定義された具体的な関数が与えられた時，その停止性は保証されていても，その関数を F の項として（高階の原始再帰的な形で）コード化する事は自明と呼ぶに程遠い．その理由は，関数を F で表現する事はその停止性を構成的な手段のみ — しかも非常に限定された形 — で証明する事と等しく，関数の停止性の通常の証明 — 集合論や古典論理といった非構成的手段を自在に用いる事が許される — に比べて技術的に大幅に難しいからである．

である．この事は Reynolds (1984) により証明されている．但し，集合論でも古典論理でなく直観主義論理に基づく構成主義的集合論を用いれば，2 階の型付き λ-計算の集合論的意味論を与え得る事は Pitts (1987) が示した．

実際，不動点演算を含まず停止する計算のみを許す F に対する表示的意味は，大堀 (1997) の 5.3 節での様に領域論的な枠組を用いて与えられる事が多い．F および関連する 2 階の型付き λ-計算に対する領域論的な表示的意味の定義の歴史に関しては本節の最後に纏める．

本節の内容はオリジナルでなく，Scott の領域論に関する標準的な諸文献 — Plotkin (1983), Barendregt (1984), Jung (1989), Davey and Priestley (1990), Gunter (1992), Amadio and Curien (1998) 等 — の内容のうち，次節での Funiq の表示的意味論の議論上で必要最小限の領域論に関する諸定義と基本的性質との纏めであり，基本的性質に対する証明は与えない．更に，領域論の枠組を用いての 2 階の型付き λ-計算の意味論に関する従来研究の概観を本節の最後に示す．後の議論の便宜上，一部の定義に関しては変更を加えたが，原則として断わらない．

■ 完備半順序集合・Scott 位相・連続関数・領域

定義 5.1 （ω-上昇鎖）

半順序集合 $\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ の部分集合 $V \subseteq D$ が **ω-上昇鎖**であるとは，V が以下の 2 条件を満たす事である：

(1) V の全ての要素は自然数によって添字付ける事が可能である．即ち，

$$V = \{d_i \in D \mid i \in \omega\};$$

(2) (1) での添字付けを適当に選ぶ事により次を満たす事ができる．即ち，

$$d_i \sqsubseteq_{\mathbf{D}} d_j \quad \Longleftrightarrow \quad i \leq j.$$

以下，V が ω-上昇鎖であると言う場合，常に (2) を満たす添字付けを選んでいる事を仮定する．即ち，$d_0 \sqsubseteq_{\mathbf{D}} d_1 \sqsubseteq_{\mathbf{D}} d_2 \sqsubseteq_{\mathbf{D}} \cdots$ と仮定する．

なお，(1) では同一要素を重複して添字付けるのを許しているので，V は有限集合でも良い事を注意しておく（但し，(2) より $\sqsubseteq_{\mathbf{D}}$ は V 上で必ず線型順序となる）．

ω-上昇鎖の概念を一般化すると次の有向集合の概念を得る．

定義 5.2 （有向集合）

半順序集合 $\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ の部分集合 $V \subseteq D$ が**有向**であるとは，V の任意の 2 要素の上界が V 自身の中に存在する事である．即ち，

$$\forall d, d' \in V. \exists d'' \in V. \big[d \sqsubseteq_{\mathbf{D}} d'' \text{ かつ } d' \sqsubseteq_{\mathbf{D}} d'' \big].$$

$V \subseteq D$ が ω-上昇鎖であれば，明らかに V は有向集合である．

次に，その中の任意の有向集合（或いは ω-上昇鎖）に対して，その「極限」の存在を保証する特殊な半順序集合のクラスを定義する．

定義 5.3 （完備半順序集合 cpo）

半順序集合 $\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ が**有向完備半順序集合 (dcpo)** であるとは，以下の 2 条件が満たされる事である：

(1) D は半順序 $\sqsubseteq_{\mathbf{D}}$ に関し最小要素を持つ．この最小要素を $\bot_{\mathbf{D}}$ で示す；

(2) D の任意の有向部分集合 $V \subseteq D$ は D 中に上限を持つ．この上限を $\bigsqcup^{\mathbf{D}} V$ で表わす．

(2) での上限の存在を有向集合ではなく ω-上昇鎖に対して要請する，即ち，

(2′) D 中の任意の ω-上昇鎖 $V \subseteq D$ は D 中に上限を持つ

とした場合，$\mathbf{D}$ を **ω-上昇鎖完備半順序集合 (ccpo)** と呼ぶ．

一般の半順序集合 $\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ に対して，D を $\mathbf{D}$ の**台集合**と呼び，$|\mathbf{D}|$ で表わす場合がある．また，半順序・最小要素・有向集合の上限が何れの cpo に関するものであるのか文脈より明らかな場合，それらの添字としての cpo 名を省く場合がある．

有向集合で定義される dcpo と ω-上昇鎖による ccpo とは関数等の再帰的定義を解釈する目的からは同等である事が Smyth and Plotkin (1982) で指摘されている．そこで，以下，両者を区別せず単に**完備半順序集合 (cpo)** と呼び，定義としては (2) の有向集合によるのと (2′) の ω-上昇鎖によるのとの便利な方を用いる（基本的には有向集合による定義を用いる）．なお，文献によっては cpo の定義で条件 (1) を要請せず条件 (2)（或いは (2′)）のみを課す場合もある．その場合，(1) をも満たすものを改めて**点付けされた** cpo と呼ぶが，ここでは両条件を満たすものを cpo と呼ぶ．

Cpo 間の関数が最小要素 $\bot$ を保存するか否かは，しばしば — 既に 2.1 節でそれに対応する構文上の概念として「形式的厳格性」を導入した事でも判る通り — 重要なポイントとなるので，最小要素を保存する関数を表わす用語を導入しておく．

定義 5.4 （厳格性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を共に cpo とする時，関数 $f : D \to E$ が**厳格**であるとは，f が最小要素を保存する事，即ち，

$$f(\bot_{\mathbf{D}}) = \bot_{\mathbf{E}}$$

を満たす事である．

Funiq の基本型は内部構造を持たないと仮定した．その様な単純な型を解釈する為に非常に単純な構造を持つ cpo のクラスを定義しておく．

定義 5.5 （平坦 cpo）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ が平坦 cpo である，とは，D 上の半順序関係 $\sqsubseteq_{\mathbf{D}}$ が次を満たす事である：

$$\forall d, d' \in D.\big[d \sqsubseteq_{\mathbf{D}} d' \quad \Longrightarrow \quad d = d' \text{ または } d = \bot_{\mathbf{D}}\big].$$

即ち，平坦 cpo では最小要素 $\bot_{\mathbf{D}}$ 以外の要素同士の間には順序関係が無い．平坦 cpo の例として真理値の cpo と自然数の cpo とを図 5.1 と図 5.2 とに示す．なお，両図の標題中の “$(\,\cdot\,)_\bot$” の意味に関しては後の定義 5.25 を参照せよ．

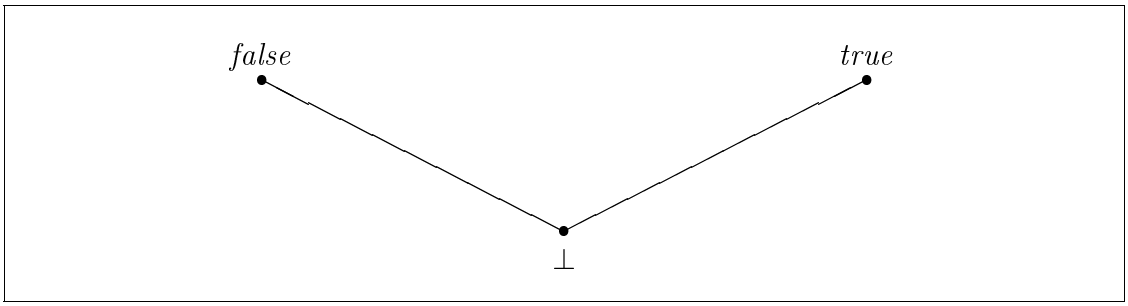


図 5.1 真理値のなす平坦 cpo — $\mathbf{T} \triangleq \mathbb{T}_\perp$

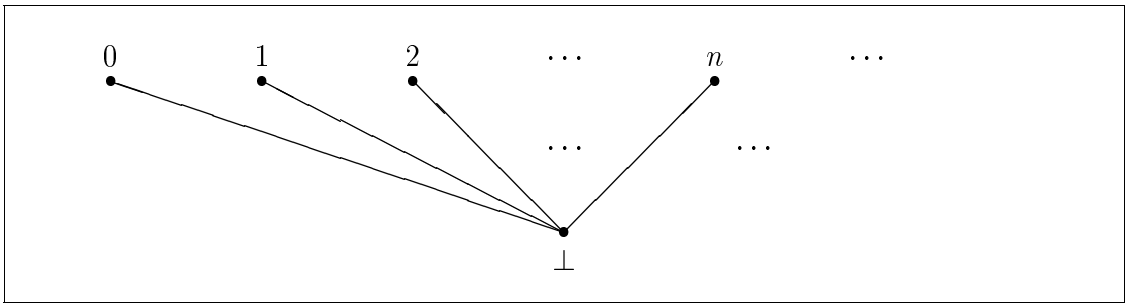


図 5.2 自然数のなす平坦 cpo — $\mathbf{N} \triangleq \mathbb{N}_\perp$

次に， cpo 上にその順序関係と上限により表わされている収束概念とに基づいた位相を与えるが，その前に半順序に関して下に或いは上に閉じている部分集合を簡潔に表現する為の一連の記号を約束しておく．

定義 5.6 （下方閉・上方閉）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を半順序集合とする時，任意の $d \in D$ と $X \subseteq D$ とに対して，以下の D の部分集合を定める：

$$\begin{aligned} \downarrow d &\triangleq \{d' \in D \mid d' \sqsubseteq_{\mathbf{D}} d\}; \\ \downarrow X &\triangleq \bigcup_{d \in X} \downarrow d; \\ \uparrow d &\triangleq \{d' \in D \mid d \sqsubseteq_{\mathbf{D}} d'\}; \\ \uparrow X &\triangleq \bigcup_{d \in X} \uparrow d. \end{aligned}$$

定義 5.7 （Scott 位相）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，$\mathbf{D}$ 上の **Scott 位相**とは以下を満たす集合族 $\mathcal{O} \subseteq \mathfrak{P}D$ を開集合族とする位相である．即ち，任意の $O \in \mathcal{O}$ について，

(1)　$\forall d \in O. \forall d' \in \uparrow d. \left[d' \in O\right]$;

(2)　$\forall$有向集合 $V \subseteq D. \left[\bigsqcup^{\mathbf{D}} V \in O \implies V \cap O \neq \varnothing\right]$.

以下，cpo $\mathbf{D}$ 上の Scott 位相が定める開集合族を $\mathcal{O}_{\mathbf{D}}$ で表わし，D の部分集合が Scott 位相の意味での開集合・閉集合である事を，各々，**Scott 開**・**Scott 閉**と呼ぶ.

ここで定義した Scott 位相は非常に弱い位相で，T_0-分離公理しか満たさない．以上の Scott 位相の定義は，順序集合上の順序位相として昔から知られている Alexandroff 位相の定義 — 条件 (1) — に cpo での極限の概念を反映した条件 (2) を追加したものである．

以下，Scott 位相の意味での連続関数を定義するが，それは通常の関数の連続性の定義そのものである．即ち，

定義 5.8 （連続性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，関数 $f : D \to E$ が**連続**であるとは，E の任意の開部分集合 $O \in \mathcal{O}_{\mathbf{E}}$ の f による逆像が D の開部分集合となる事である．即ち，

$$\forall O \in \mathcal{O}_{\mathbf{E}}. \left[f^{-1}(O) \in \mathcal{O}_{\mathbf{D}}\right].$$

Scott 位相の意味での連続関数は，単調かつ有向集合（或いは ω-上昇鎖）の上限を保存する関数として特徴付けられる．即ち，

定義 5.9 （単調性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，関数 $f : D \to E$ が**単調**であるとは， f が以下の条件を満たす事である：

$$\forall d_1, d_2 \in D. \left[d_1 \sqsubseteq_{\mathbf{D}} d_2 \quad \Longrightarrow \quad f(d_1) \sqsubseteq_{\mathbf{E}} f(d_2) \right].$$

事実 5.10 （Scott 連続性の特徴付け）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，関数 $f : D \to E$ が連続である事は以下の 2 条件を共に満たす事と同値である：

(1) f は単調である；

(2) 任意の有向集合の上限を保存する．即ち，

$$\forall \text{有向な } V \subseteq D. \left[f\left(\bigsqcup\nolimits^{\mathbf{D}} V \right) = \bigsqcup\nolimits^{\mathbf{E}} f(V) \right].$$

Scott 位相の意味でのコンパクトな（有限な）要素の概念を定義する．その為の準備として，順序関係 $\sqsubseteq$ よりも強い cpo での上限操作に配慮した関係を導入する．

定義 5.11 （道中関係）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，その任意の 2 要素 $d, d' \in D$ について， d が d' への**道中にある (way-below)** — $\boldsymbol{d \ll d'}$ —とは，次を満たす事である：

$$\forall \text{有向な } V \subseteq D. \left[d' \sqsubseteq_{\mathbf{D}} \bigsqcup\nolimits^{\mathbf{D}} V \quad \Longrightarrow \quad \exists d'' \in V. (d \sqsubseteq_{\mathbf{D}} d'') \right].$$

$d \ll d'$ とは， d' に有向集合（或いは ω-上昇鎖）の極限により抑えられるならば d へは有限に届く — 有向集合（或いは ω-上昇鎖）の具体的な要素によって抑える事ができる — 事を意味している．

事実 5.12（道中関係の基本的性質）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時,

(1) 任意の $d, d' \in D$ について,

$$d \ll d' \quad \Longrightarrow \quad d \sqsubseteq_{\mathbf{D}} d'.$$

(2) 任意の $d_1, d_2, d_3, d_4 \in D$ について,

$$d_1 \sqsubseteq_{\mathbf{D}} d_2 \ll d_3 \sqsubseteq_{\mathbf{D}} d_4 \quad \Longrightarrow \quad d_1 \ll d_4.$$

以下で cpo の要素のコンパクト性を定義するが，或る要素がコンパクト（有限）であるとは，その要素が有向集合（或いは ω-上昇鎖）の極限で抑えられる場合には必ず有限な段階で抑え得る，という事である．

定義 5.13（コンパクト性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時,

(1) 任意の $d \in D$ について，以下の部分集合を定める：

$$\begin{aligned} \downarrow\!\!\downarrow d &\triangleq \{d' \in D \mid d' \ll d\}; \\ \uparrow\!\!\uparrow d &\triangleq \{d' \in D \mid d \ll d'\}. \end{aligned}$$

(2) $d \in D$ が**コンパクト**（もしくは**有限**）であるとは，それ自身の道中にある — 即ち，$d \ll d$ — という事である．

(3) $\mathbf{D}$ のコンパクトな要素全ての集合を $\mathcal{K}(\mathbf{D})$ で表わす．

(4) $d \in D$ に対して，d を近似するコンパクト要素全ての集合 $\mathcal{K}(d) \subseteq D$ を以下の様に定義する：

$$\mathcal{K}(d) \triangleq \mathcal{K}(\mathbf{D}) \cap \downarrow d.$$

以上の要素に関するコンパクト性に基づき cpo の代数性を定義する．この名前は代数系の部分代数の成す束の性質に由来する． Scott (1972) が型無し λ-計算のモデ

ルとして用いた連続束での束の連続性の概念は cpo にも一般化できる．連続 cpo は本論文で用いないのでその定義は省略し，代数的 cpo の定義のみを与える．

定義 5.14 （代数的 cpo）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時， $\mathbf{D}$ が**代数的**とは，次を満たす事である：

$$\forall d \in D. \left[\mathcal{K}(d) \text{ は有向集合 かつ } d = \bigsqcup\nolimits^{\mathbf{D}} \mathcal{K}(d) \right].$$

更に $\mathcal{K}(\mathbf{D})$ が可算集合である場合， **ω-代数的**と呼ぶ．

要するに，代数的 cpo とは，その任意の要素がその要素を近似するコンパクトな要素のみで復元可能である様な cpo である．なお，上の代数的 cpo の定義中の「有向集合」は「ω-上昇鎖」で置き換えられない，という点に注意する必要がある．その理由は， cpo $\mathbf{D}$ 自身が鎖（線型順序集合）とか平坦 cpo とかの特殊な場合を除き，$\mathcal{K}(d)$ は一般に鎖とならないからである．

上で触れた cpo の連続性は， cpo 中で有向集合（或いは ω-上昇鎖）に対する上限操作 $\bigsqcup$ が奇妙な振舞いをしない事を保証する． Cpo の代数性は連続性よりも強い性質 — 代数的 cpo は常に連続的 cpo — であり，代数的 cpo でも上限操作は大人しく振舞う事が約束されている．その事は次の事実から垣間見る事ができる．

事実 5.15 （Scott 位相の基）

$\mathbf{D}$ を代数的 cpo とする．この時，集合族 $\{\uparrow e \mid e \in \mathcal{K}(\mathbf{D})\}$ は $\mathbf{D}$ の Scott 位相の基（として可能なものの一つ）を成す．

以上の cpo の代数性だけでは，プログラミング言語や本論文の不動点演算を含む型付き λ-計算の意味を捉える為の意味領域を組み上げる為の素材が満たすべき条件としては不十分である．その理由は， Jung (1990) が示した通り，連続関数空間の構成（定義 5.27 ）に関して cpo の代数性（やそれを緩めた連続性）だけでは閉じない（事実 5.28 (2) を参照）事にある．連続関数空間の構成に関しても閉じた cpo のクラスを得る為には，次の新たな性質（条件）を必要とする．

定義 5.16（整合完備性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする．この時，

(1) 部分集合 $X \subseteq D$ が**整合的 (consistent)** であるとは，X の任意の 2 要素の上界が D に存在する事である．即ち，

$$\forall d, d' \in X. \exists d'' \in D. \left[d \sqsubseteq_{\mathbf{D}} d'' \text{ かつ } d' \sqsubseteq_{\mathbf{D}} d'' \right].$$

(2) $\mathbf{D}$ が**整合完備**である（もしくは**整然 (coherent)** としている）とは，その任意の整合的な部分集合 X に対して，X の上限が D 中に存在する事である．

本論文で意味領域として用いるのは，以下のクラスの cpo である．なお，今まで一般用語的に漠然と用いて来た「領域 (domain)」という単語は，今後，次で定義されるクラスの cpo を指す技術用語とする．

定義 5.17（領域）

Cpo $\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ が **Scott 領域**（或いは単に**領域**）であるとは，$\mathbf{D}$ が整合完備かつ代数的だという事である．

以下に示す通り，領域という性質は単純な cpo から複合的な cpo を組み上げる為の様々な構成方法で保存されるが，その前に，最も単純な cpo である平坦 cpo が領域である事を確認しておく．

事実 5.18

平坦 cpo は領域である．

■ 複合的 cpo の構築の為の cpo 構成子

複合的な cpo の構成方法（**cpo 構成子**）として最初に示すのは，可算個の cpo の「積」である．cpo の「積」としては，成分 cpo の最小要素の扱いの違いにより二通りの構成法が存在する．それらの各々を順に示す．

定義 5.19（cpo の直積）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，それらの**直積**を以下で定まる半順序集合 $\mathbf{D} \times \mathbf{E}$ として定義する：

- $|\mathbf{D} \times \mathbf{E}| \triangleq D \times E$;
- $\langle d, e\rangle \sqsubseteq_{\mathbf{D}\times\mathbf{E}} \langle d', e'\rangle \overset{\text{def}}{\iff} d \sqsubseteq_{\mathbf{D}} d'$ かつ $e \sqsubseteq_{\mathbf{E}} e'$.

cpo の直積はそれ自身が cpo を成し，代数性や整合完備性を保存する．更に，直積構成に関連する基本演算は連続関数となる．即ち，

事実 5.20（直積 cpo）

(1) 上で定義した半順序集合 $\mathbf{D} \times \mathbf{E}$ は $\bot_{\mathbf{D}\times\mathbf{E}} = \langle \bot_{\mathbf{D}}, \bot_{\mathbf{E}}\rangle$ を最小要素とする cpo である．

(2) $\mathbf{D}, \mathbf{E}$ が共に代数的（もしくは整合完備）ならば $\mathbf{D} \times \mathbf{E}$ も同様である．

(3) 直積 cpo $\mathbf{D} \times \mathbf{E}$ の各成分への射影演算

$$\_.1 : D \times E \to D : \langle d, e\rangle \mapsto d,$$
$$\_.2 : D \times E \to E : \langle d, e\rangle \mapsto e$$

は共に連続関数である．

(4) 対化演算

$$\langle \_, \_\rangle : |\mathbf{D}| \times |\mathbf{E}| \to |\mathbf{D} \times \mathbf{E}| : d, e \mapsto \langle d, e\rangle$$

は各引数毎に連続である．

（注意：この対化演算の変域・値域は共に $D \times E$ そのものであるが，二つの cpo $\mathbf{D}$ と $\mathbf{E}$ 各々の要素の対を直積 cpo の一つの要素に写すという事を強調する為，変域を $|\mathbf{D}| \times |\mathbf{E}|$ で値域を $|\mathbf{D} \times \mathbf{E}|$ で表わした）

(5) 個々の引数毎に連続な 2 引数関数は，各々の引数の変域 cpo を成分とする直積 cpo 上の単一引数関数として連続である．

以上の直積の定義は可算個（可算無限個でも可）の cpo に対して一般化する事が可能であり，$\mathbf{D}_1 \times \mathbf{D}_2 \times \cdots \times \mathbf{D}_n$ とか $\prod_{i\in\omega} \mathbf{D}_i$ という形で表わす．

もう一つの「積」は，上の直積構成が順序も最小要素も成分毎に扱ったのに対し，順序対の成分が最小要素である場合は常に積での最小要素として扱う構成である．

定義 5.21（cpo の厳格積）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，それらの**厳格積**を以下で定まる半順序集合 $\mathbf{D} \otimes \mathbf{E}$ として定義する：

- $|\mathbf{D} \otimes \mathbf{E}| \triangleq (D \setminus \{\bot_{\mathbf{D}}\}) \times (E \setminus \{\bot_{\mathbf{E}}\}) \cup \{\bot_{\mathbf{D}\otimes\mathbf{E}}\}$;
- $z \sqsubseteq_{\mathbf{D}\otimes\mathbf{E}} z' \overset{\text{def}}{\iff} z = \bot_{\mathbf{D}\otimes\mathbf{E}}$ または $z = \langle d, e\rangle$ かつ $z' = \langle d', e'\rangle$ かつ $d \sqsubseteq_{\mathbf{D}} d'$ かつ $e \sqsubseteq_{\mathbf{E}} e'$.

厳格積の場合も cpo である事や代数性・整合完備性といった性質を保存する．関連する基本演算は — 直積構成の場合と同様に連続なだけでなく — 厳格でもある．即ち，

事実 5.22（厳格積 cpo）

(1) 上で定義した半順序集合 $\mathbf{D} \otimes \mathbf{E}$ は $\bot_{\mathbf{D}\otimes\mathbf{E}}$ を最小要素とする cpo である．

(2) $\mathbf{D}, \mathbf{E}$ が共に代数的（もしくは整合完備）ならば $\mathbf{D} \otimes \mathbf{E}$ も同様である．

(3) 厳格積 cpo $\mathbf{D} \otimes \mathbf{E}$ の各成分への射影演算

$$\_ \bullet 1 : |\mathbf{D} \otimes \mathbf{E}| \to D : \left\{ \begin{array}{l} \bot_{\mathbf{D}\otimes\mathbf{E}} \mapsto \bot_{\mathbf{D}}, \\ \langle d, e\rangle \mapsto d \end{array} \right\},$$

$$\_ \bullet 2 : |\mathbf{D} \otimes \mathbf{E}| \to E : \left\{ \begin{array}{l} \bot_{\mathbf{D}\otimes\mathbf{E}} \mapsto \bot_{\mathbf{E}}, \\ \langle d, e\rangle \mapsto e \end{array} \right\}$$

は共に厳格な連続関数である．

(4) 厳格対化演算

$$\langle \_, \_ \rangle_{\otimes} : |\mathbf{D}| \times |\mathbf{E}| \to |\mathbf{D} \otimes \mathbf{E}| : \left\{ \begin{array}{ll} \bot_{\mathbf{D}}, e \mapsto \bot_{\mathbf{D}\otimes\mathbf{E}} & \\ d, \bot_{\mathbf{E}} \mapsto \bot_{\mathbf{D}\otimes\mathbf{E}} & \\ d, e \mapsto \langle d, e\rangle & (d \neq \bot_{\mathbf{D}},\ e \neq \bot_{\mathbf{E}}) \end{array} \right\}$$

は各引数毎に厳格かつ連続である．

(5) 個々の引数毎に厳格かつ連続な 2 引数関数は，各々の引数の変域 cpo を成分とする厳格積 cpo 上の単一引数関数として厳格かつ連続である．

直積の場合と同様，厳格積も可算個（可算無限個でも可）の cpo に対して一般化する事ができ，$\mathbf{D}_1 \otimes \mathbf{D}_2 \otimes \cdots \otimes \mathbf{D}_n$ とか $\bigotimes_{i\in\omega} \mathbf{D}_i$ という形で表わす．

次に，可算個の cpo の「和」の構成を与える．

定義 5.23（cpo の融合和）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，それらの**融合和**を以下で定まる半順序集合 $\mathbf{D} \oplus \mathbf{E}$ として定義する：

- $|\mathbf{D} \oplus \mathbf{E}| \triangleq \{1\} \times (D \setminus \{\bot_{\mathbf{D}}\}) \cup \{2\} \times (E \setminus \{\bot_{\mathbf{E}}\}) \cup \{\bot_{\mathbf{D}\oplus\mathbf{E}}\}$;
- $z \sqsubseteq_{\mathbf{D}\oplus\mathbf{E}} z' \overset{\text{def}}{\iff} z = \bot_{\mathbf{D}\oplus\mathbf{E}}$
  または $z = \langle 1, d\rangle$ かつ $z' = \langle 1, d'\rangle$ かつ $d \sqsubseteq_{\mathbf{D}} d'$
  または $z = \langle 2, e\rangle$ かつ $z' = \langle 2, e'\rangle$ かつ $e \sqsubseteq_{\mathbf{E}} e'$.

事実 5.24（融合和 cpo）

(1) 上で定義した半順序集合 $\mathbf{D} \oplus \mathbf{E}$ は $\bot_{\mathbf{D}\oplus\mathbf{E}}$ を最小要素とする cpo である.

(2) $\mathbf{D}, \mathbf{E}$ が共に代数的（もしくは整合完備）ならば $\mathbf{D} \oplus \mathbf{E}$ も同様である.

(3) 融合和 cpo $\mathbf{D} \oplus \mathbf{E}$ の成分判別演算

$$is_1 : |\mathbf{D} \oplus \mathbf{E}| \to |\mathbf{T}| : \left\{ \begin{array}{l} \bot_{\mathbf{D}\oplus\mathbf{E}} \mapsto \bot_{\mathbf{T}}, \\ \langle 1, d\rangle \mapsto \mathit{true}, \\ \langle 2, e\rangle \mapsto \mathit{false}, \end{array} \right\},$$

$$is_2 : |\mathbf{D} \oplus \mathbf{E}| \to |\mathbf{T}| : \left\{ \begin{array}{l} \bot_{\mathbf{D}\oplus\mathbf{E}} \mapsto \bot_{\mathbf{T}}, \\ \langle 1, d\rangle \mapsto \mathit{false}, \\ \langle 2, e\rangle \mapsto \mathit{true} \end{array} \right\}$$

は共に厳格な連続関数である.

(4) 融合和 cpo $\mathbf{D} \oplus \mathbf{E}$ への各成分からの入射演算

$$in_1 : D \to |\mathbf{D} \oplus \mathbf{E}| : \left\{ \begin{array}{l} \bot_{\mathbf{D}} \mapsto \bot_{\mathbf{D}\oplus\mathbf{E}}, \\ \quad d \mapsto \langle 1, d\rangle \quad (d \neq \bot_{\mathbf{D}}) \end{array} \right\},$$

$$in_2 : E \to |\mathbf{D} \oplus \mathbf{E}| : \left\{ \begin{array}{l} \bot_{\mathbf{E}} \mapsto \bot_{\mathbf{D}\oplus\mathbf{E}}, \\ \quad e \mapsto \langle 2, e\rangle \quad (e \neq \bot_{\mathbf{E}}) \end{array} \right\}$$

は共に厳格な連続関数である.

(5) 融合和 cpo $\mathbf{D} \oplus \mathbf{E}$ の各成分への射影演算

$$out_1 : |\mathbf{D} \oplus \mathbf{E}| \to D : \left\{ \begin{array}{l} \bot_{\mathbf{D}\oplus\mathbf{E}} \mapsto \bot_{\mathbf{D}}, \\ \langle 1, d\rangle \mapsto d, \\ \langle 2, e\rangle \mapsto \bot_{\mathbf{D}} \end{array} \right\},$$

$$out_2 : |\mathbf{D} \oplus \mathbf{E}| \to E : \left\{ \begin{array}{l} \bot_{\mathbf{D}\oplus\mathbf{E}} \mapsto \bot_{\mathbf{E}}, \\ \langle 1, d\rangle \mapsto \bot_{\mathbf{E}} \\ \langle 2, e\rangle \mapsto e \end{array} \right\}$$

は共に厳格な連続関数である.

以上の融合和の定義は可算個（可算無限個でも可）の cpo に対して一般化する事ができ，$\mathbf{D}_1 \oplus \mathbf{D}_2 \oplus \cdots \oplus \mathbf{D}_n$ とか $\bigoplus_{i\in\omega} \mathbf{D}_i$ という形で表わす．

なお直積構成 “$\times$” ・厳格積構成 “$\otimes$” ・融合和構成 “$\oplus$” は何れも結合則を満たす．例えば，$\mathbf{C}\times\mathbf{D}\times\mathbf{E}$ と $(\mathbf{C}\times\mathbf{D})\times\mathbf{E}$ と $\mathbf{C}\times(\mathbf{D}\times\mathbf{E})$ とは cpo として同型である（$\otimes$ や $\oplus$ の場合も同様）．以下，読み易さや判り易さの為に以上の構成に伴う基本演算子名でどの成分 cpo かを指定している “1”, “2” 等の代わりに具体的な cpo 名を用いる — 例えば “$in_{\mathbf{E}}$” — 事がある．

次の構成方法は最小要素を新たに追加する構成であり，自明な — 各要素間にそれ自身以外と順序関係がない — 半順序集合としての単なる集合を平坦 cpo にする事にも使用できる．これにより変数の集合 **Var** 等の字句クラスも cpo 化可能となる．

定義 5.25 （底上げ）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を半順序集合とする時，その**底上げ** $\mathbf{D}_\perp$ を次の様に定義する：

- $|\mathbf{D}_\perp| \triangleq \{0\} \times D \cup \{\perp_{\mathbf{D}_\perp}\}$;
- $z \sqsubseteq_{\mathbf{D}_\perp} z' \overset{\text{def}}{\iff} z = \perp_{\mathbf{D}_\perp}$
  または $z = \langle 0, d\rangle$ かつ $z' = \langle 0, d'\rangle$ かつ $d \sqsubseteq_{\mathbf{D}} d'$.

事実 5.26 （底上げ cpo）

(1) 集合 S を $\mathbf{S} = (S, Id_S)$ なる半順序集合（ここで Id_S は集合 S 上の恒等関係）と看做した時，その底上げ $\mathbf{S}_\perp$ は平坦 cpo である．

(2) $\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，その底上げ $\mathbf{D}_\perp$ も cpo である．

(3) $\mathbf{D}$ が代数的（もしくは整合完備）ならば $\mathbf{D}_\perp$ も同様である．

(4) 底上げ演算

$$up : |\mathbf{D}| \to |\mathbf{D}_\perp| : d \mapsto \langle 0, d\rangle$$

は連続関数である．

(5) 上げ底除去演算

$$down : |\mathbf{D}_\perp| \to |\mathbf{D}| : \left\{ \begin{array}{l} \perp_{\mathbf{D}_\perp} \mapsto \perp_{\mathbf{D}}, \\ \langle 0, d\rangle \mapsto d \end{array} \right\}$$

は連続関数である．

Cpo の間の連続関数の集まりは，それ自体で一つの cpo をなす．即ち，

定義 5.27 （cpo 間の連続関数空間）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，それらの間の**連続関数空間**を以下で定まる半順序集合 $[\mathbf{D} \to \mathbf{E}]$ として定義する：

- $|[\mathbf{D} \to \mathbf{E}]| \triangleq \{f : D \to E \mid f \text{ は連続}\}$;
- $f \sqsubseteq_{[\mathbf{D}\to\mathbf{E}]} f' \overset{\text{def}}{\iff} \forall d \in D.\, \big[f(d) \sqsubseteq_{\mathbf{E}} f'(d)\big]$.

以後，「関数 $f : |\mathbf{D}| \to |\mathbf{E}|$ は連続」の代わりに “$f \in |[\mathbf{D} \to \mathbf{E}]|$” （或いは更に省略して “$f : [\mathbf{D} \to \mathbf{E}]$”）と書く場合がある．

事実 5.28 （連続関数空間 cpo）

(1) 上で定義した半順序集合 $[\mathbf{D} \to \mathbf{E}]$ は $\bot_{[\mathbf{D}\to\mathbf{E}]} \triangleq \boldsymbol{\lambda}\, d \in D.\bot_{\mathbf{E}}$ を最小要素とする cpo である．

(2) $\mathbf{D}, \mathbf{E}$ が共に**領域**ならば $[\mathbf{D} \to \mathbf{E}]$ も領域である．

(3) $\mathbf{C}, \mathbf{D}, \mathbf{E}$ を cpo とし，$f : [(\mathbf{C} \times \mathbf{D}) \to \mathbf{E}]$ とする時，その抽象化（Curry 化）$\hat{f}$ を

$$\hat{f} \triangleq \boldsymbol{\lambda}\, c \in |\mathbf{C}|.\big(\boldsymbol{\lambda}\, d \in |\mathbf{D}|.f(c,d)\big)$$

と定める時，

(a) 任意の $c \in |\mathbf{C}|$ に対して，$\hat{f}(c) = \boldsymbol{\lambda}\, d \in |\mathbf{D}|.f(c,d)$ は連続関数である．即ち，$\hat{f}(c) : [\mathbf{D} \to \mathbf{E}]$；

(b) $\hat{f}$ は連続関数である．即ち，$\hat{f} : [\mathbf{C} \to [\mathbf{D} \to \mathbf{E}]]$；

(c) 関数抽象化演算

$$abs \triangleq \boldsymbol{\lambda}\, f.\hat{f} : |[(\mathbf{C} \times \mathbf{D}) \to \mathbf{E}]| \to |[\mathbf{C} \to [\mathbf{D} \to \mathbf{E}]]|$$

は連続である．

(4) 関数適用演算

$$app : |[\mathbf{D} \to \mathbf{E}]| \times |\mathbf{D}| \to |\mathbf{E}| : f, d \mapsto f(d)$$

は各引数に関して連続である（ので，直積 cpo $[\mathbf{D} \to \mathbf{E}] \times \mathbf{D}$ 上の連続関数である）．

上に纏めた連続関数空間 cpo のポイントは，成分 cpo の $\mathbf{D},\mathbf{E}$ が共に領域（ω-代数的**かつ**整合完備）な場合に $[\mathbf{D}\to\mathbf{E}]$ も領域となる，という点で，直積・厳格積・融合和での場合の様に代数性という性質が単独で保存される訳ではない，という事である．その為に領域という概念を導入しておく必要があった．

まず，連続関数空間 cpo のコンパクトな要素を特徴付けておく．

事実 5.29（コンパクトな連続関数）

$\mathbf{D},\mathbf{E}$ を領域とする時，その間の連続関数空間 cpo の任意のコンパクトな要素 $f\in\mathcal{K}([\mathbf{D}\to\mathbf{E}])$ は以下の形の有限な段数の階段関数として表わせる：

$$f=\boldsymbol{\lambda}d\in|\mathbf{D}|.\begin{cases}\bot_{\mathbf{E}}, & (d\sqsubset_{\mathbf{D}} d_0)\\ e_0, & (d_0\sqsubseteq_{\mathbf{D}} d\sqsubset_{\mathbf{D}} d_1)\\ \cdots & \\ e_i, & (d_i\sqsubseteq_{\mathbf{D}} d\sqsubset_{\mathbf{D}} d_{i+1})\\ \cdots & \\ e_n. & (d_n\sqsubseteq_{\mathbf{D}} d)\end{cases}$$

ここで，$n\geq 0$ であり，また，$d_0,\ldots,d_n\in\mathcal{K}(\mathbf{D})$ かつ $d_0\sqsubset_{\mathbf{D}}\cdots\sqsubset_{\mathbf{D}} d_n$，および，$e_0,\ldots,e_n\in\mathcal{K}(\mathbf{E})$ かつ $e_0\sqsubset_{\mathbf{E}}\cdots\sqsubset_{\mathbf{E}} e_n$ とする．

或る cpo $\mathbf{D}$ からそれ自身への任意の連続関数は最小不動点を持ち，連続関数に対してその最小不動点を与える最小不動点演算子自身も連続関数である．即ち，

事実 5.30（最小不動点定理）

$\mathbf{D}$ を cpo とする．この時，

(1) 任意の $f:[\mathbf{D}\to\mathbf{D}]$ は $|\mathbf{D}|$ 中に不動点を持つ．

(2) 任意の関数 $f:[\mathbf{D}\to\mathbf{D}]$ に対してその最小不動点を与える最小不動点演算子 $\mathit{fix}:|[\mathbf{D}\to\mathbf{D}]|\to|\mathbf{D}|$ が存在する．

(3) fix は連続である．即ち，$\mathit{fix}:[[\mathbf{D}\to\mathbf{D}]\to\mathbf{D}]$.

(4) $\mathit{fix}=\boldsymbol{\lambda}f\in|[\mathbf{D}\to\mathbf{D}]|.\bigsqcup_{i\in\omega} f^i(\bot_{\mathbf{D}})$.

領域間の連続関数として厳格な — 即ち，最小要素 $\bot$ を保存する — ものに限定した場合にも，その集まりは領域を成す．即ち，

定義 5.31 （cpo 間の厳格連続関数空間）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$, $\mathbf{E} = (E, \sqsubseteq_{\mathbf{E}})$ を cpo とする時，それらの間の**厳格連続関数空間**を以下で定まる半順序集合 $[\mathbf{D} \to_{\perp} \mathbf{E}]$ として定義する：

- $|[\mathbf{D} \to_{\perp} \mathbf{E}]| \triangleq \{f : [\mathbf{D} \to \mathbf{E}] \mid f(\perp_{\mathbf{D}}) = \perp_{\mathbf{E}}\}$;
- $f \sqsubseteq_{[\mathbf{D} \to_{\perp} \mathbf{E}]} f' \overset{\text{def}}{\iff} \forall d \in D.\, \big[f(d) \sqsubseteq_{\mathbf{E}} f'(d)\big]$.

事実 5.32 （厳格連続関数空間 cpo）

(1) 上で定義した半順序集合 $[\mathbf{D} \to_{\perp} \mathbf{E}]$ は $\perp_{[\mathbf{D} \to_{\perp} \mathbf{E}]} \triangleq \boldsymbol{\lambda} d \in D.\perp_{\mathbf{E}}$ を最小要素とする cpo である．

(2) $\mathbf{D}, \mathbf{E}$ が共に領域ならば $[\mathbf{D} \to_{\perp} \mathbf{E}]$ も領域である．

(3) $\mathbf{C}, \mathbf{D}, \mathbf{E}$ を cpo とし，$f : [(\mathbf{C} \otimes \mathbf{D}) \to_{\perp} \mathbf{E}]$ とする時，その抽象化（Curry 化）$\hat{f}$ を

$$\hat{f} \triangleq \boldsymbol{\lambda}_{\perp} c \in |\mathbf{C}|.\big(\boldsymbol{\lambda}_{\perp} d \in |\mathbf{D}|.f(c,d)\big)$$

と定める — ここで，“$\boldsymbol{\lambda}_{\perp}$” は厳格な関数抽象化を表わす — 時，

(a) 任意の $c \in |\mathbf{C}|$ に対して，$\hat{f}(c) = \boldsymbol{\lambda}_{\perp} d \in |\mathbf{D}|.f(c,d)$ は厳格な連続関数である．即ち，$\hat{f}(c) : [\mathbf{D} \to_{\perp} \mathbf{E}]$；

(b) $\hat{f}$ は厳格な連続関数である．即ち，$\hat{f} : [\mathbf{C} \to_{\perp} [\mathbf{D} \to_{\perp} \mathbf{E}]]$；

(c) 厳格関数抽象化演算

$$abs_{\perp} \triangleq \boldsymbol{\lambda}_{\perp} f.\hat{f} : |[(\mathbf{C} \otimes \mathbf{D}) \to_{\perp} \mathbf{E}]| \to |[\mathbf{C} \to_{\perp} [\mathbf{D} \to_{\perp} \mathbf{E}]]|$$

は厳格かつ連続である．

(4) 関数適用演算

$$app_{\perp} : |[\mathbf{D} \to_{\perp} \mathbf{E}]| \times |\mathbf{D}| \to |\mathbf{E}| : f, d \mapsto f(d)$$

は各引数に関して厳格かつ連続な関数である（ので，厳格積 cpo $[\mathbf{D} \to_{\perp} \mathbf{E}] \otimes \mathbf{D}$ 上の厳格な連続関数である）．

(5) $\mathbf{D}, \mathbf{E}$ を領域とする時，その間の厳格連続関数空間 cpo の任意のコンパクトな要素 $f \in \mathcal{K}([\mathbf{D} \to_{\perp} \mathbf{E}])$ は事実 5.29 に示した有限な段数の階段関数に厳格性の為の $d_0 \neq \perp_{\mathbf{D}}$ という条件を追加した（或いは “$\boldsymbol{\lambda}$” の代わりに “$\boldsymbol{\lambda}_{\perp}$” として）ものとして表わせる．

厳格な連続関数に関しても最小不動点定理が成立する．即ち，

事実 5.33（最小不動点定理 — 厳格版）

$\mathbf{D}$ を cpo とする．この時，

(1) 任意の $f : [\mathbf{D} \to_{\perp} \mathbf{D}]$ は $|\mathbf{D}|$ 中に不動点を持つ．

(2) 任意の関数 $f : [\mathbf{D} \to_{\perp} \mathbf{D}]$ に対してその最小不動点を与える最小不動点演算子 $\mathit{fix} : |[\mathbf{D} \to_{\perp} \mathbf{D}]| \to |\mathbf{D}|$ が存在する．

(3) fix は連続かつ厳格である．即ち，$\mathit{fix} : [[\mathbf{D} \to_{\perp} \mathbf{D}] \to_{\perp} \mathbf{D}]$.

(4) $\mathit{fix} = \boldsymbol{\lambda} f \in |[\mathbf{D} \to_{\perp} \mathbf{D}]|. \bigsqcup_{i \in \omega} f^i(\perp_{\mathbf{D}})$.

■ 牽縮射・射影対

以下の議論や次節での 2 階の型付き λ-計算に対する領域論的な意味定義の略史での使用の為に，cpo 上で冪等に振舞う関数のクラスを表わす用語を導入しておく．

定義 5.34（牽縮）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とし，$r : [\mathbf{D} \to \mathbf{D}]$ とする．この時，

(1) r が**牽縮射**であるとは，r が冪等な写像だという事である．即ち，

$$r = r \circ r.$$

(2) D の部分集合 D' が D の**牽縮**であるとは，D' が或る牽縮射による D の像となっている事である．即ち，

$$\exists \text{ 牽縮射 } r : [\mathbf{D} \to \mathbf{D}]. \big[D' = r(D)\big].$$

(3) (2) での牽縮 D' の事を，それを像とする牽縮射 r を用いて $|r|$ で表わす場合がある．

牽縮射の中でも連続関数空間 $[\mathbf{D} \to \mathbf{D}]$ 上の順序 $\sqsubseteq_{[\mathbf{D} \to \mathbf{D}]}$ に関して恒等射 $id_{\mathbf{D}}$ 以下もしくは以上である下記の各クラスのものは様々な面で重要である．

定義 5.35 （射影・閉包）

$\mathbf{D}$ を cpo とし， r をその上の牽縮射とする時,

(1) r が**射影**であるとは， $r \sqsubseteq_{[\mathbf{D}\to\mathbf{D}]} id_{\mathbf{D}}$ という事である.

(2) r が**閉包射**（もしくは単に**閉包**）であるとは， $id_{\mathbf{D}} \sqsubseteq_{[\mathbf{D}\to\mathbf{D}]} r$ という事である.

Cpo $\mathbf{D}$ 上の牽縮射 r の像 $r(|\mathbf{D}|)$ が領域となる場合は特に重要である．その条件を満たす牽縮射（射影・閉包）は，本節の最後に略史として纏めている通り， 2 階の型付き λ-計算に対して領域論的な意味論を構築する上で何度も活用されて来た.

定義 5.36 （有限性）

領域 $\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ 上の牽縮射 r が**有限的 (finitary)** であるとは, $(r(D), \sqsubseteq_{\mathbf{D}} \cap (r(D) \times r(D)))$ が領域となっている事である.

射影の概念は異なる cpo 間の関数の対へと一般化できる．その一般化された射影対の概念は次項の再帰的領域方程式の解法で基本的な役割を果たす.

定義 5.37 （射影対）

$\mathbf{D}, \mathbf{E}$ を cpo とし， $\varphi : [\mathbf{D} \to \mathbf{E}]$, $\psi : [\mathbf{E} \to \mathbf{D}]$ とする時，対 (φ, ψ) が $\mathbf{D}$ と $\mathbf{E}$ との間の**射影対**（或いはより正確には**埋め込み-射影対**）であるとは，以下の 2 条件を満たしている事である：

(1) $\psi \circ \varphi = id_{\mathbf{D}}$;

(2) $\varphi \circ \psi \sqsubseteq_{[\mathbf{E}\to\mathbf{E}]} id_{\mathbf{E}}$.

この時， φ を $\mathbf{D}$ の $\mathbf{E}$ 中への**埋め込み**， ψ を $\mathbf{E}$ の $\mathbf{D}$ 上への**射影**，と呼ぶ.

事実 5.38（射影対）

定義 5.37 の射影対を成す埋め込みと射影とは互いに相手を一意的に定める．即ち，$\mathbf{D}, \mathbf{E}$ を cpo とする時，

(1) $\varphi : [\mathbf{D} \to \mathbf{E}]$ が与えられた時，定義 5.37 の (1)，(2) を共に満たす $\psi : [\mathbf{E} \to \mathbf{D}]$ は存在するとすれば唯一である．故に，与えられた φ に対して φ を埋め込みとして射影対を成す射影 ψ（が存在する場合，その ψ）を φ^{R} で表わす事がある．

(2) $\psi : [\mathbf{E} \to \mathbf{D}]$ が与えられた時，定義 5.37 の (1)，(2) を共に満たす $\varphi : [\mathbf{D} \to \mathbf{E}]$ は存在するとすれば唯一である．故に，与えられた ψ に対して ψ を射影として射影対を成す埋め込み φ（が存在する場合，その φ）を ψ^{L} で表わす事がある．

射影対は合成する事ができる．即ち，

定義 5.39（射影対の合成）

$\mathbf{C}, \mathbf{D}, \mathbf{E}$ を cpo とし，$(\varphi, \psi), (\varphi', \psi')$ を，各々，$\mathbf{C}, \mathbf{D}$ 間および $\mathbf{D}, \mathbf{E}$ 間の射影対とする．この時，これら二つの射影対の合成 $(\varphi', \psi') \circ (\varphi, \psi)$ を以下で定まる対として定義する：

$$(\varphi', \psi') \circ (\varphi, \psi) \triangleq (\varphi' \circ \varphi, \psi \circ \psi')$$

当然ながら以上の二つの射影対の合成は射影対である．即ち，

事実 5.40（合成射影対）

上で定義された射影対の合成 $(\varphi', \psi') \circ (\varphi, \psi)$ は $\mathbf{C}, \mathbf{E}$ 間の射影対である．

以下，前に挙げた cpo 構成子の各々に対してその構造を持つ複合 cpo 同士の間の射影対を与える射影対構成子を定義する．

最初は射影対同士の直積である．

定義 5.41 （射影対の直積）

$\mathbf{D},\mathbf{E},\mathbf{D}',\mathbf{E}'$ を cpo とし，(φ,ψ) と (φ',ψ') とを，各々，$\mathbf{D},\mathbf{E}$ 間および $\mathbf{D}',\mathbf{E}'$ 間の射影対とする時，両者の直積 $(\varphi,\psi)\times(\varphi',\psi')$ を以下の様に定義する：

$$(\varphi,\psi)\times(\varphi',\psi') \triangleq (\boldsymbol{\lambda}\, x \in |\mathbf{D}\times\mathbf{D}'|.\langle\varphi(x.1),\varphi'(x.2)\rangle,\\ \boldsymbol{\lambda}\, y \in |\mathbf{E}\times\mathbf{E}'|.\langle\psi(y.1),\psi'(y.2)\rangle\ \ ).$$

射影対同士の直積は直積 cpo 間の射影対となっている．即ち，

事実 5.42 （直積射影対）

上で定義された射影対の直積 $(\varphi,\psi)\times(\varphi',\psi')$ は二つの直積 cpo $\mathbf{D}\times\mathbf{D}'$ と $\mathbf{E}\times\mathbf{E}'$ との間の射影対である．

次は射影対同士の厳格積である．

定義 5.43 （射影対の厳格積）

$\mathbf{D},\mathbf{E},\mathbf{D}',\mathbf{E}'$ を cpo とし，(φ,ψ) と (φ',ψ') とを，各々，$\mathbf{D},\mathbf{E}$ 間および $\mathbf{D}',\mathbf{E}'$ 間の射影対とする時，両者の厳格積 $(\varphi,\psi)\otimes(\varphi',\psi')$ を以下の様に定義する：

$$(\varphi,\psi)\otimes(\varphi',\psi') \triangleq (\boldsymbol{\lambda}_{\perp}\, x \in |\mathbf{D}\otimes\mathbf{D}'|.\langle\varphi(x\bullet 1),\varphi'(x\bullet 2)\rangle_{\otimes},\\ \boldsymbol{\lambda}_{\perp}\, y \in |\mathbf{E}\otimes\mathbf{E}'|.\langle\psi(y\bullet 1),\psi'(y\bullet 2)\rangle_{\otimes}\ \ ).$$

射影対同士の厳格積の場合も厳格積 cpo 間の射影対となっている．即ち，

事実 5.44 （厳格積射影対）

上で定義された射影対の厳格積 $(\varphi,\psi)\otimes(\varphi',\psi')$ は二つの厳格積 cpo $\mathbf{D}\otimes\mathbf{D}'$ と $\mathbf{E}\otimes\mathbf{E}'$ との間の射影対である．

射影対の融合和を作る事も可能である．

定義 5.45（射影対の融合和）

$\mathbf{D},\mathbf{E},\mathbf{D}',\mathbf{E}'$ を cpo とし，(φ,ψ) と (φ',ψ') とを，各々，$\mathbf{D},\mathbf{E}$ 間および $\mathbf{D}',\mathbf{E}'$ 間の射影対とする時，両者の融合和 $(\varphi,\psi)\oplus(\varphi',\psi')$ を以下の様に定義する：

$$(\varphi,\psi)\oplus(\varphi',\psi') \triangleq (\boldsymbol{\lambda}_{\perp} x \in |\mathbf{D}\oplus\mathbf{D}'|.\ \mathsf{if}\ is_{\mathbf{D}}(x)\ \mathsf{then}\ in_{\mathbf{E}}(\varphi(out_{\mathbf{D}}(x)))\ \mathsf{else}\ in_{\mathbf{E}'}(\varphi'(out_{\mathbf{D}'}(x))),\ \boldsymbol{\lambda}_{\perp} y \in |\mathbf{E}\oplus\mathbf{E}'|.\ \mathsf{if}\ is_{\mathbf{E}}(y)\ \mathsf{then}\ in_{\mathbf{D}}(\psi(out_{\mathbf{E}}(y)))\ \mathsf{else}\ in_{\mathbf{D}'}(\psi'(out_{\mathbf{E}'}(y)))).$$

射影対の融合和の場合も融合和 cpo 間の射影対となっている．

事実 5.46（融合和射影対）

上で定義された射影対の直積 $(\varphi,\psi)\oplus(\varphi',\psi')$ は二つの融合和 cpo $\mathbf{D}\oplus\mathbf{D}'$ と $\mathbf{E}\oplus\mathbf{E}'$ との間の射影対である．

次は射影対の連続関数空間構成である．

定義 5.47（射影対の連続関数空間構成）

$\mathbf{D},\mathbf{E},\mathbf{D}',\mathbf{E}'$ を cpo とし，(φ,ψ) と (φ',ψ') とを，各々，$\mathbf{D},\mathbf{E}$ 間および $\mathbf{D}',\mathbf{E}'$ 間の射影対とする時，両者の連続関数空間構成 $[(\varphi,\psi)\to(\varphi',\psi')]$ を以下の様に定義する：

$$[(\varphi,\psi)\to(\varphi',\psi')] \triangleq (\boldsymbol{\lambda}\, f \in |[\mathbf{D}\to\mathbf{D}']|.\varphi'\circ f\circ\psi, \boldsymbol{\lambda}\, g \in |[\mathbf{E}\to\mathbf{E}']|.\psi'\circ g\circ\varphi).$$

この場合も，構成された対は連続関数空間 cpo 間の射影対となる．

事実 5.48（連続関数空間射影対）

上で定義された射影対の連続関数空間構成 $[(\varphi,\psi)\to(\varphi',\psi')]$ は二つの連続関数空間 cpo $[\mathbf{D}\to\mathbf{D}']$ と $[\mathbf{E}\to\mathbf{E}']$ との間の射影対である．

射影対の厳格連続関数空間構成も同様に行なえる.

定義 5.49 （射影対の厳格連続関数空間構成）

$\mathbf{D},\mathbf{E},\mathbf{D}',\mathbf{E}'$ を cpo とし，(φ,ψ) と (φ',ψ') とを，各々，$\mathbf{D},\mathbf{E}$ 間と $\mathbf{D}',\mathbf{E}'$ 間との射影対とする時，両者の厳格連続関数空間構成 $[(\varphi,\psi)\to_\perp(\varphi',\psi')]$ を以下の様に定義する：

$$[(\varphi,\psi)\to_\perp(\varphi',\psi')] \triangleq (\boldsymbol{\lambda}_\perp f\in|[\mathbf{D}\to_\perp\mathbf{D}']|.\varphi'\circ f\circ\psi, \boldsymbol{\lambda}_\perp g\in|[\mathbf{E}\to_\perp\mathbf{E}']|.\psi'\circ g\circ\varphi).$$

事実 5.50 （厳格連続関数空間射影対）

上で定義された射影対の厳格連続関数空間構成 $[(\varphi,\psi)\to_\perp(\varphi',\psi')]$ は二つの厳格連続関数空間 cpo $[\mathbf{D}\to_\perp\mathbf{D}']$ と $[\mathbf{E}\to_\perp\mathbf{E}']$ との間の射影対である.

射影対の構成の最後として，射影対に対する底上げは以下の通りである.

定義 5.51 （射影対の底上げ）

$\mathbf{D},\mathbf{E}$ を cpo とし，(φ,ψ) を両者の間の射影対とする時，(φ,ψ) の底上げ $(\varphi,\psi)_\perp$ を以下の様に定義する：

$$(\varphi,\psi)_\perp \triangleq (\boldsymbol{\lambda}_\perp x\in|\mathbf{D}_\perp|.up(\varphi(down(x))), \boldsymbol{\lambda}_\perp y\in|\mathbf{E}_\perp|.up(\psi(down(y))))$$

構成された対は底上げ cpo 間の射影対となっている.

事実 5.52 （底上げ射影対）

上で定義された射影対の底上げ $(\varphi,\psi)_\perp$ は二つの底上げ cpo $\mathbf{D}_\perp$ と $\mathbf{E}_\perp$ との間の射影対である.

次は射影対構成子ではないが，平坦 cpo $\mathbf{T}$, $\mathbf{N}$ 等の既知の cpo を表わす名前 — 以下では **cpo 定数**と呼ぶ — に対して常に存在する事が保証されている自明な射影対である．

定義 5.53（自明な射影対）

$\mathbf{A}$ を既知の cpo を表わす cpo 定数とする時，$\mathbf{A}$ から $\mathbf{A}$ 自身への自明な射影対 $(\varphi_{\mathbf{A}}, \psi_{\mathbf{A}})$ を

$$(\varphi_{\mathbf{A}}, \psi_{\mathbf{A}}) \triangleq (id_{\mathbf{A}}, id_{\mathbf{A}})$$

と定める．

以上の準備から，次の一般的な補題を示す事が可能となる．

補題 5.54（射影対に関する主補題）

$F(\mathbf{X})$ を，cpo を値とするメタ変数 — 以下，**cpo 変数**と呼ぶ — として唯一のメタ変数 $\mathbf{X}$ を含み，cpo 定数 $\mathbf{A}_1, \ldots, \mathbf{A}_n$ $(n \geq 0)$ とこれまでに挙げた cpo 構成子 $\_ \times \_$, $\_ \otimes \_$, $\_ \oplus \_$, $[\_ \to \_]$, $[\_ \to_{\perp} \_]$, $(\cdot)_{\perp}$ を用いて構成される cpo を表わす任意の表式 — 以下，**cpo 式**と呼ぶ — とする時，$F(\mathbf{X})$ に対応して射影対を表わす式 F^{epp} を以下の様に定義する：

(1) cpo 変数 $\mathbf{X}$ の出現箇所は関数を表わすメタ変数 e, p の対 (e, p) で置き換える；
(2) 各 cpo 定数 $\mathbf{A}_i$ $(1 \leq i \leq n)$ は対応する自明な射影対 $(\varphi_{\mathbf{A}_i}, \psi_{\mathbf{A}_i}) = (id_{\mathbf{A}_i}, id_{\mathbf{A}_i})$ で置き換える；
(3) 各 cpo 構成子は対応する射影対構成子と看做す；
(4) (1) で導入した関数対 (e, p) に関して $\boldsymbol{\lambda}$-抽象化する．

即ち，

$$F^{epp} \triangleq \boldsymbol{\lambda}(e,p).\big((F(\mathbf{X}))[(e,p)/\mathbf{X}, (id_{\mathbf{A}_1}, id_{\mathbf{A}_1})/\mathbf{A}_1, \ldots, (id_{\mathbf{A}_n}, id_{\mathbf{A}_n})/\mathbf{A}_n]\big).$$

この時，$\mathbf{D}, \mathbf{E}$ が cpo で (φ, ψ) が両者の間の射影対ならば，$F^{epp}(\varphi, \psi)$ は $F(\mathbf{D}), F(\mathbf{E})$ 間の射影対である．

証明

Cpo 式 $F(\mathbf{X})$ の構造に関する帰納法で示せば良い.

証明終

この補題のお蔭で，最初の種としての射影対 (φ, ψ) さえ用意すれば，如何に複雑な構成の cpo の間にも射影対を与える事が可能となる．そして，この事は次の再帰的領域方程式を逆極限法で解く場合に本質的な役割を果たす.

■ 再帰的領域方程式の逆極限解法

以上に挙げた領域構成子と平坦 cpo と領域を表わす変数を用いた再帰的領域方程式による領域の再帰的定義は必ず解を持ち，その解は Scott の逆極限法によって求められる．その為に，まず射影系と逆極限の概念を定義する.

定義 5.55 （射影系・逆極限）

$\mathbf{D}_0, \mathbf{D}_1, \ldots$ を可算個の cpo の列とし，各 $i \in \omega$ について $f_i : [\mathbf{D}_{i+1} \to \mathbf{D}_i]$ とする．この時,

(1) 列 $(\mathbf{D}_i, f_i)_{i\in\omega}$ を**射影系**と呼ぶ.

(2) 射影系 $(\mathbf{D}_i, f_i)_{i\in\omega}$ の**逆極限**もしくは**射影的極限** — $\varprojlim (\mathbf{D}_i, f_i)$ — は次で定まる半順序集合 $\mathbf{D}_\infty = (D_\infty, \sqsubseteq_{\mathbf{D}_\infty})$ である：

(a) $D_\infty \triangleq \big\{ \langle d_0, d_1, \ldots \rangle \;\big|\; \forall i \in \omega. \big[d_i \in |\mathbf{D}_i| \text{ かつ } f_i(d_{i+1}) = d_i \big] \big\}$;

(b) $\langle \vec{d} \rangle \sqsubseteq_{\mathbf{D}_\infty} \langle \vec{e} \rangle \quad \overset{\text{def}}{\Longleftrightarrow} \quad \forall i \in \omega. \big[d_i \sqsubseteq_{\mathbf{D}_i} e_i \big]$.

以下，“$\sqsubseteq_{\mathbf{D}_\infty}$” の代わりに “$\sqsubseteq_\infty$” と書く.

(3) 以下，D_∞ の要素の無限列 $\langle d_0, d_1, \ldots \rangle$ を写像 $d : \omega \to \bigcup_i |\mathbf{D}_i| : i \mapsto d_i$ と同一視する.

Cpo 列の逆極限はやはり cpo であり，更に列中の各 cpo が全て領域であればその逆極限も領域である．即ち,

事実 5.56（逆極限 cpo）

(1) $(\mathbf{D}_i, f_i)_{i\in\omega}$ を射影系とする時，その逆極限 $\mathbf{D}_\infty = \varprojlim (\mathbf{D}_i, f_i)$ は次で有向集合（または ω-上昇鎖） $V \subseteq |\mathbf{D}_\infty|$ の上限が与えられる cpo である：

$$\bigsqcup^{\mathbf{D}_\infty} V \triangleq \boldsymbol{\lambda} i \in \omega. \bigsqcup^{\mathbf{D}_i} \{d(i) \mid d \in V\}.$$

以下，“$\bigsqcup^{\mathbf{D}_\infty}$” の代わりに “$\bigsqcup^{\infty}$” と書く．

(2) 各 $\mathbf{D}_i$ $(i \in \omega)$ が領域であれば $\mathbf{D}_\infty = (D_\infty, \sqsubseteq_\infty)$ も領域である．

以下，再帰的領域方程式 $\mathbf{D} \cong F(\mathbf{D})$ の逆極限法による解 — ここで「解」とは両辺を同型にする領域 — の構成の要点を示す．ここで F は補題 5.54 で述べた形の cpo を表わす式であるが，cpo 変数としては補題での $\mathbf{X}$ の代わりに従来の慣習に従い $\mathbf{D}$ を用いる．

逆極限法による解の構成は，$\mathbf{D}_0$ という初期領域を与え，それを元に $\mathbf{D}_1, \mathbf{D}_2, \ldots$ という領域の列と各 $\mathbf{D}_i, \mathbf{D}_{i+1}$ 間の射影対 (φ_i, ψ_i) を同時に組み上げて，最終的に射影系 $(\mathbf{D}_i, \psi_i)_{i\in\omega}$ の逆極限 $\varprojlim (\mathbf{D}_i, \psi_i)$ としての $\mathbf{D}_\infty \triangleq \prod_{i\in\omega} \mathbf{D}_i$ を得る，という手順で行なわれる．この際，F の形により $\mathbf{D}_0$ として最初に与えるべき領域が異なる．即ち，

$$F(\mathbf{D}) \cong [\mathbf{D} \to \mathbf{D}]$$

の様な場合は初期領域 $\mathbf{D}_0$ としては要素数が 2 以上の領域を与える必要があり，その選び方は — 通常は 2 点 cpo $\mathbf{0} = \{\bot, \top\}$ を選ぶが — 或る意味では恣意的である[2)]．その理由は，$\mathbf{D}_0$ に最も単純な cpo である 1 点 cpo $\{\bot\}$ を選ぶと全ての $\mathbf{D}_i$ が 1 点 cpo のままで留まり自明な解しか得られないからである．一方，

$$F(\mathbf{D}) \cong \mathbf{T} \oplus [\mathbf{D} \to \mathbf{D}]$$

の様に，例えば真理値の cpo $\mathbf{T}$ が融合和成分として入っている場合等では，以上の問題は生じない．

2) この点を指して，Abramsky (1990) は「型無し λ-計算の定める領域方程式 $\mathbf{D} \cong [\mathbf{D} \to \mathbf{D}]$ には正準 (canonical) な解は存在しない」と述べている．

本論文の言語 Funiq の意味領域を求める為に次節で与える領域方程式は後者の形であるので，以下，後者の形の F に対する逆極限法による解領域の構成を示す．この部分の記法は主として Wadsworth (1976) に負っている．

まず，初期領域 $\mathbf{D}_0$ は

$$\mathbf{D}_0 \triangleq \{\bot\}$$

と与える．次に， $\mathbf{D}_1$ を

$$\mathbf{D}_1 \triangleq F(\mathbf{D}_0)$$

とし， $\varphi_0 : [\mathbf{D}_0 \to \mathbf{D}_1]$ と $\psi_0 : [\mathbf{D}_1 \to \mathbf{D}_0]$ とを

$$\varphi_0 \triangleq \boldsymbol{\lambda}\, x \in |\mathbf{D}_0|.\bot_{\mathbf{D}_1},$$
$$\psi_0 \triangleq \boldsymbol{\lambda}\, y \in |\mathbf{D}_1|.\bot_{\mathbf{D}_0}$$

と定める時，両者の対 (φ_0, ψ_0) は明らかに $\mathbf{D}_0, \mathbf{D}_1$ 間の射影対である．

以下，各 $i \geq 1$ に対して， $\mathbf{D}_{i+1}$ を

$$\mathbf{D}_{i+1} \triangleq F(\mathbf{D}_i)$$

とし， $\varphi_i : [\mathbf{D}_i \to \mathbf{D}_{i+1}]$ と $\psi_i : [\mathbf{D}_{i+1} \to \mathbf{D}_i]$ とを補題 5.54 の言明中にある F で定まる F^{epp} を用いて，

$$(\varphi_i, \psi_i) \triangleq F^{epp}(\varphi_{i-1}, \psi_{i-1})$$

により定義すると，補題 5.54 から (φ_i, ψ_i) は $\mathbf{D}_i$ と $\mathbf{D}_{i+1}$ との間の射影対である．

すると， $(\mathbf{D}_i, \psi_i)$ は射影系の定義を満たし，従って，元の領域方程式の右辺 $F(\mathbf{D})$ 中の cpo 定数 $\mathbf{A}_i$ の何れもが領域であるならば，事実 5.56 より射影系の逆極限 $\varprojlim (\mathbf{D}_i, \psi_i)$ として

$$D_\infty \triangleq \left\{ \langle d_i \rangle_{i \in \omega} \;\middle|\; \forall i \in \omega.\big[\, d_i = \psi_i(d_{i+1}) \text{ かつ } d_i \in |\mathbf{D}_i| \,\big] \right\},$$
$$\sqsubseteq_\infty \triangleq \boldsymbol{\lambda}\, d, d' \in D_\infty.\Big(\forall i \in \omega.\big[d_i \sqsubseteq_{\mathbf{D}_i} d'_i\big]\Big)$$

から得られる $\mathbf{D}_\infty = (D_\infty, \sqsubseteq_\infty)$ も領域であると保証された事になる．更に，以上の構成から

$$\Phi : \mathbf{D}_\infty \overset{\sim}{\longleftrightarrow} F(\mathbf{D}_\infty) : \Psi$$

という同型性も保証される．同時に，この時，各成分 cpo $\mathbf{D}_i$ $(i \in \omega)$ との間には

$$\varphi_{i\infty} : \mathbf{D}_i \longleftrightarrow \mathbf{D}_\infty : \psi_{\infty i}$$

という射影対が存在する事は明らかである．

本項で示した事は，圏論の言葉で言えば，全ての cpo とそれらの間の射影対は各 cpo を対象とし各射影対を射として一つの圏を成し，再帰的領域方程式はその圏の中で解を持ち，その解は射影的極限により（同時に埋め込みを用いての帰納的極限 $\varinjlim (\mathbf{D}_i, \varphi_i)$ としても）得られる，という事（の主要な部分）である．以上の議論を圏論で書き直せばより洗練され一般化可能な形となるが，本論文では上に示した以上の一般化は必要としないので，圏論を用いた形での提示は行なわない．

■ Cpo 上の述語の完備性

次は cpo の構造— 最小要素や上限等 — を尊重する述語のクラスを定義する．最初のは最小要素を尊重する述語のクラスである．

定義 5.57（ω-述語の厳格性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，$\mathbf{D}$ 上の述語 P が**厳格**であるとは，$\mathbf{D}$ の最小要素に対して P が成立する事である．即ち，

$$P(\bot_{\mathbf{D}}).$$

次は ω-上昇鎖の上限操作を尊重する述語のクラスである．

定義 5.58（ω-帰納性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，$\mathbf{D}$ 上の述語 P が **ω-帰納的**であるとは，P が $\mathbf{D}$ 中の ω-上昇鎖の上限を尊重する事である．即ち，$V = \{d_i\}_{i\in\omega}$ を $\mathbf{D}$ 中の任意の ω-上昇鎖とする時，

$$\big(\forall d \in V.\, P(d)\big) \quad \Longrightarrow \quad P\Big(\bigsqcup\nolimits^{\mathbf{D}} V\Big).$$

なお，ω-帰納的述語は，しばしば**許容的 (admissible) 述語**とも呼ばれるが，ここでは Plotkin (1983) の用語法に従う．

両者を組み合わせれば cpo の基本構造を尊重する述語のクラスを得る．

定義 5.59 （完備性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，$\mathbf{D}$ 上の述語 P が**完備**であるとは，P が厳格かつ ω-帰納的だという事である．

なお，述語の厳格性と完備性の用語は標準的でなく本論文限りのものである．完備な述語は次節での詳細化型に対する解釈を与える上で基本的な役割を果たす．

最後は，代数的 cpo でのコンパクト要素に関する性質を尊重するクラスである．このクラスは Funiq の意味定義では用いないが，後の議論の為に導入しておく．

定義 5.60 （一様性）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，$\mathbf{D}$ 上の述語 P が**一様**であるとは，P が或る要素 d に対して成立すれば d を近似する任意のコンパクトな要素で P が成立するという事である．即ち，

$$\forall d \in |\mathbf{D}|.\Big[P(d) \quad \Longrightarrow \quad \big(\forall e \in \mathcal{K}(d).\, P(e)\big)\Big].$$

以上の述語の「一様性」の用語も一般的なものでなく本論文限りのものであるが，この「一様性」という用語 — 寧ろ cpo との関係で言えば「代数性」の方が相応しいかも知れない — は，次節で述べる部分同値関係の或るクラスを表わすのに Amadio (1991) が用いた “uniform” に由来する．

■ Cpo 上のイデアル

本節および次節では用いないが，後の議論で用いる為に，cpo でのイデアルの概念を定義しておく．

定義 5.61（イデアル）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とする時，$I \subseteq D$ が $\mathbf{D}$ の**イデアル**であるとは，以下を満たす事である：

(1) I は空集合ではない．即ち，

$$I \neq \varnothing;$$

(2) I は下に閉じている．即ち，

$$\forall d \in I. \forall d' \in {\downarrow} d. \big[d' \in I\big];$$

(3) I は ω-上昇列の上限操作に関して閉じている．即ち，

$$\forall \omega\text{-上昇列 } V \subseteq I. \left[\bigsqcup\nolimits^{\mathbf{D}} V \in I\right];$$

(4) I は整合的な 2 要素の上限に関して閉じている．即ち，

$$\forall d_1, d_2 \in I. \big[d_1, d_2 \text{ は } \mathbf{D} \text{ 中で整合的} \implies d_1 \sqcup d_2 \in I\big].$$

Cpo $\mathbf{D}$ 上のイデアル全てのなす集合を $\mathfrak{I}(\mathbf{D})$ で表わす．

任意のイデアルは Scott 位相の意味での閉集合である．即ち，

事実 5.62（イデアルの特徴付け）

$\mathbf{D}$ を cpo とし，$I \subseteq |\mathbf{D}|$ を $\mathbf{D}$ のイデアルとする時，I は Scott 閉である．

更に，cpo 上のイデアル全ての集まりは包含関係 $\subseteq$ によって cpo を成す．即ち，

事実 5.63（イデアルの成す cpo）

$\mathbf{D} = (D, \sqsubseteq_{\mathbf{D}})$ を cpo とし，$\mathfrak{I}(\mathbf{D}) \subseteq \mathfrak{P}D$ を $\mathbf{D}$ 上の全てのイデアルの集合とする時，

(1) $(\mathfrak{I}(\mathbf{D}), \subseteq)$ は cpo である．

(2) そのコンパクトな要素としてのイデアルは，$\mathbf{D}$ のコンパクト要素の有限集合 $E \subseteq \mathcal{K}(\mathbf{D})$ を用いて ${\downarrow} E$ という形で表わせる．

(3) $\mathbf{D}$ が領域である場合でも，$(\mathfrak{I}(\mathbf{D}), \subseteq)$ は領域を成すとは限らない．

■ 2 階の型付き λ-計算に対する領域論的意味の研究略史

プログラミング言語での多相性のモデル化という計算機科学の立場から Girard とは独立に F と同等の体系を発見した Reynolds (1974) では，領域論的な立場からの多相型の解釈と表現独立性を示す事を試み，その過程に於いて，脚注 (1) で述べた多相型の持つ非可述性が領域論的に多相型を解釈する上で具体的にどの様な技術的障害を与えるかを計算機科学の立場から始めて明らかにした．失敗[3)]には終わったが，Donahue (1979) は領域上の牽縮射により多相型を解釈しようと試みた．

計算機科学として多相型に解釈を与える事に始めて成功したのは McCracken (1979) で，領域上の有限的閉包の概念を用いて多相型の意味論を作った．また，Scott (1981) の Exercise 8.27 では領域上の有限的射影を用いて多相型の解釈を構成する事を問うている． McCracken (1984) は領域上の有限的牽縮を用いて多相型に対する新たな領域論的モデルを与えた．

Bruce and Meyer (1984) と Mitchell (1984) とは，独立に 2 階の型付き λ-計算のモデル — 領域論的なモデルには限定しない — が満たすべき一般的な枠組としての “frame” の概念を与えた．それに従った形での最初の具体的な多相型の解釈を与えるモデルとして， Amadio, Bruce and Longo (1986) は Scott (1982) のアイデアに基づき具体的に 2 階の型付き λ-計算のモデルを構成して見せた．なお， Bruce and Meyer の仕事と Mitchell の仕事との最終形は，両者の共同論文の形で Bruce, Meyer and Mitchell (1990) として発表されている．

以上の牽縮（或いはそれの限定されたクラス）を型の表示に用いるアプローチは元を質すと表示的意味論と領域論の確立の上でエポックメイキングな Scott (1976) “Data Types as Lattices” に由来する．その論文では， Scott が型の表示を普遍領域 $\mathbf{P}\boldsymbol{\omega}$ 上の牽縮によって構成する方法を詳細に与えている．但し，この場合の「型」はプログラミング言語上での構文的に指定された型ではなく，意味領域で共通な振舞をする値の集まりというメタ言語レベルでの「型」であり， 2 階の型付き λ-計算の型を牽縮を用いて解釈する以上に挙げた様々な研究は，対象言語上での「型の束[4)]」

3) 失敗の直接の原因は，領域上の全ての牽縮全体としては一つの牽縮（の値域）を成さない事であるが，その根本的な原因は，脚注 (1) で述べた通り，彼の意味定義では多相型の非可述性を見落としていた事にある．

4) ここでの「束」とは束論での「束 (lattice)」という技術用語の意味ではなく，一般用語としての「たば」の意味である．

としての多相型をメタ言語レベルでの型の「束」に如何にして系統的に対応付けるかであったとも言える.

一方，牽縮を用いたのとは異なる — しかし，やはり領域論的な — アプローチとしては，$\mathbf{D}_\infty$ 上のイデアルを型の表示とするアプローチがある．このアプローチは**「型 ＝ 値の集合」**という素朴な直感に対する理論的な裏付けを与えるアプローチである.

このアプローチを最初に用いたのは，MacQueen and Sethi (1982) で，大雑把に言えば Girard の F_ω に相当する型付き λ-計算の型や高階の型（型や型構成子に関する「型」— しばしば “kind” と呼ばれる）を許し値や型のみならず型構成子に対しても不動点再帰を許す体系での型や kind に対して，一定の条件の下ではイデアルがそれらの表示として用いる事が可能な事を示した．ここでは，再帰型を解釈するのには，cpo 上の順序構造に基づく Tarski の不動点定理では関数型構成子の第 1 引数に関する反単調性の問題を解決できない為，領域 $\mathbf{D}_\infty$ 上のイデアルの集まり $\mathcal{I}(\mathbf{D}_\infty)$ 上にイデアル間の距離を定義し，Tarski の不動点定理よりも強力な計量空間上の Banach の不動点定理を用いている．彼らは更にこのアプローチを発展させ，MacQueen, Plotkin and Sethi (1984, 1986) では共通集合型や合併集合型をも許す型の体系に対してイデアルを用いた表示的意味を与えた.

型の表示を $\mathbf{D}_\infty$ なる「可能な全ての値の集合」の部分集合であるイデアルとするのは直感的に自然であるので，型のイデアルモデルは他の研究でも用いられた．その中で最も重要なのは，本研究のベースとした研究の一つである Cardelli (1984, 1988) での多重継承のモデル化の為のイデアルの使用である．この中では，レコード型や可変型といった現実のプログラミング言語で用いられている型もイデアルとして自然に意味が与えられる事が示され，レコード型の間の部分型関係 “<:” としての継承関係もイデアル間の包含関係 “$\subseteq$” に自然に対応付けられる事が示された.

型の間の部分型関係を捉えるのに Cardelli はイデアル間の包含関係を用いた，という点を少し詳しく述べる．牽縮（やそれに制約を加えた射影や閉包）に基づく型の意味論ではプログラミング言語（やその理論的モデルとしての型理論）での部分型関係を捉える事はできない．何故ならば，牽縮をベースとする型のモデルでは，関数型構成子 “$\_ \to \_$” が左側の型引数に関しては部分型関係について反単調にはならず単

調になってしまうからである．即ち，図 2.2 での [S-ARROW] 則の代わりに

$$\frac{C \rhd \sigma <: \sigma' \quad C \rhd \tau <: \tau'}{C \rhd \sigma \to \tau <: \sigma' \to \tau'}$$

という部分型規則でなければ牽縮ベースの意味論に対して健全とならない．しかし，この新しい部分型関係規則はプログラミング言語で部分型に対して直感的に期待する事 — 計算機でのデータの内部表現の話を別にすれば，整数を引数に取れる関数は引数として自然数しか取れない関数が現れて良い箇所ならどこでも使える筈だ — に全く反している．一方，イデアルに基づいて関数型構成子 "_ → _" を解釈すると，左側の型引数に関して反単調性となり，従って，[S-ARROW] 則はイデアルモデルで健全な推論規則となる．

以上に述べた通り，イデアルに基づく型の意味論は，直感的にも判り易く関数型構成子の反単調性の問題も自然に解決できるので，優れたモデルであるが，型のイデアルモデルの最大の問題は，事実 5.62 で述べた通り，イデアルが Scott 位相での閉集合だという点にある．従って，イデアルである事は，無限個のイデアルの共通集合化 $\cap$ に関しては保たれる — 実際，多相型のイデアルモデルでの解釈は無限個のイデアルの共通集合として与えられる — が，無限個のイデアルの合併集合化 $\cup$ に関して保たれるとは保証されない．その結果，イデアルでは抽象型 $\exists$ の意味を与える事ができない．

この問題を解決する為に，Cartwright (1985) はイデアル間の包含関係という順序に基づきイデアルの区間 (interval) という概念を定義し，型をイデアルの区間として解釈するアプローチを提案した．残念ながら彼の論文には誤りがあったが，Martini (1987) は問題点を解決して F に対するイデアル区間モデルを与え，更に，Martini (1988) では範囲限定の多相型と抽象型とを含む Fun の型システム対して，イデアル区間に基づく意味が与えられた．

一方，イデアルとして $\mathbf{D}_\infty$ に含まれる全てのイデアルを許すのではなく，一定の条件で生成されるイデアルだけを型の表示に用いる事によって，イデアルモデルの中で抽象型 $\exists$ の意味付けに成功したのは Abadí, Pierce and Plotkin (1989) である．彼らは，イデアルが合併集合に関して閉じない問題をイデアル完備化 — 合併集合を含む最小のイデアルとする操作 — によって解決した．

さて，多相型に関しては再帰を含まなくとも集合論では表示的意味を与える事ができない，という Reynolds (1984) の結果は本節の最初の概説で述べたが，多相型だけの場合は，たとえその多相型が F や Fun（や本論文の Funiq）での様な非可述的なものであっても領域論的な半順序を使用しない解釈の与え方が可能である．

実際，数理論理学の側では，以前から HRO (Hereditarily Recursive Operations) や HEO (Hereditarily Effective Operations) といった再帰的関数論での概念を用いて Troelstra (1971, 1971b) 等にある形で F のモデルが与えられて来た． HRO や HEO は Kleene に源流を持つ任意の有限階の再帰的部分（汎）関数の自然数による枚挙 — 個々の再帰的部分関数を一つの自然数で表わす — と自然数としてコード化された関数を引数としての自然数に適用する適用演算のコード化に基づいており，領域論と同じく部分関数の概念を含んではいるが，領域論での計算の近似としての半順序構造は含んでいない．その意味で，これらの再帰的関数論的な構造に基づくモデルは非領域論的であり，順序構造の入っていない自然数の集合を用いて構築されているという点では，広い意味での集合論的モデルと言っても間違いではない．

数理論理学的な立場からの F に対するモデルの歴史に関しては本論文の範囲を超えているので，詳細は Girard (1986) での纏めに譲る事とするが，そこで指摘されている通り理論計算機科学の側でも以上の概念の変形としての ω-set や modest set 等の概念を用いて例えば Bruce and Longo (1988, 1990) 見られる様な多相型のモデルが与えられて来た．

Modest set は Kleene の再帰的部分関数のコードとしての自然数の集合 ω 上の部分同値関係 (partial equivalence relation, per) — 集合全体でなく部分集合を定義域とする同値関係 — であるが，その後，自然数の集合 ω 上のでなく領域 $\mathbf{D}_\infty$ 上の部分同値関係の概念に発展させられ， Abadí and Plotkin (1990) では範囲限定の多相型と再帰型とを許す体系に対して部分同値関係によるモデルが与えられた．ここでの技術的課題は範囲限定の上限の型と再帰型との干渉（再帰的に定義される型が定義式としての範囲限定多相型の上限の型として現れ得る）を解決する事であった． Bruce and Mitchell (1992) はそれに加え部分型関係をも許す体系に対しモデルを与えた．更に， Amadio (1991) は $\mathbf{D}_\infty$ 上の部分同値関係に**一様性**という概念を導入し， $\mathbf{D}_\infty$ 上の一様な部分同値関係を用い，全く制限のない形の再帰型を F に追加した体系のモデルが与えた． Cardone (1989, 1991) も独立に同様の部分同値関係を用い，型の

表示の構築に於いて逆極限法と同様の反復手法を適用する事により，Fun から抽象型 $\exists$ を除き再帰型を追加した体系に対してモデルを与えた．即ち，型の意味関数自体を無限の近似列によって構成した事になる． Cardone らは更にこのアプローチを押し進め， Cardone, Dezani-Ciancaglini and de'Liguoro (1994) では，抽象型をも含む Fun （レコード型と可変型は除かれているがこの欠落は本質的ではない）に共通集合型 $\wedge$ と合併集合型 $\vee$ を加えた非常に一般的な型体系に対してモデルを与える事に成功した．この後も部分同値関係に基づいた型の意味論に関しては数多くの研究が存在するが，細部に入り過ぎており略史としては相応しくないので省略する．

なお，多相型の意味論が満たすべき一般的特性は圏論の言葉によって定式化する事が可能であるが，最初に多相型の圏論モデルを与えたのは Seely (1984) である．その後， F （やそのバリエーションとしての Fun や $\mathrm{F}_{\leq}$）やより強力な型システムに関する圏論モデルに関しては相当量の研究が成されて来た．しかし，それらは本論文の主題と直接の関連はないので，この方面に関しては，型付きのみならず型無し λ-計算をも含めての圏論モデルに関する古典である Lambek and Scott (1986)，型理論の圏論モデルの基本部分を纏めた Crole (1993)，型理論の圏論モデル研究の中心人物の一人の学位論文 Jacobs (1991) と同じ著者による型理論への圏論的手法に関する集大成 Jacobs (1998) を挙げるに止める．

5.2 Funiq の表示的意味の構成

本節では，前節で準備した Scott の領域論が与える完備半順序構造に基づき Funiq の表示的意味を与える．

5.2.1 型付き解釈と型無し解釈

Funiq の様な型付き λ-計算への表示的意味の与え方としては，大別すると次の二通りのアプローチがある：

(1) **型付き解釈 (typed interpretation)**
型を持つ λ-式をその型も含めた形で解釈する．即ち， Bruce, Meyer and Mitchell (1990) での frame を用いて，式そのものではなく，式とその型と型付け文脈とを含んだ判別式に対して直接に意味を与える．

(2) **型無し解釈 (untyped interpretation)**

式の型や式の内部の型情報 — 例えば, λ-抽象での束縛変数への型指定等 — を消去して，全ての式は型無し λ-式と看做し，それを（定数や様々な構文で拡張された）応用 (applied) 型無し λ-計算に対して表示的意味を与える方法で解釈する.

(1) の型付き解釈は, Fun 等の狭い（プログラミング言語での文法の様に構文生成規則で生成されるという）意味での言語に対して意味を与えるのでなく，型付けや型の情報をも含めた判別式全体に対して直接に表示的意味を与える，つまり, FUN の様な型理論自体に対して直接に意味を与えるアプローチである．故に，数理論理学的な観点から型理論に対して意味を与える方法としては適切だと言える.

このアプローチを採る場合，本論文でいう式型付けの情報を全て含んだ形の（型付け）判別式全体を「式」[5] と呼び，本論文で言うところの「式」は「準式 (pseudo-term)」と呼ぶ事が多い.

しかしながら，型付き解釈で判別式に意味を与える場合, FUN や FUNIQ の様に部分型関係を許す型理論では「部分型に関する推論規則の適用順序に依存せず判別式の意味は一意に定まる」という斉一性定理 (coherence theorem) を証明する必要が生ずる.

一方, (2) の型無し解釈は，型付きプログラミング言語に対する表示的意味で広く用いられている方式である．この場合，意味を直接に与える対象は式や型そのものであり，式の意味と型の意味との間の整合性は，別途，証明する必要がある．つまり，型付け文脈 C, Γ, Δ で式 e が型 σ を持つ，という判別式が表わす意味が与えられた（式や型に対する）表示的意味で適切に捉えられている事は各構文クラスに対する意味関数の定義とは別に証明する必要がある.

即ち，型無し解釈で型理論全体に対する解釈を得るには，判別式の妥当性の概念を式や型に対する意味定義を元に与え，その妥当性に基づいて型理論の健全性を新たに証明する必要がある.

本論文では, (2) の型無し解釈を採用する．その理由は，上に述べた通り，型付きプログラミング言語という型付き λ-計算の応用の立場での表示的意味の与え方とし

5) 数理論理学的な文献では寧ろ「項」と呼ばれる事の方が多いが，本論文ではプログラミング言語の言葉遣いを踏襲して「式」と呼ぶ

ては (2) のアプローチの方が広く採用されている事と，直接的に意味を与える対象が判別式というメタ理論的な対象でなく式や型というプログラミング言語の立場で基本的な対象と看做せるものであり，その結果，与えている意味の内容が直感的に単純明解である事とによる．

5.2.2 式の表示的意味

■ 式の意味定義の為のメタ言語

既に前節の領域に関する記述で部分的に使用したが，本節で式の表示的意味定義に用いるメタ言語は，以下の構文から成る形式化されていない型付き λ-記法である．

項 ::= 定数記号としての $\bot$
　　| $\mathbb{T}$ や $\mathbb{N}$ の要素を表わす定数記号
　　| 前節で示した cpo の構成等に関連する基本演算子
　　($\langle \_, \_ \rangle, \_.1, \_.2, is_{\mathbf{X}}, in_{\mathbf{X}}, out_{\mathbf{X}}, \mathit{fix}$ 等)
　　| $\boldsymbol{\lambda}$ メタ変数 $\in |$cpo 式$|$. 項
　　| $\boldsymbol{\lambda}_{\bot}$ メタ変数 $\in |$cpo 式$|$. 項
　　| 項$_1$(項$_2$)
　　| if 基本演算 $is_{\mathbf{X}}$ の呼出し項 then 項$_1$ elseif 項$_2$ elseif ... else 項$_{n+1}$
　　| let 束縛$_1$ and 束縛$_2$ and ... and 束縛$_{n+1}$ in 項 end;
束縛 ::= _ = 項
　　| メタ変数 = 項
　　| $\langle$メタ変数$_1, \ldots,$ メタ変数$_{n+1}\rangle$ = 項.

各構文の意味は明らか（束縛の最後の構文はパターンマッチングによる直積成分への分解）なので説明は加えない．上の非形式的な型付き λ-記法の項が表わす関数は常に連続である事も明らかである．

■ 式の型情報の消去

本節では言語としての型無し解釈の流儀により Funiq の式に対して表示的意味を与える．本章の最初に断わった通り，抽象型に関しては定義 3.1 に示した多相型と関数型とでコード化されたものとして扱うので，図 2.1 の **Exp** に対する構文規則で示された構文の中で抽象型に対する **pack** 式と **unpack** 式とは除き，残りの式の構文より型情報を消去せねばならない．その為の写像を以下の様に構造帰納的に与える：

定義 5.64 （型消去式）

Funiq の式から型を消去する写像 $\langle\!\langle \cdot \rangle\!\rangle$ を

$$
\begin{aligned}
\langle\!\langle c \rangle\!\rangle &= c, \\
\langle\!\langle x \rangle\!\rangle &= x, \\
\langle\!\langle r \rangle\!\rangle &= r, \\
\langle\!\langle \lambda\, x{:}\,\sigma.e \rangle\!\rangle &= \lambda\, x.\langle\!\langle e \rangle\!\rangle, \\
\langle\!\langle e_1 e_2 \rangle\!\rangle &= \langle\!\langle e_1 \rangle\!\rangle \langle\!\langle e_2 \rangle\!\rangle, \\
\langle\!\langle \{l_1 = e_1, \ldots, l_n = e_n\} \rangle\!\rangle &= \{l_1 = \langle\!\langle e_1 \rangle\!\rangle, \ldots, l_n = \langle\!\langle e_n \rangle\!\rangle\}, \\
\langle\!\langle e.l \rangle\!\rangle &= \langle\!\langle e \rangle\!\rangle.l, \\
\langle\!\langle [\, l = e\,] \rangle\!\rangle &= [\, l = \langle\!\langle e \rangle\!\rangle\,], \\
\langle\!\langle \mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n \rangle\!\rangle &= \mathbf{case}\ \langle\!\langle e_0 \rangle\!\rangle\ \mathbf{of}\ l_1\ \mathbf{then}\ \langle\!\langle e_1 \rangle\!\rangle, \ldots, l_n\ \mathbf{then}\ \langle\!\langle e_n \rangle\!\rangle, \\
\langle\!\langle \Lambda\, t{<:}\sigma.e \rangle\!\rangle &= \langle\!\langle e \rangle\!\rangle, \\
\langle\!\langle e[\sigma] \rangle\!\rangle &= \langle\!\langle e \rangle\!\rangle, \\
\langle\!\langle \mathbf{fix}\ e \rangle\!\rangle &= \mathbf{fix}\ \langle\!\langle e \rangle\!\rangle, \\
\langle\!\langle e{:}\,\sigma \rangle\!\rangle &= \langle\!\langle e \rangle\!\rangle
\end{aligned}
$$

と定義し，それの適用結果としての型を消去された式を**型消去式 (erasure)** と呼ぶ．従って，型消去式の構文は，以下の通りである：

$$
\begin{aligned}
M \in \mathbf{Erasure} ::= {} & c \quad | \quad x \quad | \quad r \\
& | \ \lambda\, x.M \quad | \quad M_1 M_2 \\
& | \ \{l_1 = M_1, \ldots, l_n = M_n\} \quad | \quad M.l \\
& | \ [\, l = M\,] \quad | \quad \mathbf{case}\ M_0\ \mathbf{of}\ l_1\ \mathbf{then}\ M_1, \ldots, l_n\ \mathbf{then}\ M_n \\
& | \ \mathbf{fix}\ M.
\end{aligned}
$$

■ 式の値の成す領域 D の再帰的定義

型無し解釈では，個々の式は「値」を，また，個々の型は「値の集まり」を表わす形で意味を考える．その為には，「値」の集まりとしての領域を定めねばならない．

上の型消去式の定義を見れば判る通り，型消去式の構文は定数とレコードの扱いとタグ付けの扱いとを追加した応用型無し λ-計算を定める．

この「型無し」という点に関してより正確に言えば，上の構文が与える言語は，Barendregt (1984) 等に於ける型無し λ-計算本来の意味 — 任意の式の組合せを許すという意味 — での「型無し」ではない．例えば，関数適用 $M_1 M_2$ の最初の式 M_1

は関数を表わす式でなければならず，自然数を表わす定数 0 等である場合は，**実行時の型のエラー**として扱わねばならない．即ち，上の構文が定める言語は，実行時に値の型の検査が必要な**動的に型付けられた**言語である．

Funiq に対する型無し解釈の為の値の成す領域 $\mathbf{D}$ は，上に示した応用型無し λ-計算の式が表示し得る値をカバーすれば良いので，Funiq に最初から組み込まれている基本型の個数を n $(n \geq 0)$ とすると，領域 $\mathbf{D}$ を構成する値としては，

(1) 個々の基本型に属する値,
(2) 関数値[6],
(3) レコード値,
(4) タグ付けされた値
(5) 実行時の型エラーを表わす値

の $(n+4)$ 種類がある．

そこで，Funiq の式の値を捉えるべき意味領域 $\mathbf{D}$ は，以下の再帰的領域方程式の解として与えられるものである：

$$d \in \mathbf{D} \cong \mathbf{A}_1 \oplus \cdots \oplus \mathbf{A}_n \oplus \mathbf{F} \oplus \mathbf{R} \oplus \mathbf{V} \oplus \mathbf{W} \qquad (\bigstar)$$

ここで，補助的に用いている領域の意味は,

- 各 $\mathbf{A}_i$ $(1 \leq i \leq n)$ は，i 番目の基本型 ι_i に属する値の集まりとしての平坦 cpo（故に領域）である；
- $f, g \in \mathbf{F} \triangleq [\mathbf{D} \to \mathbf{D}]$ は関数値の成す cpo である；
- $q, r \in \mathbf{R} \triangleq [\mathbf{L} \to_{\perp} \mathbf{D}]$ はレコード値の成す cpo である．ここで，$\mathbf{L} \triangleq \mathbf{Label}_{\perp}$ は字句カテゴリとしてのラベル／タグの集合 $\mathbf{Label}$ を底上げした平坦 cpo；
- $u, v \in \mathbf{V} \triangleq \mathbf{L} \times \mathbf{D}$ はタグ付き値の成す cpo である；
- $\mathbf{W} \triangleq \{?\}_{\perp}$ は実行時の型エラーを表わす値である ? と最小要素だけから成る平坦 cpo である．

前節の逆極限法の項で示した通り，再帰的領域方程式 $(\bigstar)$ は解を持ち，また，基本型に対する各 $\mathbf{A}_i$ が領域なので，逆極限解 $\mathbf{D}$ は領域となる事が保証される．

6) ここの「関数値」は，引数が与えられた時の関数の返す値ではなく，関数型の式が表わすべき関数という（高階の）値を意味する．

以下，再帰的領域方程式 (★) の逆極限解を構成する．より具体的である為，基本領域 $\mathbf{A}_i$ の個数 n は 2 とし $\mathbf{A}_1 = \mathbf{T}$, $\mathbf{A}_2 = \mathbf{N}$ とする．これは，意味を与えるべき言語 Funiq に組み込まれている基本型が真理値の型 bool と自然数の型 nat との二つであるとした場合に相当する．

なお，融合和の成分領域の中で求めるべき領域 $\mathbf{D}$ を構成要素として含んでいる $\mathbf{F}, \mathbf{R}, \mathbf{V}$ に相当する第 i 世代 $(i \geq 1)$ でのもの — つまり各々の中の $\mathbf{D}$ が $\mathbf{D}_{i-1}$ であるもの — を，各々， $\mathbf{F}_i, \mathbf{R}_i, \mathbf{V}_i$ で表わす事にする．即ち，各 $i \geq 1$ に対して，

$$\begin{aligned} f, g \in \mathbf{F}_i &\triangleq [\mathbf{D}_{i-1} \to \mathbf{D}_{i-1}], \\ q, r \in \mathbf{R}_i &\triangleq [\mathbf{L} \to_\perp \mathbf{D}_{i-1}], \\ u, v \in \mathbf{V}_i &\triangleq \mathbf{L} \times \mathbf{D}_{i-1} \end{aligned}$$

とする．

上の領域方程式 (★) より，前節の逆極限法での領域構成子 F は，

$$F \triangleq \boldsymbol{\lambda}\, \mathbf{X}.\mathbf{T} \oplus \mathbf{N} \oplus [\mathbf{X} \to \mathbf{X}] \oplus [\mathbf{L} \to_\perp \mathbf{X}] \oplus (\mathbf{L} \times \mathbf{X}) \oplus \mathbf{W}$$

となるので，それによって定まる射影対構成子 F^{epp} は，

$$\begin{aligned} F^{epp} \triangleq \boldsymbol{\lambda}\, (e,p). \quad & (id_{\mathbf{T}}, id_{\mathbf{T}}) \oplus (id_{\mathbf{N}}, id_{\mathbf{N}}) \\ & \oplus [(e,p) \to (e,p)] \\ & \oplus [(id_{\mathbf{L}}, id_{\mathbf{L}}) \to_\perp (e,p)] \\ & \oplus ((id_{\mathbf{L}}, id_{\mathbf{L}}) \times (e,p)) \\ & \oplus (id_{\mathbf{W}}, id_{\mathbf{W}}) \end{aligned}$$

である．

まず，最初の種領域 $\mathbf{D}_0$ を

$$\mathbf{D}_0 \triangleq \{\perp\}$$

と与える（半順序に関しては省略．以下同様）．これを用いて， $\mathbf{D}_1$ を求めると，

$$\begin{aligned} \mathbf{D}_1 &\triangleq F(\mathbf{D}_0) \\ &= \mathbf{T} \oplus \mathbf{N} \oplus [\{\perp\} \to \{\perp\}] \oplus [\mathbf{L} \to_\perp \{\perp\}] \oplus (\mathbf{L} \times \{\perp\}) \oplus \mathbf{W} \\ &= \mathbf{T} \oplus \mathbf{N} \oplus \{\perp \mapsto \perp\} \oplus \{\boldsymbol{\lambda}\, l \in \mathbf{L}.\perp\} \oplus \{\langle l, \perp\rangle \mid l \in \mathbf{L}\} \oplus \{?\}_\perp \\ &= \big(\{\mathit{false}, \mathit{true}\} \cup \{0, 1, 2, \ldots\} \cup \{\langle l, \perp\rangle \mid l \in \mathbf{Label}\} \cup \{?\}\big)_\perp \end{aligned}$$

となる．一方，$\mathbf{D}_0, \mathbf{D}_1$ 間の射影対 (φ_0, ψ_0) は，

$$\begin{aligned}\varphi_0 &\triangleq \boldsymbol{\lambda}\, d_0 \in |\mathbf{D}_0|.\bot_{\mathbf{D}_1},\\ \psi_0 &\triangleq \boldsymbol{\lambda}\, d_1 \in |\mathbf{D}_1|.\bot_{\mathbf{D}_0}\\ &= \boldsymbol{\lambda}\, d_1 \in |\mathbf{D}_1|.\bot\end{aligned}$$

である．

以下，$i \geq 1$ とし，$\mathbf{D}_i$ と $(\varphi_{i-1}, \psi_{i-1})$ とが既に構成されているとして，$\mathbf{D}_{i+1}$ と (φ_i, ψ_i) との構成を具体的に示す．まず，$\mathbf{D}_{i+1}$ に関しては，

$$\begin{aligned}\mathbf{D}_{i+1} &\triangleq F(\mathbf{D}_i)\\ &= \mathbf{T} \oplus \mathbf{N} \oplus [\mathbf{D}_i \to \mathbf{D}_i] \oplus [\mathbf{L} \to_\bot \mathbf{D}_i] \oplus (\mathbf{L} \times \mathbf{D}_i) \oplus \mathbf{W}\\ &= \mathbf{T} \oplus \mathbf{N} \oplus \mathbf{F}_{i+1} \oplus \mathbf{R}_{i+1} \oplus \mathbf{V}_{i+1} \oplus \mathbf{W}\end{aligned}$$

となる．一方，射影対 (φ_i, ψ_i) に関しては，

$$\begin{aligned}(\varphi_i, \psi_i) &\triangleq F^{epp}(\varphi_{i-1}, \psi_{i-1})\\ &= \quad (id_{\mathbf{T}}, id_{\mathbf{T}}) \oplus (id_{\mathbf{T}}, id_{\mathbf{T}})\\ &\quad \oplus [(\varphi_{i-1}, \psi_{i-1}) \to (\varphi_{i-1}, \psi_{i-1})]\\ &\quad \oplus [(id_{\mathbf{L}}, id_{\mathbf{L}}) \to_\bot (\varphi_{i-1}, \psi_{i-1})]\\ &\quad \oplus ((id_{\mathbf{L}}, id_{\mathbf{L}}) \times (\varphi_{i-1}, \psi_{i-1}))\\ &\quad \oplus (id_{\mathbf{W}}, id_{\mathbf{W}})\\ &= \quad (id_{\mathbf{T}}, id_{\mathbf{T}}) \oplus (id_{\mathbf{T}}, id_{\mathbf{T}})\\ &\quad \oplus (\boldsymbol{\lambda}\, f \in |\mathbf{F}_i|.\varphi_{i-1} \circ x \circ \psi_{i-1}, \boldsymbol{\lambda}\, g \in |\mathbf{F}_{i+1}|.\psi_{i-1} \circ y \circ \varphi_{i-1})\\ &\quad \oplus (\boldsymbol{\lambda}_\bot q \in |\mathbf{R}_i|.\varphi_{i-1} \circ q \circ id_{\mathbf{L}}, \boldsymbol{\lambda}_\bot r \in |\mathbf{R}_{i+1}|.\psi_{i-1} \circ r \circ id_{\mathbf{L}})\\ &\quad \oplus (\boldsymbol{\lambda}\, u \in |\mathbf{V}_i|.\langle id_{\mathbf{L}}(u.1), \varphi_{i-1}(u.2)\rangle, \boldsymbol{\lambda}\, v \in |\mathbf{V}_{i+1}|.\langle id_{\mathbf{L}}(v.1), \psi_{i-1}(v.2)\rangle)\\ &\quad \oplus (id_{\mathbf{W}}, id_{\mathbf{W}})\end{aligned}$$

となる．従って，

$$\begin{aligned}
\varphi_i = {} & \boldsymbol{\lambda}_\perp d \in |\mathbf{D}_i|. \\
& \mathsf{if}\ is_{\mathbf{T}}(d)\ \mathsf{then}\ in_{\mathbf{T}}(id_{\mathbf{T}}(out_{\mathbf{T}}(d))) \\
& \mathsf{elseif}\ is_{\mathbf{N}}(d)\ \mathsf{then}\ in_{\mathbf{N}}(id_{\mathbf{N}}(out_{\mathbf{N}}(d))) \\
& \mathsf{elseif}\ is_{\mathbf{F}_i}(d)\ \mathsf{then}\ in_{\mathbf{F}_i}((\boldsymbol{\lambda} f \in |\mathbf{F}_i|.\varphi_{i-1} \circ f \circ \psi_{i-1})(out_{\mathbf{F}_i}(d))) \\
& \mathsf{elseif}\ is_{\mathbf{R}_i}(d)\ \mathsf{then}\ in_{\mathbf{R}_i}((\boldsymbol{\lambda} q \in |\mathbf{R}_i|.\varphi_{i-1} \circ q \circ id_{\mathbf{L}})(out_{\mathbf{R}_i}(d))) \\
& \mathsf{elseif}\ is_{\mathbf{V}_i}(d)\ \mathsf{then}\ in_{\mathbf{V}_i}((\boldsymbol{\lambda} u \in |\mathbf{V}_i|.\langle id_{\mathbf{L}}(u.1), \varphi_{i-1}(u.2)\rangle)(out_{\mathbf{V}_i}(d))) \\
& \mathsf{elseif}\ is_{\mathbf{W}}(d)\ \mathsf{then}\ in_{\mathbf{W}}(id_{\mathbf{W}}(out_{\mathbf{W}}(d))) \\
= {} & \boldsymbol{\lambda}_\perp d \in |\mathbf{D}_i|.\, \mathsf{if}\ is_{\mathbf{T}}(d)\ \mathsf{then}\ \langle 1, out_{\mathbf{T}}(d)\rangle \\
& \qquad \mathsf{elseif}\ is_{\mathbf{N}}(d)\ \mathsf{then}\ \langle 2, out_{\mathbf{N}}(d)\rangle \\
& \qquad \mathsf{elseif}\ is_{\mathbf{F}_i}(d)\ \mathsf{then}\ \langle 3, \varphi_{i-1} \circ (out_{\mathbf{F}_i}(d)) \circ \psi_{i-1}\rangle \\
& \qquad \mathsf{elseif}\ is_{\mathbf{R}_i}(d)\ \mathsf{then}\ \langle 4, \varphi_{i-1} \circ (out_{\mathbf{R}_i}(d))\rangle \\
& \qquad \mathsf{elseif}\ is_{\mathbf{V}_i}(d)\ \mathsf{then}\ in_{\mathbf{V}_i}(\langle 5, \langle out_{\mathbf{V}_i}(d).1, \varphi_{i-1}(out_{\mathbf{V}_i}(d).2)\rangle\rangle \\
& \qquad \mathsf{elseif}\ is_{\mathbf{W}}(d)\ \mathsf{then}\ \langle 6, out_{\mathbf{W}}(d)\rangle
\end{aligned}$$

である． Standard ML 風のパターンマッチングを許す case 記法を導入して融合和成分への射影演算 out を暗黙化して式を整理すると，埋め込み $\varphi_i : [\mathbf{D}_i \to \mathbf{D}_{i+1}]$ は，

$$\begin{aligned}
\varphi_i = {} & \boldsymbol{\lambda}_\perp d \in |\mathbf{D}_i|.\, \mathsf{case}\ d\ \mathsf{of} \\
& \qquad \langle 1, b\rangle \quad \mathsf{then}\ \langle 1, b\rangle \\
& \qquad \langle 2, n\rangle \quad \mathsf{then}\ \langle 2, n\rangle \\
& \qquad \langle 3, f\rangle \quad \mathsf{then}\ \langle 3, \varphi_{i-1} \circ f \circ \psi_{i-1}\rangle \\
& \qquad \langle 4, q\rangle \quad \mathsf{then}\ \langle 4, \varphi_{i-1} \circ q\rangle \\
& \qquad \langle 5, \langle l, y\rangle\rangle \quad \mathsf{then}\ \langle 5, \langle l, \varphi_{i-1}(y)\rangle\rangle \\
& \qquad \langle 6, _\rangle \quad \mathsf{then}\ \langle 6, ?\rangle \\
& \mathsf{end}
\end{aligned}$$

と表わせる．同様に，射影 $\psi_i : [\mathbf{D}_{i+1} \to \mathbf{D}_i]$ の方も

$$\begin{aligned}
\psi_i = {} & \boldsymbol{\lambda}_\perp d \in |\mathbf{D}_{i+1}|.\, \mathsf{case}\ d\ \mathsf{of} \\
& \qquad \langle 1, b\rangle \quad \mathsf{then}\ \langle 1, b\rangle \\
& \qquad \langle 2, n\rangle \quad \mathsf{then}\ \langle 2, n\rangle \\
& \qquad \langle 3, f\rangle \quad \mathsf{then}\ \langle 3, \psi_{i-1} \circ f \circ \varphi_{i-1}\rangle \\
& \qquad \langle 4, q\rangle \quad \mathsf{then}\ \langle 4, \psi_{i-1} \circ q\rangle \\
& \qquad \langle 5, \langle l, y\rangle\rangle \quad \mathsf{then}\ \langle 5, \langle l, \psi_{i-1}(y)\rangle\rangle \\
& \qquad \langle 6, _\rangle \quad \mathsf{then}\ \langle 6, ?\rangle \\
& \mathsf{end}
\end{aligned}$$

となる． φ_i と ψ_i とが共に連続で更に $\psi_i \circ \varphi_i = id_{\mathbf{D}_i}$ と $\varphi_i \circ \psi_i \sqsubseteq id_{\mathbf{D}_i}$ とを満たし射影対を成している事は，$(\varphi_{i-1}, \psi_{i-1})$ が射影対である事を用いれば簡単に示す事ができる．

前節の逆極限法の一般論に纏めてある通り，以上の構成の超限回の反復により，

$$\Phi : \mathbf{D} \xleftrightarrow{\sim} \mathbf{T} \oplus \mathbf{N} \oplus [\mathbf{D} \to \mathbf{D}] \oplus [\mathbf{L} \to_{\perp} \mathbf{D}] \oplus (\mathbf{L} \times \mathbf{D}) \oplus \mathbf{W} : \Psi$$

という同型性を満たす領域 $\mathbf{D}$ と同型射 Φ, Ψ を得る事ができる．

なお，以下で式の意味方程式の記述を簡潔に行なう為に，右辺の融合和成分の値の操作に関する基本演算と同型射との合成に関する記法を導入しておく．

記法 5.65 （融合和成分に関する演算）

$\mathbf{X}$ を $\mathbf{A}_i$ $(\mathbf{T}, \mathbf{N})$, $\mathbf{F}$, $\mathbf{R}$, $\mathbf{V}$ の何れかとする時，

(1) $Is_{\mathbf{X}} \triangleq is_{\mathbf{X}} \circ \Phi$;

(2) $In_{\mathbf{X}} \triangleq \Psi \circ in_{\mathbf{X}}$;

(3) $Out_{\mathbf{X}} \triangleq out_{\mathbf{X}} \circ \Phi$

と定める．また，実行時の型エラーを表わす値 ? の $\mathbf{D}$ での像を

(4) $wrong \triangleq (\Psi \circ in_{\mathbf{W}})(?)$

で表わす．

■ 式の意味関数

以上で得られた領域 $\mathbf{D}$ の要素を式の表示として用い，その為の式の意味関数 $\mathcal{E}$ を本項で定義する．

式には変数の自由な出現が存在し得るので，そういった自由変数に対する付値 — **環境**と呼ぶ — を決めなければ式の値は決らない． Funiq の場合，変数としては通常変数と実現変数との 2 種類があるので，通常変数に関する環境 **OEnv** と実現変数に関する環境 **IEnv** とが必要である．これら各々は対応する変数の集合 **Var** もしくは **IVar** を底上げして得られる平坦 cpo から全ての値のなす領域 $\mathbf{D}$ への厳格な連続関数とする．

これらの変数に対する付値が厳格でなければならない理由は，付値が連続である為には単調でなければならず，付値が単調である為には（値が未定義の変数に対する付値は $\bot_{\mathbf{D}}$ であるので）底上げで各字句カテゴリ $\mathbf{Var}, \mathbf{IVar}$ に追加された最小要素 $\bot$ に対しては $\bot_{\mathbf{D}}$ 以外の値を対応付ける事が許されないからである．例として x がその様な値が未定義の変数だとする．この時，$\bot_{\mathbf{Var}_\bot} \sqsubseteq_{\mathbf{Var}_\bot} \mathrm{x}$ であるので，連続性の必要条件としての単調性から通常変数に対する任意の付値 $\zeta \in \mathbf{OEnv}$ について $\zeta(\bot_{\mathbf{Var}_\bot}) \sqsubseteq_{\mathbf{D}} \zeta(\mathrm{x})$ でなければならない．この右辺の $\zeta(\mathrm{x})$ は $\bot_{\mathbf{D}}$ であるので左辺も同じく $\bot_{\mathbf{D}}$ でなければならない．以上から ζ の厳格性が導かれた．

故に，変数に対する付値としての環境の集まりの成す意味領域 **Env** 全体はこれら 2 種類の変数に対する付値それぞれが成す厳格連続関数空間の直積であり，**Env** とその二つの成分 $\mathbf{OEnv}, \mathbf{IEnv}$ は以下の様に定義される：

$$\begin{aligned} \varepsilon \in \mathbf{Env} &\triangleq \mathbf{OEnv} \times \mathbf{IEnv}, \\ \zeta \in \mathbf{OEnv} &\triangleq [\mathbf{Var}_\bot \to_\bot \mathbf{D}], \\ \xi \in \mathbf{IEnv} &\triangleq [\mathbf{IVar}_\bot \to_\bot \mathbf{D}]. \end{aligned}$$

従って，式の意味関数 $\mathcal{E}[\![\cdot]\!]$ は，式の集合 **Exp** から **Env** の要素としての環境が与えられれば **D** の要素を返す連続関数空間 $[\mathbf{Env} \to \mathbf{D}]$ への写像である．

本論文では型無し解釈を採用するので，本来は，型消去式に対する意味関数，即ち集合 **Erasure** の各要素に対して環境を貰い **D** の要素を返す連続関数を対応付ける写像 $\mathcal{E}'$ を定義し，それと型消去写像 $\langle\!\langle \cdot \rangle\!\rangle$ とから両者の合成

$$\mathcal{E}[\![\cdot]\!] \triangleq \mathcal{E}'[\![\cdot]\!] \circ \langle\!\langle \cdot \rangle\!\rangle$$

として Funiq の式の意味関数 $\mathcal{E}$ を与えるべきである．しかし，Funiq の式が表わす意味が何であるかをより判り易く提示する為に，Funiq の式の意味関数 $\mathcal{E}$ を直接に与える事とする．その為，$\mathcal{E}[\![\cdot]\!]$ を定める意味方程式を図 5.3 に示す．ここで，定数記号 c_{ij} に対する意味方程式の右辺中の “$\mathcal{K}_i[\![\cdot]\!]$” は i 番目の基本型 ι_i の定数記号に解釈を与える為の意味関数であり，$\mathcal{K}_i : [\mathbf{Const}_{i\bot} \to_\bot \mathbf{A}_i]$ というアリティ（但し，$\mathbf{Const}_i$ は i 番目の基本型 ι_i の定数記号から成る **Const** の部分集合）を持つものとして予め用意されているものとする．

$$\mathcal{E} : \mathbf{Exp} \to [\mathbf{Env} \to \mathbf{D}]$$

$$\mathcal{E}[\![c_{ij}]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ In_{\mathbf{A}_i}(\mathcal{K}_i[\![c_{ij}]\!]);$$

$$\mathcal{E}[\![x]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ \mathsf{let}\ \langle \zeta, \xi \rangle = \varepsilon\ \mathsf{in}\ \ \zeta(x)\ \ \mathsf{end};$$

$$\mathcal{E}[\![r]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ \mathsf{let}\ \langle \zeta, \xi \rangle = \varepsilon\ \mathsf{in}\ \ \xi(r)\ \ \mathsf{end};$$

$$\mathcal{E}[\![\lambda\, x{:}\sigma.e]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ \mathsf{let}\ \langle \zeta, \xi \rangle = \varepsilon\ \ \mathsf{in}\ \ In_{\mathbf{F}}(\boldsymbol{\lambda}\, d \in |\mathbf{D}|.\mathcal{E}[\![e]\!]\langle \zeta[x \mapsto d], \xi \rangle)\ \ \mathsf{end};$$

$$\mathcal{E}[\![e_1 e_2]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon)\ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon)(\mathcal{E}[\![e_2]\!]\varepsilon)\ \mathsf{else}\ \ wrong;$$

$$\begin{aligned}\mathcal{E}[\![\{l_1 = e_1, \ldots, l_n = e_n\}]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ & In_{\mathbf{R}}(\boldsymbol{\lambda}_{\perp}\, l \in |\mathbf{L}|.\ \mathsf{if}\ \ l = l_1\ \mathsf{then}\ \ \mathcal{E}[\![e_1]\!]\varepsilon \\ & \qquad \mathsf{elseif}\ \ \ldots \\ & \qquad \mathsf{elseif}\ \ l = l_n\ \mathsf{then}\ \ \mathcal{E}[\![e_n]\!]\varepsilon \\ & \qquad \mathsf{else}\ \ \perp_{\mathbf{D}});\end{aligned}$$

$$\mathcal{E}[\![e.l]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ \mathsf{if}\ \ Is_{\mathbf{R}}(\mathcal{E}[\![e]\!]\varepsilon)\ \mathsf{then}\ \ Out_{\mathbf{R}}(\mathcal{E}[\![e]\!]\varepsilon)(l)\ \mathsf{else}\ \ wrong;$$

$$\mathcal{E}[\![[\, l = e\,]]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ In_{\mathbf{V}}(\langle l, \mathcal{E}[\![e]\!]\varepsilon \rangle);$$

$$\begin{aligned}\mathcal{E}[\![\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ & \mathsf{if}\ \ Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\varepsilon)\ \mathsf{then} \\ & \quad \mathsf{let}\ \langle l, v \rangle = Out_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\varepsilon)\ \mathsf{in} \\ & \qquad \mathsf{if}\ \ l = l_1\ \mathsf{then} \\ & \qquad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon)\ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon)(v) \\ & \qquad\quad \mathsf{else}\ \ wrong \\ & \qquad \mathsf{elseif}\ \ \ldots \\ & \qquad \mathsf{elseif}\ \ l = l_n\ \mathsf{then} \\ & \qquad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\varepsilon)\ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\varepsilon)(v) \\ & \qquad\quad \mathsf{else}\ \ wrong \\ & \qquad \mathsf{else}\ \ \perp_{\mathbf{D}} \\ & \quad \mathsf{end} \\ & \mathsf{else}\ \ wrong;\end{aligned}$$

$$\mathcal{E}[\![\Lambda\, t{<:}\sigma.e]\!] = \mathcal{E}[\![e]\!];$$

$$\mathcal{E}[\![e[\sigma]]\!] = \mathcal{E}[\![e]\!];$$

$$\begin{aligned}\mathcal{E}[\![\mathbf{fix}\ e]\!] = \boldsymbol{\lambda}\, \varepsilon \in |\mathbf{Env}|.\ & \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e]\!]\varepsilon)\ \mathsf{then} \\ & \quad \mathsf{let}\ f = Out_{\mathbf{F}}(\mathcal{E}[\![e]\!]\varepsilon)\ \mathsf{in}\ fix(f)\ \mathsf{end} \\ & \mathsf{else}\ \ wrong;\end{aligned}$$

$$\mathcal{E}[\![e{:}\tau]\!] = \mathcal{E}[\![e]\!].$$

図 5.3 Funiq の式に関する意味方程式

以上の式の意味関数 $\mathcal{E}$ に対して次の補題が成立する．この補題は Funiq の表明が表わす $\mathbf{D}$ 上の述語は常に完備である事を保証し，詳細化型に意味を与える上で最も基本的な役割を果たす．

補題 5.66 （式に関する形式的厳格性）

e を式，r' を e で形式的厳格な実現変数とする時，任意の $\zeta \in \mathbf{OEnv}$ と $\xi \in \mathbf{IEnv}$ とに対して，関数 $\boldsymbol{\lambda} d \in |\mathbf{D}|.\mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto d]\rangle$ は厳格である．

証明

証明は式 e の構造に関する帰納法を用いて，形式的厳格性の定義 2.5 の条件 (1) ∼ (11) の各場合について $\mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle = \bot_{\mathbf{D}}$ である事を示す．

(1) $e \equiv v$ の場合

この場合，形式的厳格性の定義から $v \equiv r'$ でなければならず，従って，$\mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle = \mathcal{E}[\![r']\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle = \bot_{\mathbf{D}}$ となり成立．

(2) $e \equiv \lambda x{:}\sigma.e_1$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$
\begin{aligned}
& \mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= \mathcal{E}[\![\lambda x{:}\sigma.e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= In_{\mathbf{F}}(\boldsymbol{\lambda} d \in |\mathbf{D}|.\mathcal{E}[\![e_1]\!]\langle\zeta[x \mapsto d], \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle) && \text{関数抽象式の意味方程式より} \\
&= In_{\mathbf{F}}(\boldsymbol{\lambda} d \in |\mathbf{D}|.\bot_{\mathbf{D}}) && r' \text{ は } e_1 \text{ で形式的厳格ゆえ} \\
& && \text{帰納法の仮定から} \\
&= In_{\mathbf{F}}(\bot_{\mathbf{F}}) && \bot_{\mathbf{F}} \triangleq \boldsymbol{\lambda} d \in |\mathbf{D}|.\bot_{\mathbf{D}} \text{ なので} \\
&= \bot_{\mathbf{D}}
\end{aligned}
$$

となるので成立．

(3) $e \equiv e_1 e_2$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$
\begin{array}{ll}
& \mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathcal{E}[\![e_1 e_2]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then} \\
& \quad Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(\mathcal{E}[\![e_2]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle) \\
& \mathsf{else} \\
& \quad wrong & \text{関数適用式の意味方程式より} \\
= & \mathsf{if}\ \ Is_{\mathbf{F}}(\bot_{\mathbf{D}})\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\bot_{\mathbf{D}})(\mathcal{E}[\![e_2]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle) \\
& \ \ \mathsf{else}\ \ wrong & r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
= & \mathsf{if}\ \ \bot_{\mathbf{D}}\ \ \mathsf{then}\ \ \cdots \text{中略} \cdots\ \ \mathsf{else}\ \ wrong \\
& & Is_{\mathbf{F}}(\bot_{\mathbf{D}}) = \bot_{\mathbf{D}} \text{ なので} \\
= & \bot_{\mathbf{D}} & \text{メタ言語での if 記法の条件部に関する厳格性より}
\end{array}
$$

となるので成立.

(4) $e \equiv \{l_1 = e_1, \ldots, l_n = e_n\}$ の場合

この場合，形式的厳格性の定義から r' は $e_1, \ldots, e_n$ 全てで形式的厳格である．従って，

$$
\begin{array}{lll}
& \mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle & \\
= & \mathcal{E}[\![\{l_1 = e_1, \ldots, l_n = e_n\}]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle & \\
= & In_{\mathbf{R}}(\boldsymbol{\lambda}_{\perp} l \in |\mathbf{L}|.\mathsf{if}\ l = l_1\ \mathsf{then}\ \mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle & \\
& \qquad \mathsf{elseif}\ \ldots & \\
& \qquad \mathsf{elseif}\ l = l_n\ \mathsf{then}\ \mathcal{E}[\![e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle & \\
& \qquad \mathsf{else}\ wrong) & \text{レコード式の意味方程式より} \\
= & In_{\mathbf{R}}(\boldsymbol{\lambda}_{\perp} l \in |\mathbf{L}|.\mathsf{if}\ l = l_1\ \mathsf{then}\ \bot_{\mathbf{D}} & \\
& \qquad \mathsf{elseif}\ \ldots & \\
& \qquad \mathsf{elseif}\ l = l_n\ \mathsf{then}\ \bot_{\mathbf{D}} & \\
& \qquad \mathsf{else}\ \bot_{\mathbf{D}}) & r' \text{ は各 } e_i \text{ で形式的厳格ゆえ} \\
& & \text{帰納法の仮定から} \\
= & In_{\mathbf{R}}(\boldsymbol{\lambda}_{\perp} l \in |\mathbf{L}|.\bot_{\mathbf{D}}) & \text{if 記法の全ての分岐で } \bot_{\mathbf{D}} \text{ なので} \\
= & In_{\mathbf{R}}(\bot_{\mathbf{R}}) & \bot_{\mathbf{R}} \triangleq \boldsymbol{\lambda}_{\perp} l \in |\mathbf{L}|.\bot_{\mathbf{D}} \text{ なので} \\
= & \bot_{\mathbf{D}} & In_{\mathbf{R}}(\bot_{\mathbf{R}}) = \bot_{\mathbf{D}} \text{ なので}
\end{array}
$$

となるので成立.

(5) $e \equiv e_1.l$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$
\begin{array}{ll}
& \mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathcal{E}[\![e_1.l]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathsf{if}\ \ Is_{\mathbf{R}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then}\ \ Out_{\mathbf{R}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(l) \\
& \quad \mathsf{else}\ \ wrong & \text{フィールド選択式の意味方程式より} \\
= & \mathsf{if}\ \ Is_{\mathbf{R}}(\bot_{\mathbf{D}})\ \ \mathsf{then}\ \ Out_{\mathbf{R}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(l) \\
& \quad \mathsf{else}\ \ wrong & r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
= & \mathsf{if}\ \ \bot_{\mathbf{D}}\ \ \mathsf{then}\ \ \cdots \text{中略} \cdots\ \ \mathsf{else}\ \ wrong \\
& & Is_{\mathbf{R}}(\bot_{\mathbf{D}}) = \bot_{\mathbf{D}} \text{ なので} \\
= & \bot_{\mathbf{D}} & \text{メタ言語での if 記法の条件部に関する厳格性より}
\end{array}
$$

となるので成立.

(6) $e \equiv \mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n$ の場合

この場合，形式的厳格性の定義から r' は e_0 で形式的厳格である．従って，

$$
\begin{array}{ll}
& \mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathcal{E}[\![\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathsf{if}\ \ Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then} \\
& \quad \mathsf{let}\ \ \langle l, v\rangle = Out_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{in} \\
& \qquad \mathsf{if}\ \ l = l_1\ \ \mathsf{then} \\
& \qquad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(v) \\
& \qquad\quad \mathsf{else}\ \ wrong \\
& \qquad \mathsf{elseif}\ \ \ldots \\
& \qquad \mathsf{elseif}\ \ l = l_n\ \ \mathsf{then} \\
& \qquad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(v) \\
& \qquad\quad \mathsf{else}\ \ wrong \\
& \qquad \mathsf{else}\ \ \bot_{\mathbf{D}} \\
& \quad \mathsf{end} \\
& \quad \mathsf{else}\ \ wrong & \text{タグ選択式の意味方程式より} \\
= & \mathsf{if}\ \ Is_{\mathbf{V}}(\bot_{\mathbf{D}})\ \ \mathsf{then}\ \ \cdots \text{中略} \cdots\ \ \mathsf{else}\ \ wrong \\
& & r' \text{ は } e_0 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
= & \mathsf{if}\ \ \bot_{\mathbf{D}}\ \ \mathsf{then}\ \ \cdots \text{中略} \cdots\ \ \mathsf{else}\ \ wrong \\
& & Is_{\mathbf{V}}(\bot_{\mathbf{D}}) = \bot_{\mathbf{D}} \text{ なので} \\
= & \bot_{\mathbf{D}} & \text{メタ言語での if 記法の条件部に関する厳格性より}
\end{array}
$$

となるので成立.

(7) $e \equiv \Lambda\, t{<:}\sigma.e_1$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$\begin{aligned}
&\mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= \mathcal{E}[\![\Lambda\, t{<:}\sigma.e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= \mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle && \text{型抽象式の意味方程式より} \\
&= \bot_{\mathbf{D}} && r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から}
\end{aligned}$$

となるので成立.

(8) $e \equiv e_1[\tau]$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$\begin{aligned}
&\mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= \mathcal{E}[\![e_1[\tau]]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= \mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle && \text{型適用式の意味方程式より} \\
&= \bot_{\mathbf{D}} && r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から}
\end{aligned}$$

となるので成立.

(9) $e \equiv \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$\begin{aligned}
&\mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= \mathcal{E}[\![\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&= \mathcal{E}[\![e_1[\rho](\Lambda\, t{<:}\sigma.\lambda\, x{:}\,\tau.e_2)]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
&\qquad\qquad && \text{定義 3.1 の } \mathbf{unpack} \text{ 式のコード化より} \\
&= \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1[\rho]]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then} \\
&\qquad Out_{\mathbf{F}}(\mathcal{E}[\![e_1[\rho]]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(\mathcal{E}[\![\Lambda\, t{<:}\sigma.\lambda\, x{:}\,\tau.e_2]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle) \\
&\quad \mathsf{else} \\
&\qquad \mathit{wrong} && \text{関数適用式の意味方程式より} \\
&= \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then} \\
&\qquad Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(\mathcal{E}[\![\lambda\, x{:}\,\tau.e_2]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle) \\
&\quad \mathsf{else}\ \ \mathit{wrong} && \mathcal{E} \text{ の各呼出しを整理} \\
&= \mathsf{if}\ \ Is_{\mathbf{F}}(\bot_{\mathbf{D}})\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\bot_{\mathbf{D}})(\mathcal{E}[\![\lambda\, x{:}\,\tau.e_2]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle) \\
&\quad \mathsf{else}\ \ \mathit{wrong} && r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
&= \mathsf{if}\ \ \bot_{\mathbf{D}}\ \ \mathsf{then}\ \ \cdots \text{中略} \cdots\ \ \mathsf{else}\ \ \mathit{wrong} \\
&\qquad\qquad && Is_{\mathbf{F}}(\bot_{\mathbf{D}}) = \bot_{\mathbf{D}} \text{ なので} \\
&= \bot_{\mathbf{D}} && \text{メタ言語での } \mathsf{if} \text{ 記法の条件部に関する厳格性より}
\end{aligned}$$

となるので成立.

(10) $e \equiv \mathbf{fix}\ e_1$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$
\begin{array}{ll}
& \mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathcal{E}[\![\mathbf{fix}\ e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle \\
= & \mathsf{if}\ \ \mathit{Is}_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{then} \\
& \quad \mathsf{let}\ \ f = \mathit{Out}_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{in}\ \ \mathit{fix}(f)\ \ \mathsf{end} \\
& \mathsf{else} \\
& \quad \mathit{wrong} & \text{不動点再帰式の意味方程式より} \\
= & \mathsf{if}\ \ \mathit{Is}_{\mathbf{F}}(\bot_{\mathbf{D}})\ \ \mathsf{then} \\
& \quad \mathsf{let}\ \ f = \mathit{Out}_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)\ \ \mathsf{in}\ \ \mathit{fix}(f)\ \ \mathsf{end} \\
& \mathsf{else} \\
& \quad \mathit{wrong} & r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
= & \bot_{\mathbf{D}}\ \ \mathsf{then}\ \ \cdots \text{中略} \cdots\ \ \mathsf{else}\ \ \mathit{wrong} \\
& & \mathit{Is}_{\mathbf{F}}(\bot_{\mathbf{D}}) = \bot_{\mathbf{D}} \text{ なので} \\
= & \bot_{\mathbf{D}} & \text{メタ言語での if 記法の条件部に関する厳格性より}
\end{array}
$$

となるので成立.

(11) $e \equiv e_1 : \tau$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$
\begin{array}{lll}
& \mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle & \\
= & \mathcal{E}[\![e_1 : \tau]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle & \\
= & \mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle & \text{型制約式の意味方程式より} \\
= & \bot_{\mathbf{D}} & r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から}
\end{array}
$$

となるので成立.

以上から全ての場合について補題の言明が成立する事が示された.

証明終

以上の補題から形式的厳格性が意味領域 $\mathbf{D}$ での関数の厳格性に対応する構文上の概念である事が判る．なお，上の証明から判る通り，通常変数が形式的厳格な場合も全く同様の言明が成立するが，以下での型／表明に対する表示的意味の定義では実現変数に関する上の補題だけで充分なので，通常変数に関する同様の補題は示さない.

5.2.3 型と表明の表示的意味

本項では完備部分同値関係という構造を用いて型に対する表示を与える．そこで，まず部分同値関係の概念と関連する記法とを定義する．

定義 5.67 （部分同値関係 per）

S を集合とする．この時，

(1) P が S 上の**部分同値関係 (per)** であるとは， P が S 上の対称的かつ推移的な 2 項関係だという事である．即ち， P は以下の 3 条件 (a) ∼ (c) を全て満たすという事である：

(a) $P \subseteq S \times S$;

(b) $\forall s_1, s_2 \in S.\big[\langle s_1, s_2\rangle \in P \implies \langle s_2, s_1\rangle \in P\big]$;

(c) $\forall s_1, s_2, s_3 \in S.$
$\big[\langle s_1, s_2\rangle \in P$ かつ $\langle s_2, s_3\rangle \in P \implies \langle s_1, s_3\rangle \in P\big]$.

(2) P を S 上の per とする時， P の**定義域** — $|\boldsymbol{P}|$ — を以下によって定義する：

$$|P| \triangleq \left\{ s \in S \;\middle|\; \exists s' \in S.\big[\langle s, s'\rangle \in P\big] \right\}.$$

任意の per はその定義域上の同値関係である．即ち，

補題 5.68 （per の反射性）

S を集合とし， P を S 上の per とする時， 2 項関係 P は $|P|$ 上で反射的である．即ち，

$$\forall s \in |P|.\big[\langle s, s\rangle \in P\big]$$

である．故に，

$$|P| = \{ s \in S \mid \langle s, s\rangle \in P \}.$$

証明

任意に与えられた $s \in |P|$ について， $s \in |P|$ であるので或る $s' \in S$ が存在して $\langle s, s' \rangle \in P$ である． P の対称性より $\langle s', s \rangle \in P$. 故に P の推移性より $\langle s, s \rangle \in P$. よって， P は $|P|$ 上で反射的である．

以上から $|P| \subseteq \{s \in S \mid \langle s, s \rangle \in P\}$ が示された．逆向きの包含関係は $|P|$ の定義から明らか．

証明終

定義 5.69 （per の制限）

P を S 上の per とし， X を S の部分集合とする時， P の X 上への**制限** — $\boldsymbol{P\lceil X}$ — を以下によって定義する：

$$P\lceil X \triangleq P \cap (X \times X) = \{\langle s_1, s_2 \rangle \in P \mid s_1, s_2 \in X\}.$$

Per P の部分集合上への制限は，その部分集合上の per である．即ち，

補題 5.70 （制限 per）

上の (3) で定義された $P\lceil X$ は X 上の — 故に S 上の — per である．

証明

$P\lceil X$ が定義 5.67 (1) の条件 (a) ∼ (c) の各々を満たす事を示せば良い．

(a) X 上の 2 項関係である事：

$P\lceil X = P \cap (X \times X) \subseteq X \times X \subseteq S \times S$ であるので， $P\lceil X$ は X 上および S 上の 2 項関係である．

(b) 対称的である事：

任意に $\langle s_1, s_2 \rangle \in P\lceil X$ を選べば， $P\lceil X$ の定義から $\langle s_1, s_2 \rangle \in P$ かつ $s_1, s_2 \in X$ である． $\langle s_1, s_2 \rangle \in P$ と P の対称性とから $\langle s_2, s_1 \rangle \in P$ である．これと $s_1, s_2 \in X$ である事とより $\langle s_2, s_1 \rangle \in P\lceil X$ を得る．故に， $P\lceil X$ は対称則 (b) を満たす．

(c) 推移的である事：

任意に $\langle s_1, s_2 \rangle, \langle s_2, s_3 \rangle \in P\lceil X$ を選べば， $P\lceil X$ の定義から $\langle s_1, s_2 \rangle, \langle s_2, s_3 \rangle \in P$ かつ $s_1, s_2, s_3 \in X$ である． $\langle s_1, s_2 \rangle, \langle s_2, s_3 \rangle \in P$ と P の推移性とから $\langle s_1, s_3 \rangle \in P$

である．これと $s_1, s_3 \in X$ である事とより $\langle s_1, s_3\rangle \in P\lceil X$ を得る．故に，$P\lceil X$ は推移則 (c) を満たす．

証明終

与えられた二つの集合間の関数の成す集合上にも，次の形で per を構成できる．

定義 5.71 （per の関数空間）

S, T を集合とし，P と Q とを各々 S 上と T 上との per とする．この時，P から Q への**関数空間** — $\boldsymbol{P \to Q}$ — を以下によって定義する：

$$\begin{array}{l}\langle f, f'\rangle \in P \to Q \\ \quad \stackrel{\text{def}}{\Longleftrightarrow} \\ f, f' : S \to T \text{ かつ } \forall d, d' \in S.\big[\langle d, d'\rangle \in P \implies \langle f(d), f'(d')\rangle \in Q\big].\end{array}$$

補題 5.72 （関数空間 per）

上で定義された $P \to Q$ は $S \to T$ 上の per である．

証明

$P \to Q$ が $S \to T$ 上の 2 項関係である事は定義から明らかなので，対称性と推移性とを共に満たす事を示す．

(a) 対称的である事：

$\langle f, f'\rangle \in P \to Q$ を任意に選ぶ．すると任意に与えられた $\langle d, d'\rangle \in P$ に対し $\langle f(d), f'(d')\rangle \in Q$ である．故に Q の対称性から $\langle f'(d'), f(d)\rangle \in Q$ である．一方，$\langle d, d'\rangle \in P$ と補題 5.68 とから $\langle d', d'\rangle \in P$ であるので，$P \to Q$ の定義より $\langle f(d'), f'(d')\rangle \in Q$．同様にして $\langle f(d), f'(d)\rangle \in Q$．$\langle f(d'), f'(d')\rangle \in Q$ と $\langle f'(d'), f(d)\rangle \in Q$ とに Q の推移性を適用し，$\langle f(d'), f(d)\rangle \in Q$．これと $\langle f(d), f'(d)\rangle \in Q$ とに Q の推移性を適用して $\langle f(d'), f'(d)\rangle \in Q$ を得るが，これに Q の対称性を適用すれば $\langle f'(d), f(d')\rangle \in Q$．これが任意の $\langle d, d'\rangle \in P$ に対して成立するので，$\langle f', f\rangle \in P \to Q$．即ち，$P \to Q$ は対称的である．

(b) 推移的である事：

$\langle f_1, f_2\rangle, \langle f_2, f_3\rangle \in P \to Q$ を任意に選ぶ．この時，任意に選ばれた $\langle d, d'\rangle \in P$ に対して，$\langle f_1(d), f_2(d')\rangle \in Q$．$\langle d, d'\rangle \in P$ と P の定義域での反射性より $\langle d', d'\rangle \in P$ なので $\langle f_2(d'), f_3(d')\rangle \in Q$．以上に Q の推移性を適用すれば $\langle f_1(d), f_3(d')\rangle \in Q$．これが任意の $\langle d, d'\rangle \in P$ に対して成立するので，$\langle f_1, f_3\rangle \in P \to Q$．即ち，$P \to Q$ は推移的である．

以上 (a), (b) より，$P \to Q$ は $S \to T$ 上の per である．

証明終

同一の集合上の任意個の per の共通部分はやはり per となる．即ち，

補題 5.73 （per の共通部分）
S を集合，I を添字集合とし，各 $i \in I$ について P_i は S 上の per とする．この時，これらの共通部分 $\bigcap_{i\in I} P_i$ も S 上の per である．

証明

$\bigcap_{i\in I} P_i$ が S 上の 2 項関係である事は明らかなので，対称性と推移性とを共に満たす事を順に示す．

(a) 対称的である事：

任意に $\langle s, s'\rangle \in \bigcap_{i\in I} P_i$ を選ぶ．すると，各 $i \in I$ について $\langle s, s'\rangle \in P_i$ である．各 P_i の対称性より $\langle s', s\rangle \in P_i$．よって，$\langle s', s\rangle \in \bigcap_{i\in I} P_i$ となるので，$\bigcap_{i\in I} P_i$ は対称的である．

(b) 推移的である事：

任意に $\langle s_1, s_2\rangle, \langle s_2, s_3\rangle \in \bigcap_{i\in I} P_i$ を選ぶ．すると各 $i \in I$ について $\langle s_1, s_2\rangle, \langle s_2, s_3\rangle \in P_i$ である．各 P_i の推移性より $\langle s_1, s_3\rangle \in P_i$．よって，$\langle s_1, s_3\rangle \in \bigcap_{i\in I} P_i$ となるので，$\bigcap_{i\in I} P_i$ は推移的である．

以上 (a), (b) より，$\bigcap_{i\in I} P_i$ は S 上の per である．

証明終

Cpo （の台集合）上の per にも以上の定義はそのまま適用できるが， cpo 上の per に関しては， cpo が与える順序構造や位相構造（上限操作に対する振舞）に対応する構造を付加する事が可能となる．次に示す完備部分同値関係は，それらの構造が付加された部分同値関係であり，本論文ではこの構造を型の表示として用いる．

定義 5.74 （完備部分同値関係 cper）

$\mathbf{X} = (X, \sqsubseteq_{\mathbf{X}})$ を任意の cpo とし， P を台集合 X 上の per とする時， P が**完備**であるとは， P が以下の 2 条件を満たす事である：

(a) 厳格性：$\langle \bot_{\mathbf{X}}, \bot_{\mathbf{X}} \rangle \in P$;

(b) ω-帰納性：$\{d_i\}_{i\in\omega}, \{d'_i\}_{i\in\omega}$ を共に $\mathbf{X}$ 中の任意の ω-上昇鎖とする時，

$$\forall i \in \omega.\big[\langle d_i, d'_i \rangle \in P\big] \quad \Longrightarrow \quad \left\langle \bigsqcup_{i\in\omega}^{\mathbf{X}} d_i, \bigsqcup_{i\in\omega}^{\mathbf{X}} d'_i \right\rangle \in P.$$

従って， P が cpo $\mathbf{X}$ 上の cper であるとは， $|P|$ が $\mathbf{X}$ 上の或る完備な述語を真とする値の成す $|\mathbf{X}|$ の部分集合となっている，と言い替え得る．更に，

定義 5.75 （部分 cpo）

$\mathbf{X} = (X, \sqsubseteq_{\mathbf{X}})$ を cpo とする時，半順序集合 $\mathbf{Y} = (Y, \sqsubseteq_{\mathbf{Y}})$ が $\mathbf{X}$ の**部分 cpo** であるとは，以下の各条件を満たす事である：

(a) $\mathbf{Y}$ は cpo である；

(b) $Y \subseteq X$;

(c) $\bot_{\mathbf{X}} \in Y$;

(d) $\sqsubseteq_{\mathbf{Y}} = \sqsubseteq_{\mathbf{X}} \cap (Y \times Y)$;

(e) $V \subseteq Y$ を $\mathbf{Y}$ 中の任意の有向集合（或いは ω-上昇鎖）とする時，

$$\bigsqcup^{\mathbf{Y}} V = \bigsqcup^{\mathbf{X}} V.$$

で定義される部分 cpo の概念を用いると， P が cpo $\mathbf{X}$ 上の cper であるとは $|P|$ が $\mathbf{X}$ の或る部分 cpo の台集合である事と同値である．

定義 5.69 では或る集合上の per の元の集合の部分集合上への制限もその部分集合上の per となる事が示されたが， cper の場合は完備性という性質が付加されている per であるので，その性質を保つ為には制限の土台としては完備な部分集合としての部分 cpo を用いれば良い．即ち，

補題 5.76（制限 cper）

$\mathbf{S}$ を cpo とし， P を $\mathbf{S}$ 上の cper， $\mathbf{X}$ を $\mathbf{S}$ の或る部分 cpo とする．この時，$P\lceil|\mathbf{X}|$ は $\mathbf{X}$ 上の — 故に $\mathbf{S}$ 上の — cper である．

証明

補題 5.70 より $P\lceil|\mathbf{X}|$ は $|\mathbf{X}|$ 上の — 故に $|\mathbf{S}|$ 上の — per である．従って， $P\lceil|\mathbf{X}|$ の完備性の為の条件である厳格性と ω-帰納性とを共に満たす事を示せば良い．

(a) 厳格である事：

P は $\mathbf{S}$ 上の cper であるので， $\langle\perp_{\mathbf{S}},\perp_{\mathbf{S}}\rangle \in P$ である．また， $\mathbf{X}$ は $\mathbf{S}$ の部分 cpo なので， $\perp_{\mathbf{S}} \in |\mathbf{X}|$ である．従って，制限の定義から $\langle\perp_{\mathbf{S}},\perp_{\mathbf{S}}\rangle \in P\lceil|\mathbf{X}|$ となり，$P\lceil|\mathbf{X}|$ は厳格である．

(b) ω-帰納的である事：

$\mathbf{X}$ 中の ω-上昇鎖 $\{d_i\}_{i\in\omega},\{d'_i\}_{i\in\omega}$ で各 $i\in\omega$ について $\langle d_i,d'_i\rangle \in P\lceil|\mathbf{X}|$ であるものを任意に選ぶ．この時， $\mathbf{X}$ が $\mathbf{S}$ の部分 cpo である事から $\{d_i\}_{i\in\omega},\{d'_i\}_{i\in\omega}$ は共に $\mathbf{S}$ 中の ω-上昇鎖でもあり，更に，それらの上限に関して $\bigsqcup^{\mathbf{S}}_{i\in\omega} d_i = \bigsqcup^{\mathbf{X}}_{i\in\omega} d_i \in |\mathbf{X}|$ および $\bigsqcup^{\mathbf{S}}_{i\in\omega} d'_i = \bigsqcup^{\mathbf{X}}_{i\in\omega} d'_i \in |\mathbf{X}|$ である．これと， P が $\mathbf{S}$ 上で ω-帰納的である事とから $\left\langle \bigsqcup^{\mathbf{X}}_{i\in\omega} d_i, \bigsqcup^{\mathbf{X}}_{i\in\omega} d'_i \right\rangle = \left\langle \bigsqcup^{\mathbf{S}}_{i\in\omega} d_i, \bigsqcup^{\mathbf{S}}_{i\in\omega} d'_i \right\rangle \in P$ である．従って，制限の定義から $\left\langle \bigsqcup^{\mathbf{X}}_{i\in\omega} d_i, \bigsqcup^{\mathbf{X}}_{i\in\omega} d'_i \right\rangle \in P\lceil|\mathbf{X}|$ となるので， $P\lceil|\mathbf{X}|$ は ω-帰納的である．

以上より， $P\lceil|\mathbf{X}|$ は $\mathbf{X}$ 上および $\mathbf{S}$ 上の cper である．

証明終

関数型の解釈に用いる連続関数空間 cper は以下の様に定義される．

定義 5.77 （cper の連続関数空間）

$\mathbf{X}, \mathbf{Y}$ を cpo とし， P と Q とを各々 $\mathbf{X}$ 上と $\mathbf{Y}$ 上との cper とする．この時,

(1) P から Q への**連続関数空間** — $[\boldsymbol{P \to Q}]$ — を以下によって定義する：

$$\langle f, f' \rangle \in [P \to Q] \overset{\text{def}}{\iff} f, f' : [\mathbf{X} \to \mathbf{Y}] \text{ かつ } \forall d, d' \in |\mathbf{X}|. \big[\langle d, d' \rangle \in P \implies \langle f(d), f'(d') \rangle \in Q \big].$$

(2) P から Q への**厳格連続関数空間** — $[\boldsymbol{P \to_\perp Q}]$ — を以下によって定義する：

$$\langle f, f' \rangle \in [P \to_\perp Q] \overset{\text{def}}{\iff} f, f' : [\mathbf{X} \to_\perp \mathbf{Y}] \text{ かつ } \forall d, d' \in |\mathbf{X}|. \big[\langle d, d' \rangle \in P \implies \langle f(d), f'(d') \rangle \in Q \big].$$

補題 5.78 （連続関数空間 cper）

(1) 上で定義された $[P \to Q]$ は $[\mathbf{X} \to \mathbf{Y}]$ 上の cper である．

(2) 上で定義された $[P \to_\perp Q]$ は $[\mathbf{X} \to_\perp \mathbf{Y}]$ 上の cper である．

証明

(1) について

補題 5.72 より $[P \to Q]$ が $[\mathbf{X} \to \mathbf{Y}]$ 上の per である事は保証されているので, $[P \to Q]$ が完備である為の条件としての厳格性と ω-帰納性とを順に示す.

(a) 厳格である事：

$\perp_{[\mathbf{X} \to \mathbf{Y}]} = \boldsymbol{\lambda} d \in |\mathbf{X}|. \perp_{\mathbf{Y}} \in |[P \to Q]|$ を示せば良いが， Q は cper なので特に厳格である．故に任意の $\langle d, d' \rangle \in P$ に対し $\langle \perp_{[\mathbf{X} \to \mathbf{Y}]}(d), \perp_{[\mathbf{X} \to \mathbf{Y}]}(d') \rangle = \langle \perp_{\mathbf{Y}}, \perp_{\mathbf{Y}} \rangle \in Q$．従って， $\langle \perp_{[\mathbf{X} \to \mathbf{Y}]}, \perp_{[\mathbf{X} \to \mathbf{Y}]} \rangle \in [P \to Q]$ であるので， $[P \to Q]$ は厳格である．

(b) ω-帰納的である事：

$[\mathbf{X} \to \mathbf{Y}]$ 中の ω-上昇鎖 $\{f_i\}_{i \in \omega}, \{f'_i\}_{i \in \omega}$ で各 $i \in \omega$ について $\langle f_i, f'_i \rangle \in [P \to Q]$ なるものを任意に選ぶ．この時，任意の $\langle d, d' \rangle \in P$ に対して， $i \in \omega$ の各々について $\langle f_i(d), f'_i(d') \rangle \in Q$ であり，また， $\{f_i\}_{i \in \omega}$ と $\{f'_i\}_{i \in \omega}$ とは共に $[\mathbf{X} \to \mathbf{Y}]$

中の ω-上昇鎖であるので，$\{f_i(d)\}_{i\in\omega}$ も $\{f'_i(d')\}_{i\in\omega}$ も双方共に $\mathbf{Y}$ 中の ω-上昇鎖を成している．従って，関数適用演算 app の連続性と Q の ω-帰納性とから $\left\langle \left(\bigsqcup_{i\in\omega}^{[\mathbf{X}\to\mathbf{Y}]} f_i\right)(d), \left(\bigsqcup_{i\in\omega}^{[\mathbf{X}\to\mathbf{Y}]} f'_i\right)(d') \right\rangle = \left\langle \bigsqcup_{i\in\omega}^{[\mathbf{X}\to\mathbf{Y}]} \left(f_i(d)\right), \bigsqcup_{i\in\omega}^{[\mathbf{X}\to\mathbf{Y}]} \left(f'_i(d')\right) \right\rangle \in Q$.
ここで，$\langle d, d'\rangle \in P$ は任意であったので，$\left\langle \bigsqcup_{i\in\omega}^{[\mathbf{X}\to\mathbf{Y}]} f_i, \bigsqcup_{i\in\omega}^{[\mathbf{X}\to\mathbf{Y}]} f'_i \right\rangle \in [P \to Q]$.
よって，$[P \to Q]$ は ω-帰納的である．

以上 (a), (b) より，$[P \to Q]$ は完備である．従って，$[P \to Q]$ は $[\mathbf{X} \to \mathbf{Y}]$ 上の cper である．

(2) について

(1) と同様.

証明終

補題 5.73 での各 per が同一 cpo 上の完備な per である場合，それら全ての共通部分はやはり cper となる．即ち，

補題 5.79（cper の共通部分）

$\mathbf{X}$ を cpo，I は添字集合とし，各 $i \in I$ について P_i は $\mathbf{X}$ 上の cper とする．
この時，これらの共通部分 $\bigcap_{i\in I} P_i$ も $\mathbf{X}$ 上の cper である．

証明

補題 5.73 より $\bigcap_{i\in I} P_i$ が $|\mathbf{X}|$ 上の per である事は保証されているので，$\bigcap_{i\in I} P_i$ が完備である為の条件としての厳格性と ω-帰納性とを順に示す.

(a) 厳格である事：

各 $i \in I$ に対し P_i は cper であるので特に厳格である．故に $\bot_{\mathbf{X}} \in |P_i|$．従って，$\bot_{\mathbf{X}} \in \bigcap_{i\in I} |P_i| = |\bigcap_{i\in I} P_i|$ となり，$\bigcap_{i\in I} P_i$ は厳格である．

(b) ω-帰納的である事：

各 $j \in \omega$ に対し $\langle d_j, d'_j\rangle \in \bigcap_{i\in I} P_i$ なる ω-上昇鎖 $\{d_j\}_{j\in\omega}, \{d'_j\}_{j\in\omega}$ を任意に選ぶ．この時，任意の $i \in I$ に対して，各 $j \in \omega$ について $\langle d_j, d'_j\rangle \in P_i$ が成り立つが，P_i の ω-帰納性より $\langle \bigsqcup_{j\in\omega}^{\mathbf{X}} d_j, \bigsqcup_{j\in\omega}^{\mathbf{X}} d'_j\rangle \in P_i$．これが全ての $i \in I$ について成立するので，$\langle \bigsqcup_{j\in\omega}^{\mathbf{X}} d_j, \bigsqcup_{j\in\omega}^{\mathbf{X}} d'_j\rangle \in \bigcap_{i\in I} P_i$．よって，$\bigcap_{i\in I} P_i$ は ω-帰納的である．

以上 (a), (b) より，$\bigcap_{i\in I} P_i$ は $\mathbf{X}$ 上の cper である．

証明終

型の表示となり得るのは実行時の型エラーを表わす値？（の $\mathbf{D}$ での像 $wrong$）を含まない $\mathbf{D}$ 上の cper である．即ち，

定義 5.80（型の表示領域）

(1) $\mathbf{D}$ 上の全ての cper の集まりを **CPER** で表わす．

(2) $\mathbf{D}$ 上の cper で定義域に $wrong$ を含まないもの全ての集まりを **TYPES** で表わす．即ち，

$$\mathbf{TYPES} \triangleq \{P \in \mathbf{CPER} \mid wrong \notin |P|\}.$$

補題 5.81（cpo としての TYPES）

半順序集合 $(\mathbf{TYPES}, \subseteq)$ は完備束（任意の部分集合に対して上限と下限とが存在する束）であり，故に cpo でもある．

証明

TYPES が $\subseteq$ に関して完備束を成す事を示す為には，任意の部分集合 $X \subseteq \mathbf{TYPES}$ の $\subseteq$ に関する下限が **TYPES** に含まれる事を示せば良い．

さて，X の $\subseteq$ に関する下限は共通部分 $\bigcap X$ である．$\bigcap X \in \mathbf{CPER}$ である事 — 即ち $\mathbf{D}$ 上の cper である事 — は補題 5.79 より成立．

また，任意の $P \in X$ に対して $P \in \mathbf{TYPES}$ であるから $wrong \notin |P|$．よって $wrong \notin \bigcap X$．

以上から $\bigcap X \in \mathbf{TYPES}$ であるので，**TYPES** は完備束である．

証明終

以上で定義した **TYPES** は連続関数空間，制限，共通部分といった cper の構成に関して閉じている．即ち，

補題 5.82（TYPES の閉包性）

(1) $P, Q \in \mathbf{TYPES}$ とする．この時，$In_{\mathbf{F}}([P \to Q]), In_{\mathbf{F}}([P \to_{\perp} Q]) \in \mathbf{TYPES}$. ここで，$In_{\mathbf{F}}([P \to Q])$ は以下で定義されるものである：

$$In_{\mathbf{F}}([P \to Q]) \triangleq \{\langle In_{\mathbf{F}}(f), In_{\mathbf{F}}(f')\rangle \mid \langle f, f'\rangle \in [P \to Q]\}.$$

$In_{\mathbf{F}}([P \to_{\perp} Q])$ に関しても同様.

(2) $P \in \mathbf{TYPES}$ とし，$\mathbf{S}$ を $\mathbf{D}$ の部分 cpo とする．この時，$P\lceil|\mathbf{S}| \in \mathbf{TYPES}$.

(3) $\{P_i\}_{i\in I} \subseteq \mathbf{TYPES}$ を cper の族（ここで I は添字集合）とする．この時，$\bigcap_{i\in I} P_i \in \mathbf{TYPES}$.

証明

何れに関しても，問題の各 cper の定義域が実行時型エラー ? の像を含まない事は明らか — 例えば (1) での入射演算の引数となっている各 cper は $\mathbf{F}$ 上のものであり $\mathbf{W}$ 上のではない等 — なので，以下，各 cper が $\mathbf{CPER}$ の要素である事を示せば良いが，それは以下に示す通り，今までに示した各補題から直ちに従う．

(1) について

補題 5.78 より.

(2) について

補題 5.76 より.

(3) について

補題 5.79 より.

証明終

■ 型と表明の意味関数

以下，$\mathbf{TYPES}$ に属する cper を用いて型の表示を与える．型の意味は，表明を含む詳細型の存在の為に，表明の意味に依存する．一方，表明の意味は全称化での通常変数の取り得る値の範囲を規定する型の意味に依存する．即ち，**表明の意味は（式の意味とは異なり）完全な型無し解釈にはできない**．その詳しい理由は次節の健全性定理 5.89 の証明の直後で述べる．この結果，型の意味関数 $\mathcal{T}$ と表明の意味関数 $\mathcal{A}$ とは相互再帰的に定義されている事を注意しておく．

$$\mathcal{A} : \mathbf{Assertion} \to \mathbf{Env} \to \mathbf{TEnv} \to \{true, false\}$$

$$\mathcal{A}[\![e_1 \leqslant e_2 : \sigma]\!]\varepsilon\eta = (\mathcal{E}[\![e_1]\!]\varepsilon \sqsubseteq_{\mathbf{D}} \mathcal{E}[\![e_2]\!]\varepsilon)$$

$$\mathcal{A}[\![\mathbf{forall}\, x{:}\,\sigma.\,\gamma]\!]\varepsilon\eta = \mathsf{let}\ \langle\zeta,\xi\rangle = \varepsilon\ \mathsf{in}\ \forall d \in |\mathcal{T}[\![\sigma]\!]\eta|.\mathcal{A}[\![\gamma]\!]\langle\zeta[x \mapsto d],\xi\rangle\eta\ \mathsf{end}.$$

図 5.4 Funiq の表明に関する意味方程式

型は型変数の自由な出現を含み得るので，型の意味を与えるには（式での自由変数に対したのと同様）自由型変数に対する付値 — **型環境**と呼ぶ — が必要である．この場合，型変数に割り当てられる「値」は **TYPES** の要素としての cper である．そこで，型環境全ての成す集まり **TEnv** を以下の様に定義する：

$$\eta \in \mathbf{TEnv} \triangleq [\mathbf{TVar}_\perp \to_\perp \mathbf{TYPES}]$$

なお，図 5.5 での $\mathcal{T}$ の定義から判る通り，$\mathcal{T}$ に対しては連続性を要請しない．故に型環境も連続である必要はなく，**TEnv** を単なる関数の集合 $\mathbf{TVar} \to \mathbf{TYPES}$ と定義しても本節の目的には適う．しかし，将来の発展 — 再帰型の導入等 — の為，上の様に定義しておく．なお，型環境の定義を写像の集合でなく厳格連続関数空間とする事は，型環境に対して新たな制限を科すものではない[7)]．

表明に関する意味関数 $\mathcal{A}$ を定める意味方程式を図 5.4 に示す．注意すべき点は，表明の表わす真理値は表明の構文要素の式の値が未定義であっても真か偽かで確定せねばならぬ点である．故に表明が（環境等が与えられ）最終的に表示する値の集合は真理値の cpo **T** でなくて $\mathbb{T} = \{true, false\}$ である．この事は，de Bakker (1980) で示されている通り，手続き的プログラムに対する Hoare の三つ組での表明がプログラムの停止性とは無関係に真か偽かを表わさねばならない事と同様である．

この $\mathcal{A}$ に対して次の補題が成立する．

7) まず，平坦 cpo を定義域とし cpo を値域とする厳格な関数は必ず連続である，という簡単な事実を注意しておく．よって，一般に集合 S を変域とし cpo **X** を値域とする全域関数の集合は，要素の各写像に対応 $\perp_{S_\perp} \mapsto \perp_{\mathbf{X}}$ を追加する事で，平坦 cpo $S_\perp$ から **X** への厳格連続関数空間に拡張できる．

$$\mathcal{T} : \mathbf{Type} \to \mathbf{TEnv} \to \mathbf{TYPES}$$

$$\mathcal{T}[\![\mathbf{Top}]\!]\eta = (|\mathbf{D}| \setminus \{wrong\}) \times (|\mathbf{D}| \setminus \{wrong\});$$

$$\mathcal{T}[\![\iota_i]\!]\eta = \{\langle In_{\mathbf{A}_i}(a), In_{\mathbf{A}_i}(a)\rangle \mid a \in |\mathbf{A}_i|\};$$

$$\mathcal{T}[\![t]\!]\eta = \eta(t);$$

$$\mathcal{T}[\![\sigma_1 \to \sigma_2]\!]\eta = In_{\mathbf{F}}\left([\mathcal{T}[\![\sigma_1]\!]\eta \to \mathcal{T}[\![\sigma_2]\!]\eta]\right);$$

$$\begin{aligned}\mathcal{T}[\![\{l_1{:}\,\sigma_1, \ldots, l_n{:}\,\sigma_n\}]\!]\eta = {} & \left(In_{\mathbf{R}}(|\mathbf{R}|) \times In_{\mathbf{R}}(|\mathbf{R}|)\right) \\ & \cap \bigcap_{i=1}^{n} \{\langle In_{\mathbf{R}}(q), In_{\mathbf{R}}(q')\rangle \mid q, q' \in |\mathbf{R}| \text{ かつ } \langle q(l_i), q'(l_i)\rangle \in \mathcal{T}[\![\sigma_i]\!]\eta\};\end{aligned}$$

$$\mathcal{T}[\![[\,l_1{:}\,\sigma_1, \ldots, l_n{:}\,\sigma_n\,]]\!]\eta = \langle \bot_{\mathbf{D}}, \bot_{\mathbf{D}}\rangle \cup \bigcup_{i=1}^{n} \{\langle In_{\mathbf{V}}(\langle l_i, v\rangle), In_{\mathbf{V}}(\langle l_i, v'\rangle)\rangle \mid \langle l_i, v\rangle, \langle l_i, v'\rangle \in |\mathbf{V}| \text{かつ} \langle v, v'\rangle \in \mathcal{T}[\![\sigma_i]\!]\eta\};$$

$$\mathcal{T}[\![\forall t{<:}\sigma.\tau]\!]\eta = \bigcap \{\mathcal{T}[\![\tau]\!]\eta[t \mapsto P] \mid P \in |\mathbf{TYPES}| \text{ かつ } P \subseteq \mathcal{T}[\![\sigma]\!]\eta\};$$

$$\begin{aligned}\mathcal{T}[\![\{r{:}\,\sigma \mid \gamma_1, \ldots, \gamma_m\}]\!]\eta = {} & \mathsf{let}\ \ P = \mathcal{T}[\![\sigma]\!]\eta \\ & \mathsf{and}\ \ S = \left\{ d \in |\mathbf{D}| \;\middle|\; \bigwedge_{i=1}^{m} \mathcal{A}[\![\gamma_i]\!]\langle \bot_{\mathbf{OEnv}}, \bot_{\mathbf{IEnv}}[r \mapsto d]\rangle\eta = true \right\} \\ & \mathsf{in}\ \ P\lceil S\ \ \mathsf{end}.\end{aligned}$$

図 5.5 Funiq の型に関する意味方程式

補題 5.83 （表明に関する形式的厳格性）

γ を 2.1 節の定義 2.8 の条件 (b) を満たす表明とする．即ち，或る実現変数 r が γ の左辺式で形式的に厳格であるとする．この時，任意の $\zeta \in \mathbf{OEnv}$, $\xi \in \mathbf{IEnv}$, $\eta \in \mathbf{TEnv}$ に対して， $\mathcal{A}[\![\gamma]\!]\langle\zeta, \xi[r \mapsto \bot_{\mathbf{D}}]\rangle\eta = true$ である．

証明

γ が原始表明の場合，補題 5.66 より γ の左辺式は環境 $\langle\zeta, \xi[r' \mapsto d]\rangle$ で $\bot_{\mathbf{D}}$ を表示する．

一般の形の γ の場合，全称化 **forall** の数に関する帰納法で示す． **forall** による全称化表明の意味方程式右辺は， **forall** で束縛された通常変数に対する値に関する $\forall$ による（メタ言語での）全称化の形である．この $\forall$-全称化の本体は帰納法の仮定から問題の環境で常に $true$ なので，意味方程式右辺全体も $true$ である．

証明終

表明に対する $\mathcal{A}$ を用い，型の意味関数 $\mathcal{T}$ は図 5.5 に示す意味方程式で定義できる．

図 5.5 で与えられた意味関数 $\mathcal{T}$ は整定である．即ち，

定理 5.84 （$\mathcal{T}$ の整定性）
任意の型 $\rho \in \mathbf{Type}$ と型環境 $\eta \in \mathbf{TEnv}$ とに対して， $\mathcal{T}[\![\rho]\!]\eta \in \mathbf{TYPES}$.

証明

図 5.5 の場合も図 5.7 の場合も同様の証明となるので，名前呼びの図 5.5 の $\mathcal{T}$ に対して示す．補題 5.82 と補題 5.83 とを用いて型 ρ に関する構造帰納法で証明する．

(1) $\rho \equiv \mathbf{Top}$ の場合

(2) $\rho \equiv \iota_i$ の場合

これらの場合は明らか.

(3) $\rho \equiv t$ の場合

$\mathcal{T}[\![\rho]\!]\eta = \eta(t) \in \mathbf{TYPES}$.

(4) $\rho \equiv \sigma_1 \to \sigma_2$ の場合

帰納法の仮定より $\mathcal{T}[\![\sigma_1]\!]\eta, \mathcal{T}[\![\sigma_2]\!]\eta \in \mathbf{TYPES}$．これらに補題 5.82 (1) を適用.

(5) $\rho \equiv \{l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\}$ の場合

各 $i = 1, \ldots, n$ について意味方程式右辺の

$$\{\langle In_{\mathbf{R}}(q), In_{\mathbf{R}}(q')\rangle \mid q, q' \in |\mathbf{R}| \text{ かつ } \langle q(l_i), q'(l_i)\rangle \in \mathcal{T}[\![\sigma_i]\!]\eta\}$$

が **TYPES** の要素である諸条件 — 対称性・推移性・厳格性・ ω-帰納性・ $wrong$ を含まぬ事 — は， $\mathcal{T}[\![\sigma_i]\!]\eta$ に帰納法の仮定を適用し得られる対応する条件から従う．それら n 個の cper の共通部分が **TYPES** の要素である事は補題 5.82 (3) による.

(6) $\rho \equiv [\,l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\,]$ の場合

厳格性に関しては， $\langle \perp_{\mathbf{D}}, \perp_{\mathbf{D}}\rangle$ を要素として含む事から直ちに成立する．残りの条件に関しては以下の通りである.

各 $i = 1, \ldots, n$ について，意味方程式右辺の

$$P_i \triangleq \{\langle in_{\mathbf{V}}(\langle l_i, v\rangle), in_{\mathbf{V}}(\langle l_i, v'\rangle)\rangle \mid \langle l_i, v\rangle, \langle l_i, v'\rangle \in |\mathbf{V}| \text{ かつ } \langle v, v'\rangle \in \mathcal{T}[\![\sigma_i]\!]\eta\}$$

が残りの各条件を満たす事は $\mathcal{T}[\![\sigma_i]\!]\eta$ に関する対応する条件（帰納法の仮定の適用で得られる）から簡単に従うので，これら n 個の P_i は何れも cper である．

各 $i \neq j$ について $l_i \neq l_j$ であるので，$|P_i| \cap |P_j| = \varnothing$ である．故に，相異なる P_i, P_j に属する 2 項対同士が推移則の対象となる事はあり得ず，これら二つの per の双方に跨る ω-上昇鎖も存在しない．従って，この場合には cper の条件を乱さずに以上 n 個の cper（各 P_i は厳格でないので単なる per であるが $\langle \bot_{\mathbf{D}}, \bot_{\mathbf{D}} \rangle \cup P_i$ は cper である事に注意）の合併を取る事が許され，それらの合併も **CPER** の要素である．

最後に，それらの合併が *wrong* を含まない事に関しては，帰納法の仮定より各 i について $\mathit{wrong} \notin |P_i|$ である事から直ちに従う．

(7) $\rho \equiv \forall t{<}{:}\sigma.\tau$ の場合

帰納法の仮定から $\mathcal{T}[\![\tau]\!]\eta[t \mapsto P] \in \mathbf{TYPES}$ であるので，これを前提として補題 5.82 (3) を適用すれば良い．

(8) $\rho \equiv \{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}$ の場合

意味方程式右辺中の S が厳格である事は補題 5.83 より直ちに従う．S の ω-帰納性に関しては，式の表示が連続関数である事，つまり任意の $e \in \mathbf{Exp}$ に対して $\mathcal{E}[\![e]\!] : [\mathbf{Env} \to \mathbf{D}]$ である事から直ちに導かれる．故に，S は $\mathbf{D}$ の部分 cpo（の台集合）である．一方，同じく右辺中の $P = \mathcal{T}[\![\sigma]\!]\eta$ は帰納法の仮定から **TYPES** の要素である．従って，以上に補題 5.82 (2) を適用すれば求める結論が得られる．

証明終

5.3　型理論 FUNIQ の健全性

本節では，3.2 節で定義した型理論 FUNIQ が前節で与えた Funiq の表示的意味に対して健全である事を示す．その準備として Funiq の表示的意味に対する判別式の妥当性の概念を定義する．

定義 5.85（妥当性）

(1) 型環境 η が型変数に対する制約集合 C を**尊重する**（記法：$\boldsymbol{\eta \models C}$）とは，任意の制約 $(t <: \sigma) \in C$ に対して $\eta(t) \subseteq \mathcal{T}[\![\sigma]\!]\eta$ なる事である．

(2) 型環境 η と環境 $\varepsilon = \langle \zeta, \xi \rangle$ の対が型付け基底 Γ, Δ を**尊重する**（記法：$\boldsymbol{\eta, \varepsilon \models \Gamma, \Delta}$ もしくは $\boldsymbol{\eta, \zeta, \xi \models \Gamma, \Delta}$）とは，以下の 2 条件を共に満たす事である：

(a) $\eta, \zeta \models \Gamma$．即ち，$x \in \mathrm{dom}(\Gamma)$ なる任意の通常変数 x について $\zeta(x) \in |\mathcal{T}[\![\Gamma(x)]\!]\eta|$；かつ

(b) $\eta, \xi \models \Delta$．即ち，$r \in \mathrm{dom}(\Delta)$ なる任意の実現変数 r について $\xi(r) \in |\mathcal{T}[\![\Delta(r)]\!]\eta|$．

(3) 型環境 η と環境 $\varepsilon = \langle \zeta, \xi \rangle$ の対が型付け文脈 C, Γ, Δ を**尊重する**（記法：$\boldsymbol{\eta, \varepsilon \models C, \Gamma, \Delta}$ もしくは $\boldsymbol{\eta, \zeta, \xi \models C, \Gamma, \Delta}$）とは，$\eta \models C$ かつ $\eta, \varepsilon \models \Gamma, \Delta$ という事である．

(4) Σ を FUNIQ の判別式とする時，Σ が型環境 η と環境 ε の下で**充足される**（記法：$\boldsymbol{\eta, \varepsilon \models \Sigma}$）とは，$\Sigma$ のクラスに応じて以下の何れかの条件が満たされる事である：

(a) $\Sigma \equiv C \triangleright \sigma <: \tau$ の場合

$$\eta \models C \quad \Longrightarrow \quad \mathcal{T}[\![\sigma]\!]\eta \subseteq \mathcal{T}[\![\tau]\!]\eta;$$

(b) $\Sigma \equiv C, \Gamma, \Delta \triangleright e : \sigma$ の場合

$$\eta, \varepsilon \models C, \Gamma, \Delta \quad \Longrightarrow \quad \mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\sigma]\!]\eta|;$$

(c) $\Sigma \equiv C, \Gamma, \Delta \triangleright \gamma$ の場合

$$\eta, \varepsilon \models C, \Gamma, \Delta \quad \Longrightarrow \quad \mathcal{A}[\![\gamma]\!]\varepsilon\eta = \mathit{true}.$$

(5) Σ を FUNIQ の判別式とする時，Σ が**妥当**である（記法：$\boldsymbol{\models \Sigma}$）とは，任意の型環境 $\eta \in \mathbf{TEnv}$ と環境 $\varepsilon \in \mathbf{Env}$ とに対して $\eta, \varepsilon \models \Sigma$ という事である．

(6) $\Sigma, \Sigma_1, \ldots, \Sigma_n$ $(n \geq 0)$ を FUNIQ の判別式とする時，Σ が $\Sigma_1, \ldots, \Sigma_n$ に対して**相対的に妥当**である（記法：$\boldsymbol{\Sigma_1, \ldots, \Sigma_n \models \Sigma}$）とは，任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して以下が満たされる事である：

$$\Big(\forall i \in \{1, \ldots, n\}.\big[\eta, \varepsilon \models \Sigma_i\big]\Big) \quad \Longrightarrow \quad \eta, \varepsilon \models \Sigma.$$

次の補題は，式中の変数への構文的代入が環境での値の結合と対応付く事を示す．λ-計算等の変数束縛を含む言語に表示的意味を与える上で極めて基本的な命題である．

補題 5.86 （式に関する代入補題）

(1) $\vdash C,\Gamma[x:\sigma'],\Delta \triangleright e:\sigma$ かつ $\vdash C,\Gamma,\Delta \triangleright e':\sigma'$ ならば，
$\eta,\zeta,\xi \models \Gamma,\Delta$ なる任意の $\langle\zeta,\xi\rangle \in \mathbf{Env}$ に対して，
$$\mathcal{E}[\![e[x:=e']]\!]\langle\zeta,\xi\rangle = \mathcal{E}[\![e]\!]\langle\zeta[x\mapsto\mathcal{E}[\![e']\!]\langle\zeta,\xi\rangle],\xi\rangle.$$

(2) $\vdash C,\Gamma,\Delta[r:\sigma'] \triangleright e:\sigma$ かつ $\vdash C,\Gamma,\Delta \triangleright e':\sigma'$ ならば，
$\eta,\zeta,\xi \models \Gamma,\Delta$ なる任意の $\langle\zeta,\xi\rangle \in \mathbf{Env}$ に対して，
$$\mathcal{E}[\![e[r:=e']]\!]\langle\zeta,\xi\rangle = \mathcal{E}[\![e]\!]\langle\zeta,\xi[r\mapsto\mathcal{E}[\![e']\!]\langle\zeta,\xi\rangle]\rangle.$$

証明

(1), (2) 共に式 e の構造に関する帰納法によって証明する．(1) で e が関数抽象式（λ-抽象）以外の場合，並びに，(2) での全ての場合，帰納法の仮定の適用から直線的に求める結論が得られる．

問題は (1) で e が関数抽象式の場合であるが，その場合の証明は様々な文献で示されている．例えば，Barendregt (1984) の Lemma 5.3.3 で型無し λ-計算の場合に与えられた証明は，本論文の場合にも（細かな記法等の変更を別にすれば）そのまま通用するので，証明は省略する．

証明終

表明に関しても同様の言明が成立する．

補題 5.87 （表明に関する代入補題）

(1) $\vdash C,\Gamma[x:\sigma],\Delta \triangleright \gamma$ かつ $\vdash C,\Gamma,\Delta \triangleright e:\sigma$ ならば，$\eta,\zeta,\xi \models C,\Gamma,\Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\langle\zeta,\xi\rangle \in \mathbf{Env}$ とに対して，
$$\mathcal{A}[\![\gamma[x:=e]]\!]\langle\zeta,\xi\rangle\eta = \mathcal{A}[\![\gamma]\!]\langle\zeta[x\mapsto\mathcal{E}[\![e]\!]\langle\zeta,\xi\rangle],\xi\rangle\eta.$$

(2) $\vdash C,\Gamma,\Delta[r:\sigma] \triangleright \gamma$ かつ $\vdash C,\Gamma,\Delta \triangleright e:\sigma$ ならば，$\eta,\zeta,\xi \models C,\Gamma,\Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\langle\zeta,\xi\rangle \in \mathbf{Env}$ とに対して，
$$\mathcal{A}[\![\gamma[r:=e]]\!]\langle\zeta,\xi\rangle\eta = \mathcal{A}[\![\gamma]\!]\langle\zeta,\xi[r\mapsto\mathcal{E}[\![e]\!]\langle\zeta,\xi\rangle]\rangle\eta.$$

証明

(1), (2) 共に直前の補題を用いて表明 γ の構造に関する帰納法で示せば良い.

証明終

型に関しても型変数の構文的代入について同様の言明が成立する.

補題 5.88 （型に関する代入補題）

$\mathrm{FTV}(\sigma,\tau) \subseteq \mathrm{dom}(\eta)$ なる任意の型環境 $\eta \in \mathbf{TEnv}$ に対して,

$$\mathcal{T}[\![\sigma[t := \tau]]\!]\eta = \mathcal{T}[\![\sigma]\!]\eta[t \mapsto \mathcal{T}[\![\tau]\!]\eta].$$

証明

型 σ の構造に関する帰納法で示せば良い.

証明終

必要な準備が整ったので, FUNIQ の健全性を主張する定理の証明を与える.

定理 5.89 （健全性定理）

理論 FUNIQ は意味関数 $\mathcal{E}, \mathcal{A}, \mathcal{T}$ で与えられる名前呼びの Funiq の表示的意味に対して健全である. 即ち, 任意の FUNIQ の判別式 $\Sigma, \Sigma_1, \ldots, \Sigma_n$ $(n \geq 0)$ に対して,

$$\Sigma_1, \ldots, \Sigma_n \vdash \Sigma \quad \Longrightarrow \quad \Sigma_1, \ldots, \Sigma_n \models \Sigma.$$

証明

左辺の導出に関する帰納法によって証明する.

(A) 帰納法のベース：

この場合, 判別式 Σ は FUNIQ の何れかの公理（のインスタンス）でなければならない. 従って, 図 2.2 ～ 図 2.4 の各公理の妥当性を確認すれば良い. 以下, 公理 — 即ち, 判別式 Σ — の種類別に, その妥当性を確認する.

(A1) Σ が図 2.2 の部分型関係に関する公理の場合

● [S-TOP] 則の場合

定理 5.84 より任意の型 σ の表示する cper の定義域は *wrong* を含まない，という事から直ちに従う．

● [S-IDENTITY1], [S-IDENTITY2], [S-TVAR] 則の場合

明らか．

(A2) Σ が図 2.3 の型付けに関する公理の場合

● [T-CONST] 則の場合

意味関数 $\mathcal{K}_i$ の値域が $\mathbf{A}_i$ である事から直ちに従う．

● [T-VAR] 則の場合

示すべき事は，$\eta,\zeta,\xi \models C,\Gamma[x:\sigma],\Delta$ を満足する任意の型環境 $\eta \in \mathbf{TEnv}$ と環境 $\langle\zeta,\xi\rangle \in \mathbf{Env}$ とに対して $\mathcal{E}[\![x]\!]\langle\zeta,\xi\rangle \in |\mathcal{T}[\![\sigma]\!]\eta|$ が成立する事，つまり $\zeta(x) \in |\mathcal{T}[\![\sigma]\!]\eta|$ が成り立つ事である．

条件より $\eta,\zeta \models \Gamma[x:\sigma]$ であるので，任意の変数 $y \in \mathrm{dom}(\Gamma[x:\sigma])$ に対して $\zeta(y) \in |\mathcal{T}[\![(\Gamma[x:\sigma])(y)]\!]\eta|$ が成立している．そこで特に $y \equiv x$ と選べば $\zeta(x) \in |\mathcal{T}[\![(\Gamma[x:\sigma])(x)]\!]\eta| = |\mathcal{T}[\![\sigma]\!]\eta|$ となり，求める結論が得られる．[T-IVAR] 則の場合も全く同様である．

(A3) Σ が図 2.4 の表明に関する公理の場合

明らか．

(B) 帰納法のステップ：

Σ の導出の最後に適用した推論規則によって場合分けする．

(B1) Σ が部分型判別式の場合

Σ の導出の最後に適用された規則は図 2.2 の部分型関係に関する推論規則の何れかでなければならない．それらの規則の中で [S-REFINE] 則以外に関しては特に問題なく証明できるので，以下，[S-REFINE] 則の場合についてのみ証明する．

● [S-REFINE] 則の場合

示すべきは，以下の事柄である．即ち，$\Sigma_1, \ldots, \Sigma_n$ の全てを充足する型環境 $\eta_0 \in \mathbf{TEnv}$ と環境 $\varepsilon_0 \in \mathbf{Env}$ とが任意に与えられたとして，

$$\eta_0, \varepsilon_0 \models C \triangleright \{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\} <: \{r{:}\tau \mid \delta_1, \ldots, \delta_n\} \tag{1}$$

である事を以下の (a), (b) という二つの事柄を仮定して示せば良い：

(a) $\eta_0, \varepsilon_0 \models C \triangleright \sigma <: \tau$ （帰納法の仮定を適用）．或いは，これと等価な

$$\eta_0 \models C \implies \mathcal{T}[\![\sigma]\!]\eta_0 \subseteq \mathcal{T}[\![\tau]\!]\eta_0; \tag{2}$$

(b) 任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して，もしも

- $\eta, \varepsilon \models C \triangleright \sigma <: \tau$，かつ，
- 各 $1 \leq i \leq m$ に対して $\eta, \varepsilon \models C, \varnothing, \{r : \sigma\} \triangleright \gamma_i$

であるならば，$\eta, \varepsilon \models C, \varnothing, \{r : \tau\} \triangleright \delta_j$ が各 $1 \leq j \leq n$ に対して成立する．或いは，この条件の代わりに，定義からこれと等価な

$$\begin{aligned}
&\forall \eta \in \mathbf{TEnv}, \varepsilon \in \mathbf{Env}. \\
&\left[\left(\left(\eta \models C \implies \mathcal{T}[\![\sigma]\!]\eta \subseteq \mathcal{T}[\![\tau]\!]\eta\right) \text{かつ} \right.\right. \\
&\qquad \left.\left. \bigwedge_{i=1}^{m} \left(\eta, \varepsilon \models C, \varnothing, \{r : \sigma\} \implies \mathcal{A}[\![\gamma_i]\!]\varepsilon\eta = \mathit{true}\right) \right) \right. \\
&\qquad \implies \\
&\qquad \left. \left(\bigwedge_{j=1}^{n} \left(\eta, \varepsilon \models C, \varnothing, \{r : \tau\} \implies \mathcal{A}[\![\delta_j]\!]\varepsilon\eta = \mathit{true}\right) \right) \right].
\end{aligned} \tag{3}$$

従って，(2), (3) を仮定して (1) を示せば良い．

故に，(2), (3) を仮定して，充足性の定義に基づき (1) を書き直した

$$\eta_0 \models C \implies \mathcal{T}[\![\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}]\!]\eta_0 \subseteq \mathcal{T}[\![\{r{:}\tau \mid \delta_1, \ldots, \delta_n\}]\!]\eta_0 \tag{4}$$

を示せば良いが，この (4) は，詳細化型に対する意味方程式より，

$$\eta_0 \models C \quad \Longrightarrow \quad (P_\sigma \lceil S_\sigma) \subseteq (P_\tau \lceil S_\tau) \tag{5}$$

ここで

$$P_\sigma \triangleq \mathcal{T}[\![\sigma]\!]\eta_0,$$

$$S_\sigma \triangleq \left\{ v \in \mathbf{D} \;\middle|\; \bigwedge_{i=1}^{m} \mathcal{A}[\![\gamma_i]\!]\langle \perp_{\mathbf{OEnv}}, \perp_{\mathbf{IEnv}}[r \mapsto v]\rangle \eta_0 = true \right\},$$

$$P_\tau \triangleq \mathcal{T}[\![\tau]\!]\eta_0,$$

$$S_\tau \triangleq \left\{ v \in \mathbf{D} \;\middle|\; \bigwedge_{j=1}^{n} \mathcal{A}[\![\delta_j]\!]\langle \perp_{\mathbf{OEnv}}, \perp_{\mathbf{IEnv}}[r \mapsto v]\rangle \eta_0 = true \right\}$$

と等価である．従って，以下，この (5) を (2), (3) から示す事を目標とする．

$\eta_0 \not\models C$ の時は (5) は自明に成立する．故に $\eta_0 \models C$ であると仮定する．すると，(2) より，

$$\mathcal{T}[\![\sigma]\!]\eta_0 \subseteq \mathcal{T}[\![\tau]\!]\eta_0 \tag{6}$$

を得る． $\eta_0 \models C$ である事と (6) と (3) とから，

$$\forall \varepsilon \in \mathbf{Env}. \Bigg(\Big(\eta_0, \varepsilon \models \varnothing, \{r : \sigma\} \quad \Longrightarrow \quad \bigwedge_{i=1}^{m} \mathcal{A}[\![\gamma_i]\!]\varepsilon\eta_0 = true \Big) \\ \Longrightarrow \quad \Big(\eta_0, \varepsilon \models \varnothing, \{r : \tau\} \quad \Longrightarrow \quad \bigwedge_{j=1}^{n} \mathcal{A}[\![\delta_j]\!]\varepsilon\eta_0 = true \Big) \Bigg) \tag{7}$$

を得る．

任意に $d_0 \in |P_\sigma \lceil S_\sigma|$ を選ぶ．即ち，

$$d_0 \in |P_\sigma| = |\mathcal{T}[\![\sigma]\!]\eta_0|, \tag{8}$$

かつ，

$$\bigwedge_{i=1}^{m} \mathcal{A}[\![\gamma_i]\!]\langle \perp_{\mathbf{OEnv}}, \perp_{\mathbf{IEnv}}[r \mapsto d_0]\rangle \eta_0 = true. \tag{9}$$

すると，(6) と (8) とから，

$$d_0 \in |P_\tau| = |\mathcal{T}[\![\tau]\!]\eta_0|. \tag{10}$$

即ち，

$$\eta_0, \langle \bot_{\mathbf{OEnv}}, \bot_{\mathbf{IEnv}}[r \mapsto d_0] \rangle \models \varnothing, \{r : \tau\} \tag{11}$$

である．

従って，(7), (9), (11) より，

$$\bigwedge_{j=1}^{n} \mathcal{A}[\![\delta_j]\!]\langle \bot_{\mathbf{OEnv}}, \bot_{\mathbf{IEnv}}[r \mapsto d_0] \rangle \eta_0 = true \tag{12}$$

である．

(10) と (12) とから，$d_0 \in |P_\tau \lceil S_\tau|$ である．以上より (5) が示され，よって，(1) が証明された．

(B2) Σ が型付け判別式の場合

Σ の導出の最後に適用された規則は図 2.3 の型付けに関する推論規則の何れかでなければならない．

● [T-WEAK], [T-IWEAK], [T-TWEAK] 則（型付け文脈の特定の成分に要素対の追加を行なう推論規則）の場合

各規則での帰結側の判別式の型付け文脈に対する条件の方が強くなるので明らか．

● [T-SUBTYPE] 則の場合

帰納法の仮定を適用する事で直ちに示せる．

● [T-ABS] 則の場合

示すべき事は，$\eta, \zeta, \xi \models C, \Gamma[x : \sigma], \Delta$ を満足する任意の $\eta \in \mathbf{TEnv}$, $\zeta \in \mathbf{OEnv}$, $\xi \in \mathbf{IEnv}$ の組合せに対して $\mathcal{E}[\![e]\!]\langle \zeta, \xi \rangle \in |\mathcal{T}[\![\tau]\!]\eta|$ が成り立つ，という仮定の下で，$\eta', \zeta', \xi' \models C, \Gamma, \Delta$ を満足する任意の $\eta' \in \mathbf{TEnv}$, $\zeta' \in \mathbf{OEnv}$, $\xi' \in \mathbf{IEnv}$ の組合せに対して $\mathcal{E}[\![\lambda\, x{:}\sigma.e]\!]\langle \zeta', \xi' \rangle \in |\mathcal{T}[\![\sigma \to \tau]\!]\eta'|$ が成立する，という事である．

さて，図 5.5 の関数型に対する意味方程式と定義 5.77 (1) とから，

$$\begin{aligned}
&\mathcal{T}[\![\sigma \to \tau]\!]\eta' \\
&= In_{\mathbf{F}}([\mathcal{T}[\![\sigma]\!]\eta' \to \mathcal{T}[\![\tau]\!]\eta']) \\
&= \{\langle In_{\mathbf{F}}(f), In_{\mathbf{F}}(f')\rangle \mid \langle f, f'\rangle \in [\mathcal{T}[\![\sigma]\!]\eta' \to \mathcal{T}[\![\tau]\!]\eta']\} \\
&= \left\{\langle In_{\mathbf{F}}(f), In_{\mathbf{F}}(f')\rangle \mid \forall\langle d, d'\rangle \in \mathcal{T}[\![\sigma]\!]\eta'.\left[\langle f(d), f'(d')\rangle \in \mathcal{T}[\![\tau]\!]\eta'\right]\right\}
\end{aligned}$$

であるので，$\mathcal{E}[\![\lambda\, x{:}\sigma.e]\!]\langle\zeta', \xi'\rangle \in |\mathcal{T}[\![\sigma \to \tau]\!]\eta'|$ を示すには，

$$\forall d \in |\mathcal{T}[\![\sigma]\!]\eta'|.\left[Out_{\mathbf{F}}(\mathcal{E}[\![\lambda\, x{:}\sigma.e]\!]\langle\zeta', \xi'\rangle)(d) \in |\mathcal{T}[\![\tau]\!]\eta'|\right]$$

が成り立つ，という事を示せばよい．この式を図 5.3 の関数抽象式に対する意味方程式を用いて展開し整理すると，

$$\forall d \in |\mathcal{T}[\![\sigma]\!]\eta'|.\left[\mathcal{E}[\![e]\!]\langle\zeta'[x \mapsto d], \xi'\rangle \in |\mathcal{T}[\![\tau]\!]\eta'|\right] \tag{13}$$

となる．

今，$d_0 \in |\mathcal{T}[\![\sigma]\!]\eta'|$ を任意に選んで固定する．この時，η', ζ', ξ' に対する条件より $\eta', \zeta'[x \mapsto d_0], \xi' \models C, \Gamma[x:\sigma], \Delta$ であるので，仮定での η, ζ, ξ に対する条件を満たしている．従って，仮定より $\mathcal{E}[\![e]\!]\langle\zeta'[x \mapsto d_0], \xi'\rangle \in |\mathcal{T}[\![\tau]\!]\eta'|$ である．

故に (13) の成立が示されたので，[T-ABS] 則の場合に関する証明は完了した．

● [T-APP] 則の場合

示すべき事は，$\eta, \varepsilon \models C, \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e_1]\!]\varepsilon \in |\mathcal{T}[\![\sigma \to \tau]\!]\eta|$ が成り立ち，また，$\eta', \varepsilon' \models C, \Gamma, \Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e_2]\!]\varepsilon' \in |\mathcal{T}[\![\sigma]\!]\eta'|$ が成り立つ，という二つの仮定の下で，$\eta'', \varepsilon'' \models C, \Gamma, \Delta$ なる任意の $\eta'' \in \mathbf{TEnv}$ と $\varepsilon'' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e_1\, e_2]\!]\varepsilon'' \in |\mathcal{T}[\![\tau]\!]\eta''|$ が成立する，という事である．

今，$\eta_0, \varepsilon_0 \models C, \Gamma, \Delta$ なる $\eta_0 \in \mathbf{TEnv}$ と $\varepsilon_0 \in \mathbf{Env}$ とを任意に選び固定する．$\mathcal{E}[\![e_1]\!]\varepsilon_0 \in |\mathcal{T}[\![\sigma \to \tau]\!]\eta_0|$ であるので，或る $f \in |[\mathcal{T}[\![\sigma]\!]\eta_0 \to \mathcal{T}[\![\tau]\!]\eta_0]|$ が存在して

$\mathcal{E}[\![e_1]\!]\varepsilon_0 = In_{\mathbf{F}}(f)$ と表わせる．故に，

$$\begin{aligned}\mathcal{E}[\![e_1e_2]\!]\varepsilon_0 &= \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon_0)\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon_0)(\mathcal{E}[\![e_2]\!]\varepsilon_0)\ \ \mathsf{else}\ \ wrong \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{関数適用式の意味方程式より} \\ &= \mathsf{if}\ \ Is_{\mathbf{F}}(In_{\mathbf{F}}(f))\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(In_{\mathbf{F}}(f))(\mathcal{E}[\![e_2]\!]\varepsilon_0)\ \ \mathsf{else}\ \ wrong \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{上の注意より} \\ &= f(\mathcal{E}[\![e_2]\!]\varepsilon_0)\end{aligned}$$

となる．

$f \in |[\mathcal{T}[\![\sigma]\!]\eta_0 \to \mathcal{T}[\![\tau]\!]\eta_0]|$ であり，また，$\mathcal{E}[\![e_2]\!]\varepsilon_0 \in |\mathcal{T}[\![\sigma]\!]\eta_0|$ であるので，$f(\mathcal{E}[\![e_2]\!]\varepsilon_0) \in |\mathcal{T}[\![\tau]\!]\eta_0|$，即ち，$\mathcal{E}[\![e_1e_2]\!]\varepsilon_0 \in |\mathcal{T}[\![\tau]\!]\eta_0|$ が成立している．

● [T-RECORD], [T-SELECT], [T-VARIANT], [T-CASE] 則（多相型以外の型の導入と除去に関する残りの規則） や [T-FIX] 則の場合

以上の [T-ABS] 則や [T-APP] 則の場合と類似の議論により証明できる．なお，構造を持つ型（関数型，レコード型，可変型）の値を分解する構文（関数適用式，フィールド選択式，タグ選択式）や不動点再帰式の意味方程式には，実行時型エラーを表わす $wrong$ が含まれている．このエラー値が最終的に推論規則の帰結部の型付け判別式の式の表示とならない事[8]は，推論規則前提部の各分解構文の部分式（[T-APP] 則の場合は e_1，[T-SELECT] 則の場合は e，[T-CASE] 則の場合は e_0，[T-FIX] 則の場合は e）に関する型付け判別式についての帰納法の仮定によって保証されている．

● [T-TABS] 則の場合

示すべき事は，$\eta, \varepsilon \models C[t <: \sigma], \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\tau]\!]\eta|$ が成り立つ，という仮定の下で，$\eta', \varepsilon' \models C, \Gamma, \Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![\Lambda\, t{<:}\sigma.e]\!]\varepsilon' \in |\mathcal{T}[\![\forall t{<:}\sigma.\tau]\!]\eta'|$ が成立する，という事である．

仮定の型環境 η に対する条件より，仮定で用いる事のできる η は，常に $\eta \models C[t <: \sigma]$ つまり $\eta(t) \subseteq \mathcal{T}[\![\sigma]\!]\eta$ を満たさねばならない．従って，$t \in \mathrm{dom}(\eta)$ で

8) 実行時型エラー値 $wrong$ は定理 5.84 より型が表示する cper （**TYPES** の要素）の定義域に属し得ないので，式が $wrong$ を表示すると健全性は成立しない．従って，型付けに関する推論規則の適用は，帰結部の判別式での型付けの対象となる式の表示が $wrong$ となる状況を必ず排除する，という保証を与えねばならない．

ある．そこで，有限写像 η を $\eta \triangleq \eta^\circ[t \mapsto P]$ と表わす．ここで，P は $P \in \mathbf{TYPES}$ かつ $P \subseteq \mathcal{T}[\![\sigma]\!]\eta$ を満たす cper であり，また，この時，型環境 η° は η に対する条件と [T-TABS] 則での付帯条件 $t \notin \mathrm{FTV}(C, \Gamma, \Delta, \sigma)$ とから $\eta^\circ, \varepsilon \models C, \Gamma, \Delta$ を満たしている．

すると，仮定より $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\tau]\!]\eta^\circ[t \mapsto P]|$ が上の条件を満たす任意の cper P に対して成立している．

従って，

$$\begin{aligned}
\mathcal{E}[\![\Lambda\, t{<:}\sigma.e]\!]\varepsilon &= \mathcal{E}[\![e]\!]\varepsilon && \text{型抽象式の意味方程式より}\\
&\in \bigcap\{|\mathcal{T}[\![\tau]\!]\eta^\circ[t \mapsto P]| \mid P \in \mathbf{TYPES} \text{ かつ } P \subseteq \mathcal{T}[\![\sigma]\!]\eta^\circ[t \mapsto P]\}\\
&= \bigcap\{|\mathcal{T}[\![\tau]\!]\eta^\circ[t \mapsto P]| \mid P \in \mathbf{TYPES} \text{ かつ } P \subseteq \mathcal{T}[\![\sigma]\!]\eta^\circ\}\\
& && t \notin \mathrm{FTV}(\sigma) \text{ より}\\
&= |\mathcal{T}[\![\forall t{<:}\sigma.\tau]\!]\eta^\circ| && \text{多相型の意味方程式より}
\end{aligned}$$

となるが，η が仮定の条件を満たす範囲で任意であったので，η° も $\eta^\circ, \varepsilon \models C, \Gamma, \Delta$ を満たすという制約の範囲で任意である．また，ε も上の制限の範囲で任意なので，求める結論が得られた．

● [T-TAPP] 則の場合

示すべき事は，$\eta, \varepsilon \models C, \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\forall t{<:}\rho.\tau]\!]\eta|$ が成り立ち，また，$\eta' \models C$ なる任意の $\eta' \in \mathbf{TEnv}$ に対して $\mathcal{T}[\![\sigma]\!]\eta' \subseteq \mathcal{T}[\![\rho]\!]\eta'$ が成り立つ，という二つの仮定の下で，$\eta'', \varepsilon'' \models C, \Gamma, \Delta$ なる任意の $\eta'' \in \mathbf{TEnv}$ と $\varepsilon'' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e[\sigma]]\!]\varepsilon'' \in |\mathcal{T}[\![\tau[t := \sigma]]\!]\eta''|$ が成立する，という事である．

今，$\eta_0, \varepsilon_0 \models C, \Gamma, \Delta$ なる $\eta_0 \in \mathbf{TEnv}$ と $\varepsilon_0 \in \mathbf{Env}$ とを任意に選んで固定する．すると，これらは仮定での η, ε や η', ε' に対する条件も満たすので，

$$\begin{aligned}
\mathcal{E}[\![e[\sigma]]\!]\varepsilon_0 &= \mathcal{E}[\![e]\!]\varepsilon_0 && \text{型適用式の意味方程式より}\\
&\in |\mathcal{T}[\![\forall t{<:}\rho.\tau]\!]\eta_0| && \text{1 番目の仮定より}\\
&= \bigcap\{|\mathcal{T}[\![\tau]\!]\eta_0[t \mapsto P]| \mid P \in \mathbf{TYPES} \text{ かつ } P \subseteq \mathcal{T}[\![\rho]\!]\eta_0\}\\
& && \text{多相型の意味方程式より}\\
&\subseteq |\mathcal{T}[\![\tau]\!]\eta_0[t \mapsto \mathcal{T}[\![\sigma]\!]\eta_0]| && \text{2 番目の仮定より}\\
&= |\mathcal{T}[\![\tau[t := \sigma]]\!]\eta_0| && \text{補題 5.88 より}
\end{aligned}$$

となる．η_0, ε_0 は任意に選んだので，求める結論が得られた．

● [T-RESTRICT] 則の場合

$\mathcal{E}[\![e{:}\tau]\!]\varepsilon = \mathcal{E}[\![e]\!]\varepsilon$ であるので，この規則に関する証明は [T-SUBTYPE] 則の場合と全く同一になる．

● [T-REFINE] 則の場合

示すべき事は，$\eta, \varepsilon \models C, \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\sigma]\!]\eta|$ が成り立ち，また，各 $1 \leq i \leq m$ について $\eta', \varepsilon' \models C, \Gamma, \Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{A}[\![\gamma_i[r := e]]\!]\varepsilon'\eta' = \mathit{true}$ が成り立つ，という $(m+1)$ 個の仮定の下で，$\eta'', \varepsilon'' \models C, \Gamma, \Delta$ なる任意の $\eta'' \in \mathbf{TEnv}$ と $\varepsilon'' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon'' \in |\mathcal{T}[\![\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}]\!]\eta''|$ が成立する，という事である．

今，$\eta_0, \varepsilon_0 \models C, \Gamma, \Delta$ なる $\eta_0 \in \mathbf{TEnv}$ と $\varepsilon_0 = \langle \zeta_0, \xi_0 \rangle \in \mathbf{Env}$ とを任意に選び固定する．補題 5.87 (2) より，各 $1 \leq i \leq m$ について

$$\mathcal{A}[\![\gamma_i]\!]\langle \zeta_0, \xi_0[r \mapsto \mathcal{E}[\![e]\!]\varepsilon_0] \rangle \eta_0 = \mathcal{A}[\![\gamma_i[r := e]]\!]\varepsilon_0\eta_0 = \mathit{true}$$

であるが，定義 2.8 (a) より各 γ_i は r 以外の自由変数を含まないので，$1 \leq i \leq m$ の各々について

$$\mathcal{A}[\![\gamma_i]\!]\langle \bot_{\mathbf{OEnv}}, \bot_{\mathbf{IEnv}}[r \mapsto \mathcal{E}[\![e]\!]\varepsilon_0] \rangle \eta_0 = \mathcal{A}[\![\gamma_i]\!]\langle \zeta_0, \xi_0[r \mapsto \mathcal{E}[\![e]\!]\varepsilon_0] \rangle \eta_0 = \mathit{true}$$

である．

従って，P と S とを，各々，図 5.5 中の詳細化型に対する意味方程式の右辺中でのそれらとする時，$\mathcal{E}[\![e]\!]\varepsilon_0 \in |P|$ かつ $\mathcal{E}[\![e]\!]\varepsilon_0 \in S$ であり，故に，

$$\mathcal{E}[\![e]\!]\varepsilon_0 \in |P \lceil S| = |\mathcal{T}[\![\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}]\!]\eta_0|$$

である．ε_0 と η_0 とは所定の条件を満たす限りに於いて任意であったので，求める結論が得られた．

(B3) Σ が表明判別式の場合

Σ の導出の最後に適用された規則は図 2.4 の表明に関する推論規則の何れかでなければならない．

● [A-TRANS] 則の場合

半順序 $\sqsubseteq_{\mathbf{D}}$ の推移性より明らか.

● [A-WEAK], [A-IWEAK], [A-TWEAK] 則（型付け文脈の成分への要素対の追加に関する推論規則）の場合

各規則での帰結側の判別式の型付け文脈に関する条件の方が強くなるので明らか.

● [A-SUBTYPE] 則の場合

帰納法の仮定を適用する事で直ちに示せる.

● [A-**forall**-I] 則の場合

示すべき事は，$\eta,\zeta,\xi \models C,\Gamma[x:\sigma],\Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon = \langle\zeta,\xi\rangle \in \mathbf{Env}$ とに対して $\mathcal{A}[\![\gamma]\!]\varepsilon\eta = true$ が成り立つ，という仮定の下で，$\eta',\zeta',\xi' \models C,\Gamma,\Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' = \langle\zeta',\xi'\rangle \in \mathbf{Env}$ とに対して $\mathcal{A}[\![\mathbf{forall}\ x{:}\sigma.\,\gamma]\!]\varepsilon'\eta' = true$ が成立する，という事である.

最後の $\mathcal{A}$ に関する式を図 5.4 での全称表明に関する意味方程式に従って展開し整理すると,

$$\forall d \in |\mathcal{T}[\![\sigma]\!]\eta'|.\mathcal{A}[\![\gamma]\!]\langle\zeta'[x \mapsto d],\xi'\rangle\eta' = true$$

となるので，この式の成立を示せば良い.

今，$\eta_0,\zeta_0,\xi_0 \models C,\Gamma,\Delta$ を満たす $\eta_0 \in \mathbf{TEnv}$ と $\varepsilon_0 = \langle\zeta_0,\xi_0\rangle \in \mathbf{Env}$，ならびに，$d_0 \in |\mathcal{T}[\![\sigma]\!]\eta_0|$ を任意に選んで固定する．この時，$\eta_0,\zeta_0[x \mapsto d_0],\xi_0 \models C,\Gamma[x:\sigma],\Delta$ であるので，仮定での型環境／環境に求められていた要件を満たす．従って，仮定より，$\mathcal{A}[\![\gamma]\!]\langle\zeta_0[x \mapsto d],\xi_0\rangle\eta_0 = true$ となり，求める結論が得られた.

● [A-**forall**-E] 則の場合

補題 5.87 (1) を用いれば簡単に示せる.

● 式の構文との両立性を与える為の各規則の場合

これらの規則は型付けに関する推論規則と対応しており，式に対する意味関数 $\mathcal{E}$ の連続性（が導く単調性）を用いる事で，対応する型付け規則の場合と同様にして証明できるので省略する.

● [A-$\beta_{\_}$] 則の場合

示すべき事は，$\eta,\varepsilon \models C,\Gamma[x:\sigma],\Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e_1]\!]\varepsilon \in |\mathcal{T}[\![\tau]\!]\eta|$ が成り立ち，また，$\eta',\varepsilon' \models C,\Gamma,\Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e_2]\!]\varepsilon' \in |\mathcal{T}[\![\sigma]\!]\eta'|$ が成り立つ，という二つの仮定の下で，$\eta'',\varepsilon'' \models C,\Gamma,\Delta$ なる任意の $\eta'' \in \mathbf{TEnv}$ と $\varepsilon'' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![(\lambda\, x{:}\sigma.e_1)e_2]\!]\varepsilon'' \sqsubseteq_{\mathbf{D}} \mathcal{E}[\![e_1[x := e_2]]\!]\varepsilon''$ が成立する，という事である．

関数適用式と関数抽象式とに対する意味方程式および補題 5.86 (1) を用いると，

$$\begin{aligned}\mathcal{E}[\![(\lambda\, x{:}\sigma.e_1)e_2]\!]\varepsilon'' &= \mathsf{let}\ \langle\zeta'',\xi''\rangle = \varepsilon''\ \mathsf{in}\ \mathcal{E}[\![e_1]\!]\langle\zeta''[x \mapsto \mathcal{E}[\![e_2]\!]\varepsilon''],\xi''\rangle\ \mathsf{end}\\ &= \mathcal{E}[\![e_1[x := e_2]]\!]\varepsilon''\end{aligned}$$

となり，求める（不）等式が得られた．

● [A-$\beta_{\{\ldots\}}$] 則の場合

示すべき事は，各 $1 \le i \le n$ について $\eta,\varepsilon \models C,\Gamma,\Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e_i]\!]\varepsilon \in |\mathcal{T}[\![\sigma_i]\!]\eta|$ が成り立つ，という n 個の仮定の下で，$\eta',\varepsilon' \models C,\Gamma,\Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![\{l_1 = e_1,\ldots,l_n = e_n\}.l_i]\!]\varepsilon' \sqsubseteq_{\mathbf{D}} \mathcal{E}[\![e_i]\!]\varepsilon'$ が成立する，という事である．

フィールド選択式とレコード式とに対する意味方程式を用いると，

$$\begin{aligned}&\mathcal{E}[\![\{l_1 = e_1,\ldots,l_n = e_n\}.l_i]\!]\varepsilon'\\ &= \mathsf{if}\ \ Is_{\mathbf{R}}(\mathcal{E}[\![\{l_1 = e_1,\ldots,l_n = e_n\}]\!])\varepsilon'\\ &\quad \mathsf{then}\ \ Out_{\mathbf{R}}(\mathcal{E}[\![\{l_1 = e_1,\ldots,l_n = e_n\}]\!]\varepsilon')(l_i)\\ &\quad \mathsf{else}\ \ \mathit{wrong}\\ &= \mathsf{if}\ \ Is_{\mathbf{R}}(In_{\mathbf{R}}(\boldsymbol{\lambda}_{\perp} l \in |\mathbf{L}|.\,\mathsf{if}\ \ l = l_1\ \ \mathsf{then}\ \ \mathcal{E}[\![e_1]\!]\varepsilon'\\ &\qquad\qquad\qquad\qquad \mathsf{elseif}\ \ \ldots\\ &\qquad\qquad\qquad\qquad \mathsf{elseif}\ \ l = l_n\ \ \mathsf{then}\ \ \mathcal{E}[\![e_n]\!]\varepsilon'\\ &\qquad\qquad\qquad\qquad \mathsf{else}\ \ \perp_{\mathbf{D}}))\\ &\quad \mathsf{then}\ \ Out_{\mathbf{R}}(In_{\mathbf{R}}(\boldsymbol{\lambda}_{\perp} l \in |\mathbf{L}|.\,\mathsf{if}\ \ l = l_1\ \ \mathsf{then}\ \ \mathcal{E}[\![e_1]\!]\varepsilon'\\ &\qquad\qquad\qquad\qquad\qquad \mathsf{elseif}\ \ \ldots\\ &\qquad\qquad\qquad\qquad\qquad \mathsf{elseif}\ \ l = l_n\ \ \mathsf{then}\ \ \mathcal{E}[\![e_n]\!]\varepsilon'\\ &\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \ \perp_{\mathbf{D}}))(l_i)\\ &\quad \mathsf{else}\ \ \mathit{wrong}\\ &= (\boldsymbol{\lambda}_{\perp} l \in |\mathbf{L}|.\,\mathsf{if}\ l = l_1\ \mathsf{then}\ \mathcal{E}[\![e_1]\!]\varepsilon'\ \mathsf{elseif}\ \ldots\ \mathsf{elseif}\ l = l_n\ \mathsf{then}\ \mathcal{E}[\![e_n]\!]\varepsilon'\ \mathsf{else}\ \perp_{\mathbf{D}})(l_i)\\ &= \mathcal{E}[\![e_i]\!]\varepsilon'\end{aligned}$$

となり，求める（不）等式が得られた．

● [A-$\beta_{[\ldots]}$] 則の場合

示すべき事は，$\eta, \varepsilon \models C, \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\sigma_i]\!]\eta|$ が成り立ち，各 $1 \leq j \leq n$ について $\eta', \varepsilon' \models C, \Gamma, \Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e_j]\!]\varepsilon \in |\mathcal{T}[\![\sigma_j \to \tau]\!]\eta|$ が成り立つ，という $(n+1)$ 個の仮定の下で，$\eta'', \varepsilon'' \models C, \Gamma, \Delta$ なる任意の $\eta'' \in \mathbf{TEnv}$ と $\varepsilon'' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![\mathbf{case}\,[\,l_i = e\,]\,\mathbf{of}\,l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n]\!]\varepsilon'' \sqsubseteq_{\mathbf{D}} \mathcal{E}[\![e_i e]\!]\varepsilon''$ が任意の $1 \leq i \leq n$ について成立する，という事である．

タグ付け式，タグ選択式，関数適用式それぞれに対する意味方程式を用いれば，

$$
\begin{array}{l}
\quad \mathcal{E}[\![\mathbf{case}\,[\,l_i = e\,]\,\mathbf{of}\,l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n]\!]\varepsilon'' \\
= \mathsf{if}\ \ Is_{\mathbf{V}}(\mathcal{E}[\![[\,l_i = e\,]]\!]\varepsilon'')\ \ \mathsf{then} \\
\quad \mathsf{let}\ \langle l, v\rangle = Out_{\mathbf{V}}(\mathcal{E}[\![[\,l_i = e\,]]\!]\varepsilon'')\ \ \mathsf{in} \\
\quad\quad \mathsf{if}\ \ l = l_1\ \ \mathsf{then} \\
\quad\quad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon'')\ \ \mathsf{then}\ \ out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon'')(v)\ \ \mathsf{else}\ \ wrong \\
\quad\quad \mathsf{elseif}\ \ \ldots \\
\quad\quad \mathsf{elseif}\ \ l = l_n\ \ \mathsf{then} \\
\quad\quad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\varepsilon'')\ \ \mathsf{then}\ \ out_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\varepsilon'')(v)\ \ \mathsf{else}\ \ wrong \\
\quad\quad \mathsf{else}\ \ \bot_{\mathbf{D}} \\
\quad \mathsf{end} \\
\quad \mathsf{else}\ \ wrong \\
= \mathsf{if}\ \ Is_{\mathbf{V}}(In_{\mathbf{V}}(\langle l_i, \mathcal{E}[\![e]\!]\varepsilon''\rangle))\ \ \mathsf{then} \\
\quad \mathsf{let}\ \langle l, v\rangle = Out_{\mathbf{V}}(In_{\mathbf{V}}(\langle l_i, \mathcal{E}[\![e]\!]\varepsilon''\rangle))\ \ \mathsf{in} \\
\quad\quad \mathsf{if}\ \ l = l_1\ \ \mathsf{then} \\
\quad\quad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon'')\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\varepsilon'')(v)\ \ \mathsf{else}\ \ wrong \\
\quad\quad \mathsf{elseif}\ \ \ldots \\
\quad\quad \mathsf{elseif}\ \ l = l_n\ \ \mathsf{then} \\
\quad\quad\quad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\varepsilon'')\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\varepsilon'')(v)\ \ \mathsf{else}\ \ wrong \\
\quad\quad \mathsf{else}\ \ \bot_{\mathbf{D}} \\
\quad \mathsf{end} \\
\quad \mathsf{else}\ \ wrong \\
= \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e_i]\!]\varepsilon'')\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}[\![e_i]\!]\varepsilon'')(\mathcal{E}[\![e]\!]\varepsilon'')\ \ \mathsf{then}\ \ wrong \\
= \mathcal{E}[\![e_i e]\!]\varepsilon''
\end{array}
$$

となり，求める（不）等式が得られた．

● [A-$\beta_\forall$] 則の場合

示すべき事は，$\eta, \varepsilon \models C[t <: \sigma], \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\tau]\!]\eta|$ が成り立ち，また，$\eta' \models C$ なる任意の $\eta \in \mathbf{TEnv}$ に対して $\mathcal{T}[\![\rho]\!]\eta' \subseteq \mathcal{T}[\![\sigma]\!]\eta'$ が成り立つ，という二つの仮定の下で，$\eta'', \varepsilon'' \models C, \Gamma, \Delta$ なる任意の $\eta'' \in \mathbf{TEnv}$ と $\varepsilon'' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![(\Lambda\, t{<:}\sigma.e)[\rho]]\!]\varepsilon'' \sqsubseteq_{\mathbf{D}} \mathcal{E}[\![e[t := \rho]]\!]\varepsilon''$ が成立する，という事である．

型適用式と型抽象式とに対する意味方程式を用いると，

$$\mathcal{E}[\![(\Lambda\, t{<:}\sigma.e)[\rho]]\!]\varepsilon'' = \mathcal{E}[\![\Lambda\, t{<:}\sigma.e]\!]\varepsilon'' = \mathcal{E}[\![e]\!]\varepsilon'' = \mathcal{E}[\![e[t := \rho]]\!]\varepsilon''$$

となり（最後の等式の成立は式の解釈が型無しである事から殆ど自明であるが，厳密に示すならば式 e の構造に関する帰納法を用いれば良い），求める（不）等式が得られた．

● [A-β_{fix}] 則の場合

示すべき事は，$\eta, \varepsilon \models C, \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\sigma \to \sigma]\!]\eta|$ が成り立つ，という仮定の下で，$\eta', \varepsilon' \models C, \Gamma, \Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{E}[\![\mathbf{fix}\ e]\!]\varepsilon' \sqsubseteq_{\mathbf{D}} \mathcal{E}[\![e(\mathbf{fix}\ e)]\!]\varepsilon'$ が任意の $1 \leq i \leq n$ について成立する，という事である．

e に関する仮定と関数型に対する意味方程式とより或る $g \in |[\mathcal{T}[\![\sigma]\!]\eta' \to \mathcal{T}[\![\sigma]\!]]\eta'|$ が存在し $\mathcal{E}[\![e]\!]\varepsilon' = In_{\mathbf{F}}(g)$ と表わせる．すると，不動点再帰式と関数適用式とに対する意味方程式を用いれば，

$$\begin{aligned}
&\mathcal{E}[\![\mathbf{fix}\ e]\!]\varepsilon' \\
&= \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e]\!]\varepsilon')\ \ \mathsf{then\ let}\ \ f = Out_{\mathbf{F}}(\mathcal{E}[\![e]\!]\varepsilon')\ \ \mathsf{in}\ \ \mathit{fix}(f)\ \ \mathsf{end\ else}\ \ \mathit{wrong} \\
&= \mathsf{if}\ \ Is_{\mathbf{F}}(In_{\mathbf{F}}(g))\ \ \mathsf{then\ let}\ \ f = Out_{\mathbf{F}}(In_{\mathbf{F}}(g))\ \ \mathsf{in}\ \ \mathit{fix}(f)\ \ \mathsf{end\ else}\ \ \mathit{wrong} \\
&= \mathit{fix}(g) \\
&= g(\mathit{fix}(g)) \\
&= \mathsf{if}\ \ Is_{\mathbf{F}}(In_{\mathbf{F}}(g))\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(In_{\mathbf{F}}(g))(\mathit{fix}(g))\ \ \mathsf{else}\ \ \mathit{wrong} \\
&= \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}[\![e]\!]\varepsilon')\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(In_{\mathbf{F}}(\mathcal{E}[\![e]\!]\varepsilon'))(\mathcal{E}[\![\mathbf{fix}\ e]\!]\varepsilon')\ \ \mathsf{else}\ \ \mathit{wrong} \\
&= \mathcal{E}[\![e(\mathbf{fix}\ e)]\!]\varepsilon'
\end{aligned}$$

となり，求める（不）等式が得られた．

● [A-ASSERT] 則の場合

示すべき事は，$\eta, \varepsilon \models C, \Gamma, \Delta$ なる任意の $\eta \in \mathbf{TEnv}$ と $\varepsilon \in \mathbf{Env}$ とに対して $\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\{r{:}\sigma \mid \gamma_1, \ldots, \gamma_m\}]\!]\eta|$ が成り立つ，という仮定の下で，各 $1 \leq i \leq m$ について $\eta', \varepsilon' \models C, \Gamma, \Delta$ なる任意の $\eta' \in \mathbf{TEnv}$ と $\varepsilon' \in \mathbf{Env}$ とに対して $\mathcal{A}[\![\gamma_i[r := e]]\!]\varepsilon'\eta' = \mathit{true}$ が成り立つ，という事である．

$\eta_0, \varepsilon_0 \models C, \Gamma, \Delta$ なる η_0 と ε_0 とを任意に選んで固定する．この時，詳細化型の意味方程式より，

$$\mathcal{E}[\![e]\!]\varepsilon_0 \in \left| \mathcal{T}[\![\sigma]\!]\eta_0 \left[\left\{ d \in |\mathbf{D}| \;\middle|\; \bigwedge_{i=1}^{m} \mathcal{A}[\![\gamma_i]\!]\langle \bot_{\mathbf{OEnv}}, \bot_{\mathbf{IEnv}}[r \mapsto d] \rangle \eta_0 = \mathit{true} \right\} \right] \right|$$

となるので，特に，

$$\mathcal{E}[\![e]\!]\varepsilon_0 \in \left\{ d \in |\mathbf{D}| \;\middle|\; \bigwedge_{i=1}^{m} \mathcal{A}[\![\gamma_i]\!]\langle \bot_{\mathbf{OEnv}}, \bot_{\mathbf{IEnv}}[r \mapsto d] \rangle \eta_0 = \mathit{true} \right\}$$

である．即ち，各 $1 \leq i \leq m$ について

$$\mathcal{A}[\![\gamma_i]\!]\langle \bot_{\mathbf{OEnv}}, \bot_{\mathbf{IEnv}}[r \mapsto \mathcal{E}[\![e]\!]\varepsilon_0] \rangle \eta_0 = \mathit{true}$$

となるが，γ_i は r 以外の自由変数を含まないので，γ_i の表示する真理値は r 以外の変数への付値には影響されない．そこで，上で任意に選んだ環境 $\varepsilon_0 = \langle \zeta_0, \xi_0 \rangle$ に対して

$$\mathcal{A}[\![\gamma_i]\!]\langle \zeta_0, \xi_0[r \mapsto \mathcal{E}[\![e]\!]\varepsilon_0] \rangle \eta_0 = \mathit{true}$$

となる．これに補題 5.87 (2) を適用すれば，

$$\mathcal{A}[\![\gamma_i[r := e]]\!]\varepsilon_0\eta_0 = \mathit{true}$$

となり，求める結論が得られた．

● [A-LEAST] 則の場合

帰結部の表明の左辺式 $\mathbf{fix}\ (\lambda x{:}\sigma.x)$ の表示が $\bot_{\mathbf{D}}$ である事は容易に示せる．故に帰結部の表明の表示は自明に true となり，この場合も正しい．

証明終

以上の健全性の証明に関連して一つ注意しておく． 5.2 節の型と表明の意味関数の項の冒頭で，表明の意味は式の意味と異なり完全な型無し解釈とする事はできない，という事，そして，その原因が図 5.4 での全称化表明に対する意味方程式で， **forall** による全称化の解釈としてのメタ言語上の $\forall$ により束縛されるメタ変数 d が採れる値の範囲が $|\mathbf{D}|$ 全体でなく元の変数 x の型 σ の表示（の定義域） $|\mathcal{T}[\![\sigma]\!]\eta|$ に限定せねばならない点にある，という事を注意した．この $\forall$ による全称束縛での範囲限定が必要な理由は以下の通りである．

その理由とは， [A-**forall**-I] 則に関する証明で示した通り，仮定に対する環境 $\varepsilon = \langle\zeta, \xi\rangle$ （の中でも特に ζ）に課せられた条件である $\eta, \zeta, \xi \models C, \Gamma[x : \sigma], \Delta$ を x に対する付値 d に関して全称化された（表明本体の）解釈へと反映する為である．

即ち，上の ζ に対しては， $\zeta(x) \in |\mathcal{T}[\![\sigma]\!]\eta|$ という制約が課されている．その為，[A-**forall**-I] 則での帰納法の仮定では，本体表明 γ の解釈 $\mathcal{A}[\![\gamma]\!]\langle\zeta, \xi\rangle\eta$ では， x への付値は型 σ （の表示）に属する値である事が保証されている．

この x に対する付値の範囲の保証を結論部の全称化表明の解釈で保つには，全称化の解釈に於けるメタ言語上の $\forall$ で束縛されるメタ変数 d の範囲を前提 — つまり帰納法の仮定 — での x への付値の範囲である $|\mathcal{T}[\![\sigma]\!]\eta|$ に限定する必要がある．

実際，全称化表明に対する意味方程式で $\forall$ により束縛される変数 d の走る範囲を $|\mathbf{D}|$ 全体にした場合， [A-**forall**-I] 則は容易に健全性を失う．例えば [A-**forall**-I] 則の次の適用例を見よ（ここで σ はどんな型でも良い）：

$$\frac{\varnothing, \{\mathrm{z} : \sigma \to \sigma, \mathrm{w} : \sigma\}, \varnothing \ \triangleright\ (\lambda\,\mathrm{x}{:}\sigma.\lambda\,\mathrm{y}{:}\sigma.\mathrm{y})(\mathrm{zw}) \leqslant \lambda\,\mathrm{y}{:}\sigma.\mathrm{y} : \sigma \to \sigma}{\varnothing, \{\mathrm{w} : \sigma\}, \varnothing \ \triangleright\ \mathbf{forall}\ \mathrm{z}{:}\sigma \to \sigma.\,(\lambda\,\mathrm{x}{:}\sigma.\lambda\,\mathrm{y}{:}\sigma.\mathrm{y})(\mathrm{zw}) \leqslant \lambda\,\mathrm{y}{:}\sigma.\mathrm{y} : \sigma \to \sigma}.$$

結論部の **forall**-全称化の解釈が $|\mathbf{D}|$ 全体を値の範囲とするならば， z への付値として関数でない値 — $Is_{\mathbf{F}}(d) = \mathit{false}$ となる値 d でその典型は $\mathit{wrong} = In_{\mathbf{W}}(?)$ — を選ぶ事が許される．その結果，不等式左辺の関数適用式が表示する値は wrong となり右辺式が表示する値 $In_{\mathbf{F}}(\boldsymbol{\lambda}\, d' \in |\mathbf{D}|.d')$ とは $\sqsubseteq_{\mathbf{D}}$ の関係にならない．

故に，全称化表明の意味方程式での付値 d に対するメタ言語レベルの $\forall$-全称化では $|\mathcal{T}[\![\sigma]\!]\eta|$ という範囲限定が必要となる．以上の理由により，表明の意味を型の意味に依存しない形で定義する事はできない．

健全性定理は Cardelli (1984) での様に特定の場合に特化された形で提示される事も多い．例えば，式の値は式の型が表わす集合に属する事を主張する形の

系 5.90 （意味論的健全性定理）

型付け可能な Funiq の式は実行時型エラーを引き起こさない．即ち，

$$\vdash C, \Gamma, \Delta \triangleright e : \sigma$$
$$\Longrightarrow \quad \forall \varepsilon \in \mathbf{Env}, \eta \in \mathbf{TEnv} \text{ s.t. } \eta, \varepsilon \models C, \Gamma, \Delta. \Big[\mathcal{E}[\![e]\!]\varepsilon \in |\mathcal{T}[\![\sigma]\!]\eta| \Big].$$

言い替えれば，

$$\vdash C, \Gamma, \Delta \triangleright e : \sigma$$
$$\Longrightarrow \quad \forall \varepsilon \in \mathbf{Env}, \eta \in \mathbf{TEnv} \text{ s.t. } \eta, \varepsilon \models C, \Gamma, \Delta. \Big[\mathcal{E}[\![e]\!]\varepsilon \neq \mathit{wrong} \Big].$$

および，部分型が型に属する値の集合の間の包含関係と対応する事を主張する形の

系 5.91 （意味論的部分型付け定理）

Funiq の任意の型 $\sigma, \tau \in \mathbf{Type}$ に対して，

$$\vdash C \triangleright \sigma <: \tau \quad \Longrightarrow \quad \forall \eta \models C. \Big[\mathcal{T}[\![\sigma]\!]\eta \subseteq \mathcal{T}[\![\tau]\!]\eta \Big].$$

等である．

以上から， Funiq の型システムの定式化としての型理論 FUNIQ は， Funiq に対する完備部分同値関係に基づく表示的意味に対して健全である事が示された．

なお，前章で定義した Funiq の β-簡約（操作的意味）は本章で与えた名前呼びの表示的意味に対して健全である．即ち，式の簡約に於いて式の表示（値）は変化しない事が保証される．

定理 5.92 （β-簡約の健全性）

前章で定義した Funiq の β-簡約は本章の図 5.3，図 5.4，図 5.5 が定める表示的意味論に対して健全である．即ち，任意の $e_1, e_2 \in \mathbf{Exp}$ に対して，

$$e_1 \longrightarrow\!\!\!\!\!\rightarrow_\beta e_2 \quad \Longrightarrow \quad \forall \varepsilon \in \mathbf{Env}. \big[\mathcal{E}[\![e_1]\!]\varepsilon = \mathcal{E}[\![e_2]\!]\varepsilon \big].$$

証明

定理 4.6 より $\langle e_1, e_2\rangle \in \boldsymbol{\beta}_{\exists}$ ならば $\|e_1\| \longrightarrow\!\!\!\!\!\rightarrow^{+}_{\beta^{-\{\exists\}}} \|e_2\|$ であるので，簡約概念 $\boldsymbol{\beta}_{\exists}$ の健全性は $\beta^{-\{\exists\}}$-簡約の健全性に帰着できる．従って，存在限量化型の式に対する $\beta_{\exists}$-簡約を含まない $\beta^{-\{\exists\}}$-簡約に対して定理の言明を証明すれば良い．この証明は $e_1 \longrightarrow\!\!\!\!\!\rightarrow_{\beta^{-\{\exists\}}} e_2$ のステップ数 s に関する帰納法で行なう．帰納法の基底である $s = 0$ — $e_1 \equiv e_2$ — の場合は自明である．帰納法のステップは， e_1 自身が簡約基でない時には e_1' を e_1 中の簡約基とすると $e_1 \equiv D[e_1']$ と表わせるので，文脈 $D[\ \ ]$ の構造に関する帰納法で容易に示す事ができる．従って，以下では， e_1 が簡約基で e_2 がその縮約結果 — つまり対 $\langle e_1, e_2\rangle$ が（$\boldsymbol{\beta}_{\exists}$ 以外の）簡約概念の要素 — の場合を示す．

(1) $\langle e_1, e_2\rangle \in \boldsymbol{\beta}_{\rightarrow}$ の場合

定理 5.89 の証明での [A-$\beta_{\rightarrow}$] 則の場合に示した式変形から直ちに成立する．なお，同証明中での型付け文脈を尊重するという環境や型環境に対する条件は，実際には式変形で必要としていない事（この事は以下の各場合でも同様）に注意せよ．

(2) $\langle e_1, e_2\rangle \in \boldsymbol{\beta}_{\{\ldots\}}$ の場合

同じく [A-$\beta_{\{\ldots\}}$] 則の場合に示した式変形から直ちに成立する．

(3) $\langle e_1, e_2\rangle \in \boldsymbol{\beta}_{[\ldots]}$ の場合

同じく [A-$\beta_{[\ldots]}$] 則の場合に示した式変形から直ちに成立する．

(4) $\langle e_1, e_2\rangle \in \boldsymbol{\beta}_{\forall}$ の場合

同じく [A-$\beta_{\forall}$] の場合に示した式変形から直ちに成立する．

(5) $\langle e_1, e_2\rangle \in \boldsymbol{\beta}_{\mathbf{fix}}$ の場合

同じく [A-β_{fix}] 則の場合に示した式変形から成立する．この場合だけは式変形で $e_1 \equiv \mathbf{fix}\ e$ の e の型が環境に加える制限を用いているが，それを満たさない場合，$\mathbf{fix}\ e$ も $e(\mathbf{fix}\ e)$ も共に実行時型エラー値 $wrong$ を表示する事は容易に示される．

証明終

同様に，形式的厳格性を簡約論的に特徴付ける為に導入した $\perp$-簡約に関しても，定数記号 $\perp$ に込められていた「値が未定義の式」という当初の意図に従って，定数記号 $\perp$ に対する表示を，

$$\mathcal{E}[\![\perp]\!] \triangleq \boldsymbol{\lambda}\varepsilon \in |\mathbf{Env}|.\perp_{\mathbf{D}}$$

と定義して，式の意味関数 $\mathcal{E}$ の定義域を $\mathbf{Exp}\perp_\perp$ に拡大すれば，その健全性を容易に証明できる．

定理 5.93 （⊥-簡約の健全性）

⊥-簡約は健全である．即ち，

$$e_1 \longrightarrow\!\!\!\!\!\rightarrow_\perp e_2 \quad \Longrightarrow \quad \forall \varepsilon \in \mathbf{Env}.\big[\mathcal{E}[\![e_1]\!]\varepsilon = \mathcal{E}[\![e_2]\!]\varepsilon\big].$$

証明

証明で本質的な箇所は $\langle e_1, e_2\rangle \in \perp$ の場合に $\mathcal{E}[\![e_1]\!]\varepsilon = \perp_{\mathbf{D}}$ を示す点であるが，これは $\perp$ を定義する各成分簡約概念 $\perp_{xxx}$ $(xxx \in \{\rightarrow, \{\cdots\}, [\cdots], \forall, \exists, \mathrm{fix}, :\})$ の場合に分けて確認すれば良く非常に簡単に行なえる．

証明終

式 e 中の形式的厳格な変数に対する付値が $\perp_{\mathbf{D}}$ ならば e の値も $\perp_{\mathbf{D}}$ となる，という補題 5.66 を証明したが，上の ⊥-簡約の健全性と簡約論的厳格性定理 4.31 とを用いれば，より簡単に証明できる．

系 5.94 （式に関する形式的厳格性）

e を式，v を e で形式的厳格な変数とする時，

$$\forall \varepsilon \in \mathbf{Env}.\big[\mathcal{E}[\![e]\!]\varepsilon[v \mapsto \perp_{\mathbf{D}}] = \perp_{\mathbf{D}}\big].$$

証明

$$\begin{aligned}
\mathcal{E}[\![e]\!]\varepsilon[v \mapsto \perp_{\mathbf{D}}] &= \mathcal{E}[\![e]\!]\varepsilon[v \mapsto \mathcal{E}[\![\perp]\!]\varepsilon] && \mathcal{E}[\![\perp]\!]\varepsilon \text{ の定義より}\\
&= \mathcal{E}[\![e[v := \perp]]\!]\varepsilon && \text{式に関する代入補題 5.86 より}\\
&= \mathcal{E}[\![\perp]\!]\varepsilon && \text{簡約論的厳格性定理 4.31 と}\\
& && \perp\text{-簡約の健全性定理 5.93 より}\\
&= \perp_{\mathbf{D}}. && \mathcal{E}[\![\perp]\!]\varepsilon \text{ の定義より}
\end{aligned}$$

証明終

5.4 値呼びに基づく $\mathrm{Funiq}_\perp$ に関する表示的意味論

値呼び意味論に基づく $\mathrm{Funiq}_\perp$ の場合，関数の引数の式の値が未定義 $\perp$ の場合は関数の値も未定義 $\perp$ となるので， $\mathrm{Funiq}_\perp$ での関数は常に厳格である．従って，関数値の集まりとしての cpo $\mathbf{F}$ は厳格連続関数空間でなければならない．可変型の式に関しても，値呼び意味論の場合，タグ付けの対象となる式の値が未定義であればタグ付けした結果の値も未定義であると考えるのが自然であるので，タグ付き値の集まりとしての cpo $\mathbf{V}$ としては厳格積構成を採用するのが適切である．従って意味領域 $\mathbf{D}$ の対応する成分領域 $\mathbf{F}$ と $\mathbf{V}$ とは以下の様に定義を変更されねばならない：

- $f, g \in \mathbf{F} \triangleq [\mathbf{D} \to_\perp \mathbf{D}]$;
- $u, v \in \mathbf{V} \triangleq \mathbf{L} \otimes \mathbf{D}$.

以上の意味領域の変更に伴い，式の意味関数 $\mathcal{E}$ の定義としての意味方程式も影響を受けるが， $\mathrm{Funiq}_\perp$ の式の意味方程式に関しては，この意味領域の変更に伴うだけでなく，補題 5.66 相当の補題 — この補題は型の表示を定義する上で最も基本的な性質 — を成立させる為に，定義 2.6 で与えた形式的厳格性の定義の変更点に対応した修正を加えねばならない．

以上の考察に基づき $\mathrm{Funiq}_\perp$ の式に対する意味関数 $\mathcal{E}_\perp$ の意味方程式で名前呼び意味論に基づく Funiq に対する $\mathcal{E}$ の対応する方程式のと異なる（図 5.3 中の $\mathcal{E}$ を $\mathcal{E}_\perp$ と読み替えて得られない）ものだけを図 5.6 に示す．

同図の意味方程式の中で，関数抽象式・タグ付け式・ **case** 式に関するものは，上述の対応する成分領域（各々， $\mathbf{F}$ と $\mathbf{V}$）を作る領域構成子（各々， $[\_ \to \_]$ と $\_ \times \_$）の変更に伴いその領域の値に対する演算（各々， $\boldsymbol{\lambda}$ と $\langle \_, \_ \rangle / \_.i$）を変更後の領域構成子（各々， $[\_ \to_\perp \_]$ と $\_ \otimes \_$）に対応するもの（各々， $\boldsymbol{\lambda}_\perp$ と $\langle \_, \_ \rangle_\otimes / \_ \bullet i$）へと変更しただけである．一方，関数適用式に関する意味方程式は引数式 e_2 の値が未定義 $\perp_{\mathbf{D}}$ か否かを最初に調べる形に変更されているが，この変更は定義 2.6 での条件 $(3')$ に対応するもので，正に「値呼び」という事を反映した意味方程式となっている．

なお，形式的厳格性に関する同定義での他の条件の追加・変更に関しては，追加条件 (12) がタグ付け式の意味方程式の変更で対処されており， **unpack** 式に関する条件の変更 $(9')$ と **pack** 式に関する追加条件 (13) とは関数適用式に関する変更等で対処されているので，上記以外の意味方程式に対しては変更する必要がない．

$$
\begin{aligned}
\mathcal{E}_\perp[\![\lambda x{:}\sigma.e]\!] =& \boldsymbol{\lambda}\,\varepsilon \in |\mathbf{Env}|.\ \mathsf{let}\ \langle \zeta, \xi\rangle = \varepsilon\ \mathsf{in}\ \mathit{In}_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\mathcal{E}_\perp[\![e]\!]\langle \zeta[x \mapsto d], \xi\rangle)\ \mathsf{end};\\
\\
\mathcal{E}_\perp[\![e_1 e_2]\!] =& \boldsymbol{\lambda}\,\varepsilon \in |\mathbf{Env}|.\\
& \quad (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\\
& \qquad \mathsf{if}\ \mathit{Is}_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\varepsilon)\ \mathsf{then}\ \mathit{Out}_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\varepsilon)(d)\ \mathsf{else}\ \mathit{wrong})(\mathcal{E}_\perp[\![e_2]\!]\varepsilon);\\
\\
\mathcal{E}_\perp[\![[l = e]]\!] =& \boldsymbol{\lambda}\,\varepsilon \in |\mathbf{Env}|.\ \mathit{In}_{\mathbf{V}}(\langle l, \mathcal{E}_\perp[\![e]\!]\varepsilon\rangle_\otimes);\\
\\
\mathcal{E}_\perp[\![\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n]\!] =& \boldsymbol{\lambda}\,\varepsilon \in |\mathbf{Env}|.\ \mathsf{if}\ \ \mathit{Is}_{\mathbf{V}}(\mathcal{E}_\perp[\![e_0]\!]\varepsilon)\ \ \mathsf{then}\\
& \qquad \mathsf{let}\ \ l = (\mathit{Out}_{\mathbf{V}}(\mathcal{E}_\perp[\![e_0]\!]\varepsilon)) \bullet 1\\
& \qquad \mathsf{and}\ \ v = (\mathit{Out}_{\mathbf{V}}(\mathcal{E}_\perp[\![e_0]\!]\varepsilon)) \bullet 2\ \ \mathsf{in}\\
& \qquad\quad \mathsf{if}\ \ l = l_1\ \ \mathsf{then}\\
& \qquad\qquad \mathsf{if}\ \ \mathit{Is}_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\varepsilon)\ \ \mathsf{then}\ \ \mathit{Out}_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\varepsilon)(v)\\
& \qquad\qquad \mathsf{else}\ \ \mathit{wrong}\\
& \qquad\quad \mathsf{elseif}\ \ \ldots\\
& \qquad\quad \mathsf{elseif}\ \ l = l_n\ \ \mathsf{then}\\
& \qquad\qquad \mathsf{if}\ \ \mathit{Is}_{\mathbf{F}}(\mathcal{E}_\perp[\![e_n]\!]\varepsilon)\ \ \mathsf{then}\ \ \mathit{Out}_{\mathbf{F}}(\mathcal{E}_\perp[\![e_n]\!]\varepsilon)(v)\\
& \qquad\qquad \mathsf{else}\ \ \mathit{wrong}\\
& \qquad\quad \mathsf{else}\ \perp_{\mathbf{D}}\\
& \qquad \mathsf{end}\\
& \quad \mathsf{else}\ \ \mathit{wrong}.
\end{aligned}
$$

図 5.6　$\mathrm{Funiq}_\perp$ の式に関する意味方程式（$\mathrm{Funiq}_\perp$ 固有分，他は図 5.3 の $\mathcal{E}$ に同じ）

以上の意味方程式に対して，期待通り補題 5.66 に相当する次の補題が成立する．

補題 5.95（式に関する形式的厳格性 — 値呼び版）

e を $\mathrm{Funiq}_\perp$ の式，r' を e で（$\mathrm{Funiq}_\perp$ に対する定義 2.6 の意味で）形式的厳格な実現変数とする時，任意の $\zeta \in \mathbf{OEnv}$ と $\xi \in \mathbf{IEnv}$ とに対して，関数 $\boldsymbol{\lambda}\, d \in |\mathbf{D}|.\mathcal{E}_\perp[\![e]\!]\langle \zeta, \xi[r' \mapsto d]\rangle$ は厳格である．

証明

定義 2.6 での定義 2.5 に対する変更／追加条件に関してのみ示す（関数抽象式の意味方程式の変更 — $\boldsymbol{\lambda}$ から厳格な $\boldsymbol{\lambda}_\perp$ へ — は条件 (2) の証明には影響しない）．

(3′) $e \equiv e_1 e_2$ の場合

この場合，形式的厳格性の定義から r' は e_1, e_2 の少なくとも一方で形式的厳格である． e_1 で形式的厳格な場合は補題 5.66 の証明での場合 (4) として示したのと同様であるので，以下， e_2 で形式的厳格な場合を示す．

$$
\begin{aligned}
&\mathcal{E}_\perp[\![e]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= \mathcal{E}_\perp[\![e_1 e_2]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|. \\
&\qquad \mathsf{if}\ Is_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)\ \mathsf{then}\ Out_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)(d)\ \mathsf{else}\ wrong) \\
&\quad (\mathcal{E}_\perp[\![e_2]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle) && \text{関数適用式の意味方程式より} \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|. \\
&\qquad \mathsf{if}\ Is_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)\ \mathsf{then}\ Out_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)(d)\ \mathsf{else}\ wrong) \\
&\quad (\perp_{\mathbf{D}}) && r' \text{ は } e_2 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
&= \perp_{\mathbf{D}} && \boldsymbol{\lambda}_\perp \text{ の厳格性から}
\end{aligned}
$$

となるので成立.

$(9')$ $e \equiv \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2$ の場合

定義から r' は e_1 で形式的厳格か $r' \neq x$ かつ e_2 で形式的厳格かの何れかだが，常に $r' \neq x$ （$\mathbf{Var} \cap \mathbf{IVar} = \varnothing$）なので，結局， r' は e_1, e_2 の少なくとも一方で形式的厳格である． e_1 の場合は補題 5.66 証明での (10) と同様なので， e_2 の場合を示す.

$$
\begin{aligned}
&\mathcal{E}_\perp[\![e]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= \mathcal{E}_\perp[\![\mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{with}\ t\ \mathbf{in}\ e_2]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= \mathcal{E}_\perp[\![e_1[\rho](\Lambda\, t{<:}\sigma.\lambda\, x{:}\,\tau.e_2)]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
& && \text{定義 3.1 の } \mathbf{unpack} \text{ 式のコード化より} \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|. \\
&\qquad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1[\rho]]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1[\rho]]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)(d) \\
&\qquad \mathsf{else}\ \ wrong) \\
&\quad (\mathcal{E}_\perp[\![(\Lambda\, t{<:}\sigma.\lambda\, x{:}\,\tau.e_2)]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle) \\
& && \text{関数適用式の意味方程式より} \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|. \\
&\qquad \mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)\ \ \mathsf{then}\ \ Out_{\mathbf{F}}(\mathcal{E}_\perp[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)(d) \\
&\qquad \mathsf{else}\ \ wrong) \\
&\quad (\mathcal{E}_\perp[\![(\lambda\, x{:}\,\tau.e_2)]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle) && \mathcal{E}_\perp \text{ の各呼出しを整理} \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\ \cdots\text{中略}\cdots\ )(In_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\mathcal{E}_\perp[\![e_2]\!]\langle\zeta[x \mapsto d], \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle) \\
& && \text{関数抽象式の意味方程式より} \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\ \cdots\text{中略}\cdots\ )(In_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\perp_{\mathbf{D}})) \\
& && r' \text{ は } e_2 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\ \cdots\text{中略}\cdots\ )(In_{\mathbf{F}}(\perp_{\mathbf{F}})) && \perp_{\mathbf{F}} \triangleq \boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\perp_{\mathbf{D}} \text{ なので} \\
&= (\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\ \cdots\text{中略}\cdots\ )(\perp_{\mathbf{D}}) && In_{\mathbf{F}}(\perp_{\mathbf{F}}) = \perp_{\mathbf{D}} \text{ なので} \\
&= \perp_{\mathbf{D}} && \boldsymbol{\lambda}_\perp \text{ の厳格性から}
\end{aligned}
$$

となるので成立.

(12) $e \equiv [\, l = e_1 \,]$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$
\begin{aligned}
&\mathcal{E}_\perp[\![e]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= \mathcal{E}_\perp[\![[\, l = e_1 \,]]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= In_{\mathbf{V}}(\langle l, \mathcal{E}_\perp[\![e]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle\rangle_\otimes) && \text{タグ付け式の意味方程式より} \\
&= In_{\mathbf{V}}(\langle l, \perp_{\mathbf{D}}\rangle_\otimes) && r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
&= In_{\mathbf{V}}(\perp_{\mathbf{V}}) && \text{厳格対化 } \langle \_, \_ \rangle_\otimes \text{ の各成分に関する厳格性より} \\
&= \perp_{\mathbf{D}} && In_{\mathbf{V}}(\perp_{\mathbf{V}}) = \perp_{\mathbf{D}} \text{ なので}
\end{aligned}
$$

となるので成立.

(13) $e \equiv \mathbf{pack}_{\exists t<:\rho.\sigma}\ e_1\ \mathbf{with}\ \tau$ の場合

この場合，形式的厳格性の定義から r' は e_1 で形式的厳格である．従って，

$$
\begin{aligned}
&\mathcal{E}_\perp[\![e]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= \mathcal{E}_\perp[\![\mathbf{pack}_{\exists t<:\rho.\sigma}\ e_1\ \mathbf{with}\ \tau]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&= \mathcal{E}_\perp[\![\Lambda\, s{<:}\mathbf{Top}.\lambda\, x{:}\, \forall t{<:}\rho.(\sigma \to s).x[\tau]e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&\qquad\qquad \text{定義 3.1 の } \mathbf{pack} \text{ 式のコード化より} \\
&= \mathcal{E}_\perp[\![\lambda\, x{:}\, \forall t{<:}\rho.(\sigma \to s).x[\sigma]e_1]\!]\langle\zeta, \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle \\
&\qquad\qquad \text{型抽象式の意味方程式より} \\
&= In_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\mathcal{E}_\perp[\![x[\sigma]e_1]\!]\langle\zeta[x \mapsto d], \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle) \\
&\qquad\qquad \text{関数抽象式の意味方程式より} \\
&= In_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.(\mathsf{if}\ \ Is_{\mathbf{F}}(\mathcal{E}_\perp[\![x[\sigma]]\!]\langle\zeta[x \mapsto d], \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle)\ \ \mathsf{then} \\
&\qquad\qquad\qquad Out_{\mathbf{F}}(\mathcal{E}_\perp[\![x[\sigma]]\!]\langle\zeta[x \mapsto d], \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle) \\
&\qquad\qquad \mathsf{else} \\
&\qquad\qquad\qquad wrong) \\
&\qquad\qquad (\mathcal{E}_\perp[\![e_1]\!]\langle\zeta[x \mapsto d], \xi[r' \mapsto \perp_{\mathbf{D}}]\rangle))) \\
&\qquad\qquad \text{関数適用式の意味方程式より} \\
&= In_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.(\boldsymbol{\lambda}_\perp d' \in |\mathbf{D}|.\cdots \text{中略} \cdots)(\perp_{\mathbf{D}})) \\
&\qquad\qquad r' \text{ は } e_1 \text{ で形式的厳格ゆえ帰納法の仮定から} \\
&= In_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\perp_{\mathbf{D}}) && \boldsymbol{\lambda}_\perp \text{ の厳格性から} \\
&= In_{\mathbf{F}}(\perp_{\mathbf{F}}) && \perp_{\mathbf{F}} \triangleq \boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.\perp_{\mathbf{D}} \text{ なので} \\
&= \perp_{\mathbf{D}} && In_{\mathbf{F}}(\perp_{\mathbf{F}}) = \perp_{\mathbf{D}} \text{ なので}
\end{aligned}
$$

となるので成立.

証明終

$$\mathcal{T}_\perp[\![\sigma_1 \to \sigma_2]\!]\eta = In_{\mathbf{F}}\left([\mathcal{T}_\perp[\![\sigma_1]\!]\eta \to_\perp \mathcal{T}_\perp[\![\sigma_2]\!]\eta]\right);$$

$$\mathcal{T}_\perp[\![[l_1\!:\!\sigma_1, \ldots, l_n\!:\!\sigma_n\,]]\!]\eta = \langle \perp_{\mathbf{D}}, \perp_{\mathbf{D}} \rangle \cup \bigcup_{i=1}^{n} \{\langle In_{\mathbf{V}}(\langle l_i, v\rangle_\otimes), In_{\mathbf{V}}(\langle l_i, v'\rangle_\otimes)\rangle \mid \langle l_i, v\rangle_\otimes, \langle l_i, v'\rangle_\otimes \in |\mathbf{V}| \text{ かつ } \langle v, v'\rangle \in \mathcal{T}_\perp[\![\sigma_i]\!]\eta\}.$$

図 5.7 $\mathrm{Funiq}_\perp$ の型に関する意味方程式（$\mathrm{Funiq}_\perp$ 固有分，他は図 5.5 の $\mathcal{T}$ に同じ）

本節冒頭の成分意味領域 $\mathbf{F}, \mathbf{V}$ の変更に伴い，$\mathrm{Funiq}_\perp$ の型に対する意味関数 $\mathcal{T}_\perp$ は図 5.7 に示す通りである．これも名前呼びに対する図 5.5 での $\mathcal{T}$ の意味方程式の $\mathcal{T}/\mathcal{A}$ を各々 $\mathcal{T}_\perp/\mathcal{A}_\perp$ に読み替えても得られないものだけを示す．なお，表明に対する意味関数 $\mathcal{A}_\perp$ の意味方程式は，全て図 5.4 の $\mathcal{A}$ に関する対応する方程式の $\mathcal{E}/\mathcal{T}$ を各々 $\mathcal{E}_\perp/\mathcal{T}_\perp$ に読み替えて得られるので省略する．

上で定義した $\mathrm{Funiq}_\perp$ に対する値呼びの表示的意味論に関しても型理論 FUNIQ は次に示す通り健全である．証明は定理 5.89 の場合と同様に行なえるので省略する．

定理 5.96（健全性定理 — 値呼び版）

理論 FUNIQ は意味関数 $\mathcal{E}_\perp, \mathcal{A}_\perp, \mathcal{T}_\perp$ で与えられる値呼びの $\mathrm{Funiq}_\perp$ の表示的意味に対して健全（定義 5.85 の充足性の定義での $\mathcal{E}/\mathcal{A}/\mathcal{T}$ は各々 $\mathcal{E}_\perp/\mathcal{A}_\perp/\mathcal{T}_\perp$ に読み替える）である．即ち，任意の FUNIQ の判別式 $\Sigma, \Sigma_1, \ldots, \Sigma_n$ $(n \geq 0)$ に対して，

$$\Sigma_1, \ldots, \Sigma_n \vdash \Sigma \quad \Longrightarrow \quad \Sigma_1, \ldots, \Sigma_n \models \Sigma.$$

4.6 節で定義された値呼び評価の β^v-簡約や $\perp^v$-簡約に関しても，以下の様に $\mathrm{Funiq}_\perp$ の表示的意味に対する健全性が成り立つ．証明は名前呼びの対応する定理と同様であるので省略する．

定理 5.97（β^v-簡約の健全性）

β^v-簡約は $\mathrm{Funiq}_\perp$ の表示的意味に対して健全である．即ち，

$$e_1 \longrightarrow\!\!\!\!\!\rightarrow_{\beta^v} e_2 \quad \Longrightarrow \quad \forall \varepsilon \in \mathbf{Env}.\left[\mathcal{E}_\perp[\![e_1]\!]\varepsilon = \mathcal{E}_\perp[\![e_2]\!]\varepsilon\right].$$

定理 5.98 （$\perp^v$-簡約の健全性）

$\perp^v$-簡約は $\mathrm{Funiq}_\perp$ の表示的意味に対して健全である．即ち，

$$e_1 \longrightarrow\!\!\!\!\!\rightarrow_{\perp^v} e_2 \quad \Longrightarrow \quad \forall \varepsilon \in \mathbf{Env}.\big[\mathcal{E}_\perp[\![e_1]\!]\varepsilon = \mathcal{E}_\perp[\![e_2]\!]\varepsilon\big].$$

一方，通常の β-簡約は $\mathrm{Funiq}_\perp$ の表示的意味に対して健全ではない．即ち，次の主張は成立しない．

非定理 5.99 （β-簡約の健全性）

$\perp^v$-簡約は $\mathrm{Funiq}_\perp$ の表示的意味に対して健全である．即ち，

$$e_1 \longrightarrow\!\!\!\!\!\rightarrow_{\beta} e_2 \quad \Longrightarrow \quad \forall \varepsilon \in \mathbf{Env}.\big[\mathcal{E}_\perp[\![e_1]\!]\varepsilon = \mathcal{E}_\perp[\![e_2]\!]\varepsilon\big].$$

反証

$e_1 \equiv (\lambda\,\mathrm{x}\!:\!\{\}.\lambda\,\mathrm{y}\!:\!\{\}.\mathrm{y})(\mathbf{fix}\ (\lambda\,\mathrm{z}\!:\!\{\}.\mathrm{z}))$，$e_2 \equiv \lambda\,\mathrm{y}\!:\!\{\}.\mathrm{y}$ と選べば，$e_1 \longrightarrow_\beta e_2$ であるが，任意の $\varepsilon \in \mathbf{Env}$ に対して $\mathcal{E}_\perp[\![\mathbf{fix}\ (\lambda\,\mathrm{z}\!:\!\{\}.\mathrm{z})]\!]\varepsilon = \perp_{\mathbf{D}}$ であるので，図 5.6 の関数適用式に対する意味方程式を用いると，

$$\mathcal{E}_\perp[\![e_1]\!]\varepsilon = \perp_{\mathbf{D}} \neq In_{\mathbf{F}}(\boldsymbol{\lambda}_\perp d \in |\mathbf{D}|.d) = \mathcal{E}_\perp[\![e_2]\!]\varepsilon$$

となる．

反証終

本節の最後として，遅延評価の場合の関数値に対する意味領域 $\mathbf{F}$ に関して触れておく．遅延評価では関数抽象式の本体式の表示が $\perp_{\mathbf{D}}$ であっても関数抽象式の表示は「関数である事」が判っている分だけ単なる $\perp_{\mathbf{D}}$ よりも情報が多いと考える．従って，名前呼び意味論で遅延評価の Funiq^ℓ の場合には

$$\mathbf{F} \triangleq [\mathbf{D} \to \mathbf{D}]_\perp$$

と，また，値呼び意味論で遅延評価の $\mathrm{Funiq}^\ell_\perp$ の場合には

$$\mathbf{F} \triangleq [\mathbf{D} \to_\perp \mathbf{D}]_\perp$$

と，遅延評価を採用しない場合の各々の $\mathbf{F}$ の定義を底上げして得られる． $\mathbf{F}$ のこれらの定義に対しても，底上げ cpo に対する基本演算の $up, down$ を用いて関数抽象式

と関数適用式との意味方程式を与え，型理論や対応する簡約に対する健全性を証明できるが，ここでは省略する．

5.5 Funiq の表示的意味論に関する今後の課題

5.5.1 式の型無し解釈に伴う問題点

式の意味として型無し解釈を採用し実行時型エラーの可能性を意味領域に持ち込んだ事により，この意味論に対して健全とする為の形式的厳格性を定める定義 2.5 での認定条件に於いて直感的には許されそうに思える場合が排除されねばならなくなっている．

具体的に言うと， **case** 式に対する形式的厳格性の条件に於いて直感的に正しいと思われる場合が排除されねばならなくなっている． 2.1 節で述べた通り，形式的厳格性の直感的な意味は，形式的厳格な変数の値が未定義であれば式全体の値が未定義になる，という事である．故に **case** e_0 **of** l_1 **then** $e_1, \ldots, l_n$ **then** e_n で v が形式的厳格である為には定義 2.5 (6) で要請されている **case** 式の場合分け対象式 e_0 に於ける v の形式的厳格性，つまり，

(a) v は e_0 で形式的厳格である，

が満たされなくとも， e_0 が l_i でタグ付けられた値を持つ場合にその値からタグを外した値に作用すべき関数を表わす各々の式 e_i $(1 \leq i \leq n)$ の全てが v に関して形式的厳格だという事が成り立つならば， v の値が未定義の時は **case** 式全体の値も未定義となる筈である．従って，定義 2.5 (6) に

(b) v は $e_1, \ldots, e_n$ 全てで形式的厳格である，

という条件を追加して， (a) か (b) かの少なくとも一方が満たされれば良い，と直感的には考えられる．

(b) を許した場合，補題 5.66 での (7) の **case** 式に対する証明を (a), (b) 各々の場合に分けて示す必要がある．その結果，条件 (b) の場合に対する証明での式変形は，

$\mathcal{E}[\![e]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle$
$= \mathcal{E}[\![\mathbf{case}\ e_0\ \mathbf{of}\ l_1\ \mathbf{then}\ e_1, \ldots, l_n\ \mathbf{then}\ e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle$
$=$ if $Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ then
  let $\langle l, v\rangle = Out_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ in
    if $l = l_1$ then
      if $Is_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ then $Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(v)$
      else $wrong$
    elseif ...
    elseif $l = l_n$ then
      if $Is_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ then $Out_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(v)$
      else $wrong$
    else $\bot_{\mathbf{D}}$
  end
  else $wrong$ $\mathcal{E}$ の定義より
$=$ if $Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ then
  let $\langle l, v\rangle = Out_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ in
    if $l = l_1$ then
      if $Is_{\mathbf{F}}(\bot_{\mathbf{D}})$ then $Out_{\mathbf{F}}(\mathcal{E}[\![e_1]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(v)$
      else $wrong$
    elseif ...
    elseif $l = l_n$ then
      if $Is_{\mathbf{F}}(\bot_{\mathbf{D}})$ then $Out_{\mathbf{F}}(\mathcal{E}[\![e_n]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)(v)$
      else $wrong$
    else $\bot_{\mathbf{D}}$
  end
  else $wrong$ r' は各 e_i で形式的厳格ゆえ帰納法の仮定から
$=$ if $Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ then
  let $\langle l, v\rangle = Out_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ in
    if $l = l_1$ then $\bot_{\mathbf{D}}$
    elseif ...
    elseif $l = l_n$ then $\bot_{\mathbf{D}}$
    else $\bot_{\mathbf{D}}$
  end
  else $wrong$ $Is_{\mathbf{F}}(\bot_{\mathbf{D}}) = \bot_{\mathbf{D}}$ なので
$=$ if $Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta, \xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ then $\bot_{\mathbf{D}}$
  else $wrong$ 内側の if 記法の各分岐が $\bot_{\mathbf{D}}$ なので

となるが，この最後の if $Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta,\xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ then $\bot_{\mathbf{D}}$ else $wrong$ は $\bot_{\mathbf{D}}$ と等しくない．

実際，**case** 式での場合分けの対象となる式 e_0 が可変型の値を表わさない式であれば **case** 式全体は実行時型エラーとなるので，それを表わす値 $wrong$ を表示せねばならない．即ち，条件 (b) は型無し解釈に基づく意味関数 $\mathcal{E}$ に対しては健全とはならない．

一方，正しく型付けされたプログラムの場合，以上の様な事は起こり得ないので，その様な正しく型付けされたプログラム中の **case** 式の場合は最後の if 記法の条件 $Is_{\mathbf{V}}(\mathcal{E}[\![e_0]\!]\langle\zeta,\xi[r' \mapsto \bot_{\mathbf{D}}]\rangle)$ が常に真となり，その結果，上の if 記法全体も常に then 側の $\bot_{\mathbf{D}}$ を値とする事になる．従って，補題 5.66 での式 e を正しく型付け可能な式に限定すれば上の問題は解決し，**case** 式に対する形式的厳格性の定義に条件 (b) を追加する事が許されそうに見える．

上のパラグラフの主張を証明するには，型付けされた式が表示し得る値の範囲に対する保証を与えている健全性定理 — 定理 5.89，特に系 5.90 の意味論的健全性定理 — が必要である．一方，定理 5.89 は $\mathcal{T}$ の整定性を与える定理 5.84 に基づいている．そして，定理 5.84 — つまり型の表示が cper である事 — を保証するには補題 5.66 が必要である．

故に，直感的には認めて良いと思われる上の条件 (b) を定義 2.5 (6) に加えると，補題 5.66 と定理 5.84 と定理 5.89 との間での循環論法が生ずるので (b) を認める事は不可能である．

この問題は型無し解釈を採用する限り実行時型エラーを意味領域 $\mathbf{D}$ に導入せざるを得ないので解決できないと思われる．型付き解釈を採用した場合に解決できるのかは不明であるが，上の意味関数の計算から判断すると，型付き解釈を採用しても実行時型エラーの概念を含む解釈を与える場合には，この問題の解決は困難であると予想される．

5.5.2 完備部分同値関係意味論に伴う問題点

完備部分同値関係を用いた本論文での表示的意味論の最も重要な問題点は，関数適用の計算が一般には型（の表示）の中で閉じない，という点である．この点をより具体的に述べよう．

例えば，適当な型付け文脈の下で，f は $f : \sigma \to \tau$ という関数型の式，引数式 e は $e : \sigma$ であるとする時，或る適当な環境 ε の下での $f(e)$ の値は，$\mathbf{D}$ と成分 cpo 間の入射／射影や逆極限法での同型射の詳細を省略すれば，$\mathcal{E}[\![f]\!]\varepsilon(\mathcal{E}[\![e]\!]\varepsilon)$ となる．そこで，以下，$\mathcal{E}[\![f]\!]\varepsilon$ を f と，また，$\mathcal{E}[\![e]\!]\varepsilon$ を e と略記する事にすると，意味論的健全性定理（系 5.90）より，適当な型環境 η の下で，$f \in |[\mathcal{T}[\![\sigma]\!]\eta \to \mathcal{T}[\![\tau]\!]\eta]|$ なる連続関数であり，また，$e \in |\mathcal{T}[\![\sigma]\!]\eta|$ である．

ところが，一般には，$f(e)$ の値は以上の二つの cper の定義域に属する $\mathbf{D}$ の要素だけでは求める事ができない．その理由は，$f(e)$ の値を求めるには，f と e との各々を近似するコンパクトな要素から成る ω-上昇鎖 $\{f_i\}_{i\in\omega}$ と $\{e_i\}_{i\in\omega}$ とを用いて，$\bigsqcup^{\mathbf{D}}_{i\in\omega} f_{i+1}(e_i)$ という上限を求める必要があるが，cper の定義域としての部分 cpo はコンパクトな要素に関して閉じているとは限らないからである．

即ち，$\mathbf{D}$ の或る要素 d が $\mathcal{T}[\![\sigma]\!]\eta$ の定義域に属していたとしても，d を近似する全コンパクト要素の集合 $\mathop{\downarrow} d$ が $\mathcal{T}[\![\sigma]\!]\eta$ の定義域に含まれる（つまり $\mathop{\downarrow} d \subseteq |\mathcal{T}[\![\sigma]\!]\eta|$）とは限らない．従って，$f(e)$ の値は，一般には $\mathbf{D}$ 全体で計算する必要がある．

上で概略として述べた $\mathcal{T}[\![\sigma]\!]\eta$ が一般の — 即ち詳細化型を含む — 型 σ に関してはコンパクト要素に関して閉じていると限らない，という点に関して，以下，より具体的に説明する．

5.2.3 項の補題 5.83 で述べた通り，本章での詳細化型に cper を用いた表示的意味を与える上でのキーポイントは，

$$P \triangleq \boldsymbol{\lambda} d \in |\mathbf{D}|.f(d) \sqsubseteq_{\mathbf{D}} g(d)$$

という形で定義される領域 $\mathbf{D}$ 上の述語が，f が厳格かつ連続で g が連続ならば完備となる，即ち，$f(\bot_{\mathbf{D}}) \sqsubseteq_{\mathbf{D}} g(\bot_{\mathbf{D}})$ で，かつ，任意の ω-上昇鎖 $\{d_i\}_{i\in\omega}$ に対して，

$$\forall i \in \omega.\big[f(d_i) \sqsubseteq_{\mathbf{D}} g(d_i)\big] \quad \Longrightarrow \quad f\left(\bigsqcup^{\mathbf{D}}_{i\in\omega} d_i\right) \sqsubseteq_{\mathbf{D}} g\left(\bigsqcup^{\mathbf{D}}_{i\in\omega} d_i\right)$$

という点にある．

問題は，後の方の性質での逆向きの含意が（一般の ω-上昇鎖でなくコンパクト要素のみからなる ω-上昇鎖の場合に限っても）成立するか，という点にある．この点を正確に議論する為に，以下の記法を導入する．

記法 5.100 （近似コンパクト要素）

$\mathbf{X}$ を $\mathbf{X} \cong F(\mathbf{X})$ なる再帰的領域方程式の逆極限法による領域解とし，$\varphi_{i\infty} : \mathbf{X}_i \longleftrightarrow \mathbf{X} : \psi_{\infty i}$ を各成分領域 $\mathbf{X}_i$ と $\mathbf{X}$ との間の射影対とし，$d \in \mathbf{X}$ をその任意の要素とする時，各 $i \in \omega$ について，

(1) 無限列 d の第 i 成分を $d^{(i)}$ $(\in \mathbf{X}_i)$ で表わす．即ち，$d^{(i)} \triangleq d(i)$.

(2) $d^{(i)}$ の $\mathbf{X}$ での像を $d^{[i]}$ で表わす．即ち，$d^{[i]} \triangleq \varphi_{i\infty}(d^{(i)})$.

一般に，任意の $d \in \mathbf{D}$ に対して，d の近似コンパクト要素の集まり $\{d^{[i]}\}_{i\in\omega}$ は ω-上昇鎖を成す．従って，問題は，f, g を上で述べた性質を満たす関数とする時，任意の $d \in \mathbf{D}$ に対して，

$$\big(f(d) \sqsubseteq_{\mathbf{D}} g(d)\big) \quad \Longrightarrow \quad \forall i \in \omega.\big[f(d^{[i]}) \sqsubseteq_{\mathbf{D}} g(d^{[i]})\big] \qquad (※)$$

が成り立つか，という事である．

しかし，この言明 (※) は，f が厳格かつ連続で g が連続という条件だけでは一般には成立しない．例えば，問題の $d \in |\mathbf{D}|$ として $d \neq \bot_{\mathbf{D}}$ かつ $d^{[1]} \sqsubset_{\mathbf{D}} d^{[2]} \sqsubseteq_{\mathbf{D}} d$ なるものを選び，

$$f \triangleq \boldsymbol{\lambda} x \in |\mathbf{D}|.\mathsf{if}\ x \sqsubset_{\mathbf{D}} d^{[1]}\ \mathsf{then}\ \bot_{\mathbf{D}}\ \mathsf{else}\ d^{[2]},$$
$$g \triangleq id_{\mathbf{D}}$$

とすると，これらは f, g に関する上記の条件を満たしており（f は事実 5.29 で示した連続関数空間のコンパクト要素であるので，当然ながら連続関数），$f(d) = d^{[2]} \sqsubseteq_{\mathbf{D}} d = g(d)$ であるが，$f(d^{[1]}) = d^{[2]} \sqsupset_{\mathbf{D}} d^{[1]} = g(d^{[1]})$ となり，(※) の右辺は成立しない．

本項の始めの方で述べた関数適用 $f(e)$ の計算がそれに関わる型の表示の中で閉じておらず意味領域 $\mathbf{D}$ 全体に渡って計算する必要があるという事は，Funiq の応用面では何ら直接的な制約を加えるものではない．しかし，純理論的な立場からすると，計算に関して閉じていない型を「型」と呼ぶに相応しいか，即ち，型として充分な正統性があるのか，というのは疑問が残る点であり，意味論的な裏付けを保ちつつ Funiq を拡張し発展させて行く上での障害となり得る．

一方では，cper を表示とする本論文での Funiq の型は，その上で任意の再帰的関数の定義を許し，また，その再帰的関数の性質を証明する為に LCF の類の不動点帰納法 — 例えば，Paulson (1987), Scott (1993) 参照 — が可能であり，少なくとも実用的な観点からは一人前の立派な型と呼ぶに相応しい条件を満たしている．

5.5.3　再帰型の表示的意味に関する課題

3.4 節で詳細化型の一般性を議論した際，定理 3.28 で再帰型を Funiq に追加して得られる型理論 FUNIQY は，Fun に追加した再帰型を言語 FunnY に対する型理論 FUNNY の保存的拡大となる事を述べた．

この事は，型理論 FUNNY が不整合的 — 1 点集合以外のモデルを持たない — でない限り，詳細化型と再帰型との双方を含む型理論 FUNIQY も整合的（無矛盾）である，という事を意味しており，再帰型と詳細化型との自在な組合せを許す型理論が証明論的には有意である，という事を表わしている．

無論，型理論 FUNNY はトリビアルでなく有意なモデルを持ち無矛盾である．例えば Cardone (1989, 1991) で FUNNY に対する具体的なモデルが与えられている．故に，FUNIQ の整合性を壊す事なく再帰型を追加して詳細化型と再帰型との双方を含み無矛盾な型理論 FUNIQY を得る事が証明論的には許されている．

そこで，詳細化型を構成要素として含む再帰型の表示的意味を与える事が新たな課題となる．再帰型に対して表示的意味を与える為には，型の表示に関する再帰方程式の解の存在を保証して具体解を求める必要がある．

5.1 節の最後の 2 階の型付き λ-計算の領域論的意味論に関する略史を見ると，再帰型と多相型との両方を含む型理論に意味を与えている方法には，大別して

(1) イデアルを用いるアプローチ：
MacQueen and Sethi (1982), MacQueen, Plotkin and Sethi (1984, 1986);

(2) $\mathbf{D}_\infty$ 上の部分同値関係を用いるアプローチ：
Abadí, Pierce and Plotkin (1989), Abadí and Plotkin (1990), Amadio (1991), Bruce and Mitchell (1992), Cardone (1989, 1991), Cardone, Dezani-Ciancaglini and de'Liguoro (1994)

という二つの流れがある．

(1) のイデアルによるアプローチの場合，二つのイデアル I, J の距離 $d(I, J)$ を

$$d(I,J) \triangleq 2^{-\min\{i\in\omega|((I\setminus J)\cup(J\setminus I))\cap|\mathbf{D}_i|\neq\varnothing\}}$$

という形で， I, J の何れか一方だけに含まれる最も古い（逆極限法で最も早く生成される）コンパクト要素に基づいて定義し，型構成子の各々がこの距離に関して縮小写像である事を示し，それに基づき Banach の不動点定理（縮小写像は不動点を持つ）を用いる事により，（型構成子の合成としての）型の上の関数の不動点を $\mathfrak{I}(\mathbf{D}_\infty)$ の中に見出す，という方法で再帰型に対する表示を与えている．

一方， (2) の per によるアプローチの場合は， $\mathbf{D}$ 上の per として本論文で用いて来た完備性だけでなく一様性 — per P が一様とは定義域 $|P|$ が一様な述語の真となる値の集合を成す事 — をも満たす cuper (complete uniform per) のみを型の表示として用いるが，そこからの再帰型の表示を求める方法には二通りある．一つは， Amadio (1991) で代表される方法で， (1) と同様にコンパクト要素に基づいて距離を定義して Banach の不動点定理で不動点の存在を保証する，というアプローチを採用するものである．もう一つは， Cardone (1989, 1991) が採用している方法で，型の表示がコンパクト要素に関して閉じている事を利用して，型の意味関数の近似列を Scott の逆極限法と類似の方法で構成し，その極限として型の意味関数を得る，という方法である．

以上の概観で判る通り，再帰型に表示的意味を与える従来の方法では，型の表示がコンパクト要素に関して閉じている事が決定的に重要であるが，既に前に見た通り，詳細化型の表示としての cper は，一般にはコンパクト要素に関して閉じていない．その結果，従来の方法やその単純な拡張では，詳細化型と再帰型との双方の自由な組合せを許す FuniqY の型に対しては表示的意味を与える事ができない．

無論，型の上の再帰性はコンパクト要素に関して閉じる事が保証される型や型構成方法のみで構成される場合に限れば，その限定された再帰型に関しては従来の方法の延長で意味を与える事は可能である．

例えば，表明中の実現変数の出現はその表明を構成する原始表明の左辺式のみに限定した場合，表明の表示を *true* とする値の集合 —図 5.5 での詳細化型に対する意味方程式の中の S — がイデアル（つまり部分 cpo として代数的かつ整合完備である）

を成し，その様な S によって一様な cper を制限した結果が一様な cper となる，という事は容易に示す事ができる．

従って，それら一様な cper を成す事が保証されている型構成法（つまり，詳細化型に関しては上のパラグラフで述べた形の表明に限定）だけで構成された型に対してのみ再帰型構成方法を許し，その上のクラスでは一般の詳細化型を許す代わりに再帰型構成を認めない事とすれば，本論文での意味論的結果もほぼそのまま適用できる．しかし，この様に型のクラスを 2 階層に分けてしまっては，詳細化型も再帰型も，最早，他の様々な型構成法 — 関数型・レコード型・可変型・多相型・抽象型 — と完全に対等な型構成法であると主張する事はできなくなる．

そもそも，上で述べた制限された形の表明では，図 2.5 での LIFO スタックの表明は記述できても図 2.7 での FIFO キューの表明の幾つかは記述できない．従って，部分 cpo を成している集合 S の代数性（S で制限された結果の cper の一様性）を保証する為に表明の形を制限するのであれば，上の様な単純だが厳しい制限ではなく，不等式が実用的に充分な記述力を持てる制限でなければならない．

従って，不等式表明の表示が *true* となる値の集合の部分 cpo としての代数性を保証する表明に対する構文的制限の検討は重要である．しかし，コンパクト性やコンパクト要素に関する閉包性という表示での性質が構文上のどの様なの性質として特徴付けられ得るのか，という事に関しては，従来も殆ど研究されていない．

更に，上の一様性／代数性を満たす為の構文的制約の探索以上に重要な課題は，詳細化型と再帰型とを共に許す型の表示的意味の構成方法[9)]を探す事であり，それは必ず達成し得る，という事である．つまり，3.4 節での定理 3.28 の意義に関して本項の最初で説明した通り，詳細化型と再帰型との両方の共存を許す型理論としての FUNIQY は証明論的には無矛盾な理論であり，その自明でないモデルは必ず存在すると保証されている．従って，詳細化型と再帰型との任意の組合せを許す型システムに対する表示的意味の構成方法の探究は，今後に残された研究課題としての価値を充分に有している．

9) Funiq に再帰型を追加しても，無論，Church-Rosser 性は成り立つので，$\mathbf{Exp}/=_\beta$（式の集合 **Exp** 上の同値関係としての β-変換可能性による商集合）に基づき FuniqY の項モデルを構成する事は可能である．しかし，項モデルは全く構文的なモデルであり，その為，無限に続く計算が或る有意な極限値へと収束するとか極限に対する近似といった概念を項モデルで捉える事はできない．ここで探究すべきだと述べているのは，無限に続き得る計算の収束性や近似の概念を捉える事が — 領域論モデルの様に — 可能な枠組に基づいた再帰型と詳細化型とを許す型のモデルである．

5.5.4 等式表明を許す上での課題

記法 2.9 で示した通り，現在の Funiq でも，左右両辺を交換した二つの対称的な不等式（に全称化を施した）表明の対を表わす略記法として，既に，等式表明が導入されている．しかし，問題は，現状で許されている等式表明の場合，その実体としての二つの不等式表明は共に定義 2.8 での意味で良形でなければならない．その結果，等式表明 **forall** $x_1{:}\sigma_1.\ \ldots.$ **forall** $x_n{:}\sigma_n.\, e_1 = e_2 : \sigma$ の本体の両辺の式 e_1, e_2 は，共に当表明が属する詳細化型で束縛されている r に関して形式的厳格でなければならない．

この両辺が共に実現変数 r に関して形式的厳格でなければならない，という条件は非常に厳しく，等式表明を用いる事が許されるケースを非常に限定している．一方では，任意の等式表明を許してしまうと，それを含む詳細化型の表示が（$\perp_{\mathbf{D}}$ さえも含まない）空集合となる事を惹き起こし，領域論的意味論の構築を完全に破壊してしまう．

さて，本論文で詳細化型の表示を構成する為に利用している表明の表示としての $\mathbf{D}$ 上の述語の完備性という性質の中で，ω-帰納性に関しては式の表示 $\mathcal{E}[\![e]\!]$ の連続性から直ちに導かれる．この事は定理 5.84 の証明の (8) で見た通りである．

従って，等式表明をより自在に仕様記述で用いる為の課題は，等式表明が表示する述語が厳格性を満たす構文的な十分条件で，上に述べた両辺共に実現変数に関して形式的厳格よりもより緩やかで実用的に有用な条件を見出す事である．

残念ながら，表示の領域論的な性質が構文の上でどういった性質によって特徴付けられるのか，という事は，$\mathbf{D}$ 上の述語の ω-帰納性を除いては従来も殆ど知られておらず，上で求めようとしている条件が存在し得るか否かさえも全く不明なのが現状であり，全ては今後の研究課題として残されている．

第6章

纏め

— 関連研究と今後の課題 —

本章では，まず，本研究に関連する研究としての仕様と実現とを単一の言語として記述する広帯域言語ならびに代数仕様と型理論的機構（特に関数型と λ-計算）との統合の試みに関する従来の研究の概観を与える．次いで，本研究の結果の総括と今後に残された研究課題とを示し，本論文の纏めとする．

6.1　関連研究

本節では，本論文の主題であった仕様と実現としてのプログラムとを同一の言語で記述する広帯域言語に関する研究，仕様とプログラムとの間の関係を明確化する事を目的とした研究，更には，代数仕様に λ-計算的な言語メカニズムを導入する研究の中で主要なものに関して纏める．

■ 広帯域言語 CIP-L

仕様とプログラムとを同一言語で記述しようという広帯域言語の概念それ自体は特に新しいものではない．特に，広帯域言語の意味に関して何らかの厳密さを求めての研究は，少なくとも Bauer らによる CIP-L — Bauer and Wössner (1982), Bauer et al. (1985, 1987) — に遡る事ができる．広帯域言語 CIP-L では，抽象データ型の仕様に関しては代数的アプローチのスタイルで記述し，同時にその抽象データ型の実現やそれを用いるプログラムを関数的プログラム風・手続き的プログラム風・機械命令風といった様々な水準で記述する事を許している．従って， CIP-L は非常に幅の広い広帯域言語であり，仕様とプログラムとの間，並びに，関数的プログラムの様な高水準の記述と機械命令風等の低水準の記述との間には変換規則を定めて異なる水準間の意味の対応を与えている．

しかしながら， CIP-L に対する変換規則が全ての文法的に許される構文の組合せをカバーしているのか，即ち，変換規則が完全に言語の意味を与え切れているのか，という点に関しては明瞭でない．ましてや，この非常に幅の広い CIP-L 言語全体に対する一貫した表示的意味が直接に与えられている訳ではない．

その点では，広帯域言語 CIP-L は — 実用的観点からは変換規則群で充分かも知れないが純粋に理論的な観点からは — その良形 (well-formed) な句全てに対する意味が完全に定義された形式言語であると看做す事はできない．

■ VDM/RAISE

現時点で実用的に用いられている幾つかの形式手法の中でも最初の形式手法として Bjørner and Jones (1978) で広く公開された VDM (Vienna Development Method) の仕様記述言語の Meta-IV は，実行可能な関数的プログラミング言語を基本としつつも仕様記述の為に一般の 1 階述語論理の論理式を用いる事を許しており，更に，

代入文等の手続き的プログラミング言語の機能をも含んでいるという意味では最初から広帯域言語であった． Meta-IV の現代版 VDM-SL — Dawes (1991) — は仕様記述で用いる関数を Meta-IV での様に具体的に関数的プログラムの形で書き下さず，それが果たすべき機能を事前／事後条件によって規定する記法も導入しており，広帯域言語としての性格は益々強くなっている．

VDM-SL は ISO (1996) として国際標準規格となり，その規格書では VDM-SL の構文と表示的意味とが VDM-SL 自身をメタ言語として用いて超循環的な形で厳密に定義されている．事前／事後条件等の記述に用いる論理式の解釈は，（論理式中の変数の値は確定しているという意味で全域的な）通常の 1 階述語論理ではなく， Barriger, Chen and Jones (1984) による LPF (Logic for Partial Functions) が採用されている．これによって，関数的プログラミング言語での再帰性に起因する部分性を扱える様になっている．

また， VDM を開発した Bjørner らのグループが VDM の後継手法として生み出した RAISE (Regorous Approach to Industrial Software Engineering) — The RAISE Method Group (1995) — の仕様記述言語である RSL — The RAISE Language Group (1992) — は抽象データ型の定義法として代数仕様と同様の記述方式を導入しており， VDM-SL 以上に広帯域言語としての性格を強めている． RSL での代数仕様的な記述と関数的あるいは手続き的プログラミング風の記述とが一貫したモデルを持つか否かは The RAISE Language Group (1992) で示された RSL の言語仕様の定義だけからは判然としないが，やはり LPF を基底論理として採用しているのであろう．

■ COLD/VVSL

代数的仕様記述に λ-計算的な側面を導入した仕様記述言語の例としては， Feijs and Jonkers (1992), Feijs (1993), Feijs, Jonkers and Middelburg (1994) の COLD も挙げる事ができる．

COLD の場合， λ-計算的な言語機構はモジュールのパラメタ化の部分で用いられているが， COLD での λ-計算的なパラメタ機構の特徴は，仕様の詳細化に基づく擬順序によって仮パラメタを置換する実引数を規制する点（Fun での Λ-抽象化での型変数の置換が部分型関係 <: によって規制されていた事に類似）にある．

Feijs らは COLD でのパラメタ化機構に用いている λ-計算の体系を $\lambda\pi$-計算と呼んでいる．彼らの $\lambda\pi$-計算は，形式的体系としての側面に関する限り，(Fun でなく Funiq の場合の様に擬順序を成す）部分型関係に基づき上限型による制限を許す $\Lambda/\forall$ による型パラメタ化と類似の性質を満たす．

なお，$\lambda\pi$-計算を仕様のパラメタ化に用いている例としては，Middelburg (1993) によって VDM-SL の拡張として生み出された VVSL もある．VVSL は状態概念に基づくモデル指向の形式的仕様記述言語であり，基底の論理としては MPL_ω (Many-sorted Partial infinitary Logic) という無限長の論理式を許す部分論理の体系が採用されている．その結果，強力な表現力を得ているが，一方で，無限論理に共通した問題である（モデル論での）コンパクト性定理を喪失している筈である．従って，VVSL の実用的な面や仕様記述での表現力に関してはともかく，理論的な面に関して言えば，VVSL のモデル — 即ち，表示的意味論 — の世界ではコンパクト性定理から導かれるべき自然な性質が失われてしまっている．

■ Extended ML

関数的プログラミング言語の拡張として仕様記述まで行なおうとする試みとして，Standard ML (Milner, Tofte, Harper and MacQueen (1997)) にモジュールレベルでの仕様記述を追加した Kahn, Sannella and Tarlecki (1997) の Extended ML が挙げられる．Extended ML の場合，仕様とプログラムとの意味の対応は Goguen and Burstall (1992) で提案された institution という枠組で与えられている．

Standard ML やその広帯域化の拡張としての Extended ML は，共にモジュールのレベル（その「型」は `signature`，その実体は `structure`，その上の「関数」は `functor`）と通常の値（型，式，関数）のレベルとの二つのレベルに分離した言語という欠点を持つ．即ち，Fun/Funiq での様にモジュールを通常のデータとして扱う事は許されない．

Standard ML/Extended ML と Fun/Funiq との間のもう一つの相違点は，前者での `structure` と呼ばれるモジュール機能は Fun/Funiq での **pack** 式の様に単一の抽象データ型（の実現を隠蔽する）為の言語機能ではなく，様々な種類の定義や宣言を纏めてパッケージ化し，外部からの自由な参照を防ぐ情報隠蔽の為の言語機能だという点である．

その意味で，Standard ML の `structure` は Fun での **pack** 式よりも遥かに融通が利き強力な言語機能であるが，その代償として，Standard ML のモジュールでの情報隠蔽に対応する型（論理）での構成としての存在限量化型 ∃ は，本論文での多相型 ∀ と関数型構成子 _ → _ でコード化可能なもの —**「弱い ∃」**と呼ばれる — ではなく，メタ理論的により強い ∃-除去規則によって定義される**「強い ∃」**と呼ばれるものでなければならない事が MacQueen (1986) によって指摘されている．しかし，Jacobs (1998) や Luo (1994) が指摘している通り，強い ∃ は，それを含む型理論やその表示的意味論の構築で様々な障害を引き起こす．

また，広帯域言語としての Extended ML に対して，そのプログラミング言語としての Standard ML の部分に対する institution と，述語論理に基づき原則的には関数記号の表わす関数を全域的に扱う筈の仕様記述言語の部分に対する institution と，これら二つの異なる institution を対応付ける為の institution morphism とが具体的に構成され Extended ML 全体の表示的意味として実際に与えられた事はない（少なくともテクニカルレポートも含め発表された事はない）．実際，Milner et al. (1997) による Standard ML の形式的定義でさえも 100 ページ以上の書籍を必要としている事を振り返ると，Extended ML 全体に対する institution （上に述べた通り，二つの部分の各々に対する institution とその間の morphism から成る）の具体的な構成は相当な規模になると考えるのが妥当である．

従って，Extended ML が実際に institution morphism で繋ぎ合わせる事が可能な一貫した表示的意味を持つ広帯域言語であるか否かは Extended ML に対して具体的に発表された範囲からは判断できず，また，仮に Extended ML 全体に対する institution が与えられたとしても，第 5 章で示した Fun/Funiq に対する比較的コンパクトなサイズの表示的意味とは異なり，最早，人間が簡単に読んで内容を理解し活用できる規模を超える代物となる，と予想せざるを得ない．

■ Larch

Guttag and Horning (1993) による Larch は C, Clu, Modula-3 といった既存の手続き的プログラミング言語に対して，そのモジュールの仕様を代数仕様の形で記述を可能にする為のモジュール仕様記述言語を付加する興味深いアプローチである．

しかし，Larch によりモジュール仕様が記述されたモジュール仕様とそのモ

ジュールの手続き的プログラミング言語による実現との間の対応ならびに正しさは，両者の間の（非形式的な）名前の対応とそれに基づく等式による証明という純粋に構文的・証明論的なもので，仕様とプログラムとの間の正当性に関する意味論的な裏付けが与えられている訳ではない．

即ち，Larch の場合，モジュール仕様のレベルでは他の代数仕様言語の多くと同様に，等号を含んだ 1 階古典述語論理をベースの論理としており，関数名が表わすのは全域関数であるのに対し，それと対応付けられるプログラム中の関数は一般には部分関数であり得る（故にその為の論理体系としては Logic of Partial Function の類が必要），という事で，プログラムの仕様に対する正当性は，Larch の場合，厳密な意味に於いては非形式的レベルに留まっている．

■ 構成的型理論の諸体系

1.1 節での (1) の論理的アプローチの発展として，高階型理論の表現力の強さを用いて，構成的高階型理論に基づき等式論理に基づく抽象データ型に対する公理的な記述を (1) の枠組の中に取り込んでしまおうという立場は Martin-Löf (1975, 1982, 1984, 2000) の直観主義的型理論 (ITT) の体系に基づく Nordström, Petersson and Smith (1990) や Luo (1990, 1994) の ECC (Extended Calculus of Constructions) に於いて採用されている．特に，Luo (1994) では，無限の型宇宙の階層を用いて，仕様や単純な要素仕様から複雑な仕様を構成する仕様構成子，更には，仕様に対する実現の正しさの証明といった仕様から実現としてのプログラムまでの形式的な構成のプロセスの全てが型理論 ECC の枠組で統一的かつ洗練された形で定式化されており，仕様と実現との間の関連を形式的に捉える研究として非常に優れた仕事である．

しかし，以上の構成的型理論からのプログラミングに関するアプローチは，構成的論理に基づくが故に，少なくとも以下の三つの欠点を共通して抱えている：

(1) 構造帰納的に定義可能な関数しか扱えない．その結果，一般のプログラミング言語で記述が許される一般の再帰性 — 代償として停止しない部分関数を含んでしまう — は最初から排除されている．実際，一般の再帰性を許す為に本論文での不動点演算再帰構成 **fix** やそれに相当する不動点演算子 — 以下，これも **fix** で表わす — を導入すると，構成的論理の体系は無条件に矛盾してしまう．何故ならば，**fix** は，任意の型 σ について $\sigma \to \sigma$ の閉じた式 e が

与えられた時に $\mathbf{fix}(e)$ は σ 型の閉じた式となるが，これは構成的論理での「型 = 命題」の立場で言えば，任意の命題 σ について，$\sigma \to \sigma$ が証明可能であれば，σ 自身も証明可能である事を意味しているが，当然ながら，$\sigma \to \sigma$ は常に証明可能であるので，任意の命題 σ が証明可能である，即ち，**fix** を追加した構成的論理の体系では任意の命題が証明可能となり，論理として矛盾した体系となってしまう．従って，「型 = 命題」の立場に基づいた構成的型理論を用いる限り，不動点演算，もしくは，それと同等の一般の再帰性を許す事は不可能である；

(2) (1) の問題と深く関連する事であるが，第 5 章の脚注 (1) の最後の段落で触れた通り，一般の再帰性や（構造帰納的でない形で）高階関数を用いて記述され停止性が保証された関数的プログラムを構成的型理論のプログラムとして許されている構造帰納的な形に — しかも時間的計算量が悪化しない様に — 書き直す事は全く自明でない．即ち，その書き直し作業は構成的論理で構成可能な整礎構造だけの使用に限るという非常に厳しい制約の下で元の関数的プログラムの停止性を証明する事と等しく，元の関数的プログラムの停止性を単に（排中律が許された古典論理で）証明する事よりも困難である．この事は，手続き的プログラミングの場合で言えば，**while** ループを用いた停止性が保証されているプログラムを実行性能も劣化させず **for** ループのみを用いて書き直す事が一般には自明な作業ではない，という事に相当する；

(3) 構成的型理論での「等式」を用いた抽象データ型の仕様に対する実現は，それの実行に於いて，実用的な意味での計算 — つまり与えられた入力データに対して仕様通りの値を返す事 — 以外に，その実現が与えられた入力データに対して返す値が仕様通りである事の証明を含んでおり，少なくとも応用という観点からは余分な計算を必然的に含むプログラムしか得られないという意味で不満が残る．

最後の欠点を解決する為に，Nordström, Peterson and Smith (1990) では Subset Theory という部分集合を扱う為の拡張を ITT に導入して証明とプログラムとの分離を図っており，実用的には計算でなく検証に過ぎない実行時の計算を排除する事には成功しているが，同時に ITT を始めとする構成的型理論一般で最大の長所とさ

れていた仕様と命題との間，および，プログラムと証明との間の de Bruijn-Curry-Howard の対応 — de Brijn (1970, 1980); Curry and Feys (1958); Curry, Hindley and Seldin (1972), Howard (1980) — を，本論文での Funiq と同様，失ってしまっている．

更に， Subset Theory の場合も一般の再帰性は許されず，構造帰納的な形でしかプログラムを書く事ができない．従って，この体系でのプログラミングは，他の構成的型理論の体系でのプログラミングと同様に (2) の欠点，つまり作ろうとするプログラムの停止性を構成的に与える事のできる整礎関係を用いるという非常に制限された形で証明する事に相当し，通常の関数的プログラミング言語での自由に再帰的定義を利用できるプログラミングよりも遥かに窮屈で面倒な作業とならざるを得ない．

■ Erik Poll による $\lambda\omega$ のプログラミング論理

本研究と最も近い研究として Erik Poll (1994) による $\boldsymbol{\lambda\omega}$ のプログラミング論理の研究を挙げる事ができる． Poll は Barendregt の PTS (Pure Type Systems) — Barendregt (1992); Kamareddine, Laan and Nederpelt (2004) — に属する型付き λ-計算の体系，とりわけ $\boldsymbol{\lambda\omega}$ — Girard (1972) での F_ω を PTS の形で定式化した体系 — で記述されたプログラムを検証する為のプログラミング論理を与え，部分 cpo に基づく表示的意味論を与えた．

Poll (1994) では，プログラミング言語としての $\boldsymbol{\lambda\omega}$ を $\boldsymbol{\lambda\omega_s}$ と呼び，同時に $\boldsymbol{\lambda\omega}$ を一つの論理体系と看做した体系を $\boldsymbol{\lambda\omega_p}$ と呼び，両者の体系を PTS でのソート $*, \Box$ を二重化する事 — 一組はプログラミング言語の体系で用いる為，もう一組は論理で用いる為 — によってプログラミング論理を含んだ体系 $\boldsymbol{\lambda\omega_L}$ を構築した．

前項の構成的型理論によるアプローチと Poll の $\boldsymbol{\lambda\omega_L}$ の最大の違いは，プログラムと証明とを切り離した事にある．即ち， $\boldsymbol{\lambda\omega_L}$ の部分言語である $\boldsymbol{\lambda\omega_s}$ の項として表現されたプログラムはあくまでもプログラムに過ぎないのであって，構成的型理論の場合のプログラムの様な何らかの命題の証明となっている訳ではない．

一方， $\boldsymbol{\lambda\omega_L}$ での証明は，部分言語 $\boldsymbol{\lambda\omega_p}$ の項であり，それは $\boldsymbol{\lambda\omega_p}$ の型として表現された命題に対するものである．以上の二つの言語の関係を簡単に纏めると，全てのデータ型の集まりを表わすソートを $*_s$ で，また，それに相当する論理 $\boldsymbol{\lambda\omega_p}$ の側のソートを $*_p$ で，各々，表わすと，通常の意味でのデータ型 σ とその型に属する値に

関する性質 P とは，各々，

- $\sigma : *_s$ ；
- $P : \sigma \to *_p$

というソートを持つ．従って $\boldsymbol{\lambda\omega_L}$ でのプログラミングとは，これら σ と P とが具体的に与えられた時に，

- プログラム M．ここで， $M : \sigma$ ；
- その正しさの証明 p_M．ここで， $p_M : P(M)$

の両者を求める事に外ならない．

つまり， $\boldsymbol{\lambda\omega_L}$ で書かれた「プログラム」は，部分言語 $\boldsymbol{\lambda\omega_s}$ で書かれるべき本来のプログラム以外にその正当性の要請を表現する $\boldsymbol{\lambda\omega_p}$ による記述が注釈の形で付加された構造を持っている．

以上の様な 2 重構造の言語 $\boldsymbol{\lambda\omega_L}$ により， Poll はプログラムと論理とを対応付けながらも分離する事を可能とし，構成的型理論では不可能であった一般的な再帰性を与える不動点演算子の導入や実用的な意味での計算とは無関係な正当性証明をプログラムの実行から排除する事を達成した．

その意味では， Poll の $\boldsymbol{\lambda\omega_L}$ は，プログラムとプログラミング論理との双方をカバーする広帯域言語の為の体系としては非常に洗練され，また，高階述語論理[1)]に基づいており，本論文での不等式表明よりも遥かに強力な表現力を持つ体系である．

但し， $\boldsymbol{\lambda\omega_L}$ の場合，抽象データ型の基本演算の振舞を定めるべき表明はプログラミング論理 $\boldsymbol{\lambda\omega_p}$ で記述されねばならない．その結果，基本演算の振舞に関する表明はプログラミング言語としての $\boldsymbol{\lambda\omega_s}$ には属さず，基本演算の振舞も含めた抽象データ型間の継承関係（振舞に基づく部分型関係）をプログラミング言語 $\boldsymbol{\lambda\omega_s}$ のレベルで定義できない．

つまり，基本演算の多寡に基づく — Cardelli (1984) 本来の意味での — 継承関係と本論文での [S-REFINE] 則で表わされる様な基本演算の振舞（に関する表明の強

1) 元の $\boldsymbol{\lambda\omega}$ 自身は高階**命題**論理に対応する体系である．しかし， PTS としての定式化に於いて， $\forall$ （或いは Π）導入規則での全称化の束縛変数が値として採り得る集合を規定する公理としてのソート対の集合（PTS のパラメタ A）の中に $(*_s, \Box_p)$ を含める事でプログラミング論理を司る部分言語 $\boldsymbol{\lambda\omega_p}$ の型 — つまり「命題」 — が $\boldsymbol{\lambda\omega_s}$ の項 — つまり「値」 — に依存する事を許している．この公理ソート対を含めた事により， $\boldsymbol{\lambda\omega_L}$ でのプログラミング論理 $\boldsymbol{\lambda\omega_p}$ はプログラミング言語 $\boldsymbol{\lambda\omega_s}$ の項（つまり値）に言及可能な高階**述語**論理の表現力を有している．

弱）に基づく継承関係とは，$\boldsymbol{\lambda\omega_L}$ では全く異なるレベルの関係として別個に定義する必要がある．一方，本論文での Funiq の場合，基本演算の多寡と基本演算の振舞に関する表明の強弱とを併せて単一の部分型関係 <: として表現するので， Meyer (1997) が用いている意味での継承関係の忠実な定式化に成功している． $\boldsymbol{\lambda\omega_L}$ の場合にはそれと対照的であり，継承概念をプログラミング言語／プログラミング論理で記述し検証で利用する上での簡潔さや一貫性に欠ける．

■ 代数仕様の高階型への拡張

1.1 での (2) の代数的アプローチでの欠点として述べた高階型（関数型）等が従来の代数仕様では扱えない，という問題に関しては，高階関数と関数型で代数仕様を拡張しようという研究が行なわれている．その代表的なものとしては， Astesiano and Cerioli (1991), Haxthausen (1997), Meinke (1992, 1997), Mossakowski, Haxthausen and Krieg-Brückner (2000), Qian (1993) を挙げる事ができる．以上の高階関数と関数型による代数仕様の拡張の研究で共通しているのは，関数的プログラミング言語での高階関数や関数型の取り扱いに比べて以下の著しい不自然さが存在する事である：

(1) 高階関数の代数的枠組での扱いの為に， *eval*（一部の論文では *apply*）を対象言語レベルで — シグニチャの一つとして — 導入している事である．関数的プログラミング言語では，これはメタ言語 — 意味論上 — で扱われている．実際，関数を表わす項を引数項に作用させる為に対象言語レベルで *eval*を必要とするのであれば，概念的には，この *eval*自身もシグニチャの一員である単なる関数記号としてとして対象言語に属しているので， *eval*自体を関数項と引数項の対という *eval*への引数項に作用させる為に，更に高い型での *eval*を必要とする事になる筈であるが，無論，上記の各論文では，*eval*はシグニチャの一員でありながら， *eval*の引数項（それは関数項とその関数項への引数項から成るが）への作用自体は特別扱いしており，関数的プログラミング言語に於ける様な一貫性に欠けると同時に不自然でもある；

(2) 以上の研究の中で Haxthausen (1997), Mossakowski, Haxthausen and Krieg-Brückner (2000), Qian (1993) は，ソート間に包含関係を入れた順序ソート（本論文での部分型関係に相当する代数的アプローチでの概念）を許しているが，関数ソート構成子 "_ → _" の第 1 引数ソートに関する反単調性

の扱いの為に，二つの部分ソート関係 — Haxthausen (1997) の記法を用いると “$\leq^{\rightarrow}$” と “$\leq^{\Rightarrow}$” — を導入する必要が生じている．一方， Fun や本論文の体系の様に，部分型関係を許す関数的プログラミング言語での部分型関係は “<:” の一種類だけで済ませる事ができている．

以上の点から，代数的アプローチでの高階関数や関数型の扱いは未だ充分に洗練されているとは言い難い．

6.2　結果の纏めと今後の課題

前章までの議論から，抽象データ型の部分正当性仕様と実現との双方を記述する為の広帯域言語の雛型として本論文で Cardelli and Wegner の Fun に不等式表明を追加する形で提案した言語 Funiq ならびにその型システムを型理論として定式化した FUNIQ は：

(1) 証明論的な観点からは，型理論 FUN に対して保存的拡大（系 3.22）となっている．即ち， FUN の良い性質を全て引き継いでいる事や FUNIQ で型付けとして表わされている実現の仕様に対する部分正当性は型検査に対して忠実に拡張された概念である（定理 3.23）事が成立している；

(2) 簡約論（操作的意味論）的な観点からは，型付き λ-計算に基づく形式的体系に対して殆どの場合に期待される諸性質，即ち，簡約での型の保存を与える主部簡約性（定理 4.13）や式の操作的な評価の一意性を保証する簡約の合流性（定理 4.26），ならびに，不動点再帰を含まない式の場合には式の評価が必ず停止する事を保証する強正規化性（定理 4.17）を満たしている；

(3) 表示的意味論的な観点からは，型理論 FUNIQ は完備部分同値関係 (cper) に基づく意味論に対して健全（名前呼び意味論に関しては定理 5.89，値呼び意味論に関しては定理 5.96）である．特に，式が型を持つ事は式の値が型の表わす値の集合の要素であって型付け可能な式は実行時に型エラーを引き起こさない事（系 5.90）を意味しており，二つの型が部分型関係にある事は双方の型の表わす値の集合間の包含関係の成立（系 5.91）を意味している．更に，簡約によって定義された操作的意味論も表示的意味論に対して健全

（名前呼び意味論に関しては定理 5.92，値呼び意味論に関しては定理 5.97）である．

といった自然で望ましい様々な性質を満足しており，少なくとも理論的見地からは，型付き計算機言語とその型システムとして充分に妥当な性質を持つものだと判った．

更に，本論文での不等式表明を用いた詳細化型の導入による型理論の拡張方法は，3.4 節で検討した通り，ベースの型理論として FUN 以外の様々な 2 階の型理論 $\mathrm{F}_{\leq}$, F-bounded FUN, FUNNY に対する拡張でも保存的拡大定理（定理 3.26，定理 3.27, 定理 3.28）を成立させるという意味で，証明論的には非常に素直な性質を持ち一般性のある拡張方法である事が判り，その適用範囲に関しては，予想 3.29 の形で予想として示した．

今後の課題としては，無論，不等式表明に基づく詳細化型による拡張を本論文での型付き λ-計算ベースの体系とは全く性質の異なる様々な体系，例えば型付き π-計算の様な型付きプロセス代数に対しても具体的に行なってその性質を検討する，という事を挙げる事ができる．

それ以外の課題としては，意味論に関する第 5 章の 5.5 節の各項で述べた現在の cper に基づく意味論での問題点の解決を挙げる事ができる．

それらの意味論的な課題の中でも最も重要であると報告者が考えているのは，5.5.3 項で議論した再帰型を含む体系に対する（項モデルの様な計算の収束の概念を表わせない構文的なモデルではなくその概念を表わせる）領域論に基づく表示的意味を如何にして与えるのか，それを与える為の枠組となる数学的構造や手段は何か，を探求するという事である．その理由は：

(1) 再帰型の与える記述能力は実用的にも様々な応用を持つ事は明らかである；

(2) 5.5.3 項で述べた通り，FUNIQ に再帰型を追加した体系 FUNIQY は定理 3.28 によって無矛盾である事が保証されている；

(3) Funiq の型の場合も，計算の収束の概念は，ω-上昇列の上限に関して型の表示が閉じている，という意味で保持されている；

(4) 一方，再帰型の表示的意味を与える既存の研究で用いられている数学的構造の全てで本質的な役割を果たしている型の表示のコンパクト要素に関する閉包性が Funiq の型に関しては成立せず，従って，FuniqY の再帰型に対する

表示的意味の定義には，全く新しい発想に基づいた数学的構造が必須だ，という事が判明している．即ち，再帰型が追加された言語 FuniqY に対する表示的意味を与える為の数学的手段の探求は，**再帰型に対する意味定義での従来の数学的手段よりも真に強力な手段を提供できる**可能性が非常に大きく，再帰型の意味論に関して，理論的な面で本当に新しい一歩を踏み出して貢献できる可能性を有する．

以上の再帰型に対する表示的意味を如何に定義するかの問題とも関連する可能性があるが，直接的には独立な今後の課題としては，T-代数あるいはそれの適当な拡張を用いての Funiq の詳細化型に対する意味を与える可能性の追求である． Plotkin (1983) の第 5 章によれば，ω-帰納的述語は T-代数と深い関係がある．一方で，T-代数の利用に関する従来の研究 — 例えば Lehmann and Smyth (1981) や Manes and Arbib (1986) — では，代数仕様の言い方を借りればシグニチャから自由に（つまり等式公理が定める同値関係によって商代数をとる事なく）生成される代数に相当する T-代数の場合にしか T-代数に基づく意味が与えられていない．実際， Pierce は，彼の著書 (1991) の p. 41 に於いて，

> ... this construction works only for algebras without equations. The framework has apparently never been extended to include algebras with equations.

と述べている．

しかし，T-代数の定義を検討すると，この「自由な生成」は T-代数の枠組での意味定義が本質的に依存する性質には見えず， Funiq の詳細化型の様な（ω-帰納的述語と対応する）不等式の場合には，T-代数意味論の構築の可能性は充分に残されており，今後，挑戦すべき課題であると思われる．

参考文献

Abadí, M. and Cardelli, L. (1996):
A Theory of Objects, Springer-Verlag.

Abadí, M., Pierce, B. and Plotkin, G. (1989):
Faithful Ideal Models for Recursive Polymorphic Types, in *the 4th IEEE Conf. on Logic in Computer Science*, 216–225.

Abadí, M. and Plotkin, G. D. (1990):
A Per Model of Polymorphism and Recursive Types, in *the 5th IEEE Conf. on Logic in Computer Science*, 355–365.

Abramsky, S. (1990):
The Lazy Lambda Calculus, in *Research Topics in Funcitonal Programming* (D. A. Turner, Ed.), 65–116, Addison-Wesley.

Amadio, R. M. (1991):
Recursion over Realizability Structures, *Inform. Comput.* **91**, 55–85.

Amadio, R., Bruce, K. and Longo, G. (1986):
The Finitary Projection Model for Second Order Lambda Calculus and Solution to Higher Order Domain Equation, in *the 1st IEEE Conf. on Logic in Computer Science*, 122–130.

Amadio, R. M. and Curien, P.-L. (1998):
Domains and Lambda Calculi, Cambridge University Press.

America, P. (1991):
A Behavioural Approach to Subtyping in Object-Oriented Programming Languages, in *Inheritance Hierarchies in Knowledge Representation and Programming Languages* (M. Lenzerini et al., Eds.), 173-190, John Wiley & Sons.

Astesiano, E. and Cerioli, M. (2000):
Partial Higher-Order Specifications, in *Mathematical Foundations of Computer Science 1991: the 16th International Symposium* (A. Tarlecki, Ed.), Lecture Notes in Computer Science **520**, 74–83, Springer-Verlag.

de Bakker, J. W. (1980):
Mathematical Theory of Program Correctness, Prentice-Hall.

Barendregt, H. P. (1984):
The Lambda Calculus: Its Syntax and Semantics (the revised second edition), North-Holland.

Barendregt, H. P. (1992):
Lambda Calculi with Types, in *Handbook of Logic in Computer Science, Vol. 1* (D. M. Gabbay, S. Abramsky and T. S. E. Maibaum, Eds.), 118–309, Oxford University Press.

Barringer, H., Cheng, H. and Jones, C. B. (1984):
A Logic Covering Undefinedness in Program Proofs, *Acta Inf.* **21**, 251–260.

Bauer, F. L. and Wössner, H. (1982):
Algorithmic Language and Program Development, Springer-Verlag.

Bauer, F. L. et al. (The CIP Language Group) (1985):
The Munich Project CIP: Vol. I, The Wide Spectrum Language CIP-L, Lecture Notes in Computer Science **183**, Springer-Verlag.

Bauer, F. L. et al. (The CIP System Group) (1987):
The Munich Project CIP: Vol. II, The Program Transformation System CIP-S, Lecture Notes in Computer Science **292**, Springer-Verlag.

Bjørner, D. and Jones, C. B. (1978):
The Vienna Development Method: the Meta-Language, Lecture Notes in Computer Science **61**, Springer-Verlag.

Böhm, C. and Berarducci, A. (1985):
Automatic Synthesis of Typed Λ-Programs on Term Algebras, *Theoret. Comput. Sci.* **39**, 135–154.

Bruce, K. B. (2002):
Foundations of Object-Oriented Languages, The MIT Press.

Bruce, K. B. and Longo, G. (1988):
A Modest Model of Records, Inheritance and Bounded Quantification, in *the 3rd IEEE Conf. on Logic in Computer Science*, 38–50.

Bruce, K. B. and Longo, G. (1990):
A Modest Model of Records, Inheritance and Bounded Quantification, *Inform. Comput.* **87**, 196–240.

Bruce, K. and Meyer, A. R. (1984):
The Semantics of Second Order Polymorphic Lambda Calculus, in Kahn, MacQueen and Plotkin, Eds. (1984), 131–144.

Bruce, K., Meyer, A. R. and Mitchell, J. C. (1990):
The Semantics of Second Order Polymorphic Lambda Calculus, *Inform. Comput.* **85**, 76–134.

Bruce, K. and Mitchell, J. C. (1992):
PER Models of Subtyping, Recursive Types and Higher-order Polymorphism, in *the 16th ACM Symp. on Principles of Programming Languages*, 316–327.

Bruce, K. and Wegner, P. (1990):
An Algebraic Model of Subtype and Inheritance, in *Advances in Database Programming Languages* (F. Bancihon and P. Buneman, Eds.), 75–96, Academic Press.

de Bruijn, N. G. (1970):
The Mathematical Language AUTOMATH, in *Symposium on Automated Demonstration* (M. Laudet et al., Eds.), Lecture Notes in Mathematics **125**, 29–61, Springer-Verlag; reprinted in Nederpelt, Geuvers and de Vrijer, Eds. (1994), 73–100.

de Bruijn, N. G. (1980):
A Survey of the Project AUTOMATH, in Seldin and Hindley, Eds. (1980), 579-606, ; reprinted in Nederpelt, Geuvers and de Vrijer, Eds. (1994), 141–161.

Canning, P. Cook, W., Hill, W., Olthoff, W. and Mitchell, J. C. (1989):
F-Bounded Polymorphism for Object-Oriented Programming, in *Proceedings of the 4th Int'l Conf. on Functional Programming Languages and Computer Architecture FPCA '89*, 273–280, ACM Press.

Cardelli, L. (1984):
A Semantics of Multiple Inheritances, in Kahn, MacQueen and Plotkin, Eds. (1984), 51–67.

Cardelli, L. (1988):
A Semantics of Multiple Inheritances, *Inform. Comput.* **76**, 138–164.

Cardelli, L., Martini, S., Mitchell, J. C. and Scedrov, A. (1991):
An Extension of System F with Subtyping, in *TACS '91* (T. Ito and A. R. Meyer, Eds.), Lecture Notes in Computer Science **526**, 750–770, Springer-Verlag.

Cardelli, L. and Wegner, P. (1985):
On Understanding Types, Data Abstraction, and Polymorphism, *ACM Comput. Surv.* **17**, 471-522.

Cardone, F. (1989):
Relational Semantics for Recursive Types and Bounded Quantification, in *the 16th Int'l Colloq. on Automata Languages and Programming: ICALP '89* (G. Ausiello et al., Eds.), Lecture Notes in Computer Science **372**, 164–178, Springer-Verlag.

Cardone, F. (1991):
Recursive Types for Fun, *Theoret. Comput. Sci.* **83**, 29–56.

Cardone, F., Dezani-Ciancaglini, M. and de'Liguoro, U. (1994):
Combining Type Disciplines, *Annals of Pure and Appl. Logic* **66**, 197–230.

Cartwright, R. (1985):
Types as Intervals, in *the 12th ACM Symp. on Principles of Programming Languages*, 22–36.

Church, A. (1941):
The Calculi of Lambda Conversion, Princeton University Press.

Compagnoni, A. B. (1995):
Higher-Order Subtyping with Intersection Types, Ph. D. Thesis, Katholieke Universiteit Nijmegen.

Coquand, Th. and Huet, G. (1985):
Constructions: a Higher Order Proof System for Mechanizing Mathematics, in *EUROCAL '85, Vol. 1* (B. Buchberger, Ed.), Lecture Notes in Computer Science **203**, 151–184, Springer-Verlag.

Coquand, Th. and Huet, G. (1988):
The Calculus of Constructions, *Inform. Comput.* **76**, 95–130.

Crole, R. L. (1993):
Categories for Types, Cambridge University Press.

Curien, P.-L. and Ghelli, G. (1991):
Subtyping + Extensionality: Confluence of $\beta\eta$Top reduction in $F_{\leq}$, in *TACS '91* (T. Ito and A. R. Meyer, Eds.), Lecture Notes in Computer Science **526**, 731–749, Springer-Verlag.

Curien, P.-L. and Ghelli, G. (1992):
Coherence of Subsumption, Minimum Typing and Type-Checking in $F_{\leq}$, *Math. Struct. Comput. Sci.* **2**, 55–91.

Curry, H. B. and Feys, R. (1958):
Combinatory Logic: Vol. I,, North-Holland.

Curry, H. B., Hindley, J. R. and Seldin, J. P. (1972):
Combinatory Logic: Vol. II,, North-Holland.

Davey, B. A. and Priestley, H. A. (1990):
Introduction to Lattices and Orders, Cambridge University Press.

Dawes, J. (1991):
The VDM-SL Reference Guide, Pitman.

Diaconescu, R. and Futatsugi, K. (1998):
CafeOBJ Report, World Scientific.

Donahue, J. (1979):
On the Semantics of "Data Types", *SIAM J. Comput.* **8**, 546–560.

Ehrig, H. and Mahr, B. (1985):
Fundamentals of Algebraic Specification 1: Equations and Initial Semantics, Springer-Verlag.

Ehrig, H. and Mahr, B. (1990):
Fundamentals of Algebraic Specification 2: Module Specification and Constraints, Springer-Verlag.

Feijs, L. M. G. and Jonkers, H. B. M. (1992):
Formal Specification and Design, Cambridge University Press.

Feijs, L. (1993):
A Formalisation of Design Methods: A λ-calculus Approach to System Design with an Application to Text Editing, Ellis Horwood.

Feijs, L. M. G., Jonkers, H. B. M. and Middelburg, C. A. (1994):
Notations for Software Design, Springer-Verlag.

Felleisen, M., Friedman, D. P., Kohlbecker, E. and Duba, B. (1987):
A Syntactic Theory of Sequential Control, *Theoret. Comput. Sci.* **52**, 205–237.

Fokkinga, M. M. (1992):
Laws and Order in Algorithmics, Ph. D. Thesis, Universiteit Twente.

Fortune, S., Leivant, D. and O'Donnell, M. (1983):
The Expressiveness of Simple and Second-Order Type Structures, *J. Assoc. Comput. Mach.* **30**, 151–185.

Girard, J.-Y. (1971):
Une extension de l'interprétation de Gödel à l'analyuse et son application l'elimination des coupures dans l'analyse et la théorie des types, in *2nd Scandinavian Logic Symp.* (J. E. Fenstad, Ed.), 63–92, North-Holland.

Girard, J.-Y. (1972):
Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse d'Etat, Université Paris VII.

Girard, J.-Y. (1986):
The System F of Variable Types, Fifteen Years Later, *Theoret. Comput. Sci.* **45**, 159–192; reprinted in Huet, Ed. (1990), 87–126.

Girard, J.-Y. (1987):
Linear Logic, *Theoret. Comput. Sci.* **50**, 1–102.

Girard, J.-Y., Lafont, Y. and Taylor, P. (1989):
Proofs and Types, Cambridge University Press.

Goguen, J. A. and Burstall, R. M. (1992):
Institutions: Abstract Model Theory for Specification and Programming, *J. Assoc. Comput. Mach.* **39**, 95–146.

Gunter, C. (1992):
Semantics of Programming Languages: Structures and Techniques, The MIT Press.

Guttag, H. V. and Horning, J. J. (1993):
Larch: Languages and Tools for Formal Specification, Springer-Verlag.

Haxthausen, A. (1997):
Order-Sorted Algebraic Specifications with Higher-Order Functions, *Theoret. Comput. Sci.* **183**, 157–185.

Hindley, J. R. (1997):
Basic Simple Type Theory, Cambridge University Press.

Hindley, J. R. and Seldin, J. P. (1986):
Introduction to Combinators and λ-Calculus, Cambridge University Press.

Howard, W. A. (1980):
The Formula-as-Types Notion of Construction, in Seldin and Hindley, Eds. (1980), 479–490.

Huet, G., Ed. (1990):
Logical Foundations of Functional Programming, Addison-Wesley.

ISO (1996):
International Standard: Information Technology — Programming Languages, their environments and system software interfaces — Vienna Development Method — Specification Languages — Part 1: Base Language, ISO/IEC 13817-1, ISO.

Jacobs, B. P. F. (1991):
Categorical Type Theory, Ph. D. Thesis, Katholieke Universiteit Nijmegen.

Jacobs, B. (1998):
Categorical Logic and Type Theory, North-Holland.

Jung, A. (1989):
Cartesian Closed Categories of Domains, CWI Tract 66, Centrum voor Wiskunde en Informatica.

Jung, A. (1990):
Cartesian Closed Categories of Algebraic CPOs, *Theoret. Comput. Sci.* **70**, 233–250.

Kahn, G., MacQueen, D. B. and Plotkin, G., Eds. (1984):
Semantics of Data Types, Proceedings of International Symposium, Sophia-Antipolis, June 1984, Lecture Notes in Computer Science **173**, Springer-Verlag.

Kahrs, S., Sannella, D. and Tarlecki, A. (1997):
The Definition of Extended ML: A Gentle Introduction, *Theoret. Comput. Sci.* **173**, 445–484.

Kamareddine, F., Laan, T. and Nederpelt, R. (2004):
A Modern Perspective on Type Theory: from Its Origins until Today, Kluwer.

Krivine, J. L. (1993):
Lambda-calculus, Types and Models, Ellis Horwood; originally published in French as *Lambda-calcul, types et modèls*, Mason (1990).

Lambek, J. and Scott, P. J. (1990):
Introduction to Higher Order Categorical Logic, Cambridge University Press.

Lehmann, D. J. and Smyth, M. B. (1981):
Algebraic Specification of Data Types: A Synthetic Approach, *Math. Syst. Theory* **14**, 97–139.

Luo, Z. (1990):
An Extended Calculus of Constructions, Ph. D. Thesis, CST-65-90, Department of Computer Science, University of Edinburgh.

Luo, Z. (1994):
Computation and Reasoning: A Type Theory for Computer Science, Oxford University Press.

MacQueen, D. (1986):
Using Dependent Types to Express Modular Structures, in *the 13th ACM Symp. on Principles of Programming Languages*, 277–286.

MacQueen, D. and Sethi, R. (1982):
A Semantic Model of Types for Applicative Languages, in *Proceedings of 1982 ACM Symp. on Lisp and FunctionalProgramming*, 243–252.

MacQueen, D., Plotkin, G. and Sethi, R. (1984):
An Ideal Model for Recursive Polymorphic Types, in *the 11th ACM Symp. on Principles of Programming Languages*, 165–174.

MacQueen, D., Plotkin, G. and Sethi, R. (1986):
An Ideal Model for Recursive Polymorphic Types, *Inform. Comput.* **71**, 95–130.

Manes, E. G. and Arbib, M. A. (1986):
Algebraic Approaches to Progaram Semantics, Springer-Verlag.

Martini, S. (1987):
An Interval Model for Second Order Lambda Calculus, in *Category Theory and Computer Science* (D. H. Pitts et al., Eds.), 219–237.

Martini, S. (1988):
Bounded Quantifiers Have Interval Models, in *1988 ACM Conf. on LISP and Functional Programming*, 164–173.

Martin-Löf, P. (1975):
An Intuitionistic Theory of Types: Predicative Part, in *Logic Colloquium '73* (H. E. Rose and J. C. Shepherdson, Eds.), 73–118, North-Holland.

Martin-Löf, P. (1982):
Constructive Mathematics and Computer Programming, in *Logic, Methodology and Philosophy of Science VI* (L. J. Cohen et al., Eds.), 153–175, North-Holland.

Martin-Löf, P. (1984):
Intuitionistic Type Theory, Bibliopolis.

Martin-Löf, P. (1998):
An Intuitionistic Theory of Types, in *Twenty-Five Years of Constructive Type Theory* (G. Sambin and J. Smith, Eds.), 127–172, Oxford University Press.

McCracken, N. J. (1979):
An Investigation of a Programming Language with a Polymorphic Type Structure, Ph. D. Thesis, Syracuse University.

McCracken, N. J. (1984):
An Investigation of a Programming Language with a Polymorphic Type Structure, Ph. D. Thesis, Syracuse University.

Meinke, K. (1992):
Universal Algebra in Higher Types, *Theoret. Comput. Sci.* **100**, 385–417.

Meinke, K. (1997):
A Completeness Theorem for the Expressive Power of Higher-Order Algebraic Specifications, *J. Comput. Syst. Sci.* **54**, 502–519.

Meyer, B. (1997):
Object-Oriented Software Construction (the 2nd edition), Prentice Hall.

Middelburg, C. A. (1993):
Logic and Specification: Extending VDM-SL for Advanced Formal Specification, Chapman & Hall.

Mitchell, J. C. (1984):
Semantic Models for Second-Order Lambda Calculus, in *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, 289–299.

Mitchell, J. C. and Plotkin, G. D. (1985):
Abstract Types Have Existential Type, in *the 12th ACM Symp. on Principles of Programming Languages*, 37–51.

Mitchell, J. C. and Plotkin, G. D. (1988):
Abstract Types Have Existential Type, *ACM Trans. Prog. Lang. Syst.* **10**, 470–502.

Mitchell, J. C. (1996):
Foundations of Programming Languages, The MIT Press.

Milner, R., Tofte, M., Harper, R. and MacQueen, D. (1997):
The Definition of Standard ML (the revised edition), The MIT Press.

Mossakowski, T., Haxthausen, A. and Krieg-Brückner, B. (2000):
Subsorted Partial Higher-Order Logic as an Extension of CASL, in *Recent Trends in Algebraic Development Techniques: the 14th International Workshop, WADT'99* (D. Bert et al., Eds.), Lecture Notes in Computer Science **1827**, 126–145, Springer-Verlag.

Nederpelt, R., Geuvers, J. H. and de Vrijer, R. C., Eds. (1994):
Selected Papers on Automath, North-Holland.

Nordström, B., Petersson, K. and Smith, J. M. (1990):
Programming in Martin-Löf's Type Theory, Oxford University Press.

大堀淳 (1997):
「プログラミング言語の基礎理論」, 共立出版.

Partsch, H. A. (1990):
Specification and Transformation of Programs: A Formal Approach to Software Development, Springer-Verlag.

Paulson, L. (1987):
Logic and Computation: Interactive Proof with Cambrige LCF, Cambrige University Press.

Pierce, B. C. (1991):
Basic Category Theory for Computer Scientists, The MIT Press.

Pierce, B. C. (1992):
Bounded Quantification Is Undecidable, in *the 19th ACM Symp. on Principles of Programming Languages*, 305–315.

Pierce, B. C. (2002):
Types and Programming Languages, The MIT Press.

Pierce, B. C. and Turner, D. N. (1994):
Simple Type-Theoretic Foundations for Object-Oriented Programming, *J. Funct. Prog.* **4**, 207–247.

Pitts, A. M. (1987):
Polymorphism is Set Theoretic, Constructively, in *Category Theory and Computer Science* (D. H. Pitts et al., Eds.), Lecture Notes in Computer Science **283**, 12–39, Springer-Verlag.

Plotkin, G. D. (1977):
LCF Considered as a Programming Language, *Theoret. Comput. Sci.* **5**, 223–255.

Plotkin, G. D. (1983):
Domains, Advanced Postgraduate Course Notes, Department of Computer Science, University of Edinburgh, 1983; rendered into TEX by H. Kondoh and Y. Kashiwagi in 1992.

Plotkin, G. D., Abad´i, M. and Cardelli, L. (1994):
Subtyping and Parametricity, in *the 9th IEEE Conf. on Logic in Computer Science*, 310–319.

Poll, E. (1994):
A Programming Logic Based on Type Theory, Ph. D. Thesis, Technische Universiteit Eindhoven.

Prawitz, D. (1965):
Natural Deduction: A Proof-Theoretical Study, Almqvist and Wiksell.

Pree, W. (1995):
Design Patterns for Object-Oriented Software Development, The ACM Press.

Qian, Z. (1993):
An Algebraic Semantics of Higher-Order Types with Subtypes, *Acta Inf.* **30**, 569–607.

The RAISE Language Group (1992):
The RAISE Specification Language, Prentice Hall.

The RAISE Method Group (1995):
The RAISE Method Manual, Prentice Hall.

Reynolds, J. C. (1974):
Towards a Theory of Type Structure, in *Proceedings of Colloque sur la Programmation* (B. Robinet, Ed.), Lecture Notes in Computer Science **19**, 408–425, Springer-Verlag.

Reynolds, J. C. (1983):
Types, Abstraction and Parametric Polymorphism, in *Information Processing '83* (R. E. A. Mason, Ed.), 513–523, North-Holland.

Reynolds, J. C. (1984):
Polymorphism is not Set-Theoretic, in *Kahn, MacQueen and Plotkin, Eds.*, 145–156.

Reynolds, J. C. (1985):
Three Approaches to Type Structures, in *TAPSOFT Vol. 1: CAAP '85* (H. Ehrig et al., Eds.), Lecture Notes in Computer Science **185**, 97–138, Springer-Verlag.

Ronchi della Rocca, S. and Paolini, L. (2004):
The Parametric Lambda Calculus: A Metamodel for Computation, Springer-Verlag.

Schmidt, D. A. (1986):
Denotational Semantics: A Methodology for Language Development, Allyn and Bacon.

Schmidt, D. A. (1994):
The Structure of Typed Programming Languages, The MIT Press.

Scott, D. S. (1972):
Continuous Lattices, in *Toposes, Algebraic Geometry and Logic* (F. W. Lawvere, Ed.), Lecture Notes in Mathematics **274**, 97–136, Springer-Verlag.

Scott, D. S. (1976):
Data Types as Lattices, *SIAM J. Comput.* **5**, 522–587.

Scott, D. S. (1981):
Lectures on a Mathematical Theory of Computation, Oxford University PRG Technical Monograph No. 19.

Scott, D. S. (1982):
Domains for Denotational Semantics, in *9th Int'l Colloq. on Automata, Language and Programming: ICALP '82* (M. Nielsen et al., Eds.), Lecture Notes in Computer Science **140**, 577–613, Springer-Verlag.

Scott, D. S. (1993):
A Type-theoretical Alternative to ISWIM, CUCH, OWHY, *Theoret. Comput. Sci.* **121**, 411–440.

Seldin, J. P. and Hindley, J. R., Eds. (1980):
To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press.

Smyth, M. B. and Plotkin, G. D. (1982):
The Category Theoretic Solution of Recursive Domain Equations, *SIAM J. Comput.* **11**, 761–783.

Sommaruga, G. (2000):
History and Philosophy of Constructive Type Theory, Kluwer.

Stenlund, S. (1972):
Combinators, λ-Terms, and Proof Theory, D. Reidel.

Stoltenberg-Hansen, V., Lindstöm, I. and Griffor, E. R. (1994):
Mathematical Theory of Domains, Cambridge University Press.

高橋正子 (1991):
「計算論：計算可能性とラムダ計算」, 近代科学社.

Tennent, R. D. (1991):
Semantics of Programming Languages, Prentice-Hall.

Thompson, S. (1991):
Type Theory and Funcional Programming, Addison-Wesley.

Troelstra, A. N. (1971):
Notions of Realizability for Intuitionistic Arithmetic and Intuitionistic Arithmetic in All Finite Types, in *the 2nd Scandinavian Logic Symp.* (J. E. Fenstad, Ed.), 369–405, North-Holland.

Troelstra, A. N. (1971b):
Notes on Intuitionistic Second Order Arithmetic, in *Cambridge Summer School in Mathematical Logic* (A. R. D. Mathias and H. Roger, Eds.), Lecture Notes in Mathematics **337**, 171–205, Springer-Verlag.

Vickers, S. J. (1989):
Topology via Logic, Cambridge University Press.

Wadler, P. (1989):
Theorems for Free!, in *Proceedings of the 4th Int'l Conf. on Funct. Prog. FPCA '89*, 347–359.

Wadsworth, C. P. (1976):
The Relation between Computational and Denotational Properties for Scott's $\mathbf{D}_\infty$-Models of the Lambda-Calculus, *SIAM J. Comput.* **5**, 488–521.