リソースに基づく代数的アーキテクチャモデルのプロセス代数CCSによる実装について

著者	新田 直也, 蔭山 信二
雑誌名	甲南大学紀要. 知能情報学編
巻	14
号	2
ページ	149-179
発行年	2022-02-01
URL	http://doi.org/10.14990/00004168

論　文

リソースに基づく代数的アーキテクチャモデルのプロセス代数CCSによる実装について

新田直也[a], 蔭山信二[b]

[a] 甲南大学 知能情報学部 知能情報学科
神戸市東灘区岡本８―９―１, 658-8501
[b] 甲南大学大学院 自然科学研究科 知能情報学専攻
神戸市東灘区岡本８―９―１, 658-8501

（受理日 2021 年 11 月 29 日）

概 要

近年, システムの高分散化に伴い, ソフトウェアアーキテクチャを構成する上で, コンポーネント間のデータ転送方式の選択の重要性が高まってきている. データの転送方式は大きく PUSH 型と PULL 型に分けることができるが, いずれのデータ転送方式を採用するかが, システム全体の最終的な性能に大きな影響を及ぼす. そのため, 開発の初期の段階で適切にデータ転送方式を評価できることが望まれる.

一般にアーキテクチャ設計を正しく評価するためには, アーキテクチャの設計内容を厳密に定義できる体系が必要となる. そのため, アーキテクチャモデル及びアーキテクチャ記述言語の研究が広く行われてきた. 本研究室では, データ転送方式の選択および評価の支援を目的とした代数的アーキテクチャモデルの提案を行っている. 提案アーキテクチャモデルは, 制御の流れに関する情報を捨象することによって, プロセス代数などのプロセスに基づく他の記述体系よりも抽象度の高い情報を記述することを可能にしている. そのため, 提案アーキテクチャモデルによって記述されたアーキテクチャ情報は, データ転送方式の選択の影響を受けないという特徴を持つ.

本論文では, 既存の記述体系としてプロセス代数 CCS を選び, 提案アーキテクチャモデルの記述から, データ転送方式の選択に応じて CCS による実装を自動生成できることを示し, 生成された CCS 実装が元のアーキテクチャモデルと等価となることを証明する. また, 等価性を保ちながらデータ転送方式を変更するための十分条件を示す. これらによって, 提案アーキテクチャモデルに基づいてデータ転送方式をリファクタリングするための理論的な基盤の構築が可能となる.

キーワード: アーキテクチャ記述言語, プロセス代数, CCS, リソースアーキテクチャ

1　はじめに

大規模ソフトウェアの開発においては, 開発の初期の段階で適切なソフトウェアアーキテクチャを選択することが重要である. しかしながら実際のソフトウェア開発では, 選択したソフトウェアアーキテクチャの不備に気付かないままプロジェクトが進められ, 後の工程になって初めて, 現状のソフトウェアアーキテクチャのままでは非機能的要件や機能的要件の実現が十分にできないことに気付くと

いったことも少なくない. 開発が進んだ段階でアーキテクチャレベルの修正を行うことは, 非常に大きな手戻りを発生することになり現実的ではない. そのため, 開発の初期の段階で適切にソフトウェアアーキテクチャの評価を行うことができる手法の確立が求められている.

一般にソフトウェアアーキテクチャ評価の目的は, アーキテクチャを分析して潜在的なリスクを特定し, 設計レベルで品質要件が満たされていることを確認することである [1]. 近年, システムの高分散化に伴い, ソフトウェアアーキテクチャを構成する上で, コンポーネント間のデータ転送方式の選択の重要性が高まってきている. データの転送方式は大きく PUSH 型と PULL 型に分けることができる. **PUSH 型のデータ転送**とは, あるソフトウェアコンポーネントから別のソフトウェアコンポーネントに制御が移るときに, 制御の移動と同じ向きにデータを転送する方法であり, **PULL 型のデータ転送**とは制御の移動と逆向きにデータを転送する方法である. 例えばコンポーネント間でのメソッド呼び出しを考えたとき, 引数を使ったデータ転送が PUSH 型に, 戻り値を使ったデータ転送が PULL 型に相当する. より高い抽象度では, Web アプリケーションにおいてサーバからクライアントへのデータ転送を PUSH 型で行うか PULL 型で行うか, MVC アーキテクチャにおいてモデル更新時の情報をビューに PUSH 型で転送するか PULL 型で転送するかなど, データ転送方式の選択はアーキテクチャ設計において大きな部分を占める. データ転送方式の選択は, 非機能的要件の実現に大きな影響を与えるため [2], [3], アーキテクチャ設計時にその影響を見積もれることが望まれる.

アーキテクチャの設計を正しく評価するには, アーキテクチャの設計内容を厳密に定義できる体系が必要となる. そのため**アーキテクチャモデル**及び**アーキテクチャ記述言語** (Architecture Description Language, 以下 ADL と略す) の研究が広く行われてきた. アーキテクチャモデルとは, ソフトウェアアーキテクチャを特定の観点から抽象化したものであり, アーキテクチャ記述言語とは, アーキテクチャモデルを記述するための表現形式である. ADL は, 大きく非形式的なものと形式的なものに分けることができる. 非形式的なものの代表として, UML などの図的表記法 [4] が挙げられる. 形式的なものの代表としては, Wright [5], Darwin [6], Acme [7], AADL [8], CHARMY [9] などを挙げることができる. 形式的な ADL の特徴として, 検証やシミュレーション, プロトタイプの自動生成などが可能であることが挙げられる [10]. 形式的な ADL の多くはプロセスに基づいたものであり (たとえば, [5], [6], [9]), 制御の流れに依存する情報と依存しない情報が混在して記述される. そのため, アーキテクチャモデル上で PUSH と PULL の間でデータ転送方式を変更することは容易ではない.

本研究室では, データ転送方式の選択および評価の支援を目的とした代数的アーキテクチャモデル (以下, 本アーキテクチャモデルと略す) の提案を行っている. 本アーキテクチャモデルは, 制御の流れに関する情報を捨象することによって, プロセスに基づく他のアーキテクチャモデルよりも抽象度の高い情報を記述することを可能にしている. 具体的には, 本アーキテクチャモデルによって, 各コンポーネントの振る舞いとコンポーネント間のデータの流れが代数を用いて形式的に記述される. そのため, 本アーキテクチャモデルによって記述されたアーキテクチャ情報は, データ転送方式の選択の影響を受けないという特徴を持つ.

本論文では, 既存の記述体系としてプロセス代数 CCS を選び, 本アーキテクチャモデルの記述から, データ転送方式の選択に応じて CCS による実装を自動生成できることを示し, 生成された CCS 実装が元のアーキテクチャモデルと等価となることを証明する. また, 一般に PUSH 型と PULL 型の間でデータ転送方式を変更した場合に振る舞いの等価性が保たれるとは限らないため, 等価性を保ちつつデータ転送方式を変更するための十分条件を示す. これらによって, 本アーキテクチャモデルに基づいてデータ転送方式をリファクタリングするための理論的な基盤の構築が可能となる.

以降の本論文の構成は次のとおりである. 2 章では, データ転送方式の解説と実装例を示す. 3 章で提案アーキテクチャモデルの概要を説明し, 4 章で提案アーキテクチャモデルの形式的な定義を与える. 5 章で提案アーキテクチャモデルの記述から CCS による実装の自動生成法を説明し, 6 章で提案アーキテクチャモデルの記述と生成された CCS 実装の等価性の証明を行う. 7 章では, 5 章の結果に基づきデータ転送方式の変更について考察を行う.

2 例題システム

最初に, データ転送方式を PUSH 型と PULL 型に変更可能なシステム例を示す. 例題システムは, Web サービスとして実装された簡単な気象観測システムである (以下, WOS と略す). 簡単のため, このシステムでは単一の観測所のデータのみを扱うものとする. システムに要求される機能的要件は次の通りである. まず, システムが気象観測を行うたびにシステムに華氏による気温が入力され, 即座に摂氏による気温が計算される. 同時に, 測定された華氏による気温を元にその日の最高気温 (華氏による) も更新される. また, 日付が変わる毎にその日の最高気温がリセットされる.

ここで, このシステムを RESTful Web サービス [11] として, 3 つのリソース, すなわち華氏による現在の気温 (`temp_f` と記述), 摂氏による現在の気温 (`temp_c` と記述), その日の最新の最高気温 (`highest` と記述) によって構成していくことを考える. これらのリソースの間のデータフローを図 1 に示す. RESTful アーキテクチャでは, 各リソースは URI で特定可能な Web 上の情報に相当する. 各リソースには GET, PUT, POST, DELETE などの決められたメソッドを定義することができる. GET メソッドは**安全性**を満たすことが求められ, GET, PUT, DELETE メソッドは**冪等性**を満たすことが求められる. ここで, 安全性とは, メソッドの適用によってリソースの状態が変わらないことを意味し, 冪等性とは, メソッドの 2 回以上の適用が, 単独の適用と同じ結果になることを意味する. 特に, GET メソッドは, リソースの状態を取得するために用いられ, PUT および POST メソッドは, リソースの状態を更新するために用いられる.

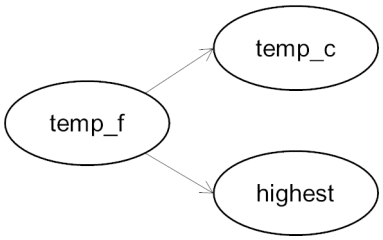
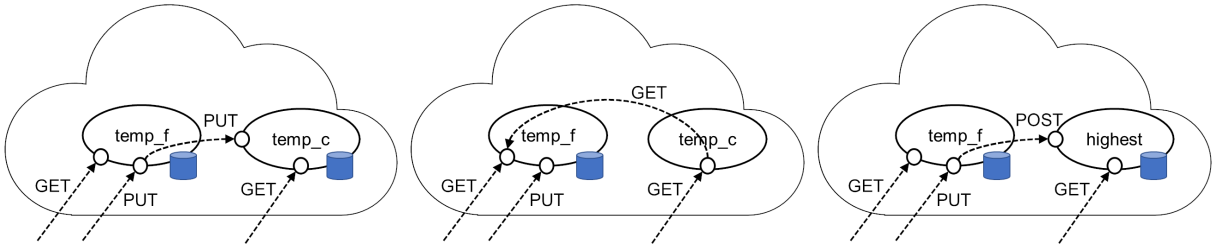


図 1: WOS のデータフロー

まず, `temp_f` リソースから `temp_c` リソースへのデータ転送を PUSH 型で実装することを考える (図 2 (a) 参照). PUSH 型では, いずれのリソース内にも最新の状態が保持され, その状態をレスポンスとして返すように GET メソッドが実装される. `temp_f` リソースの状態を新たに測定された値で更新するメソッドは, そのメソッドを 2 回以上適用しても状態が更新されないことから, PUT で実装される. `temp_c` の値は, `temp_f` に設定された値を元に即座に更新される必要があるため, PUSH 型の実装では, `temp_f` リソースの PUT メソッド内から `temp_c` リソースの PUT メソッドが呼び出されるように実装される.



(a) temp_f から temp_c への PUSH 型データ転送

(b) temp_f から temp_c への PULL 型データ転送

(c) temp_f から highest への PUSH 型データ転送

図 2: WOS において選択可能なデータ転送方式

次に, 同じデータ転送を PULL 型で実装することを考える (図 2 (b) 参照). PUSH 型の場合と同じように, temp_f リソースの最新の状態はリソース内に保持されるように実装され, その値が GET メソッドの呼び出しによって返される. しかしながら PULL 型では, temp_c リソースの最新の状態は temp_f リソースの最新の状態から常に計算して求めることができるため, どこにも保持しておく必要がない. したがって, temp_c リソースの GET メソッドは, temp_f リソースの最新の状態を得るために, 内部から temp_f リソースの GET メソッドを呼び出すように実装される. いっぽう PUSH 型の実装の場合とは異なり, temp_f リソースの PUT メソッド内から temp_c リソースの PUT メソッドは呼び出されない.

最後に, temp_f リソースから highest リソースへのデータ転送を実装することを考える. その PUSH 型による実装は, temp_f と temp_c の間のデータ転送の場合と類似している (図 2 (c) 参照). しかしながら, temp_f と highest の間のデータ転送は, PULL 型では実装することができないという点が大きく異なる. その理由は, highest リソースの場合, 最新の状態が temp_f リソースの最新の状態のみから求められないためである. highest リソースの最新の状態を得るためには, temp_f リソースの最新の状態と highest リソースの直前の状態の両方の情報が必要であり, そのため, highest リソースの状態を常にリソース内に保持しておく必要がある. さらにその状態は, temp_f リソースの状態が更新されるたびに更新される必要がある. これらの理由により, temp_f リソースから highest リソースへのデータ転送は, PUSH 型で実装されなければならない. なお, 日付が変わったときの highest リソースの状態のリセットであるが, この操作は冪等性を満たすため, highest リソースの PUT メソッドで実装すればよい.

上記の実装を比較することによって以下のことがわかる.

- **PULL 型で実装できないデータ転送がある.** 例題システムでは, temp_f リソースから highest リソースへのデータ転送を PULL 型で実装することができなかった. しかしながら, 図 1 のデータフローを見ている限りでは, temp_f と temp_c の間のデータ転送が PULL 型で実装できて, temp_f と highest の間のデータ転送が PULL 型で実装できない理由がわからない. すなわち, 転送元のリソースと転送先のリソースの間の具体的な関係がわからなければ, その間のデータ転送を PULL 型で実装できるか否かの判定をすることができない. そのため, 3 章ではリソースの状態間の関係を代数的に記述するアーキテクチャモデルを導入する. PULL 型で実装できる

か否かの厳密な判定基準については 5 章で説明するが, 直観的には転送元リソースの状態が更新されるたびに転送先リソースの状態を更新する必要があるかないかが判定基準となる.

- **状態の保持が必要なリソースと不必要なリソースがある.** 外部から見て常に固有の状態を持つかのように見え, かつその状態をいつでも取得できるようなリソースであっても, 実装上はその状態を保持する必要はない. たとえば, `temp_f` と `temp_c` の間のデータ転送を PULL 型で実装した場合は, `temp_c` リソースの状態をどこにも保持しなくても, その GET メソッドは常に正しく動作するが, PUSH 型で実装した場合はそのリソース内に状態を保持しなければ正しく動作しない. このように, リソース内部に状態を保持する必要があるかどうかは, その状態を PUSH 型のデータ転送で更新する必要があるかどうかによって決まる.
- **データ転送方式はシステムの非機能的側面に影響を及ぼす.** 本研究では, リソースの状態は常に外部から観測可能であると仮定するため, その状態変化はシステムの機能的要件と関係する. いっぽう, データ転送方式の選択は, システムの非機能的要件と強く関係する. たとえば, データ転送を PUSH 型で実装した場合, すべての転送先リソース内で状態を保持しておく必要があるため, メモリやストレージ上の記憶領域が増大する傾向がある. リソースの状態を更新する際の通信負荷は, PULL 型の実装よりも PUSH 型の実装の方が高くなるが, 逆にリソースの状態を取得する際の通信負荷は, PUSH 型の実装よりも PULL 型の実装の方が高くなる.

ここでの議論が RESTful アーキテクチャを前提としていたわけではないことに注意が必要である. 本研究では, 記述するアーキテクチャの対象を RESTful には限定していない.

3 リソースに基づく代数的アーキテクチャモデルの概要

2 章での議論を踏まえたうえで, 本研究室で提案している**代数的アーキテクチャモデル**の概要について説明する. アーキテクチャモデルの形式的な定義については次章で詳しく説明する. まず, データ転送方式の変更によって変わらない部分として, 図 1 で示したようなリソースの種類とリソース間のデータの流れを挙げることができる. さらに, システムと外部環境との相互作用, またシステム外部からの入力に応じて各リソースの状態がどのように遷移するかなども, データ転送方式の変更の影響を受けない部分である. また 2 章で議論したように, リソース間のデータ転送を PULL 型で実装できるか否かを判定するために, 転送元のリソースと転送先のリソースの間の具体的な関係を厳密に定義できる必要がある. いっぽう, データ転送方式を変更すると制御の流れが反転するため, 本アーキテクチャモデルにおいては制御の流れに関する情報が捨象される. この点がプロセス代数と大きく異なる部分である. 本研究室ではこれらの点すべてを考慮した代数的アーキテクチャモデルを提案している. 本アーキテクチャモデルを RESTful 以外のアーキテクチャにも適用できるように, 以降は**リソース**の定義を以下の条件を満たすソフトウェアコンポーネント c として一般化する.

1) c はシステムの外部から特定可能である.
2) c はそれ自身で状態を持つように見え, その状態はシステム外部から常に観測可能である.
3) c の状態は, c がメッセージを受信もしくは送信したときにのみ変化する.

本アーキテクチャモデルの概要を, 2 章で紹介した WOS の例を用いて説明する. 本アーキテクチャモデルにおいて, 各リソースは**メッセージ**を送受信することによって状態を変化させる状態遷移系としてモデル化される. いくつかの既存のアーキテクチャモデル [12], [13] においても, 各リソースがラベル付き状態遷移系 (LTS) としてモデル化されているが, LTS は状態空間が有限であるため, 自然数や実数, リストなど多くのデータ構造を表す上で十分な表現力を持たない. さらに, LTS はソースコードの自動生成にも適さない. なぜなら, 状態遷移から膨大な条件分岐が生成され, ソースコードの可読性が大きく損なわれてしまうためである. そのため本研究では, 無限の状態空間を持つことができるよう状態遷移系を代数を用いて定義する. たとえば, `temp_f` リソースの PUT メソッドをパラメータ x で呼び出したときの状態遷移は以下のように定義することができる.

$$\mathrm{temp\_f}(f, x) = x.$$

各遷移関数の第 1 引数は遷移前の状態, 第 2 引数は送受信されるメッセージを表し, 関数値は遷移後の状態を表す. 同様に `temp_c` リソースの状態と `highest` リソースの状態を更新する状態遷移関数は以下のように定義される.

$$\mathrm{temp\_c}(c, x) = (x - 32)/1.8.$$
$$\mathrm{highest}(h, x) = \mathrm{if}(x > h, x, h).$$

これらの定義において, 各リソースの遷移関数名にはリソース名をそのまま用いている. また, これら 3 つのリソースはメッセージ x を受信して同期して遷移すると仮定する.

ここで, なぜ `temp_f` から `temp_c` へのデータ転送が PULL 型で実装できるのかについて考える. 上記の式において, x は `temp_f` の新しい状態に対応している. ここで, `temp_c` の次の状態は x のみから計算できるのに対し, `highest` の次の状態 (すなわち, $\mathrm{if}(x > h, x, h)$) を求めるには, x に加えて `highest` の現在の状態 h も必要となる. この違いが, PULL 型で実装できるか否かを左右している. より一般的には, データの転送元リソースの状態遷移系からデータの転送先リソースの状態遷移系への準同型写像が存在することが, そのデータ転送が PULL 型で実装できるための十分条件になる. たとえば, `temp_f` から `temp_c` へのデータ転送では, $h(x) = (x - 32)/1.8$ という関数を考えると

$$\mathrm{temp\_c}(h(f), x) = (x - 32)/1.8 = h(x)$$

が成り立つため h は準同型写像となり, この h を用いて PULL 型の実装を行うことができる. いっぽう, `temp_f` の状態遷移系から `highest` の状態遷移系への準同型写像は存在せず, PULL 型で実装することができない. このように, 転送元のリソースと転送先のリソースの状態遷移系を比較することによって, PULL 型での実装可能性を判定することができる. また, 各メッセージの冪等性も状態遷移関数の性質として定義することができる. たとえば, 状態遷移関数 temp_c は以下のように冪等性を満たすため, temp_c の状態の更新は PUT メソッドで実装される.

$$\mathrm{temp\_c}(\mathrm{temp\_c}(c, x), x) = \mathrm{temp\_c}(c, x) = (x - 32)/1.8.$$

本アーキテクチャモデルでは, 状態遷移関数が比較できるように, データ転送の転送元リソースと転送先リソースの状態遷移関数を**チャンネル**によってグループ化する. 各チャンネルでは, 転送元リソースと転送先リソースが同一のメッセージに対して, 同期して状態遷移を行うと仮定する. 各チャンネル

は入力 (I), 出力 (O), 参照 (R) の 3 つのポートを持つものとする. 転送元のリソースと転送先のリソースは, それぞれ入力ポートと出力ポートに接続しているとする. これらの仮定は, 入力側に接続したリソースから出力側に接続したリソースにメッセージがチャンネルを通じて瞬時に転送され, 入力側リソースの状態遷移, データ転送, 出力側リソースの状態遷移がすべて同時に行われることを意味している. これによって, 本アーキテクチャモデルでは制御に関する情報が捨象され, データ転送方式の変更の影響を受けなくなる. ここで, `temp_f` から `temp_c` へデータがチャンネル c_1 を通じて転送されるとする. このとき, このデータ転送は本アーキテクチャモデルでは以下のように定義される.

$$\begin{aligned} \text{temp\_f}^{c_1,\mathrm{I}}(f, \text{convert}(y)) &= y, \\ \text{temp\_c}^{c_1,\mathrm{O}}(c, \text{convert}(y)) &= (y-32)/1.8. \end{aligned}$$

これらの式の中で, 関数名の右肩の表記は, リソースが接続しているチャンネル名とポート名を示している. 各チャンネルの入力ポートには一般に複数のリソースが接続できることに注意が必要である. システムと外部環境との相互作用は入出力用のチャンネルを通じて行われる. ただし, 各リソースの内部状態は入出力チャンネルを経由することなく常に外部から観測されているものとする.

具体的に, 2 章の WOS が本アーキテクチャモデルでどのようにしてモデル化されるかについて説明する. この例では, チャンネルとして $c_{\mathrm{I}/O_1}, c_{\mathrm{I}/O_2}, c_1, c_2$ の 4 つを用い, c_{I/O_1} および c_{I/O_2} を入出力チャンネルとする. まず, チャンネルとリソースの対応を表 1 に示す. また, 各リソースの状態遷移関数を図 3 に示す.

表 1: WOS におけるチャンネルとリソースの対応

チャンネル	入力側リソース	出力側リソース
c_{I/O_1}		temp_f
c_{I/O_2}		highest
c_1	temp_f	temp_c
c_2	temp_f	highest

本アーキテクチャモデルでは, 同一リソースの状態遷移関数が複数のチャンネルで重複して定義されるように見えるが, これらは同じ遷移の各チャンネル内での見え方の違いを表している. したがって, これらの関数間で第 1 引数の値および関数値は常に一致する. 例えば図 3 の例では, 任意の遷移において $x = y = z$ が成り立つ.

システム全体は, 外部環境から入出力チャンネルへのメッセージの入力に反応して動作する. 例えば WOS のチャンネル c_{I/O_1} にメッセージ measure(68) が入力されると, 最初に c_{I/O_1} の出力側の状態遷移関数 $\text{temp\_f}^{c_{\mathrm{I}/O_1},\mathrm{O}}$ が評価され, temp_f の状態が 68 になる. $\text{temp\_f}^{c_{\mathrm{I}/O_1},\mathrm{O}}$, $\text{temp\_f}^{c_1,\mathrm{I}}$, $\text{temp\_f}^{c_2,\mathrm{I}}$ はすべてリソース temp_f の状態遷移関数であるため, temp_f の遷移後の状態から $x = y = z = 68$ が求まる. これによりチャンネル c_1 および c_2 の入力側リソースの遷移後の状態が決定するため, c_1 および c_2 内にそれぞれメッセージ convert(68), update(68) が送信され, 出力側リソース temp_c, highest の状態がそれぞれ 20, 68 になる. ただし, リソース highest の直前の状態を 67 とする.

$$
\begin{array}{|rcl|}
\hline
\multicolumn{3}{|c|}{\mathrm{temp\_f}^{c_{\mathrm{I}/O_1},\mathrm{O}}(f, \mathrm{measure}(x)) \quad = \quad x,} \\
\hline
\multicolumn{3}{|c|}{\mathrm{highest}^{c_{\mathrm{I}/O_2},\mathrm{O}}(h, \mathrm{reset}(v)) \quad = \quad v,} \\
\hline
\mathrm{temp\_f}^{c_1,\mathrm{I}}(f, \mathrm{convert}(y)) & = & y, \\
\mathrm{temp\_c}^{c_1,\mathrm{O}}(c, \mathrm{convert}(y)) & = & (y-32)/1.8, \\
\hline
\mathrm{temp\_f}^{c_2,\mathrm{I}}(f, \mathrm{update}(z)) & = & z, \\
\mathrm{highest}^{c_2,\mathrm{O}}(h, \mathrm{update}(z)) & = & \mathrm{if}(z > h, z, h). \\
\hline
\end{array}
$$

図 3: WOS の代数的アーキテクチャモデル

4 リソースに基づく代数的アーキテクチャモデルの定義

4.1 多ソート代数

本論文では標準的な**多ソート代数** [14] を用いてチャンネルでの状態遷移関数とメッセージ転送を定義する. T をソートの全体集合とする. T 上の**ソート付きシグネチャ** Σ を $\Sigma = \langle \Sigma_{w,t} \rangle_{w \in T^*, t \in T}$ と定義する. ここで $\Sigma_{w,t}$ は, ソート列 w の引数を取り, ソート t の値を返すソート付き演算子シンボルの集合である. ϵ を空のシーケンスとし, 引数を持たない $a \in \Sigma_{\epsilon,t}$ を**定数シンボル**と呼ぶ. たとえば, $0 \in \Sigma_{\epsilon,\mathrm{int}}$, $\mathrm{true}, \mathrm{false} \in \Sigma_{\epsilon,\mathrm{bool}}$ は定数シンボルである. 変数の集合 X は, $X = \langle X_t \rangle_{t \in T}$ で定義される. ここで, X_t はソート $t \in T$ の変数の列挙可能な集合である. 与えられたソート $t \in T$ と与えられた変数の集合 X に対して, X 上の t の Σ **項**の集合を以下の式を満たす最小の集合 $T_\Sigma(X)_t$ として定義する.

1) $\Sigma_{\epsilon,t} \cup X_t \subseteq T_\Sigma(X)_t$,

2) 各 i $(1 \leq i \leq n)$ について, $f \in \Sigma_{t_1 t_2 \cdots t_n, t}$ かつ $a_i \in T_\Sigma(X)_{t_i}$ ならば, $f(a_1, a_2, \ldots, a_n) \in T_\Sigma(X)_t$.

たとえば, 図 3 では, $X_{\mathrm{real}} = \{x, y, z, v, f, c, h\}$, および $\{1.8, 32, x, y, z, v, f, c, h, y-32, (y-32)/1.8, \mathrm{temp\_f}^{c_{\mathrm{I}/O_1},\mathrm{O}}(f, \mathrm{measure}(x))\} \subset T_\Sigma(X)_{\mathrm{real}}$ となる. 以下では, 簡単のため, $T_\Sigma(\emptyset)_t$ を $T_{\Sigma,t}$ と, $\bigcup_{t \in T} T_{\Sigma,t}$ を T_Σ と書く. Σ 項 t に現れるすべての変数の集合を $\mathrm{var}\,(t)$ で表す. Σ 項 t に対して, t 内の変数 $v_1, \ldots, v_n$ のすべての出現を Σ 項 $t_1, \ldots, t_n$ に置き換えることによって得られる項を $t[t_1/v_1, \ldots, t_n/v_n]$ で表す. Σ の等式とは $l = r$ の形式の Σ 項の対であり, あるソート $t \in T$ および変数の集合 X に対して, $l, r \in T_\Sigma(X)_t$ を満たす. 本論文では, 非決定的な状態遷移が許されるよう $\mathrm{var}\,(r) \subseteq \mathrm{var}\,(l)$ を想定しない.

4.2 アーキテクチャモデルの基本定義

リソースに求められている外的振る舞いを表す代数的アーキテクチャモデルを $\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ で定義する. ただし,

- R: リソースの有限集合

- C: チャンネルの有限集合
- ρ ($\rho : C \times \{\mathrm{I}, \mathrm{O}, \mathrm{R}\} \to 2^R$): 各チャンネルの各ポート (入力, 出力, 参照) に接続しているリソースの集合
- T: ソートの有限集合
- τ ($\tau : R \to T$): 各リソースの状態のソート
- μ ($\mu : C \to T$): 各チャンネルを流れるメッセージのソート
- Σ: T 上のソート付きシグニチャ
- $\Delta = \langle \delta_r^{c,d} \rangle_{c \in C, d \in \{\mathrm{I,O,R}\}, r \in \rho(c,d)}$ (ただし, $\delta_r^{c,d} \in \Sigma_{\tau(r)\,\mu(c),\,\tau(r)}$): 状態遷移関数のシンボルの有限集合
- Γ: T 上の等式の有限集合
- s_0: 初期状態 (ただし, $r \in R$ について $s_0(r) \in \Sigma_{\epsilon,\tau(r)}$)

とする. それぞれの詳細については以下で説明する.

4.3 リソースとチャンネル

リソースとは, システム外部から見て固有の状態を持つように見え, かつその状態が常に観測できるようなコンポーネントである. システムで扱うリソース全体からなる集合を R で表す. 本論文のモデルでは R を有限集合とし, システムの実行中にリソースの生成, 消滅は発生しないものとする. なお, 各リソースは一般に無限の状態空間を持つことができる. **チャンネル**は複数のリソースの状態遷移を連動させるために用いられる. チャンネル全体からなる集合を C とし, 各チャンネルは入力ポート, 出力ポート, 参照ポートをそれぞれ 1 つずつ持つことができ, 各ポートには任意の数のリソースが接続することができる. チャンネル $c \in C$ の入力ポート, 出力ポート, 参照ポートに接続しているリソースの集合を, それぞれ $\rho(c, \mathrm{I})$, $\rho(c, \mathrm{O})$, $\rho(c, \mathrm{R})$ で表し, それぞれ入力側リソース, 出力側リソース, 参照側リソースと呼ぶ. ここで、$\rho(c, \mathrm{I}) \cap \rho(c, \mathrm{O}) = \rho(c, \mathrm{O}) \cap \rho(c, \mathrm{R}) = \rho(c, \mathrm{I}) \cap \rho(c, \mathrm{R}) = \emptyset$ と仮定する. 入力側のいずれかのリソースが状態遷移を行うと, チャンネル上にメッセージが送信され, それを受け取った出力側のリソースが状態遷移を行う. なお, C は空でない入出力チャンネルの集合 $C_{\mathrm{I/O}}$ を含むものとする. すべての入出力チャンネル $c_{\mathrm{I/O}} \in C_{\mathrm{I/O}}$ は入力側のリソースを持たない. すなわち, $\rho(c_{\mathrm{I/O}}, \mathrm{I}) = \emptyset$ が成り立つ.

4.4 ソートとシグニチャ

本論文では多ソート代数を用いてチャンネルでの状態遷移関数とメッセージ転送を定義する. まず, 各リソースの状態集合と各チャンネルを流れるメッセージの集合はともに固有のソートを持つと仮定

し, T をソート全体からなる集合とする. 各リソース $r \in R$ について, $\tau(r)$ は r の状態のソートを表し, 各チャンネル $c \in C$ に対して, $\mu(c)$ は c 上を流れうるメッセージのソートを表す.

Σ は T 上のソート付きシグニチャである. 各チャンネル $c \in C$ に対して, 少なくとも 1 つの演算シンボル $m \in \Sigma_{w,\mu(c)}$ ($w \in T^*$) が存在し, これを c に定義された **API シンボル**と呼ぶ. 例えば, 図 3 の例では, $\mathrm{measure} \in \Sigma_{\mathrm{real},\mu(c_{\mathrm{I}/O_1})}$ はチャンネル c_{I/O_1} に, $\mathrm{convert} \in \Sigma_{\mathrm{real},\mu(c_1)}$ はチャンネル c_1 に定義された API シンボルである. また各チャンネルには, API シンボルとは別に, 操作をしないことを表す単位元シンボル $e_c \in \Sigma_{\epsilon,\mu(c)}$ が定義されているものとする.

Δ は**状態遷移関数**のシンボルの有限集合であり, 各リソースの各チャンネルへの接続毎にちょうど 1 つのシンボルを含む. ここで, リソース r がチャンネル c の d 側に接続している場合 (すなわち, $r \in \rho(c,d)$ が成り立つとき), 対応する状態遷移関数のシンボルを $\delta_r^{c,d}$ で表す. このとき, $\delta_r^{c,d}$ のソートは, $\delta_r^{c,d} \in \Sigma_{\tau(r)\,\mu(c),\,\tau(r)}$ となる. すなわち, そのリソースの遷移前の状態を第 1 引数, そのチャンネルを流れるメッセージを第 2 引数にとって, 遷移後の状態を返す関数を表しており, この関数は部分関数にも全域関数にも, また単一値関数にも多値関数にもなりうる. 例えば図 3 の例では, $\tau(\mathrm{temp\_f}) = \mathrm{real}$ とすると, $\delta_{\mathrm{temp\_f}}^{c_1,\mathrm{I}} = \mathrm{temp\_f}^{c_1,\mathrm{I}} \in \Sigma_{\mathrm{real}\,\mu(c_1),\,\mathrm{real}}$ となる.

4.5 状態遷移関数の代数的定義

状態遷移関数は Σ 上の等式の有限集合 Γ を用いて定義される. 各等式は, 適当な変数の集合 $X = \langle X_t \rangle_{t \in T}$ 上の Σ 項を用いて定義される. 各状態遷移関数シンボル $\delta_r^{c,d} \in \Delta$ について, Γ は $d = \mathrm{O}$ のとき,

$$\delta_r^{c,\mathrm{O}}(x_s, t_m) = t'_s \tag{1}$$

の形の等式を, $d = \mathrm{I}$ のとき,

$$\delta_r^{c,\mathrm{I}}(x_s, t_m) = x'_s \tag{2}$$

の形の等式を, $d = \mathrm{R}$ のとき,

$$\delta_r^{c,\mathrm{R}}(x_s, t'_m) = x'_s \tag{3}$$

の形の等式をそれぞれちょうど 1 つずつ含むものとする. ただし, $x_s, x'_s \in X_{\tau(r)}$, $t'_s \in T_\Sigma(X)_{\tau(r)}$, $t_m, t'_m \in T_\Sigma(X)_{\mu(c)}$ とする. また, $\mathrm{var}(t'_s) \subseteq \mathrm{var}(t_m) \cup \{x_s\}$ とし, t'_m は x'_s を含まないものとする. なお, $\delta_r^{c,\mathrm{O}}$ は常に全域かつ単一値関数になるが, $\delta_r^{c,\mathrm{I}}$ および $\delta_r^{c,\mathrm{R}}$ については, t_m や t'_m が x_s を含むならば, それらは部分関数になる可能性があり, いっぽう t_m や t'_m が x'_s を含まないならば, それらは多値関数になる可能性があることに注意されたい.

4.6 データフローグラフとアーキテクチャモデルの妥当性

システム全体の状態遷移を定義する前に, まず代数的アーキテクチャモデルのデータフローグラフの定義を行う. $\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ を任意の代数的アーキテクチャモデルとする. このとき, $\mathcal{R}$ の**データフローグラフ**を, 有効グラフ $G_{\mathcal{R}} = \langle N_R, N_C, E \rangle$ として定義する. ただし, $N_R = R$, $N_C = C$, かつ $E = \{\langle c, r \rangle \mid c \in C, r \in \rho(c, \mathrm{O})\} \cup \{\langle r, c \rangle \mid c \in C, r \in \rho(c, \mathrm{I})\}$ とする. $\mathcal{R}$ の各入力チャンネル $c \in C_{\mathrm{I/O}}$ について, c の**動的データフローグラフ**とは, すべての頂点が頂点 $c \in N_C$ から

到達可能であるような $G_{\mathcal{R}}$ の最大部分グラフ $G_{\mathcal{R},c}$ のことをいう. $c \in C$ を任意のチャンネルとし, $\rho(c,\mathrm{I}) = \{r_1,\ldots,r_l\}$, $\rho(c,\mathrm{R}) = \{r'_1,\ldots,r'_k\}$ とする. 任意の $s_{r_1}, s'_{r_1},\ldots,s_{r_l}, s'_{r_l}, s_{r'_1}, s'_{r'_1},\ldots,s_{r'_k}, s'_{r'_k}$ に対して,

$$\begin{aligned}
\delta_{r_1}^{c,\mathrm{I}}(s_{r_1},m) &= s'_{r_1},\\
&\vdots\\
\delta_{r_l}^{c,\mathrm{I}}(s_{r_l},m) &= s'_{r_l},\\
\delta_{r'_1}^{c,\mathrm{R}}(s_{r'_l},m) &= s'_{r'_1},\\
&\vdots\\
\delta_{r'_k}^{c,\mathrm{R}}(s_{r'_k},m) &= s'_{r'_k}
\end{aligned}$$

を満たす m がただ 1 つに定まる場合, c において**メッセージが一意に定まる**という. また, 以下の条件が満たされるとき, $\mathcal{R}$ は**妥当**であるという.

- $G_{\mathcal{R}}$ は強連結成分を持たない.
- 任意の入力チャンネル $c \in C_{\mathrm{I/O}}$ について, $G_{\mathcal{R},c}$ は木構造になる.
- すべてのチャンネル $c \in C$ について, メッセージが一意に定まる.

4.7 システム全体の状態遷移

システム全体の状態は, すべてのリソースの状態の組み合わせとして定義することができる. リソース r の状態がソート $\tau(r)$ の定数シンボルで表されることから, システム全体の状態 s を, 各リソースからその状態を表す定数シンボルへの写像とみなす. すなわち, s は, 各リソース $r \in R$ について $s(r) \in \Sigma_{\epsilon,\tau(r)}$ を満たす. これは初期状態 s_0 についても同様である.

システム全体は, 外部環境から入出力チャンネルへのイベント入力に反応して動作する. 具体的には, アーキテクチャモデル $\mathcal{R}$ において, システム全体の状態が, 入出力チャンネル $c_{\mathrm{I/O}} \in C_{\mathrm{I/O}}$ に入力されたメッセージ m によって s から s' に遷移するとき,

$$s \overset{\langle m, c_{\mathrm{I/O}}\rangle}{\underset{\mathcal{R}}{\rightarrow}} s'$$

と書き, この遷移が可能であるとき, かつそのときに限り, 以下の条件を満たすような, **チャンネルへのメッセージの割り当て** $\pi : C \to T_\Sigma$ が存在するとする.

- すべての $c \in C$ について, $\pi(c) \in T_{\Sigma,\mu(c)}$.
- $\pi(c_{\mathrm{I/O}}) = m$.

- 各 $c \in C$ について, c が $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に含まれるならば, 各 $d \in \{\mathrm{I},\mathrm{O},\mathrm{R}\}$ および各 $r \in \rho(c,d)$ について,

$$\delta_r^{c,d}(s(r),\pi(c)) \begin{cases} = s'(r) & (\delta_r^{c,d}\text{が単一値関数}) \\ \ni s'(r) & (\delta_r^{c,d}\text{が多値関数}). \end{cases} \tag{4}$$

- 各 $c \in C$ について, c が $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に含まれないならば, $\pi(c) = e_c$, かつ以下の条件を満たすすべての r について $s(r) = s'(r)$.
  - $r \in \rho(c,\mathrm{I})$ を満たすか, もしくは
  - $r \in \rho(c,\mathrm{O})$ を満たし, r が $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に含まれない.

これは, 入出力チャンネル $c_{\mathrm{I/O}}$ から到達可能なチャンネルしか使わないこと, および同一のリソースはどのチャンネルから見ても同じ遷移を行うことなどを意味している.

5 代数的アーキテクチャモデルからの CCS 実装の自動生成法

並行システムを記述する形式的体系としてプロセス代数 CCS [15] が良く知られている. 本研究では, 代数的アーキテクチャモデルから実行可能なプログラムを自動生成することを目的の 1 つとしているが, その前段階として, 代数的アーキテクチャモデルを CCS を用いて実装することを考える. CCS を用いると, プロセス間のメッセージの送受信に加えて, プロセス間およびプロセス内の制御の移動も扱うことができるため, PUSH 型のデータ転送と PULL 型のデータ転送を区別して記述することが可能である. そこで本章では, 与えられた代数的アーキテクチャモデルに対して, モデル中の各データ転送を PUSH 型で行うか PULL 型で行うかを選択することによって, 選択されたデータ転送方式に応じた CCS の実装を自動で導出する手法について考える.

5.1 プロセス代数 CCS

CCS の記述はプロセスによって構成される. 通信ラベル全体からなる集合を $\mathcal{A}$, プロセス定数全体からなる集合を $\mathcal{K}$ としたとき, プロセス式 P の構文は $\mathcal{A}$ と $\mathcal{K}$ を用いて

$$P ::= \mathbf{0} \mid A(e_1,\ldots,e_n) \mid a.P \mid \overline{a}.P \mid a(x).P \mid \overline{a}(e).P \mid \tau.P \mid P_1 + P_2 \mid (P_1 \mid P_2) \mid P \setminus L \mid P[f]$$

で定義される. ここで, P, P_1, P_2 はプロセス式, $\mathbf{0}$ は停止プロセス, $A \in \mathcal{K}$ は次数 n ($n \geq 0$) のプロセス定数, $a \in \mathcal{A}$ は入力ラベル, $\overline{a} \in \overline{\mathcal{A}}$ は出力ラベル, τ は内部アクション, x は変数, $e, e_1, \ldots, e_n$ は値を表す. また, $P_1 + P_2$ は P_1 と P_2 のうちのいずれかを選択する選択動作, $P_1 \mid P_2$ は P_1 と P_2 を並行に動作させる並行動作, $P \setminus L$ は $L \subseteq \mathcal{A} \cup \overline{\mathcal{A}}$ に属するアクションによる遷移を制限する動作制限, $P[f]$ はリラベリング関数 f によるリラベリングを表す.

各プロセス定数 $A \in \mathcal{K}$ に対して, 以下のようなプロセス定義式を高々1 つ定義することができる.

$$A(x_1,\ldots,x_n) \stackrel{\mathrm{def}}{=} P$$

ただし, n は A の次数であり, P はプロセス式である.

CCS の操作的意味論は次の通りである. プロセス P がアクション α を行って P' に遷移するとき, $P \xrightarrow{\alpha} P'$ と書く. この遷移関係を以下の推論規則を満たす最小の関係として定義する.

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \text{ Act},$$

$$\frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \text{ Sum}_1, \qquad \frac{P_2 \xrightarrow{\alpha} P_2'}{P_1 + P_2 \xrightarrow{\alpha} P_2'} \text{ Sum}_2,$$

$$\frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2} \text{ Com}_1, \qquad \frac{P_2 \xrightarrow{\alpha} P_2'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1 \mid P_2'} \text{ Com}_2,$$

$$\frac{P_1 \xrightarrow{\overline{\alpha}} P_1' \quad P_2 \xrightarrow{\alpha} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'} \text{ Com}_3, \qquad \frac{P_1 \xrightarrow{\overline{\alpha}(e)} P_1' \quad P_2 \xrightarrow{\alpha(x)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'[e/x]} \text{ Com}_4,$$

$$\frac{P \xrightarrow{\alpha} P' \quad \alpha, \overline{\alpha} \notin L}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \text{ Res},$$

$$\frac{P[e_1/x_1, \ldots, e_n/x_n] \xrightarrow{\alpha} P' \quad A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P}{A(e_1, \ldots, e_n) \xrightarrow{\alpha} P'} \text{ Con}.$$

5.2 PUSH 型/PULL 型共通の実装

$\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ を代数的アーキテクチャモデルとする. $\mathcal{R}$ の CCS による実装方法はデータ転送方式の選択に応じて一般に複数通り可能であるが, 本研究では, どのような転送方式を選んでも, リソース毎に定義されたサブプロセスの並列合成によって実装されるものとする. 具体的には, リソースの集合を $R = \{r_1, r_2, \ldots, r_n\}$ とし, $\mathcal{R}$ のある状態 s が $s(r_i) \in \Sigma_{\epsilon, \tau(r_i)}$ $(1 \leq i \leq n)$ を満たすとき, $\mathcal{R}$ の CCS による実装 $\mathcal{P}_\mathcal{R}$ を,

$$\begin{aligned} &\mathcal{P}_\mathcal{R}(s, x_1, \ldots, x_L) \\ &= P_{r_1}(s(r_1), x_1^{r_1}, \ldots, x_{l_1}^{r_1}) \mid \cdots \mid P_{r_n}(s(r_n), x_1^{r_n}, \ldots, x_{l_n}^{r_n}) \end{aligned} \tag{5}$$

のような形式で表す. ただし, $L = \sum_{i=1}^{n} l_i$ とし, リソース r_i $(1 \leq i \leq n)$ 毎に P_{r_i} がパラメータ $s(r_i)$ を持つ場合と持たない場合があるとする. これは, r_i が自分の状態を保持する場合と保持しない場合があることを意味する. また, 各パラメータ $x_j^{r_i}$ は r_i が内部に保持しているキャッシュを表す. リソース $r \in R$ が与えられたときの r に対応する $\mathcal{P}_\mathcal{R}$ のサブプロセスを

$$\Phi(\mathcal{P}_\mathcal{R}(s, x_1, \ldots, x_L), r) = \begin{cases} P_r(s(r), x_1, \ldots, x_l) & (s(r) \text{ を持つ場合}) \\ P_r(x_1, \ldots, x_l) & (s(r) \text{ を持たない場合}) \end{cases}$$

で表す. 各サブプロセス P_r はさらに 1 つ以上のメソッドの選択合成によって表される. 具体的には, P_r は必ず 1 つの getter メソッド $\Psi^{\mathrm{get}}(P_r)$ を持ち, また, 出力側に r が接続している各チャンネル $c \in C$ (ただし, $r \in \rho(c, \mathrm{O})$) について, $c \in C_{\mathrm{I/O}}$ ならば c に対応した input メソッド $\Psi_c^{\mathrm{input}}(P_r)$ を 1 つと, $c \notin C_{\mathrm{I/O}}$ なら, c の入力側リソースを $\{r_1, \ldots, r_l\} = \rho(c, \mathrm{I})$ とおいたとき, c および各 r_j $(1 \leq j \leq l)$ に対応した update メソッド $\Psi_{c,r_j}^{\mathrm{update}}(P_r)$ をたかだか 1 つ持つ. すなわち,

$$P_r \stackrel{\mathrm{def}}{=} \Psi^{\mathrm{get}}(P_r) \overbrace{+\Psi_{c_{\mathrm{I/O}}}^{\mathrm{input}}(P_r)}^{c_{\mathrm{I/O}} \in C_{\mathrm{I/O}}, r \in \rho(c_{\mathrm{I/O}}, \mathrm{O})} \overbrace{+\Psi_{c,r_1}^{\mathrm{update}}(P_r) + \cdots + \Psi_{c,r_l}^{\mathrm{update}}(P_r)}^{c \notin C_{\mathrm{I/O}}, r \in \rho(c, \mathrm{O}), \{r_1, \ldots, r_l\} = \rho(c, \mathrm{I})} \tag{6}$$

となる.

入力チャンネル $c_{\mathrm{I/O}} \in C_{\mathrm{I/O}}$ に対して, $c_{\mathrm{I/O}}$ の出力側に接続しているリソースの集合を $\rho(c_{\mathrm{I/O}}, \mathrm{O}) = \{r_1, \ldots, r_k\}$ とおく. $\mathcal{R}$ における $c_{\mathrm{I/O}}$ へのメッセージ m の入力 $\langle m, c_{\mathrm{I/O}} \rangle$ に対応する $\mathcal{P}_{\mathcal{R}}$ のアクション列は $\eta_{c_{\mathrm{I/O}}, r_1}(m)\, \overline{\eta'_{c_{\mathrm{I/O}}, r_1}} \cdots \eta_{c_{\mathrm{I/O}}, r_k}(m)\, \overline{\eta'_{c_{\mathrm{I/O}}, r_k}}$ となり, これを $\widehat{\langle m, c_{\mathrm{I/O}} \rangle}$ で表す. ここで, $\eta_{c,r}(m)$ および $\eta'_{c,r}$ は, それぞれ入力チャンネル c からサブプロセス $\Phi(\mathcal{P}_{\mathcal{R}}, r)$ への入力メッセージ m とその応答を意味する. プロセス間のアクション列 α による遷移関係 $\stackrel{\alpha}{\Rightarrow}$ を以下のように定義する. a をアクションとする.

- $a \neq \tau$ のとき, $X \stackrel{a}{\rightarrow} Y$ ならば, $X \stackrel{a}{\Rightarrow} Y$.
- $X \stackrel{\alpha}{\Rightarrow} Y$ かつ $Y \stackrel{a}{\rightarrow} Z$ ならば, $X \stackrel{\alpha a}{\Rightarrow} Z$.
- $X \stackrel{\tau}{\rightarrow} Y$ かつ $Y \stackrel{\alpha}{\Rightarrow} Z$ ならば, $X \stackrel{\alpha}{\Rightarrow} Z$.
- $X \stackrel{\alpha}{\Rightarrow} Y$ かつ $Y \stackrel{\tau}{\rightarrow} Z$ ならば, $X \stackrel{\alpha}{\Rightarrow} Z$.

$\mathcal{R}$ への長さ n の入力列 $\sigma = \langle m_1, c_1 \rangle \cdots \langle m_n, c_n \rangle$ に対応するアクション列 $\widehat{\sigma} = \widehat{\langle m_1, c_1 \rangle} \cdots \widehat{\langle m_n, c_n \rangle}$ を以下では長さ n の**入力メッセージ列**と呼ぶ. また入力列 σ に対する $\mathcal{R}$ の遷移 $\stackrel{\langle m_1, c_1 \rangle}{\underset{\mathcal{R}}{\longrightarrow}} \cdots \stackrel{\langle m_n, c_n \rangle}{\underset{\mathcal{R}}{\longrightarrow}}$ を $\stackrel{\sigma}{\underset{\mathcal{R}}{\Longrightarrow}}$ と書く.

定義 1. *(*制御の到達可能性*)*

代数的アーキテクチャモデル $\mathcal{R}$ について, $\mathcal{P}_{\mathcal{R}}(s, x_1, \ldots, x_L)$ を $\mathcal{R}$ の *CCS* による実装とする. $\mathcal{P}_{\mathcal{R}}(s, x_1, \ldots, x_L)$ からリソース r に対応するサブプロセス $P_r = \Phi(\mathcal{P}_{\mathcal{R}}(s, x_1, \ldots, x_L), r)$ を取り除いたプロセスを $\mathcal{P}_{\mathcal{R}}(s, x_1, \ldots, x_L) \backslash P_r$ とおく. このとき, チャンネル $c_{\mathrm{I/O}}$ への入力 m について, 適当なプロセス P' が存在し,

$$P_r \xrightarrow{\eta_{c_{\mathrm{I/O}}, r}(m)} P'$$

が成り立つか, あるいは以下の *2* 式 *(*ただし, P', $\mathcal{P}'$, $\mathcal{P}''$ は適当なプロセスとする*)* を満たすメッセージ μ およびリソース r' が存在するとき, $\mathcal{P}_{\mathcal{R}}(s, x_1, \ldots, x_L)$ において, チャンネル $c_{\mathrm{I/O}}$ への m の入力によって**制御が r に到達**するという.

$$P_r \xrightarrow{\mu} P'.$$

$$\mathcal{P}_{\mathcal{R}}(s, x_1, \ldots, x_L) \backslash P_r \xrightarrow{\eta_{c_{\mathrm{I/O}}, r'}(m)} \mathcal{P}' \stackrel{\overline{\mu}}{\Longrightarrow} \mathcal{P}''.$$

□

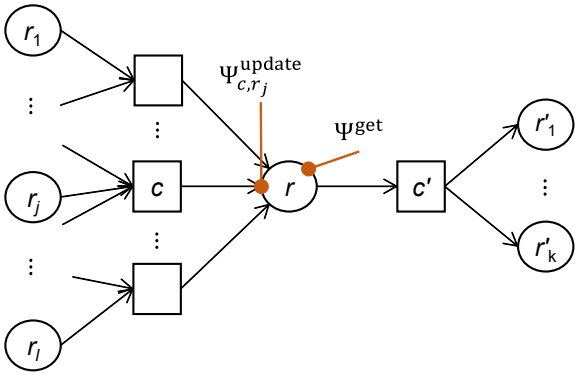


図 4: PUSH 実装

5.3 PUSH 優先の実装

$\mathcal{R}$ の CCS による PUSH 優先実装 $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ を以下のように定義する. まず, 各リソース $r \in R$ に対するサブプロセスを P^{PSH}_r とおく.

$$\Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s, x_1, \ldots, x_L), r) = P^{\mathrm{PSH}}_r(s(r), x^r_1, \ldots, x^r_l).$$

ただし, 各パラメータおよび l の内容については後述する.

次に, P^{PSH}_r の getter メソッドを以下のように定義する.

$$\Psi^{\mathrm{get}}(P^{\mathrm{PSH}}_r(s(r), x^r_1, \ldots, x^r_l)) \stackrel{\mathrm{def}}{=} \mathrm{get}_r.\overline{\mathrm{get'}_r}(s(r)).P^{\mathrm{PSH}}_r(s(r), x^r_1, \ldots, x^r_l). \tag{7}$$

ただし, get_r, $\mathrm{get'}_r$ は, r の getter メソッドの呼び出しと復帰を表す.

以下では, 簡単のため入力側に r が接続しているチャンネルが c' のみ, すなわち $r \in \rho(c'', \mathrm{I}) \Leftrightarrow c'' = c'$ である場合について考え, c' の出力側に接続しているリソースの集合を $\rho(c', \mathrm{O}) = \{r'_1, \ldots, r'_k\}$ とおく. また, 出力側に r が接続しているチャンネルに関しては, それらのチャンネルの入力側に接続しているすべてのリソースからなる集合を $\{r'' \in R \mid c'' \in C, r'' \in \rho(c'', \mathrm{I}), r \in \rho(c'', \mathrm{O})\} = \{r_1, \ldots, r_l\}$ とおく (図 4 参照). このとき, P^{PSH}_r の各パラメータ x^r_j $(1 \leq j \leq l)$ は, リソース r が内部に保持しているリソース r_j の状態のキャッシュを表すものとする. そのため $\mathcal{R}$ の状態 s に対応する P^{PSH}_r の状態は, $P^{\mathrm{PSH}}_r(s(r), s(r_1), \ldots, s(r_l))$ で表され, これを

$$P^{\mathrm{PSH}}_r(s) \stackrel{\mathrm{def}}{=} P^{\mathrm{PSH}}_r(s(r), s(r_1), \ldots, s(r_l))$$

と書く. さらに, s に対応する $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ 全体の状態を

$$\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \stackrel{\mathrm{def}}{=} P^{\mathrm{PSH}}_{r_1}(s) \mid \cdots \mid P^{\mathrm{PSH}}_{r_n}(s)$$

と書く.

また, $r \in \rho(c, \mathrm{O})$ を満たす各チャンネル c について, $c \notin C_{\mathrm{I/O}}$ なら, $r_j \in \rho(c, \mathrm{I})$ を満たす各 r_j

$(1 \leq j \leq l)$ について, c および r_j に対応した update メソッドを以下のように定義する.

$$\begin{aligned}\Psi^{\mathrm{update}}_{c,r_j}(P^{\mathrm{PSH}}_r(s(r), x^r_1, \ldots, x^r_l)) \stackrel{\mathrm{def}}{=}\ & \eta_{c,r_j,r}(y, y').\\ & \overline{\eta_{c',r,r'_1}}(x, x').\eta'_{c',r,r'_1}.\\ & \quad\vdots\\ & \overline{\eta_{c',r,r'_k}}(x, x').\eta'_{c',r,r'_k}.\\ & \overline{\eta'_{c,r_j,r}}.\\ & P^{\mathrm{PSH}}_r(x', x^r_1, \ldots, x^r_{j-1}, y', x^r_{j+1}, \ldots, x^r_l). \end{aligned} \tag{8}$$

ここで, $\eta_{c,r,r'}$ および $\eta'_{c,r,r'}$ は, リソース r' の状態を更新するために r から r' にチャンネル c を使って送るメッセージとその応答を表す. また, x' はリソース r の次の状態, すなわち $x' = \delta^{c,\mathrm{O}}_r(s(r), m)$ を表し, m は各 h $(1 \leq h \leq l)$ に対して,

$$\begin{cases} \delta^{c,\mathrm{I}}_{r_h}(y, m) = y' & (h = j)\\ \delta^{c,\mathrm{I}}_{r_h}(x^r_h, m) = x^r_h & (h \neq j) \end{cases}$$

を満たすチャンネル c 上のメッセージを表す.

また, $r \in \rho(c, \mathrm{O})$ を満たす各チャンネル c について, $c \in C_{\mathrm{I/O}}$ のとき, P^{PSH}_r の c に対応した input メソッドを以下のように定義する.

$$\begin{aligned}\Psi^{\mathrm{input}}_c(P^{\mathrm{PSH}}_r(s(r), x^r_1, \ldots, x^r_l)) \stackrel{\mathrm{def}}{=}\ & \eta_{c,r}(m).\\ & \overline{\eta_{c,r,r'_1}}(s(r), x').\eta'_{c,r,r'_1}.\\ & \quad\vdots\\ & \overline{\eta_{c,r,r'_k}}(s(r), x').\eta'_{c,r,r'_k}.\\ & \overline{\eta'_{c,r}}.\\ & P^{\mathrm{PSH}}_r(x', x^r_1, \ldots, x^r_l). \end{aligned} \tag{9}$$

ここで, $\eta_{c,r,r'}$ および $\eta'_{c,r,r'}$ の意味は上の場合と同様であり, x' は r の次の状態, すなわち $x' = \delta^{c,\mathrm{O}}_r(s(r), m)$ を表す.

5.4 PULL 型の実装

PUSH 優先実装の各データ転送を PULL 型に変更することを考える. まず, データ転送元のリソースを r とし, r に対応するサブプロセスを P_r とおく. PUSH 優先実装の場合と同様, 簡単のため入力側に r が接続しているチャンネルが c' のみ, すなわち $r \in \rho(c'', \mathrm{I}) \Leftrightarrow c'' = c'$ である場合について考え, c' の出力側に接続しているリソースの集合を $\rho(c', \mathrm{O}) = \{r'_1, \ldots, r'_k\}$ とおく. r と c' の間のデータ転送方式が PULL 型であった場合, 各 r'_i $(1 \leq i \leq k)$ に対応したサブプロセスは状態を保持しない, すなわち, $\Phi(\mathcal{P}_{\mathcal{R}}(s, \ldots), r'_i) = P_{r'_i}(\ldots)$ となり, また $P_{r'_i}$ は r と c' に対応した update メソッドを持たない. すなわち,

$$\Psi^{\mathrm{update}}_{c',r}(P_{r'_i}(\ldots)) \stackrel{\mathrm{def}}{=} \bot.$$

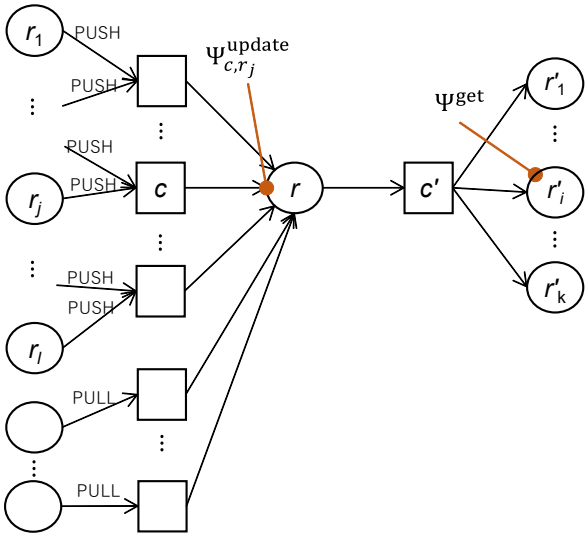


図 5: PULL 実装

さらに, $P_{r'_i}$ の getter メソッドは,

$$\begin{aligned}\Psi^{\mathrm{get}}(P_{r'_i}(\ldots)) \stackrel{\mathrm{def}}{=}\ & \mathrm{get}_{r'_i}.\\ & \vdots\\ & \overline{\mathrm{get}_r}.\mathrm{get'}_r(x_r).\\ & \vdots\\ & \overline{\mathrm{get'}_{r'_i}}(f_r(\ldots,x_r,\ldots)).\\ & P_{r'_i}(\ldots)\end{aligned} \tag{10}$$

で定義される. ただし, f_r は, PUSH 優先実装と振る舞いが等価になるような適当な関数とする. また, プロセス $P_{r'_i}$ は, r に関するキャッシュは持たない.

出力側に r が接続している各チャンネル c'' に関して, c'' の入力側に接続しているリソース r'' のうち, r'' と c'' の間のデータ転送方式が PUSH 型のもののみからなる集合を $\{r'' \in R \mid c'' \in C, r'' \in \rho(c'',\mathrm{I}), r \in \rho(c'',\mathrm{O}), r''$ と c'' の間が PUSH 型データ転送 $\} = \{r_1,\ldots,r_l\}$ とおくと (図 5 参照), 各リソース r_j $(1 \le j \le l)$, および $r_j \in \rho(c,\mathrm{I})$ かつ $r \in \rho(c,\mathrm{O})$ を満たすチャンネル c について, r_j と c に対応する r の update メソッドは以下のように定義される.

$$\begin{aligned}\Psi^{\mathrm{update}}_{c,r_j}(P_r(x^r_1,\ldots,x^r_j,\ldots,x^r_l)) \stackrel{\mathrm{def}}{=}\ & \eta_{c,r_j,r}(y,y').\\ & \overline{\eta'_{c,r_j,r}}.\\ & P_r(x^r_1,\ldots,y',\ldots,x^r_l).\end{aligned} \tag{11}$$

上記のようにして, $\mathcal{R}$ の PUSH 優先実装の 0 個以上のデータ転送を, 以下の条件を満たす範囲で PULL 型に変更した実装を $\mathcal{P}_{\mathcal{R}}$ とおく. これは CCS による $\mathcal{R}$ の可能なあらゆる実装を表す. 以下の条

件は $\mathcal{R}$ と $\mathcal{P}_{\mathcal{R}}$ の両方に関係しており, $\mathcal{P}_{\mathcal{R}}$ が $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ と等価となることを保証するためのものである. この条件を満たしているときに, $\mathcal{P}_{\mathcal{R}}$ が $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ と等価になることは次の章で証明する.

条件 1: $r \in \rho(c, \mathrm{I})$ を満たすチャンネル $c \in C$ およびリソース $r \in R$ について, r と c の間のデータ転送方式が PULL 型であるなら, $r' \in \rho(c, \mathrm{O})$ および $r' \in \rho(c', \mathrm{I})$ を満たすすべての $r' \in R$ と $c' \in C$ について,

1) r' と c' の間のデータ転送方式も PULL 型でなければならず,
2) また r' が出力側に接続しているチャンネルは c のみ, すなわち $r' \in \rho(c'', \mathrm{O}) \Rightarrow c'' = c$ を満たす.

条件 2: $r \in \rho(c, \mathrm{I})$ を満たすチャンネル $c \in C$ およびリソース $r \in R$ について, r と c の間のデータ転送方式が PULL 型であるなら, $\rho(c, \mathrm{I}) = \{r_1, \ldots, r_l\}$ とおくと, $r' \in \rho(c, \mathrm{O})$ を満たすすべてのリソース $r' \in R$ について, $\mathcal{R}$ の初期状態 s_0 に対して,

$$s_0(r') = f_{r'}(s_0(r_1), \ldots, s_0(r_l)) \tag{12}$$

が成り立ち, $\mathcal{R}$ の任意の状態 s およびチャンネル c 上の任意のメッセージ $m \in \mu(c)$ に対して,

$$\delta_{r'}^{c,\mathrm{O}}(f_{r'}(s(r_1), \ldots, s(r_l)), m) = f_{r'}(\delta_{r_1}^{c,\mathrm{I}}(s(r_1), m), \ldots, \delta_{r_l}^{c,\mathrm{I}}(s(r_l), m)) \tag{13}$$

が成り立つ. これは, チャンネル c において, 入力側リソースの状態遷移系から出力側リソースの状態遷移系への準同型写像 $f_{r'}$ が存在していることを示している.

例 1. 図 *6* に示したアーキテクチャモデルにおいて, リソース r_3 からチャンネル c_7 へのデータ転送方式が *PULL* 型であるため, 条件 *1* の *1)* より, リソース r_6 からチャンネル c_8 およびリソース r_7 からチャンネル c_9 へのデータ転送方式も *PULL* 型でなければならない. いっぽう, r_5 が出力側に接続しているチャンネルが c_5 と c_6 の *2* つ存在しているため, 条件 *1* の *2)* より, リソース r_1 からチャンネル c_5 およびリソース r_2 からチャンネル c_6 へのデータ転送方式は *PUSH* 型でなければならない. □

6 代数的アーキテクチャモデルと生成された CCS 実装の等価性

本章では, 5 章で定義された自動生成法によって代数的アーキテクチャモデルから生成された CCS 実装が, 選択したデータ転送方式によらず, 元の代数的アーキテクチャモデルと等価になることを証明する.

6.1 PUSH 優先 CCS 実装の等価性

本節では, まず生成された PUSH 優先実装が元の代数的アーキテクチャモデルと常に等価になることを証明する.

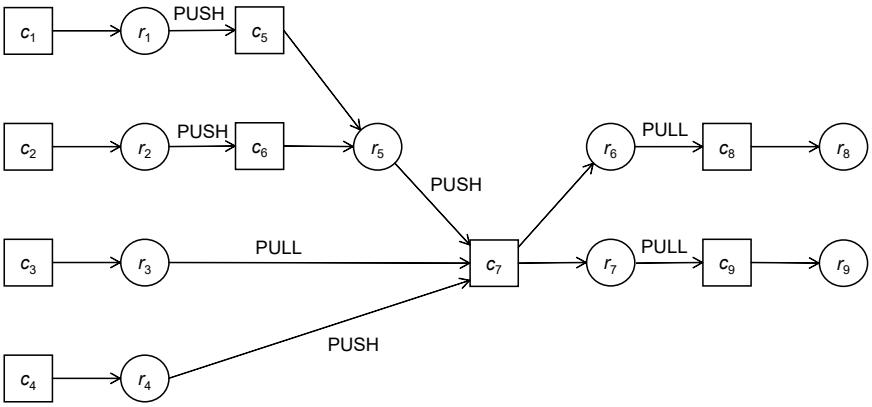


図 6: データ転送方式の PULL 型への変更可能性

補題 1. *(PUSH 優先実装における GET メソッドの安全性)*

任意の代数的アーキテクチャモデル $\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ が与えられたとき, 任意の状態 s, 任意のリソース $r \in R$ に対して, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s, x_1, \ldots, x_L) \overset{\mathrm{get}_r \widehat{\mathrm{get}'_r(s(r))}}{\Longrightarrow} \mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s, x_1, \ldots, x_L)$ が成り立つ.

証明. 式 (5), (6), (7) より明らか. □

補題 2. *(PUSH 優先実装における共通部分の等価性)*

$\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ を任意の代数的アーキテクチャモデルとし, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ を $\mathcal{R}$ の *PUSH* 優先による実装とする. また, $\mathcal{R}'$ を $\mathcal{R}$ からリソース $\tilde{r}$ *(*ただし, $\tilde{r} \in \rho(c, \mathrm{I})$ を満たす $c \in C$ は存在しない*)* を取り除いた代数的アーキテクチャモデル, $\mathcal{R}$ の任意の状態 s に対して $\mathcal{P}_{\mathcal{R}'}^{\mathrm{PSH}}(s)$ を $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ からリソース $\tilde{r}$ に対応するサブプロセス $\Phi(\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s), \tilde{r})$ を取り除き, $\mathcal{R}'$ の実装となるよう $\tilde{r}$ の *update* メソッドの呼び出し, および $\tilde{r}$ から呼び出される *update* メソッドの本体を削除したプロセスとする. これを, $\mathcal{P}_{\mathcal{R}'}^{\mathrm{PSH}}(s) = \mathrm{trim}_{\mathcal{R}'}(\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s) \setminus \Phi(\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s), \tilde{r}))$ と書く. このとき, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ および $\mathcal{P}_{\mathcal{R}'}^{\mathrm{PSH}}(s)$ への任意の入力メッセージ $\langle \widehat{m, c_{\mathrm{I/O}}} \rangle$ について,

$$\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s) \overset{\langle \widehat{m, c_{\mathrm{I/O}}} \rangle}{\Longrightarrow} P$$

かつ

$$\mathcal{P}_{\mathcal{R}'}^{\mathrm{PSH}}(s) \overset{\langle \widehat{m, c_{\mathrm{I/O}}} \rangle}{\Longrightarrow} P'$$

が成り立つならば, P, P' は $P' = \mathrm{trim}_{\mathcal{R}'}(P \setminus \Phi(P, \tilde{r}))$ を満たす.

また, 任意のリソース $r \in R \setminus \{\tilde{r}\}$ について, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ において, $c_{\mathrm{I/O}}$ への m の入力によって制御が r に到達可能であるとき, かつそのときのみ, $\mathcal{P}_{\mathcal{R}'}^{\mathrm{PSH}}(s)$ において, $c_{\mathrm{I/O}}$ への m の入力によって制御が r に到達可能である.

証明. $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ において, チャンネル $c_{\mathrm{I/O}}$ への m の入力によって制御が $\tilde{r}$ に到達可能である場合とそうでない場合に分けて考える.

$\tilde{r}$ に制御が到達可能でない場合, 到達可能性の定義より

$$\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \setminus \Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), \tilde{r}) \stackrel{\widehat{\langle m, c_{\mathrm{I/O}} \rangle}}{\Longrightarrow} P \setminus \Phi(P, \tilde{r})$$

が成り立ち, この関係は $\mathrm{trim}_{\mathcal{R}'}$ によっても保存されるため,

$$\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}'}(s) = \mathrm{trim}_{\mathcal{R}'}(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \setminus \Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), \tilde{r})) \stackrel{\widehat{\langle m, c_{\mathrm{I/O}} \rangle}}{\Longrightarrow} \mathrm{trim}_{\mathcal{R}'}(P \setminus \Phi(P, \tilde{r})) = P'$$

が成り立つ. またこのこと, および $r \in R \setminus \{\tilde{r}\}$ を満たすリソース r が $\mathcal{R}'$ に含まれることから, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ において制御が r に到達可能であるとき, かつそのときのみ, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}'}$ においても制御が r に到達可能である.

いっぽう, $\tilde{r}$ に制御が到達可能である場合, 到達可能性の定義より, 適当なメッセージ μ, 適当なプロセス P'', P''', P'''' が存在して,

$$\Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), \tilde{r}) \xrightarrow{\mu} P'',$$

$$\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \setminus \Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), \tilde{r}) \stackrel{\widehat{\langle m, c_{\mathrm{I/O}} \rangle}}{\Longrightarrow} P''' \xrightarrow{\overline{\mu}} P''''$$

となる. ここで, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ の定義より, $\overline{\mu}$ は $\tilde{r}$ の update メソッドの呼び出しに相当するメッセージであることがわかる. $\mathrm{trim}_{\mathcal{R}'}$ によって, この update メソッドの呼び出しが削除されることから, $\mathrm{trim}_{\mathcal{R}'}(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \setminus \Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), \tilde{r}))$ からの遷移が, $\tilde{r}$ の update メソッドの呼び出しで止まることはない. さらに, $\tilde{r} \in \rho(c, \mathrm{I})$ を満たす $c \in C$ が存在しないため, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s)$ からの遷移では $\tilde{r}$ からの update メソッドの呼び出しは発生せず, したがって, $\tilde{r}$ に対応するサブプロセスを取り除いた $\mathrm{trim}_{\mathcal{R}'}(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \setminus \Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), \tilde{r}))$ からの遷移が, 元の $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s)$ からの遷移よりも短くなることはない. よって,

$$\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}'}(s) = \mathrm{trim}_{\mathcal{R}'}(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \setminus \Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), \tilde{r})) \stackrel{\widehat{\langle m, c_{\mathrm{I/O}} \rangle}}{\Longrightarrow} \mathrm{trim}_{\mathcal{R}'}(P \setminus \Phi(P, \tilde{r})) = P'$$

が成り立ち, $r \in R \setminus \{\tilde{r}\}$ を満たす任意のリソース r について, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ において制御が r に到達可能であるとき, かつそのときのみ, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}'}$ においても制御が r に到達可能であるといえる. □

補題 3. *(アーキテクチャモデルと PUSH 実装の遷移関係の等価性)*

任意の代数的アーキテクチャモデル $\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ が与えられたとき, 任意の状態 s, 入力チャンネル $c_{\mathrm{I/O}}$, メッセージ m に対して, $s \stackrel{\langle m, c_{\mathrm{I/O}} \rangle}{\underset{\mathcal{R}}{\longrightarrow}} s'$ が成り立つとき, かつそのときに限り, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s) \stackrel{\widehat{\langle m, c_{\mathrm{I/O}} \rangle}}{\Longrightarrow} \mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s')$ が成り立つ.

また, $s \stackrel{\langle m, c_{\mathrm{I/O}} \rangle}{\underset{\mathcal{R}}{\longrightarrow}} s'$ が成り立つときのメッセージの割り当てを π とすると, 任意のリソース $r \in R$ について,

1) $r \in \rho(c, \mathrm{O})$ かつ $\pi(c) \neq e_c$ を満たすチャンネル $c \in C$ が存在するなら, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s)$ において, チャンネル $c_{\mathrm{I/O}}$ への m の入力によって制御が r に到達し,

2) そのような c が存在しないなら, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s)$ において, チャンネル $c_{\mathrm{I/O}}$ への m の入力によって制御は r に到達せず, $s(r) = s'(r)$ が成り立つ.

証明. リソースの数 $|R| = n$ に関する帰納法で証明する.
(基底段) リソースが1つ, かつチャンネルが入出力チャンネルのみの場合, すなわち $R = \{r\}, C \subseteq C_{\mathrm{I/O}}$, かつ各 $c \in C$ について, $\rho(c, \mathrm{O}) = \{r\}$ を満たす場合を考える. このとき, 式 (5)-(7), (9) より, $\mathcal{R}$ の PUSH 優先による実装は, $C = \{c_1, \ldots, c_n\}$ としたとき,

$$\begin{aligned}
\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s) &= P_r^{\mathrm{PSH}}(s(r)) \\
&= \eta_{c_1,r}(m).\overline{\eta'_{c_1,r}}.P_r^{\mathrm{PSH}}(\delta_r^{c_1,\mathrm{O}}(s(r), m)) \\
&\quad \vdots \\
&\quad +\eta_{c_n,r}(m).\overline{\eta'_{c_n,r}}.P_r^{\mathrm{PSH}}(\delta_r^{c_n,\mathrm{O}}(s(r), m)) \\
&\quad +\mathrm{get}_r.\overline{\mathrm{get'}_r}(s(r)).P_r^{\mathrm{PSH}}(s(r))
\end{aligned} \tag{14}$$

となる. ここで, $\mathcal{R}$ について $s \xrightarrow[\mathcal{R}]{\langle m, c_i \rangle} s'$ (ただし, $1 \leq i \leq n$) が成り立っている場合, 式 (4) より, $s'(r) = \delta_r^{c_i,\mathrm{O}}(s(r), m)$. いっぽう $\mathcal{R}$ の PUSH 優先による実装 $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ については, 式 (14) より, 適当なプロセス P'_1 が存在し, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s) \xrightarrow{\eta_{c_i,r}(m)} P'_1 \xrightarrow{\overline{\eta'_{c_i,r}}} P_r^{\mathrm{PSH}}(\delta_r^{c_i,\mathrm{O}}(s(r), m)) = \mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s'')$ (ただし, $s''(r) = \delta_r^{c_i,\mathrm{O}}(s(r), m)$)) が成り立つ. ここでリソースは r しか存在しないため, $s'(r) = s''(r)$ より $s' = s''$ が成り立ち, 基底段において題意の前半部分が成り立つ. またこのとき, リソース r について, $r \in \rho(c_i, \mathrm{O})$ および $\pi(c_i) = m \neq e_{c_i}$ が成り立ち, かつ $P_r^{\mathrm{PSH}}(s(r)) \xrightarrow{\eta_{c_i,r}(m)} P'_1$ が成り立つことから, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ において制御が r に到達する. したがって, 基底段において題意の後半部分も成り立つ.
(帰納段) $\mathcal{R}$ に対し, 以下の条件 (図 7 参照) を満たす $\mathcal{R}' = \langle R', C', \rho', T, \tau, \mu, \Sigma, \Delta', \Gamma, s'_0 \rangle$ が存在して, $\mathcal{R}'$ に関して帰納法の仮定が成り立つ.

- 適当な $r \in R$ が存在して, $r \in \rho(c', \mathrm{I})$ を満たす $c' \in C$ は存在せず, $R' = R \setminus \{r\}$ が成り立つ.
- $r \in \rho(c, \mathrm{O})$ を満たす各 $c \in C$ について,
  - $\rho(c, \mathrm{O}) = \{r\}$ ならば, $c \notin C'$.
  - $\rho(c, \mathrm{O}) \supset \{r\}$ ならば, $c \in C'$ かつ $\rho'(c, \mathrm{O}) = \rho(c, \mathrm{O}) \setminus \{r\}$.

まず, 制御の到達可能性について証明を行う. $r \in \rho(c, \mathrm{O})$ かつ $\pi(c) \neq e_c$ を満たす $c \in C$ が存在する場合とそうでない場合に分けて証明を行う.
(i) $r \in \rho(c, \mathrm{O})$ かつ $\pi(c) \neq e_c$ を満たす $c \in C$ が存在するとき:

さらに, $c \in C_{\mathrm{I/O}}$ の場合とそうでない場合に分けて証明を行う.
(i.a) $c \in C_{\mathrm{I/O}}$ のとき:

$\pi(c) \neq e_c, c \in C_{\mathrm{I/O}}$ より, $c = c_{\mathrm{I/O}}, m = \pi(c)$ が成り立つため, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ において, r に対応するサブプロセス P_r が持つ input メソッドは式 (9) より,

$$\Psi_{c_{\mathrm{I/O}}}^{\mathrm{input}}(P_r(s)) = \eta_{c_{\mathrm{I/O}},r}(m).\overline{\eta'_{c_{\mathrm{I/O}},r}}.P_r^{\mathrm{PSH}}(\delta_r^{c,\mathrm{O}}(s(r), m), s(r_1), \ldots, s(r_n))$$

を満たす. よって, 適当なプロセス P' が存在し,

$$P_r^{\mathrm{PSH}}(s(r)) \xrightarrow{\eta_{c_{\mathrm{I/O}},r}(m)} P'. \tag{15}$$

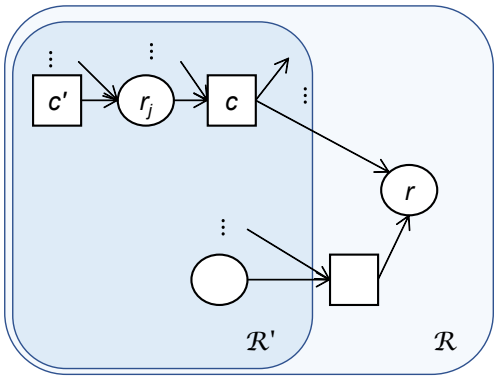


図 7: 補題 3 の証明

$c_{\mathrm{I/O}}$ への m の入力によって, r に制御が到達する.
(i.b) $c \notin C_{\mathrm{I/O}}$ のとき:

$\pi(c) \neq e_c$ および π の定義より, $\mathcal{R}$ において, $r_j \in \rho(c', \mathrm{O})$ かつ $\pi(c') \neq e_{c'}$ を満たすチャンネル $c' \in C$ およびリソース r_j が存在する. ここで, 帰納法の仮定より, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}'}$ において, チャンネル $c_{\mathrm{I/O}}$ への m の入力に対して制御が r_j に到達する. 補題 2 より, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ においても, チャンネル $c_{\mathrm{I/O}}$ への m の入力に対して, 制御が r_j に到達する. いっぽう, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ において, r_j に対応するサブプロセス P_{r_j} は, $r_j \in \rho(c', \mathrm{O})$ を満たす $c', r' \in \rho(c', \mathrm{I})$ を用いて,

$$P_{r_j}(s) = \Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s), r_j) = \begin{cases} \Psi^{\mathrm{get}}(P_r) + \Psi^{\mathrm{update}}_{c',r'}(P_{r_j}) + \cdots & (c' \notin C_{\mathrm{I/O}}) \\ \Psi^{\mathrm{get}}(P_r) + \Psi^{\mathrm{input}}_{c'}(P_{r_j}) + \cdots & (c' \in C_{\mathrm{I/O}}) \end{cases}$$

と表される. このとき, 式 (8), (9) より, 全ての r_j に対応するサブプロセス P_{r_j} は適当なメッセージ μ において,

$$P_{r_j} \xrightarrow{\mu} P'' \xrightarrow{\overline{\eta_{c,r_j,r}}(y,y')} P''' \tag{16}$$

を満たす. 制御が r_j に到達するため式 (16) より, 適当なリソース r'' が存在して

$$\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}} \backslash P_r(s_{\mathcal{R}}) \xrightarrow{\eta_{c_{\mathrm{I/O}},r''}(m)} \mathcal{P}' \overset{\overline{\eta_{c,r_j,r}}(z,z')}{\Longrightarrow} \mathcal{P}'' \tag{17}$$

が成り立つ.

$\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ において, r に対応するサブプロセス P_r が持つ $r_j \in \rho(c, \mathrm{I})$ に対応する update メソッドは式 (8) より,

$$\Psi^{\mathrm{update}}_{c,r_j}(P_r(s)) = \eta_{c,r_j,r}(y, y').\overline{\eta'_{c,r_j,r}}.P^{\mathrm{PSH}}_r(x', s(r_1), \ldots, s(r_{j-1}), y', s(r_{j+1}), \ldots, s(r_n))$$

を満たす. このとき,

$$P^{\mathrm{PSH}}_r(s(r)) \xrightarrow{\eta_{c,r_j,r}(y,y')} P' \tag{18}$$

が成り立つ. したがって, 式 (18), (17) より, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ において, $c_{\mathrm{I/O}}$ への m の入力によって r に制御が到達する.
(ii) $r \in \rho(c, \mathrm{O})$ かつ $\pi(c) \neq e_c$ を満たす $c \in C$ が存在しないとき:

$r \in \rho(c, \mathrm{O})$ を満たすすべての c について, $\pi(c) = e_c$ となり, π の定義より c は $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に含まれない. $c \in C_{\mathrm{I/O}}$ のとき, $c \neq c_{\mathrm{I/O}}$ なので, r に制御が到達しないことは明らか. さらに, $r_j \in \rho(c, \mathrm{I})$ および $r_j \in \rho(c', \mathrm{O})$ を満たすすべての r_j および c' は $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に含まれない. したがって, すべての c' について $\pi(c') = e_{c'}$ となり, 帰納法の仮定より, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}'}$ において制御が到達する r_j が存在しない. さらに補題 2 より, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ においても制御が到達する r_j が存在しないため, $c_{\mathrm{I/O}}$ への m の入力によって r に制御が到達しない.

次に, 遷移関係の等価性について証明する.

$r \in \rho(c, \mathrm{O})$ かつ $\pi(c) \neq e_c$ を満たす c が存在する場合とそうでない場合に分けて証明を行う.
(i) $r \in \rho(c, \mathrm{O})$ かつ $\pi(c) \neq e_c$ を満たす c が存在するとき:

$\mathcal{R}$ と $\mathcal{R}'$ の状態を区別するため $s_{\mathcal{R}} = s$, $s'_{\mathcal{R}} = s'$ とおく. $c \notin C_{\mathrm{I/O}}$ の場合とそうでない場合に分けて証明を行う.
(i.a) $c \notin C_{\mathrm{I/O}}$ のとき:

$\pi(c) \neq e_c$ より, c が $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に含まれるので, 式 (4) より,

$$\delta^{c,O}_r(s_{\mathcal{R}}(r), \pi(c)) = s'_{\mathcal{R}}(r) \tag{19}$$

が成り立ち, $r_j \in \rho(c, \mathrm{I})$, $r_j \in \rho(c', \mathrm{O})$ かつ $\pi(c') \neq e_{c'}$ を満たす r_j が存在し,

$$\delta^{c,I}_{r_j}(s_{\mathcal{R}}(r_j), \pi(c)) = s'_{\mathcal{R}}(r_j) \tag{20}$$

が成り立つ. このとき, $\mathcal{R}'$ における遷移後の r_j の状態を y' とすると,

$$y' = \delta^{c',O}_{r_j}(s_{\mathcal{R}'}(r_j), \pi(c')) = s'_{\mathcal{R}'}(r_j) = s'_{\mathcal{R}}(r_j) \tag{21}$$

を満たす. いっぽう $\pi(c) \neq e_c$ なので, 制御の到達可能性の帰納法の仮定より, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}'}$ において r_j に制御が到達する. さらに補題 2 より, $\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}$ においても r_j に制御が到達する. したがって, $\Phi(\mathcal{P}^{\mathrm{PSH}}_{\mathcal{R}}(s_{\mathcal{R}}), r)$ の update メソッドも呼ばれるため, 呼ばれた後の状態を x' とすると, 式 (8) より, 適当な m_c が存在して

$$x' = \delta^{c,\mathrm{O}}_r(s_{\mathcal{R}}(r), m_c) \tag{22}$$

が成り立ち, さらに遷移関係の等価性の帰納法の仮定より,

$$y' = \delta^{c,I}_{r_j}(s_{\mathcal{R}}(r_j), m_c) \tag{23}$$

が成り立つ. 式 (21), (23) より,

$$s'_{\mathcal{R}}(r_j) = \delta^{c,I}_{r_j}(s_{\mathcal{R}}(r_j), m_c), \tag{24}$$

さらに式 (20), および c におけるメッセージの一意性より,

$$m_c = \pi(c)$$

が成り立ち, 式 (19) より,

$$s'_{\mathcal{R}}(r) = \delta_r^{c,O}(s_{\mathcal{R}}(r), m_c), \tag{25}$$

さらに, 式 (22) より,

$$x' = s'_{\mathcal{R}}(r) \tag{26}$$

となるため, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s_{\mathcal{R}}) \overset{\widehat{\langle m, c_{\mathrm{I/O}} \rangle}}{\Longrightarrow} \mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s'_{\mathcal{R}})$ が成り立つ.
(i.b) $c \in C_{\mathrm{I/O}}$ のとき:

$c = c_{\mathrm{I/O}}$ となり, c が $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に含まれるので, 式 (4) より,

$$\delta_r^{c,O}(s_{\mathcal{R}}(r), m) = s'_{\mathcal{R}}(r) \tag{27}$$

を満たす. このとき, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ において, $\Phi(\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s_{\mathcal{R}}), r)$ の input メソッドが呼ばれるため, 呼ばれた後の r の状態を x' とすると, 式 (9) より

$$x' = \delta_r^{c,\mathrm{O}}(s_{\mathcal{R}}(r), m). \tag{28}$$

式 (27), (28) より, $x' = s'_{\mathcal{R}}(r)$ を満たす.
(ii) $r \in \rho(c, \mathrm{O})$ かつ $\pi(c) \neq e_c$ を満たす c が存在しないとき:

π の定義より, $\mathcal{R}$ において $s(r) = s'(r)$ が成り立つ. また, c が $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に属していないことから, $c \in C_{\mathrm{I/O}}$ のとき $c \neq c_{\mathrm{I/O}}$ となるため, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ において r に制御が到達しない. いっぽう $c \notin C_{\mathrm{I/O}}$ のとき, $r_j \in \rho(c, \mathrm{I})$ および $r_j \in \rho(c', \mathrm{O})$ を満たすすべてのリソース r_j, チャンネル c' も $G_{\mathcal{R},c_{\mathrm{I/O}}}$ に属さず, よって $\pi(c') = e'_c$ および帰納法の仮定より, $\mathcal{P}_{\mathcal{R}'}^{\mathrm{PSH}}$ においてどの r_j にも制御が到達せず, さらに補題 2 より, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ においてもどの r_j にも制御が到達しない. したがって, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ において, r にも制御が到達せず, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}$ の状態 s から s'' への遷移においても

$$s(r) = s'(r) = s''(r) \tag{29}$$

が成り立つ.
(i), (ii) より, 任意の状態 s, 入力チャンネル $c_{\mathrm{I/O}}$, メッセージ m に対して, $s \xrightarrow[\mathcal{R}]{\langle m, c_{\mathrm{I/O}} \rangle} s'$ が成り立つとき, かつそのときに限り, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s) \overset{\widehat{\langle m, c_{\mathrm{I/O}} \rangle}}{\Longrightarrow} \mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s')$ が成り立つ. □

定理 2. *(アーキテクチャモデルと PUSH 優先実装の等価性)*
任意の代数的アーキテクチャモデル $\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ が与えられたとき, $\mathcal{R}$ への任意の入力列 σ に対して, $s_0 \overset{\sigma}{\underset{\mathcal{R}}{\Longrightarrow}} s$ が成り立つとき, かつそのときに限り, $\mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s_0) \overset{\widehat{\sigma}}{\Longrightarrow} \mathcal{P}_{\mathcal{R}}^{\mathrm{PSH}}(s)$ が成り立つ.

証明. 補題 1, 3 より明らか. □

6.2 任意の CCS 実装の等価性

補題 4. *(任意の実装における共通部分の等価性)*
$\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ を任意の代数的アーキテクチャモデルとし, $\mathcal{P}_{\mathcal{R}}$ を条件 *1, 2* を満たす

$\mathcal{R}$ の任意の実装とする. また, $\mathcal{R}'$ を $\mathcal{R}$ からリソース $\tilde{r}$ *(*ただし, $\tilde{r} \in \rho(c, \mathrm{I})$ を満たす $c \in C$ は存在しない*)* を取り除いた代数的アーキテクチャモデル, $\mathcal{P}_{\mathcal{R}'} = \mathrm{trim}_{\mathcal{R}'}(\mathcal{P}_{\mathcal{R}} \setminus \Phi(\mathcal{P}_{\mathcal{R}}, \tilde{r}))$ を $\mathcal{P}_{\mathcal{R}}$ から $\Phi(\mathcal{P}_{\mathcal{R}}, \tilde{r})$ を取り除き, $\mathcal{R}'$ の実装となるよう $\tilde{r}$ の *update* メソッドの呼び出し, および $\tilde{r}$ から呼び出される *update* メソッドの本体を削除したプロセスとする. このとき, $\mathcal{P}_{\mathcal{R}}$ および $\mathcal{P}_{\mathcal{R}'}$ への任意の入力メッセージ列 σ について, $\mathcal{P}_{\mathcal{R}} \stackrel{\sigma}{\Longrightarrow} P_\sigma$, $\mathcal{P}_{\mathcal{R}'} \stackrel{\sigma}{\Longrightarrow} P'_\sigma$ が成り立つならば, P_σ, P'_σ は $P'_\sigma = \mathrm{trim}_{\mathcal{R}'}(P_\sigma \setminus \Phi(P_\sigma, \tilde{r}))$ を満たす.

証明. σ の長さ n に関する帰納法で証明する.
(基底段) $n = 0$ のとき, $\mathcal{P}_{\mathcal{R}'}$ の定義より明らか.
(帰納段) $n > 0$ の場合を考え, $\sigma = \sigma' \langle m, c_{\mathrm{I/O}} \rangle$ とおく. $\mathcal{P}_{\mathcal{R}}$ において, $c_{\mathrm{I/O}}$ への m の入力によって, 制御が $\tilde{r}$ に到達する場合と到達しない場合に分け, 補題 2 と同様に証明する. □

補題 5. *(*任意の実装間の等価性のリソース追加に対する不変性*)*
$\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ を任意の代数的アーキテクチャモデルとし, $\mathcal{R}'$ を $\mathcal{R}$ からリソース $\tilde{r}$ *(*ただし, $\tilde{r} \in \rho(c, \mathrm{I})$ を満たす $c \in C$ は存在しない*)* を取り除いた代数的アーキテクチャモデル, $R' = R \setminus \{\tilde{r}\}$ を $\mathcal{R}'$ のリソース集合とする. このとき, 任意の入力メッセージ列 σ, および任意のリソース $r' \in R'$ について, 条件 *1, 2* を満たす $\mathcal{R}'$ の任意の実装 $\mathcal{P}_{\mathcal{R}'}$ と $\mathcal{P}'_{\mathcal{R}'}$ の両方に σ と $\mathrm{get}_{r'}$ を続けて入力した後, $\mathrm{get}'_{r'}$ によってそれぞれから返される値が一致するならば, すなわち,

$$\begin{aligned} \forall \sigma. \forall_{r' \in R'}. \quad & \{s \mid \exists_{P_\sigma, P_{\sigma,r'}}. \mathcal{P}_{\mathcal{R}'}(s_0) \stackrel{\sigma}{\Rightarrow} P_\sigma \stackrel{\mathrm{get}_{r'}\ \overline{\mathrm{get}'_{r'}}(s)}{\Longrightarrow} P_\sigma\} \\ & = \{s' \mid \exists_{P'_\sigma, P'_{\sigma,r'}}. \mathcal{P}'_{\mathcal{R}'}(s_0) \stackrel{\sigma}{\Rightarrow} P'_\sigma \stackrel{\mathrm{get}_{r'}\ \overline{\mathrm{get}'_{r'}}(s')}{\Longrightarrow} P'_\sigma\} \end{aligned}$$

が成り立つならば, $\mathcal{R}$ においても, 任意の入力メッセージ σ, および任意のリソース $r \in R$ について, 条件 *1, 2* を満たす $\mathcal{R}$ の任意の実装 $\mathcal{P}_{\mathcal{R}}$ と $\mathcal{P}'_{\mathcal{R}}$ の両方に σ と get_r を続けて入力した後, get'_r によってそれぞれから返される値が一致する, すなわち,

$$\begin{aligned} \forall \sigma. \forall_{r \in R}. \quad & \{s \mid \exists_{P_\sigma, P_{\sigma,r}}. \mathcal{P}_{\mathcal{R}}(s_0) \stackrel{\sigma}{\Rightarrow} P_\sigma \stackrel{\mathrm{get}_r\ \overline{\mathrm{get}'_r}(s)}{\Longrightarrow} P_\sigma\} \\ & = \{s' \mid \exists_{P'_\sigma, P'_{\sigma,r}}. \mathcal{P}'_{\mathcal{R}}(s_0) \stackrel{\sigma}{\Rightarrow} P'_\sigma \stackrel{\mathrm{get}_r\ \overline{\mathrm{get}'_r}(s')}{\Longrightarrow} P'_\sigma\} \end{aligned}$$

が成り立つ.

証明. 補題の条件部が成り立っているという仮定の下で補題の結論部が成り立っていることを, 補題の結論部における σ の長さ n に関する帰納法で証明する.
(基底段) $n = 0$ のとき, 補題 4 が成り立っていることから, $\tilde{r}$ についてのみ, $\mathcal{P}_{\mathcal{R}}(s_0) \stackrel{\mathrm{get}_{\tilde{r}}\ \overline{\mathrm{get}'_{\tilde{r}}}(s_{\tilde{r}}^{\mathcal{P},\epsilon})}{\Longrightarrow} \mathcal{P}_{\mathcal{R}}(s_0)$ および $\mathcal{P}'_{\mathcal{R}}(s_0) \stackrel{\mathrm{get}_{\tilde{r}}\ \overline{\mathrm{get}'_{\tilde{r}}}(s_{\tilde{r}}^{\mathcal{P}',\epsilon})}{\Longrightarrow} \mathcal{P}'_{\mathcal{R}}(s_0)$ が成り立つときに, $s_{\tilde{r}}^{\mathcal{P},\epsilon} = s_{\tilde{r}}^{\mathcal{P}',\epsilon}$ となることを示せば十分である.

まず, $\tilde{r} \in \rho(c, \mathrm{I})$ を満たすチャンネル $c \in C$ を考え, c の入力側リソースを $\rho(c, \mathrm{I}) = \{r_1, \ldots, r_l\}$ とおく. このとき $\mathcal{P}_{\mathcal{R}}$ において, すべての r_j $(1 \leq j \leq l)$ について r_j と c の間のデータ転送が PUSH 型であった場合, 式 (7) より,

$$s_{\tilde{r}}^{\mathcal{P},\epsilon} = s_0(\tilde{r}) \tag{30}$$

となる. また, $\mathcal{P}'_{\mathcal{R}}$ において, ある r_j $(1 \leq j \leq l)$ について r_j と c の間のデータ転送が PULL 型であった場合, 式 (10) より,

$$s_{\tilde{r}}^{\mathcal{P}',\epsilon} = f_{\tilde{r}}(s_0(r_1), \ldots, s_0(r_l)) \tag{31}$$

となる. したがって式 (30), (31) および, 条件 2 の式 (12) より $s_{\tilde{r}}^{\mathcal{P},\epsilon} = s_{\tilde{r}}^{\mathcal{P}',\epsilon}$ が成り立つ.

いっぽう, $\mathcal{P}_{\mathcal{R}}$ においても, ある r_i $(1 \leq i \leq l)$ について r_i と c の間のデータ転送が PULL 型であった場合, 式 (10) および $\tilde{r}$ 内部のキャッシュの初期条件より, $s_{\tilde{r}}^{\mathcal{P},\epsilon} = f_{\tilde{r}}(s_0(r_1), \ldots, s_0(r_l)) = s_{\tilde{r}}^{\mathcal{P}',\epsilon}$ が成り立つ.

(帰納段) $n > 0$ のときを考え, $\sigma = \sigma' \langle m, c_{\mathrm{I/O}} \rangle$ とおく. このとき, σ' の長さは $n-1$ となる. 補題 4 が成り立っているため, ここでは, リソース $\tilde{r}$ についてのみこの補題の結論部が成り立っていることを示せば十分である. 以下では, 実装 $\mathcal{P}_{\mathcal{R}}$ および $\mathcal{P}'_{\mathcal{R}}$ に σ と get_r を続けて入力した後, get'_r によって返される値をそれぞれ $s_{\tilde{r}}^{\mathcal{P},\sigma}$, $s_{\tilde{r}}^{\mathcal{P}',\sigma}$ とおき, $s_{\tilde{r}}^{\mathcal{P},\sigma} = s_{\tilde{r}}^{\mathcal{P}',\sigma}$ であることを示す.

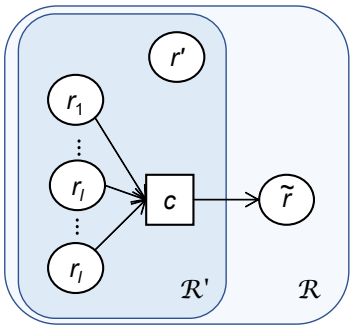


図 8: 補題 5 の証明

まず, $\tilde{r} \in \rho(c, \mathrm{O})$ を満たす $c \in C$ がただ 1 つ存在する場合を考え, c の入力側リソースを $\rho(c, \mathrm{I}) = \{r_1, \ldots, r_l\}$ とおいたとき (図 8 参照),

1) $\mathcal{P}_{\mathcal{R}}$ と $\mathcal{P}'_{\mathcal{R}}$ のいずれにおいても, すべての r_j $(1 \leq j \leq l)$ について r_j と c の間のデータ転送が PUSH 型であった場合,
2) $\mathcal{P}_{\mathcal{R}}$ のみすべての r_j $(1 \leq j \leq l)$ について r_j と c の間のデータ転送が PUSH 型であった場合,
3) $\mathcal{P}_{\mathcal{R}}$ と $\mathcal{P}'_{\mathcal{R}}$ のいずれにおいても, 少なくとも 1 つの r_j $(1 \leq j \leq l)$ について r_j と c の間のデータ転送が PULL 型であった場合,

に分けて証明する. なお, $\tilde{r} \in \rho(c, \mathrm{O})$ を満たす c が複数存在する場合は, 条件 1 の 2) より, いずれの c においてもすべての r_j $(1 \leq j \leq l)$ について r_j と c の間のデータ転送が PUSH 型でなければならないため, 上記条件の 1) に相当する.

もっとも典型的な 2) の場合について最初に考える. この場合, $\mathcal{P}_{\mathcal{R}}$ のサブプロセス $\Phi(\mathcal{P}_{\mathcal{R}}, \tilde{r})$ は状態を保持する. このとき, 式 (7) より, $\mathcal{P}_{\mathcal{R}}$ に σ' を入力した直後の $\Phi(\mathcal{P}_{\mathcal{R}}, \tilde{r})$ の状態は $s_{\tilde{r}}^{\mathcal{P},\sigma'}$ となり, 帰納法の仮定より

$$s_{\tilde{r}}^{\mathcal{P},\sigma'} = s_{\tilde{r}}^{\mathcal{P}',\sigma'} \tag{32}$$

となる. また $\mathcal{P}'_{\mathcal{R}}$ については, ある r_j $(1 \leq j \leq l)$ について, r_j と c の間のデータ転送が PULL 型となるため, 式 (7) より, $\mathcal{P}'_{\mathcal{R}}$ に σ' を入力した直後の $\Phi(\mathcal{P}'_{\mathcal{R}}, r_j)$ $(1 \leq j \leq l)$ の状態は $s_{r_j}^{\mathcal{P}',\sigma'}$ となり, $r_j \in R'$ であることから, この補題の条件部が σ', r_j に対して成り立ち, 補題 4 より

$$s_{r_j}^{\mathcal{P}',\sigma'} = s_{r_j}^{\mathcal{P},\sigma'}, \tag{33}$$

さらに式 (10) より

$$f_{\tilde{r}}(s_{r_1}^{\mathcal{P}',\sigma'},\ldots,s_{r_l}^{\mathcal{P}',\sigma'}) = s_{\tilde{r}}^{\mathcal{P}',\sigma'} \tag{34}$$

となる. 次に $\mathcal{P}_{\mathcal{R}}$ に σ を入力した直後について考える. このとき, $\Phi(\mathcal{P}_{\mathcal{R}},\tilde{r})$ の getter メソッドの戻り値 $s_{\tilde{r}}^{\mathcal{P},\sigma}$ は, チャンネル c 上の適当なメッセージ m に対して, 式 (7), (8) より,

$$s_{\tilde{r}}^{\mathcal{P},\sigma} = \delta_{\tilde{r}}^{c,\mathrm{O}}(s_{\tilde{r}}^{\mathcal{P},\sigma'},m) \tag{35}$$

となり, $\Phi(\mathcal{P}_{\mathcal{R}},r_j)$ の getter メソッドの戻り値は,

$$s_{r_j}^{\mathcal{P},\sigma} = \delta_{r_j}^{c,\mathrm{I}}(s_{r_j}^{\mathcal{P},\sigma'},m) \tag{36}$$

となる. さらに, $\mathcal{P}'_{\mathcal{R}}$ に σ を入力した直後について考えると, $\Phi(\mathcal{P}'_{\mathcal{R}},\tilde{r})$ の getter メソッドの戻り値 $s_{\tilde{r}}^{\mathcal{P}',\sigma}$ は, 式 (10) より

$$s_{\tilde{r}}^{\mathcal{P}',\sigma} = f_{\tilde{r}}(s_{r_1}^{\mathcal{P}',\sigma},\ldots,s_{r_l}^{\mathcal{P}',\sigma}) \tag{37}$$

となり, $r_j \in R'$ であることから, この補題の条件部が σ, r_j に対して成り立ち, さらに補題 4 より

$$s_{r_j}^{\mathcal{P}',\sigma} = s_{r_j}^{\mathcal{P},\sigma} \tag{38}$$

となる. ここで, 式 (33), (36)-(38) より

$$s_{\tilde{r}}^{\mathcal{P}',\sigma} = f_{\tilde{r}}(\delta_{r_1}^{c,\mathrm{I}}(s_{r_1}^{\mathcal{P}',\sigma'},m),\ldots,\delta_{r_l}^{c,\mathrm{I}}(s_{r_l}^{\mathcal{P}',\sigma'},m)) \tag{39}$$

を得ることができ, さらに式 (32), (34), (35) より

$$s_{\tilde{r}}^{\mathcal{P},\sigma} = \delta_{\tilde{r}}^{c,\mathrm{O}}(f_{\tilde{r}}(s_{r_1}^{\mathcal{P}',\sigma'},\ldots,s_{r_l}^{\mathcal{P}',\sigma'}),m) \tag{40}$$

となるため, 式 (39), (40), および条件 2 の式 (13) より, $s_{\tilde{r}}^{\mathcal{P},\sigma} = s_{\tilde{r}}^{\mathcal{P}',\sigma}$ が成り立つ.

1) の場合, $\mathcal{P}_{\mathcal{R}}$ のサブプロセス $\Phi(\mathcal{P}_{\mathcal{R}},\tilde{r})$ および $\mathcal{P}'_{\mathcal{R}}$ のサブプロセス $\Phi(\mathcal{P}'_{\mathcal{R}},\tilde{r})$ はともに状態を保持する. 1) の場合と同様, 帰納法の仮定より式 (32) が, この補題の条件部および補題 4 より式 (33), (38) が成り立つ. また, $\mathcal{P}_{\mathcal{R}}$ に σ を入力した直後については, 式 (35), (36) が成り立ち, さらに $\mathcal{P}'_{\mathcal{R}}$ に σ を入力した直後については, チャンネル c 上の適当なメッセージ m' に対して,

$$s_{\tilde{r}}^{\mathcal{P}',\sigma} = \delta_{\tilde{r}}^{c,\mathrm{O}}(s_{\tilde{r}}^{\mathcal{P}',\sigma'},m'), \tag{41}$$

$$s_{r_j}^{\mathcal{P}',\sigma} = \delta_{r_j}^{c,\mathrm{I}}(s_{r_j}^{\mathcal{P}',\sigma'},m') \tag{42}$$

が成り立つため, 式 (33), (36), (38), (42) より, $m = m'$ が成り立つ. したがって, 式 (32), (35), (41) より, $s_{\tilde{r}}^{\mathcal{P},\sigma} = s_{\tilde{r}}^{\mathcal{P}',\sigma}$ が成り立つ.

3) の場合について考える. このとき, $\mathcal{P}_{\mathcal{R}}$ に σ を入力した直後の各サブプロセス $\Phi(\mathcal{P}_{\mathcal{R}},r_i)$ $(1 \le i \le l)$ の getter メソッドの戻り値, および $\Phi(\mathcal{P}_{\mathcal{R}},r_i)$ が状態を保持する場合その状態は, 式 (7) より, ともに $s_{r_i}^{\mathcal{P},\sigma}$ となる. 同様に, $\mathcal{P}'_{\mathcal{R}}$ に σ を入力した直後の各サブプロセス $\Phi(\mathcal{P}'_{\mathcal{R}},r_i)$ $(1 \le i \le l)$ の getter メソッドの戻り値, およびその内部状態 (存在する場合) は, 式 (7) より, ともに $s_{r_i}^{\mathcal{P}',\sigma}$ となる. また, 補題 4 およびこの補題の条件部より,

$$s_{r_i}^{\mathcal{P},\sigma} = s_{r_i}^{\mathcal{P}',\sigma} \tag{43}$$

が成り立つことがわかる. したがって, $\mathcal{P}_\mathcal{R}$ および $\mathcal{P}'_\mathcal{R}$ に σ を入力した直後の $\Phi(\mathcal{P}_\mathcal{R}, \tilde{r})$ および $\Phi(\mathcal{P}'_\mathcal{R}, \tilde{r})$ の getter メソッドの戻り値は, 式 (10), (43) より

$$s_{\tilde{r}}^{\mathcal{P},\sigma} = f_{\tilde{r}}(s_{r_1}^{\mathcal{P},\sigma}, \ldots, s_{r_l}^{\mathcal{P},\sigma}) = f_{\tilde{r}}(s_{r_1}^{\mathcal{P}',\sigma}, \ldots, s_{r_l}^{\mathcal{P}',\sigma}) = s_{\tilde{r}}^{\mathcal{P}',\sigma} \tag{44}$$

となる. □

定理 3. *(PUSH 優先実装と任意の実装の等価性)*

任意の代数的アーキテクチャモデル $\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ が与えられたとき, $\mathcal{P}_\mathcal{R}$ を条件 *1, 2* を満たす $\mathcal{R}$ の任意の実装とする. このとき, 任意の入力メッセージ列 σ および任意のリソース $r \in R$ について, $\mathcal{P}_\mathcal{R}^{\mathrm{PSH}}$ と $\mathcal{P}_\mathcal{R}$ に σ と get_r を続けて入力した後, $\mathrm{get'}_r$ によってそれぞれから返される値が一致する. すなわち,

$$\begin{aligned}\forall_\sigma.\forall_{r\in R}. \quad & \{s \mid \exists_{P_\sigma, P_{\sigma,r}}.\mathcal{P}_\mathcal{R}^{\mathrm{PSH}}(s_0) \overset{\sigma}{\Rightarrow} P_\sigma \overset{\mathrm{get}_r\,\overline{\mathrm{get'}_r}(s)}{\Longrightarrow} P_\sigma\} \\ & = \{s' \mid \exists_{P'_\sigma, P'_{\sigma,r}}.\mathcal{P}_\mathcal{R}(s_0) \overset{\sigma}{\Rightarrow} P'_\sigma \overset{\mathrm{get}_r\,\overline{\mathrm{get'}_r}(s')}{\Longrightarrow} P'_\sigma\}\end{aligned}$$

が成り立つ.

証明. リソースの数 $|R| = n$ に関する帰納法で証明する.
(基底段) リソースが 1 つ, かつチャンネルが入出力チャンネル 1 つのみの場合, すなわち $C = \{c_{\mathrm{I/O}}\}$, $R = \{r\}$, $\rho(c_{\mathrm{I/O}}, \mathrm{O}) = \{r\}$ の場合を考える. このとき, データ転送は存在せず, 常に $\mathcal{P}_\mathcal{R} = \mathcal{P}_\mathcal{R}^{\mathrm{PSH}}$ となるため明らか.
(帰納段) $n > 0$ のとき, $\mathcal{R}$ のデータフローグラフ $G_\mathcal{R}$ は強連結成分を持たないため, $\tilde{r} \in \rho(c, \mathrm{I})$ を満たす $c \in C$ が存在しないようなリソース $\tilde{r} \in R$ が少なくとも 1 つ存在する. $\mathcal{R}'$ を $\mathcal{R}$ からリソース $\tilde{r}$ を取り除いた代数的アーキテクチャモデルとしたとき, $\mathcal{R}'$ のリソースの数は $|R \setminus \{\tilde{r}\}| = n - 1$ となるため, 帰納法の仮定より $\mathcal{R}'$ について題意が成り立つ. さらに補題 5 より, $\mathcal{R}$ についても題意が成り立つ. □

定理 4. *(アーキテクチャモデルと任意の CCS 実装の等価性)*

任意の代数的アーキテクチャモデル $\mathcal{R} = \langle R, C, \rho, T, \tau, \mu, \Sigma, \Delta, \Gamma, s_0 \rangle$ が与えられたとき, $\mathcal{P}_\mathcal{R}$ を条件 *1, 2* を満たす $\mathcal{R}$ の任意の実装とする. このとき, $\mathcal{R}$ への任意の入力列 σ について, $s_0 \overset{\sigma}{\underset{\mathcal{R}}{\Longrightarrow}} s$ が成り立つとき, かつそのときに限り, 適当なプロセス P が存在し $\mathcal{P}_\mathcal{R}(s_0) \overset{\widehat{\sigma}}{\Rightarrow} P$ が成り立ち, 任意のリソース $r \in R$ について, $P \overset{\mathrm{get}_r\,\overline{\mathrm{get'}_r}(s(r))}{\Longrightarrow} P$ が成り立つ.

証明. 定理 2, 3 より明らか. □

7 データ転送リファクタリング構築に向けての考察

6 章で証明したように, 任意の代数的アーキテクチャモデル $\mathcal{R}$ から生成した PUSH 優先実装 $\mathcal{P}_\mathcal{R}^{\mathrm{PSH}}$ は, 常に $\mathcal{R}$ と等価である. しかしながら, $\mathcal{R}$ の任意の CCS 実装 $\mathcal{P}_\mathcal{R}$ が $\mathcal{R}$ と等価になるためには, $\mathcal{R}$ と $\mathcal{P}_\mathcal{R}$ が一定の条件を満たしている必要がある. 5 章で示した条件 1 および条件 2 は, 等価になるための

1 つの十分条件である. これらのうち, 条件 1 の 2) および条件 2 は, 個々のデータ転送を PUSH 型から PULL 型に変更する際の, $\mathcal{R}$ そのものに対する条件である. これらのことを考慮に入れると, データ転送方式の変更を行うアーキテクチャレベルのリファクタリングと, リファクタリングの結果に基づくソースコードの自動生成を以下のように構成することができる.

1) $\mathcal{R}$ から $G_{\mathcal{R}}$ を構成し, $\mathcal{R}$ が妥当であるか否かの判定を行う.

2) $\mathcal{R}$ 中の各データ転送が PULL 型に変更可能であるか否かを, 条件 1 の 2) および条件 2 に基づいて判定する. 具体的には, 各データ転送が条件 1 の 2) の結論部および条件 2 の結論部をともに満たしている場合のみ, PULL 型に変更可能であると判定する.

3) PULL 型に変更可能な各データ転送に関して, PULL 型に変更するか否かをユーザに問い合わせる. ただし, PULL 型に変更していく上で, 条件 1 の 1) が常に満たされていることを保証し, 満たされなくなるような変更についてはユーザに問い合わせない.

4) ユーザが決定したデータ転送方式にしたがって, 5 章で示した CCS の実装方式に基づき, ソースコードを自動生成する.

上記手続きの中で, 条件 1 の 1) および 2) の判定アルゴリズムの構成は比較的容易である. しかしながら, 条件 2 の結論部の判定は準同型写像の有無を調べる必要があるため, アルゴリズムの構成が困難であると考えられる. そこで, 判定が容易でかつより強い条件として出力側リソースの遷移関数の**右単項性**を考える. 具体的には, リソース $r' \in R$ およびチャンネル c が $r' \in \rho(c, \mathrm{O})$ を満たすとき, r' の遷移関数 $\delta_{r'}^{c,\mathrm{O}}$ が右単項であるとは,

$$\delta_{r'}^{c,\mathrm{O}}(x, m) = \delta_{r'}^{c,\mathrm{O}}(y, m) \tag{45}$$

が成り立つことをいう. このとき, $\rho(c, \mathrm{I}) = \{r_1, \ldots, r_l\}$ とおき, δ^{-1} を

$$m = \delta^{-1}(\delta_{r_1}^{c,\mathrm{I}}(s_1, m), \ldots, \delta_{r_l}^{c,\mathrm{I}}(s_l, m))$$

を満たす関数とし, 関数 $f_{r'}$ を z を適当な定数として,

$$f_{r'}(s'_1, \ldots, s'_l) = \delta_{r'}^{c,\mathrm{O}}(z, \delta^{-1}(s'_1, \ldots, s'_l))$$

と定義すると, $f_{r'}$ は準同型性, すなわち式 (13) を満たす. また, 式 (45) から以下の式を導くことができることから, 右単項性が冪等性の十分条件になることもわかる.

$$\delta_{r'}^{c,\mathrm{O}}(\delta_{r'}^{c,\mathrm{O}}(y, m), m) = \delta_{r'}^{c,\mathrm{O}}(y, m).$$

8 おわりに

本論文では, 本研究室が提案しているリソースに基づく代数的アーキテクチャモデルが, プロセス代数 CCS よりも抽象度の高い記述法であることを示し, 与えられた代数的アーキテクチャモデルから, データ転送方式を選択することによって, CCS による実装を自動で生成できることを示した. また生

成された CCS 実装が, 元の代数的アーキテクチャモデルと常に等価であることを証明した. また, これらの結果から導き出されるアーキテクチャレベルリファクタリングの手順およびアルゴリズムの一部を示した. 今後, このアーキテクチャモデルおよびリファクタリングが, アーキテクチャ設計のための 1 つの理論的な基盤となっていくことが期待される.

謝辞

この研究の一部は, 私立大学等経常費補助金 特別補助「大学間連携等による共同研究」による.

参考文献

[1] Liliana Dobrica and Eila Niemela, “A survey on software architecture analysis methods,” *IEEE Transaction on Software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.

[2] Yang Zhao, “A model of computation with push and pull processing,” *Technical Report*, EECS Department, University of California, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2003/ERL-03-51.pdf 2003.

[3] Maciej Besta, Michał Podstawski, Linus Groner, Edgar Solomonik and Torsten Hoefler, “To push or to pull: On reducing communication and synchronization in graph computations,” in *Proc. 26th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 93–104, 2017.

[4] Jorge Enrique Pŕez-Martínez and Almudena Sierra-Alonso, “UML 1.4 versus UML 2.0 as languages to describe software architectures,” in *Proc. Software Architecture, European Workshop on Software Architecture (EWSA 2004)*, pp. 88–102, 2004.

[5] Robert Allen and David Garlan, “A formal basis for architectural connection,” *Transactions on Software Engineering and Methodology*, vol. 6, no.3, pp. 213–249, 1997.

[6] Jeff Magee, Jeff Kramer and Dimitra Giannakopoulou, “Behaviour analysis of software architectures,” in *Proc. Working IEEE/International Federation for Information Processing Conference Software Architecture*, pp. 35–49, 1999.

[7] David Garlan, Robert T. Monroe and David Wile, “Acme: Architectural description of component-based systems,” in: Gary T. Leavens and Murali Sitaraman (Eds.), *Foundations of Component-Based Systems*, pp. 47–67. Cambridge University Press, 2000.

[8] Peter H. Feiler, David P. Gluch and John J. Hudak, “The architecture analysis & design language (AADL): An introduction,” *Technical Report*, CMU/SEI-2006-TN-011, `http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn011.pdf`, 2006.

[9] Patrizio Pelliccione, Paola Inverardi and Henry Muccini, “CHARMY: A framework for designing and verifying architectural specifications,” *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 324–345, 2009.

[10] Mary Shaw and David Garlan, “Formulations and formalisms in software architecture,” in: Jan van Leeuwen (Ed.), *Computer Science Today: Recent Trends and Developments*, pp. 307–323. Springer, 1995.

[11] Roi T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures, Ph. D. Thesis*. University of California, 2000.

[12] Uri Klein and Kedar S. Namjoshi, “Formalization and automated verification of RESTful behavior,” in *Proc. 23rd international conference on Computer Aided Verification (CAV’11)*, pp. 541–556, 2011.

[13] Naoya Nitta, “A formal approach for guiding architecture design with data constraints,” in *Proc. IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS’16)*, pp. 1–6, 2016.

[14] 稲垣康善, 坂部俊樹, “多ソート代数と等式論理–抽象データタイプの代数的仕様記述法の基礎-1-,” 情報処理学会論文誌, vol. 25, no. 1, pp. 47–53, 1984.

[15] Robin Milner, *Communication and Concurrency*. Prentice Hall, 1989.