Aπ 計算の Coq による形式化

安武 祥平

アクターモデルは並行計算のモデルのひとつであり，アクターと呼ばれる計算主体がそれぞれ並列に動作し互いに非同期にメッセージを送りあって計算を進める．アクターモデルは，アクターの名前は一意であるという性質，他のアクターの名前あるいはメッセージとして送られてきた名前ではアクターを作れないという性質，アクターにはいつでもメッセージを送ることができるという性質，の 3 つの性質を持っている．これらの性質を満たすように π 計算に型を付け，アクターモデルとしての振る舞いを強制させたものに Aπ 計算があるが，Aπ 計算が確かにアクターモデルとしての振る舞いを示すかどうかについて形式的な証明はまだ与えられていない．そこで，本研究では，Aπ 計算およびその型システムの定義を，等価だが証明を進めやすい定義に変更するなどの工夫を行い，Aπ 計算の型システムがアクターモデルとしての振る舞いを強制することについて，定理証明支援系 Coq を用いて形式的な証明を与えた．

英語アブスト

1 はじめに

π 計算やアクターモデルは並行計算のモデルとして発展してきた．アクターモデルはアクターと呼ばれる計算主体 (computing entity) が並列に動作し，互いに非同期にメッセージを送り合うことによって計算を行う．アクターモデルの持つべき性質として，アクターの名前は一意であるという性質，メッセージとして送られてきた名前でアクターを作ることはできないという性質，アクターにはいつでもメッセージを送ることができるという性質，を持っている．

このようなアクターモデルの性質を満たすように設計された，π 計算を用いた形式化として，Aπ 計算が提案されている [2] ．Aπ 計算は，π 計算に型を付けることによってアクターモデルの性質を表現することを目的としており，May Testing と呼ばれる，二つのアクターの等価性についての理論に使われている．

このような基礎的な理論には形式的な証明があることが望ましい．しかし，Aπ 計算の性質に対して形式的な証明を与えているものはまだない．

1.1 目的

本研究では，Aπ 計算の型付けにおける健全性に対して，形式的な証明を与えることを目的とする．形式的な証明を与えるためのツールとして定理証明支援系 Coq を用いる．また，Coq で証明する際に，Aπ 計算およびその型システムについての定義をより具体的なものに定義しなおすことによって，より簡単に証明を行えるようにしている．

2 背景知識

2.1 アクターモデル

アクターモデルは非同期メッセージ通信に基づいた並行計算のモデルである [3]．アクターと呼ばれる計算主体があり，それらは名前 (またはアドレス) と内部状態をもっている．各アクターは並列に動作し，非同期にメッセージを送り合うことでコミュニケーションをとる．

アクターは，メッセージの内容に応じて一定の動作を行う．これを振る舞い (behavior) という．振る舞

A formalization of Aπ-calculus using Coq
Shohei Yasutake, 東京工業大学大学院情報理工学研究科計算工学専攻, Dept. of Computer Science, Tokyo Institute of Technology.

いには以下の様なものがある．

- 他のアクターにメッセージを送信する．
- 新しいアクターを作る．
- 自らの振る舞いを変える．

2.1.1 配置

アクターモデルにおける配置とは，アクターモデルの世界におけるその時点での状態を切り取ったものを表すための概念である．配置はアクターとそのアクターのもつ未実行の仕事から成り立っている．

配置内の，配置外からのメッセージを受け取ることのできるアクターを，窓口 (receptionist) という．アクターは，送り先の名前 (アドレス) を知ることで，その名前のアクターにメッセージを送れるようになる，よって窓口は外部のアクターに自身の名前を知られているアクターと言い換えることができる．また，窓口の集合のことを窓口集合 (receptionist set) という．

2.1.2 アクターモデルの性質

アクターモデルが満たすべき性質として，以下がある．

一意性 (uniqueness property)

アクターの名前は一意である．

新鮮性 (freshness property)

アクターが作られるとき，作られるアクターの名前はまだどのアクターの名前でもない．アクターは，メッセージとして送られてきた名前でアクターを作ることはできない．

永続性 (persistence property)

アクターは消えない．一旦アクターが作られると，いつでもそのアクターにメッセージを送ることができる．

2.2 π 計算

π 計算は，Milner らによって提唱された並行計算のモデルである [5]．プロセスと呼ばれるオブジェクト同士が通信しデータを受け渡していくことで計算を進める．また，アクターモデルとは異なり，メッセージを送受信する際にはプロセス間で同期をとる必要がある．

π 計算の抽象構文は 図 1 のようになっている．ここで，$P, P_1, P_2, ...$ はプロセスの式，$x, y, ...$ は名前である．

$P ::= 0$	何もしない
$\mid x(y).P$	x から値を受け取り,P 中の y に束縛する
$\mid \bar{x}\langle y\rangle.P$	x へ y を送る
$\mid P_1 \mid P_2$	P_1, P_2を並列に実行させる
$\mid (\nu x)P$	P 中に現れる x を束縛する
$\mid\ !P$	P を繰り返す

図 1: π 計算の文法

2.3 定理証明支援系 Coq

Coq はフランス国立情報学自動制御研究所で開発されている定理証明支援系である [1]．Coq を用いることで，プログラムがある仕様を満たすということや，数学的な定理などに対して形式的かつ厳密な証明を与えることができる．Coq によって証明されたものとしては，四色定理が有名である [4]．

2.3.1 カリー・ハワード同型対応

Coq における定理証明は，カリー・ハワード同型対応という理論に基づいている．カリー・ハワード同型とは，論理における命題と証明という構造と，コンピュータプログラムにおける型と項という構造が直接的に対応しているという理論である．これにより，ある命題を証明することが，命題に対応する型と，その型を持つ項を作ることに帰着される．

カリー・ハワード同型対応では，表 1 に示すような対応関係がある．

表 1: カリー・ハワード同型対応

論理	コンピュータプログラム
命題 P	型 `P`
命題 P の証明	型 `P` を持つ項
命題 $P \Rightarrow Q$	型 `P -> Q` (`P` から `Q` への関数)
命題 $P \wedge Q$	`P * Q` の直和型
命題 $P \vee Q$	`P + Q` の直積型

3 並列更新アルゴリズム

本節では，後続の操作をブロックしない *update* 操作を与える．基本的なアイデアは，zig-zig と zig-zag の両方について，目標節点をその深さの半分までしか浮上させない半扁平化 (semi-splaying) を用いることである (文献 [6] の半扁平化は，zig-zig のみが扁平化と異なっていた)．x を更新対象の節点とすると，アルゴリズムは以下のようになる．

(a) 空の木に対する挿入は図 2 (a1) の操作，(空でない) 木の根に対する更新は図 2(a2) の操作を行なう．

(b) zig: x が左部分木の根である場合は図 2(b1) の操作，x が存在すべき左部分木が空の場合は図 2(b2) の操作を行なう．

(c) zig-zig: 図 2(c) 左の木における $x(< b)$ の探索では，枝 ba の右回転を行なってアクセスしたパスの長さを 1 短縮する．次は 1 レベル (短縮前の長さでは 2 レベル) 下降して，部分木 A に対して再帰的に探索を行なう．

(d) zig-zag: 図 2(d) 左の木における節点 x ($b < x < a$) の探索では，枝 cb の左回転と，できた枝 ca の右回転を行ない，アクセスしたパスを 1 短縮する．$x = c$ ならばこれで探索終了である．$x < c$ ならば 2 レベル (短縮前の長さでは 3 レベル) 下降して B の中から x を再帰的に探索する．$x > c$ ならば同様に C の中から再帰的に探索する．$x \geq c$ の場合には枝 ca の回転操作を省略することも考えられる．b の右部分木が空の場合は，そこに節点 x を挿入したあと，上に述べた回転操作を行なう．

以上の操作で，アクセスしたパスの長さは最悪でも約 2/3 になる．半分でなくて 2/3 なのは，上記 zig-zag 操作の性質によるものである．

4 並列削除アルゴリズム

並列削除のための基本的な着想は，扁平化操作を，削除すべき節点を下降させるために利用することである．これまでは，扁平化操作はもっぱら，再度アクセスしそうな節点を浮上させるために用いられてきた．ここで重要なことは，削除対象の節点以外は高々 O(1) レベルしか下降させないようにすることである．以下では，z を削除対象の節点とする．

まず，根節点が削除対象節点 z である場合を考える．この場合，zipping と呼ぶ操作によってそれを “容易に” 削除できる場所まで下降させる．節点が “容易に” 削除できるとは，その左部分木，右部分木，左部分木の右部分木，右部分木の左部分木のいずれかが空であることである．根節点の下降によって，その左部分木と右部分木の縫い合せが起きる．

(a) “容易に” 削除できる場合：図 3(a1) または (a2) のように変形する．

(b) “容易に” 削除できない場合：図 3(b) のように zig-zag を施し，その結果できる b の右部分木に，(一つめとは左右対称な) zig-zag を施す．
4 回の回転で z は 2 レベル下降する．z の新たな部分木 C と F は，同じレベルにとどまる．それ以外の節点も高々1 レベルしか下降しない．z を根とする新たな部分木に対して再帰的に削除操作を行なうが，z の子孫でない節点がそれによってさらに下降することはない．

図 4 に，根節点 z の削除による木の形状の変化を示す．

削除対象節点 z が根であるとは限らない場合は，まず第 3 節の方法で z を探索する．これは根から z に至るパスを短縮する効果をもつ．つぎに，z を zipping によって下降させて削除する．

Zipping 操作はパスの短縮を行なわないが，アクセスした節点は浮上させるという原則にしたがうならば，zipping に先だって，左部分木の最大要素に至るパスと右部分木の最小要素に至るパスをそれぞれトップダウンの半扁平化 (zig-zig (図 2(c)) の繰返し) によって短縮すればよい．この短縮化は zipping と並行して行なうことができる．

Zipping は更新操作と異なり，各節点のキー値を読むことなく木を下降する．また zipping は，木 T_1 と木 T_2 (T_1 のどのキーも，T_2 のどのキーよりも小さいものとする) とのトップダウン併合操作にも応用できる．すなわち，新たな節点 (キーは任意) を調達し，その左部分木を T_1，右部分木を T_2 として一つの木を

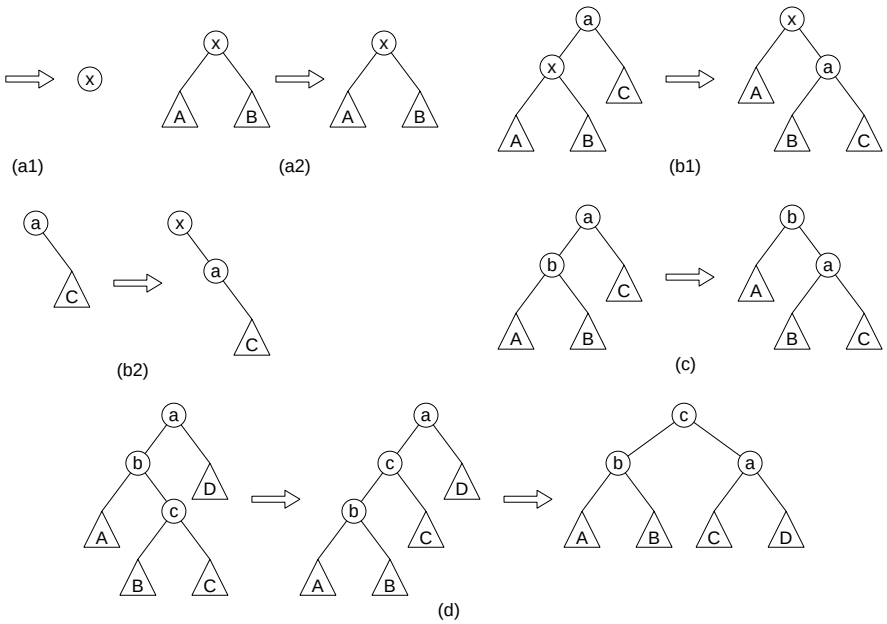

図 2: 後続操作をブロックしない更新アルゴリズムの 1 ステップ

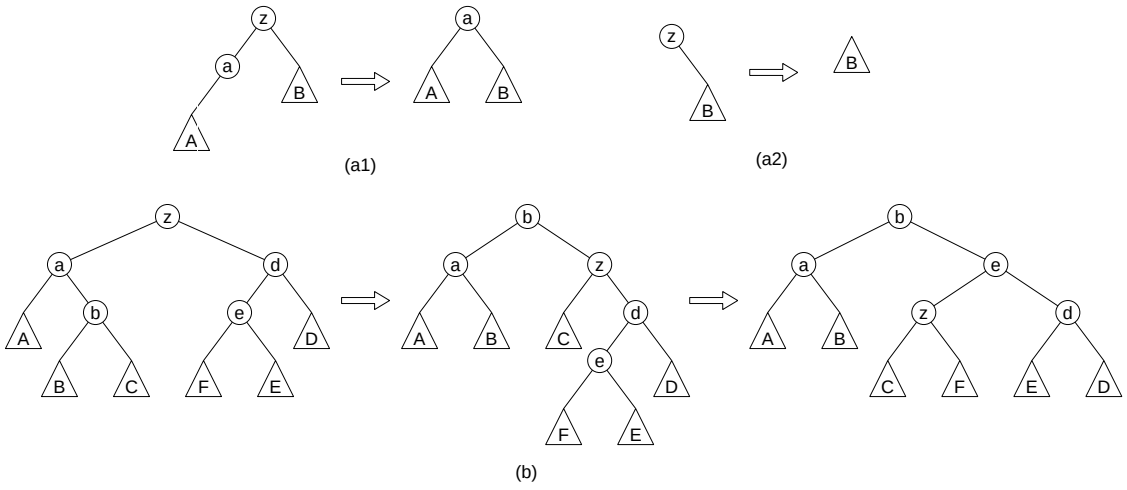

図 3: 後続操作をブロックしない削除アルゴリズムの 1 ステップ

構成したのち，調達した根節点を消去すればよい．

5 計算量に関する結果と考察

効率の二つの尺度のうち，スループットについては容易に議論ができる．すなわち，二つの操作は，レベル l (根をレベル 0 として) の節点を $O(l)$ 回 — *update* は高々$(l+2)$ 回，zipping は高々$(2l+2)$ 回 — の回転操作ののちに確定させる．さらにどちらの操作も，連続する高々3 レベルの節点を同時に施錠するだけでよい．これらのことから，木の大きさや深さによらないスループットで，操作系列をパイプライン的に並列処理することができる．

レスポンスは，*update* については，通常のスプレー木と同等の償却計算量をもつことが証明できる．具体的には，節点 x の**大きさ** $s(x)$ を x を根とする部分木の節点数と定義し，**ランク** $r(x)$ を $\log_2(s(x))$ とす

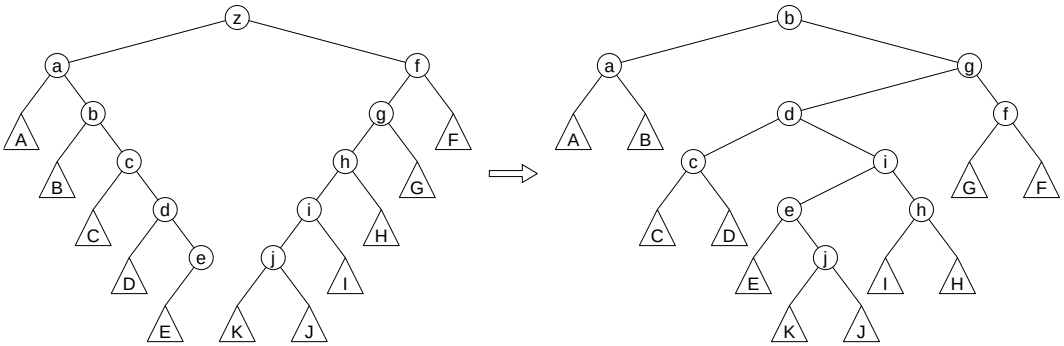

図 4: Zipping による節点 z の削除

る．そして木の**ポテンシャル**を，すべての節点のランクの和と定義する．すると，*update* の償却時間，つまり回転操作の回数で測った所要時間に操作前後のポテンシャルの変化を加えたものは，n を木の節点数として，$\mathrm{O}(\log n)$ であることを示すことができる．このことから，十分長い操作系列の平均レスポンスは，最悪でも対数的であることがわかる．文献 [6] のように，節点に異なる重みをつけて s や r を定義することにより，より強い性質を示すこともできるが，本論文では省く．

一方，*delete* については，文献 [6] の解析方法では，対数的償却計算量を導くことはできない．そのことを示すために，図 3 (b) の 4 回の回転によるポテンシャル変化を考える．

図 3(b) の一番右側の木のランク関数を r' とする．一番左側の木からのポテンシャルの変化を，k をある正定数として $k(r'(b) - r'(z))$ 以内に押さえることができることを示すのが，文献 [6] における償却計算量の証明技法の基本であった．しかし，これらの木について $s(A) = s(B) = s(C) = h \gg t = s(D) = s(E) = s(F)$ を仮定すると，ポテンシャル変化が h/t に関して $\mathrm{O}(\log(h/t))$ となる．一方 $r'(b) - r'(z)$ は h/t に関して $\mathrm{O}(1)$ であるので，上記の要請を満たす k は存在しないことがわかる．Zipping に先立ってパス短縮化を行なった場合についても，同様のことが示せる．

しかし，第 4 節の削除操作は，アクセスしたパス上の節点の深さが約半分になり (事前にパス短縮化を施した場合)，それ以外の節点も高々定数レベルしか沈まないという，節点の浮き沈みについてのスプレー木一般の性質は満たしている．では一般に，この二つの性質を満たす自己調整的な木アルゴリズムで，平均レスポンスが対数時間で押さえられないような，十分長い操作系列は存在するのだろうか？ これは未解決であるが，本論文で提案した二操作については，平均レスポンスは少なくとも $\mathrm{O}(\sqrt{n})$ (更新のみならば $\mathrm{O}(\log n)$) と予想される．

その根拠として，各節点の削除しやすさの変化を考える．節点 x の**削除困難度** $d(x)$ を，x からその直前のキー x_- をもつ節点へ至るパス長 (x_- が存在しない場合や，x_- が x の子孫でない場合は 0 と定める) と直後のキー x_+ をもつ節に至るパス長の最小値と定めると，第 4 節の *delete* は，d の大きな節点の消去には時間がかかるものの，残った各節点の d を高々 $\mathrm{O}(1)$ しか大きくしない．また第 3 節の *update* で新たに挿入した節点の d は 0 であり，*update* はすでに存在していた各節点の d も高々$\mathrm{O}(1)$ しか大きくしない．(ボトムアップ扁平化における節点の d の増加は，定数で押えることができない．) これらのことから

1. 新たな節点の d の値が k まで成長するには，他の節点の $\Omega(k)$ 回の挿入削除が必要

であることがわかる．さらに

2. 二分木における各節点の d の総和は，木をトラバースしたときに通る枝の延べ本数を上回ることはないから $\mathrm{O}(n)$

である．1. と 2. から，新たな節点の挿入と，d の大きな節点の消去が繰り返されるという最悪の操作系列を考えても，操作の平均の手間は $\mathrm{O}(\sqrt{n})$ であり，実

用上の効率は更新操作のみの場合とほとんど変わらないと予想される．

6 まとめと今後の課題

節点の浮き沈みに関する望ましい性質を保ち，かつ計算量の意味で最適なスループットをもつ自己調整二分木の並列操作 (更新，挿入，削除，併合) アルゴリズムを提案した．節点の更新や挿入に関しては対数的償却計算量を持つことが証明できており，さらにアクセスパターンの偏りや変化に対する追従性など，スプレー木の持つ強力かつ頑健な性質の多くを引き継いでいる．削除の償却計算量のより良い理論的限界を導く (またはその不存在を示す) ことは今後の課題である．また，アルゴリズムの実際的効率，並列分散環境での実装，応用の検討も今後の課題である．

謝辞 しゃじ

参 考 文 献

[1] : The Coq Proof Assistant, `http://coq.inria.fr/`.

[2] Agha, G. and Thati, P.: An Algebraic Theory of Actors and Its Application to a Simple Object-Based Language, *From Object-Orientation to Formal Methods*, Lecture Notes in Computer Science, Vol. 2635, Springer-Verlag, 2004, pp. 26–57.

[3] Agha, G. A.: *ACTORS - A Model of Concurrent Computation in Distributed Systems*, MIT Press series in artificial intelligence, MIT Press, 1990.

[4] Gonthier, G.: A computer-checked proof of the Four-Colour Theorem, 2004.

[5] Milner, R.: *Communicating and Mobile Systems: The π-calculus*, Cambridge University Press, New York, NY, USA, 1999.

[6] Sleator, D. D. and Tarjan, R. E.: Self-Adjusting Binary Search Trees, *J. ACM*, Vol. 32, No. 3(1985), pp. 652–686.