正則表現を用いた並列ごみ集めの抽象モデル検査

高橋孝一[†] 萩谷昌己[††]

モデル検査は状態空間の全探索によってシステムの検証を自動的に行う枠組みであるが，状態空間の爆発の問題をつねにかかえている．抽象モデル検査はこの問題を解決する 1 つの有力な技術であるが，メモリやネットワークなどのリンク構造を抽象化するための一般的な方法は知られていなかった．そこで本稿では，リンク構造を大きさによらず抽象化する方法を提案する．この方法では，リンク構造を構成するセルを，あらかじめ与えた正則表現を満たすか否かによって抽象化する．リンク構造全体は，セルを抽象化して得られる抽象セルの集合に抽象化される．この方法の有効性を示すために，並列ごみ集めの複数のアルゴリズムの抽象モデル検査を行った．

Abstract Model Checking of Concurrent Garbage Collection by Regular Expressions

Koichi Takahashi[†] and Masami Hagiya[††]

Model checking, which is a framework for automatically verifying a system by enumerating its entire state space, always faces the problem of state space explosion. Although abstract model checking is considered as a promising technique for solving the problem, a general method that can abstract linked structures, such as memory or network, was not known. In this paper, we propose a method for abstracting linked structures independent of their size. By the method, each cell in a linked structure is abstracted by whether it satisfies each of the predefined regular expressions. The entire linked structure is then abstracted by a set of abstract cells, each of which is a result of abstracting a cell. For showing the effectiveness of the method, we did abstract model checking of the safety property of several algorithms for concurrent garbage collection.

1. はじめに

抽象モデル検査とは状態遷移システムを抽象写像によって抽象システムに写し，抽象システムを従来の方法でモデル検査することによって，もとの状態遷移システムの正当性を検証する手法である[6),7),9),10)]．たとえ状態遷移システムが無限であっても，抽象写像によっては抽象システムは有限になることがある．また，状態数が多すぎて限られた時間内には計算できない場合に，抽象システムを用いることによって計算時間を大幅に減らすことが可能になることもある[2),5)]．

本稿では，システムの状態がリンク構造をデータとして持つ場合を考察の対象とする．リンク構造とはセルの配列もしくは集合であり，各セルはセルへのポインタをいくつか持つものとする．有向グラフもリンク構造と見なすことができる．リンク構造はコンピュータのメモリや，ネットワークをモデル化したときに自然に現れる構造である．このようなシステムをモデル検査する場合，探索空間はリンク構造の大きさに依存してしまう．したがって，リンク構造の大きさを制限しなければモデル検査は不可能である．

そこで，リンク構造の大きさに依存しないようなリンク構造の抽象化を行いたい．このような抽象化として，Shape-Analysis と呼ばれる方法がある（文献 8) など）．Shape-Analysis では，リンク構造がリストになっているか，木になっているか，サイクルになっているかといった，構造の性質をとらえることができる．しかし，各セルの内容はまったく考慮できないため，たとえば並列ごみ集めの正しさは彼らの方法では分からない．

我々はリンク構造を正則表現を用いてセルの内容も考慮に入れて抽象化する方法を提案する．この方法を用いると，抽象化された状態はリンク構造の大きさに依存しない．よって，この方法で抽象化されたシステムをモデル検査すれば，もとのリンク構造の大きさを制限することなく検証を行うことができる．

† 電子技術総合研究所
Electrotechnical Laboratory
†† 東京大学大学院理学系研究科
Graduate School of Science, University of Tokyo

本章の以下の部分では，正則表現を用いたリンク構造の抽象化の概要を説明する．リンク構造に対して，セルを状態，ポインタを状態遷移とするオートマトンを考える．各セルには，そのセルを初期状態とする有限オートマトンが対応する．我々は，このオートマトンがどのような性質を持つかに従って，セルの抽象化である抽象セルを定義する．

各セルに対応するオートマトンの性質は，あらかじめ与えておいた有限個の正則表現の集合を用いて表現する．セルを初期状態とする有限オートマトンが，各正則表現に属する文字列を受理できるかどうかの真偽値の組合せによってセルを抽象化する．したがって，抽象セルは各正則表現に対応する真偽値の組合せにほかならない．正則表現を有限個にしておいたので，抽象セルの種類も有限個になる．リンク構造全体の抽象化は，すべてのセルを抽象化した抽象セルの集合とする．もとのリンク構造を X としたとき，これを $N(X)$ と表す．

抽象モデル検査を行うためには，もとのシステムの状態遷移 t に対応した抽象状態の遷移，すなわち抽象遷移 t_A を定義しなければならない．状態がリンク構造であり，状態遷移 t がリンク構造の変化 $X \to t(X)$ を起こす場合，抽象遷移 t_A は抽象セルの集合の変化 $N \to t_A(N)$ を起こす必要がある．抽象遷移 t_A が妥当であるとは，$N \supseteq N(X)$ ならば $t_A(N) \supseteq N(t(X))$ が満たされていることと定義する．抽象遷移が妥当であるとき，初期状態において $N \supseteq N(X)$ が満たされていれば，初期状態から到達可能な任意の状態においても $N \supseteq N(X)$ が満たされる．したがって，抽象モデル検査においてエラー状態に到達しなければ，もとの状態遷移システムにおいてもエラー状態に到達しないことが分かる．

もとの状態遷移システムがエラー状態に到達しないにもかかわらず，抽象化によっては，抽象モデル検査では実際には生じないエラー状態に到達してしまう場合がある．このような状況を「抽象モデル検査に失敗した（成功しなかった）」と呼ぶことにする．逆に抽象モデル検査によって抽象システムがエラー状態に到達しないことが示せた場合に「抽象モデル検査に成功した」と呼ぶことにする．妥当性を満たしつつ，抽象モデル検査が成功するような抽象遷移を定義することは，容易なことではない．妥当な抽象遷移を定義するだけならば，$t_A(N)$ を，状態遷移 t が新たに作る可能性のある（セルに対応する）抽象セルを N に追加していくものと定義すればよい．これは自明ではないが，比較的容易である．しかし，この方法では抽象セルの集合は大きくなる一方なので，抽象化の木目が粗すぎて抽象モデル検査はほとんど成功しないであろう．抽象モデル検査が成功するためには，一般にいくつかの抽象遷移において抽象セルの集合を減らすことが必要となる．

もとの状態遷移の性質から，抽象遷移において減らすべき抽象セルが簡単に分かる場合がある．本稿では，残された抽象セルから矛盾した抽象セルを除くことによって，より木目の細かい抽象遷移を得る方法を確立した．抽象セルの集合 N は，もし仮にそれがリンク構造を反映したものであるならば，その中で存在することが矛盾している抽象セルが計算できる．これらの矛盾抽象セルを $I(N)$ と書き，$R(N)$ を $I(N)$ が空になるまで $N := N - I(N)$ を繰り返して得られたものとする．$R(N)$ は $N \supseteq N(X)$ ならば $N \supseteq R(N) \supseteq N(X)$ を満たすので，抽象遷移 t_A が妥当であれば，抽象遷移 $t_A^R(N) = R(t_A(N))$ も妥当である．しかも，$R(N)$ の方が N よりも木目が細かいので，抽象モデル検査に成功する可能性が増える．

我々は，上述した抽象化を用いて，並列ごみ集めアルゴリズムの正当性の抽象モデル検査を行った．検証したアルゴリズムの 1 つは Dijkstra ら[3] によって定式化されたオンザフライごみ集めと呼ばれるアルゴリズムであり，もう 1 つは湯浅[12),14)] によるスナップショットごみ集めと呼ばれるアルゴリズムである．これらのアルゴリズムのデータ構造はレジスタとヒープを用いてモデル化されるが，これはまさしくリンク構造を成しているので，本稿の抽象化が利用できる．我々が文献 13) において行った並列ごみ集めアルゴリズムの抽象モデル検査では，抽象遷移を定義するためにフィルタという自明でない概念を導入する必要があったが，本稿では，上述した抽象遷移を用いることにより，その必要がなくなっている．

本稿は次のように構成されている．まずリンク構造を正則表現を用いて抽象化する方法を説明し，それに基づく抽象モデル検査を考察する．次に例となる並列ごみ集めのモデルを導入し．その有限モデルと抽象モデル検査について説明する．

2. 正則表現を用いた抽象化

この章では，正則表現を用いたリンク構造の抽象化を提案し，それについて議論する．

文字の有限集合 Σ を定めておく．X が**リンク構造**であるとは，X はセルの配列であり，各セル c は文字 $\sigma(c) \in \Sigma$ と，セルへのリンク（ポインタ）を有限個持っていることをいう．より正確には，リンク構造

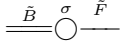
はインデックスの有限集合 I と関数 $\sigma : I \to \Sigma$ と関数 $f : I \to 2^I$ から成り立っている．

2.1 抽象セルとその集合

我々は各セルを抽象化した抽象セルの集合によってリンク構造を抽象化する．まず，その抽象セルの形式的な定義を与える．Σ 上の有限個の正則表現の集合 D を 1 つ固定する．正則表現の先頭に否定記号 $\neg$ をつけたものを，負の正則表現と呼び，D の各正則表現に否定記号をつけたものの集合を $\neg D$ と書く．両者の和は $\tilde{D}$ と書く．すなわち $\tilde{D} = D \cup \neg D$ である．

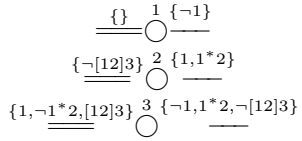
たとえば，$\Sigma = \{1, 2, 3\}$，$D = \{1, 1^*2, [12]3\}$ ならば $\neg D = \{\neg 1, \neg 1^*2, \neg[12]3\}$ である．なお，$[12]$ は正則表現 $(1 + 2)$ を表す．

定義 1 抽象セルは 3 つ組 $(\sigma, \tilde{F}, \tilde{B})$ と定義する．ただし $\sigma \in \Sigma$，$\tilde{F} \subseteq \tilde{D}$，$\tilde{B} \subseteq \tilde{D}$ である．$\tilde{F}$，$\tilde{B}$ をそれぞれ，前方条件，後方条件と呼ぶことにする．$F = \tilde{F} \cap D$，$\neg F = \tilde{F} \cap \neg D$ などと書いて，正の前方条件，負の前方条件と呼ぶ．抽象セルを図として



と表すこともある．

前の例では



などは抽象セルである．

後により正確に記述するが，抽象セルの持つ前方条件 $\tilde{F}$ は，抽象化されたもとのセルを初期状態とし，ポインタを状態遷移とするオートマトンに関する条件を表そうとしている．正の前方条件 F の各正則表現に属する文字列でオートマトンによって受理されるものがあり，負の前方条件 $\neg F$ の各正則表現については，それに属するどの文字列も決して受理することができない，ということを表す．後方条件 $\tilde{B}$ も同様であるが，こちらはオートマトンを逆向きに考える，つまり，リンクを逆にたどる点だけが異なる．

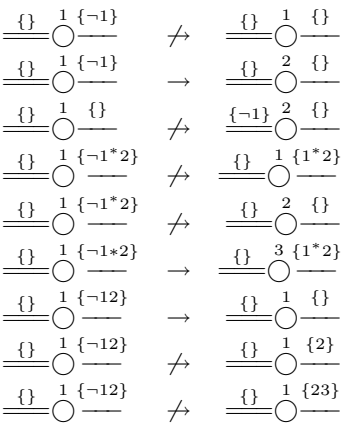
次に，抽象セルの集合から，有向グラフを構成する方法を与える．2 つの抽象セル間に辺が存在することが，抽象化されたもとの 2 つのセル間にリンクが張られている可能性を意味するように，グラフを構成する．負の前方（後方）条件に関する情報から明らかに辺が存在しないと分かるもの以外は，すべての抽象セル間に辺を張る．正確に定義しよう．

定義 2 抽象セルの集合 N と，自然数 i が与えられたとき，抽象セル間の**辺の集合** $E^i_N \subseteq N \times N$ を次のように定義する．$n_1 = (\sigma_1, \tilde{F}_1, \tilde{B}_1), n_2 = (\sigma_2, \tilde{F}_2, \tilde{B}_2) \in N$ とする．$(n_1, n_2) \notin E^i_N$ である必要十分条件を

- $\exists \neg f_1 \in \neg F_1 \,.\, (\sigma_2 \in f_1 \vee \ \exists f_2 \in F_2 \,.\, (\sigma_2 f_2 \subseteq f_1 \vee 1 \leq \exists k \leq i \,.\, \mathrm{pre}_k(\sigma_2 f_2) \subseteq f_1))$ または
- $\exists \neg b_2 \in \neg B_2 \,.\, (\sigma_1 \in b_2 \vee \ \exists b_1 \in B_1 \,.\, (\sigma_1 b_1 \subseteq b_2 \vee 1 \leq \exists k \leq i \,.\, \mathrm{pre}_k(\sigma_1 b_1) \subseteq b_2))$

が成り立つこととする．ここで，$\sigma_2 f_2$ と $\sigma_1 b_1$ は正則表現の連結である．pre は正則表現によって表現される文字列 s から前方 k 文字だけ切り取ってできる文字列 $s|_k$ の集合である．正確には $\mathrm{pre}_k(\sigma_2 f_2) = \{\, s|_k \mid s \in \sigma_2 f_2 \}$ である．ただし，文字列 s が長さ k 以上ならば $s|_k$ は s の先頭の k 文字から成る部分文字列であり，長さが k 以下ならば $s|_k = s$ である．

たとえば，$\Sigma = \{1, 2, 3\}, D = \{1, 1^*2, 12, 2, 23\}$ とすると



となる．最後の例は $k = 2$ の場合に条件が満たされるので $i \geq 2$ のときは，辺が存在しないが，$i = 1$ のときは存在する．

E^i_N の i の値は，以下の議論に影響を与えないので，ある値 i が選ばれ固定されているものと仮定する．E^i_N を単に E_N と書く．

抽象セルの間の辺の存在はその 2 つの抽象セルだけによって定まることに注意されたい．このことから次の補題は自明である．

補題 1 2 つの抽象セルの集合 N，N' に対して，抽象セル n_1，n_2 が両者に入っているならば，$(n_1, n_2) \in E_N$ と $(n_1, n_2) \in E_{N'}$ は同値である．特に N が N' の部分集合になっていれば，(N, E_N) は $(N', E_{N'})$ の部分グラフになる．

有向グラフの構成は負の前方（後方）条件に関する情報を用いたが，正の前方（後方）条件に関する情報から，抽象セルの集合の中に存在しえない元が計算できる．これを計算するために抽象セルを状態とする

オートマトンを用いる．

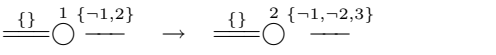
抽象セルの有向グラフ (N, E) とその元 $n \in N$ が与えられたとき，**非決定性有限オートマトン** (N, E, n) は次のように定義する．状態集合は N，初期状態は n，n_1 から n_2 へ状態遷移があるかどうかは $(n_1, n_2) \in E$ と同値とし，遷移のラベルは σ_2（ただし $n_2 = (\sigma_2, \tilde{F}_2, \tilde{B}_2)$）とする．すべての状態を受理状態とする．オートマトン (N, E, n) の定める言語を $L(N, E, n)$ と書く．

定義 3 抽象セルの集合 N が与えられたとき，$n = (\sigma, \tilde{F}, \tilde{B}) \in N$ が N **において無矛盾である**必要十分条件を

- $\forall f \in F\ .\ f \cap L(N, E_N, n) \neq \emptyset$ かつ
- $\forall b \in B\ .\ b \cap L(N, E_N^{op}, n) \neq \emptyset$

とする．ここで E_N^{op} は E_N の辺の向きを逆にしたものである．n が N において無矛盾でないとき矛盾しているという．

たとえば，抽象セルの集合 N を 2 つの抽象セル $n_1 = (1, \{\neg 1, 2\}, \{\})$，$n_2 = (2, \{\neg 1, \neg 2, 3\}, \{\})$ から成るものとする．有向グラフ (N, E_N) は



となる．オートマトンの定める言語はそれぞれ $L(N, E_N, n_1) = \{2\}$，$L(N, E_N, n_2) = \emptyset$ となる．n_1 は N において無矛盾だが，n_2 は N において矛盾している．

定義 4 $I(N) = \{n \in N \mid n$ は N において矛盾 $\}$ と書き，$I(N) = \emptyset$ になるまで $N := N - I(N)$ を繰り返して得られる N を $R(N)$ と定義する．

$R(N)$ は N から一意に定まり，$R(N)$ の元はすべて $R(N)$ において無矛盾である．

先ほどの例では n_1 は N においては無矛盾であったが，矛盾していた n_2 を除いた集合 $N - I(N) = \{n_1\}$ においては矛盾する．したがって，$R(N) = \emptyset$ である．

2.2 リンク構造の抽象化

リンク構造 X が与えられたとき，それを抽象化して得られる抽象セルの集合を定義する．まず X を次のように非決定性有限オートマトンと見なす．セルを状態として，セル c から c' へリンクがある場合に状態遷移が存在し，そのラベルは c' に割り当てられている文字とする．初期状態をセル c，受理状態を全状態としたときのオートマトンの定める言語を $L(X, c)$ と書く．またリンクを逆方向に見て，オートマトンを定義した場合 $L(X^{op}, c)$ と書く．

定義 5 D の部分集合 D_F，D_B が与えられているとする．D_F**，**D_B **によって定まる抽象セル** $n(c) = (\sigma, \tilde{F}, \tilde{B})$ を

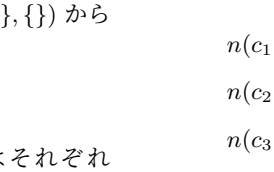
- σ は c に割り当てられている Σ の値 $\sigma(c)$.
- $\tilde{F}$ は，各 $d \in D_F$ について $d \cap L(X, c) = \emptyset$ なら $\neg d \in \tilde{F}$，そうでないなら $d \in \tilde{F}$ となる最小の集合.
- $\tilde{B}$ は，各 $d \in D_B$ について $d \cap L(X^{op}, c) = \emptyset$ なら $\neg d \in \tilde{B}$，そうでないなら $d \in \tilde{B}$ となる最小の集合.

と定義する．

前方条件は D_F の各元の正か負のどちらか一方を持ち，後方条件は B_F の各元の正か負のどちらか一方を持つ．

リンク構造全体は $N(X) = \{n(c) \mid c \in X\}$ という，抽象セルの集合として抽象化される．

たとえば，リンク構造 X が 3 つのセル c_1，, c_2，c_3 から成り，それぞれ文字 1，1，2 が割り当てられているとする．リンクは c_1 から c_2，c_2 から c_3 にあるとする．$D_F = \{1^*2\}$，$D_B = \{1\}$ とすると，$N(X)$ は



となる．

以上で定義した抽象化が正当である証拠として，次の補題が成り立つ．

補題 2 X において c_1 から c_2 に直接リンクがあったとすると，$(n(c_1), n(c_2)) \in E_{N(X)}$ となる．

証明をする．$n(c_1) = (\sigma_1, \tilde{F}_1, \tilde{B}_1)$，$n(c_2) = (\sigma_2, \tilde{F}_2, \tilde{B}_2)$ とする．もし $(n(c_1), n(c_2)) \notin E_{N(X)}$ ならば，対称性から $\exists \neg f_1 \in \neg F_1\ .\ (\sigma_2 \in f_1 \vee \exists f_2 \in F_2\ .\ (\sigma_2 f_2 \subseteq f_1 \vee 1 \leq \exists k \leq i\ .\ \mathrm{pre}_k(\sigma_2 f_2) \subseteq f_1))$ としてよい．$\sigma_2 \in f_1$ なら $\sigma_2 \in L(X, c_1)$ なので $f_1 \cap L(X, c_1) = \emptyset$ と矛盾する．$\sigma_2 f_2 \subseteq f_1$ の場合，$f_2 \cap L(X, c_2) \neq \emptyset$ であるから $\sigma_2(f_2 \cap L(X, c_2)) = \sigma_2 f_2 \cap \sigma_2 L(X, c_2) \subseteq f_1 \cap L(X, c_1)$ なので，$f_1 \cap L(X, c_1)$ も空集合ではない．これは $n(c_1)$ の構成と矛盾する．$\exists k \geq 1\ .\ \mathrm{pre}_k(\sigma_2 f_2) \subseteq f_1$ の場合，ある文字列 $s_2 \in L(X, c_2) \cap f_2$ が存在するので，$\sigma_2 s_2|_k \in f_1$ となる．しかし $\sigma_2 s_2|_k \in L(X, c_1)$ も成り立つので，これは $f_1 \cap L(X, c_1) = \emptyset$ であることと矛盾である．証明終わり．

先の例では $E_{N(X)}$ は

$$\{\quad (n(c_1), n(c_2)), (n(c_1), n(c_3)), \\ (n(c_2), n(c_2)), (n(c_2), n(c_3)) \quad \}$$

なので，確かに補題が成り立っている．また，すべての抽象セルは無矛盾である．

次の補題は矛盾している抽象セルは除去可能であることを示している．

補題 3 $N(X)$ のすべての抽象セルは $N(X)$ において無矛盾である．

証明をする．今 $n(c)$ が $N(X)$ において矛盾しているとする．対称性から，ある $f \in F$ について $f \cap L(N(X), E_{N(X)}, n(c)) = \emptyset$ となっているとしてよい．$n(c)$ の構成から $f \cap L(X, c) \neq \emptyset$ であるので，文字列 $\sigma_1 \ldots \sigma_k \in f \cap L(X, c)$ を 1 つとる．$L(X, c)$ の構成からリンクのつながっているセルの列 $cc_1 \ldots c_k$ が X に存在し，$\sigma_i = \sigma(c_i)$ となる．前の補題から $n(c), n(c_1), \ldots, n(c_k)$ は $(N(X), E_{N(X)}, n(c))$ において遷移列となりえ，$\sigma_1 \ldots \sigma_k$ は受理可能である．これは $f \cap L(N(X), E_{N(X)}, n(c)) = \emptyset$ と矛盾する．よって補題は証明された．

補題を組み合わせることによって，次の定理が簡単に導かれる．

定理 1 リンク構造 X と抽象セルの集合 N が $N \supseteq N(X)$ を満たすならば，$N \supseteq R(N) \supseteq N(X)$ である．

2.3 抽象モデル検査

状態遷移システムを直接にモデル検査する代わりに抽象化されたシステムをモデル検査するのが抽象モデル検査であった．状態遷移システムの状態として，リンク構造をそのデータの一部として持つものを考えよう．この場合，前節で定義したようなリンク構造の抽象化を用いて，抽象モデル検査を行うことが期待される．そのためには，もとのシステムにおける状態遷移 t に対して抽象状態の遷移，すなわち抽象遷移 t_A を定義しなければならない．状態遷移 t がリンク構造を $X \to t(X)$ と変化させ，抽象遷移 t_A は抽象セルの集合を $N \to t_A(N)$ と変化させる．抽象モデル検査が意味を持つためには，抽象遷移が以下の性質を持たなければならない．

定義 6 抽象遷移 t_A が妥当であるとは，$N \supseteq N(X)$ ならば $t_A(N) \supseteq N(t(X))$ を満たしていることとする．

抽象遷移が妥当であれば，初期状態において $N \supseteq N(X)$ が満たされていれば，到達可能な任意の状態に対しても $N \supseteq N(X)$ が満たされているので，抽象モデル検査においてエラー状態に到達しなければ，もとの状態遷移システムでもエラー状態に到達しないことが分かる．

このように妥当な抽象遷移を見つけることが，抽象モデル検査の鍵の 1 つである．

リンク構造を変化させる状態遷移の最も基本的なものは

- あるセルの文字を変化させる．
- あるセルからあるセルへリンクを作る．
- あるセルからあるセルへのリンクを消す．

の 3 つであろう．この 3 つの状態遷移に対して，妥当な抽象遷移を構成する指針を与える．

- あるセル c の文字 σ を σ' に変化させる場合，$n(c)$ となりうるすべての $n = (\sigma, \tilde{F}, \tilde{B}) \in N$ に対して，σ を σ' に置き換えた抽象セル n' を N に加える．また，同時に (N, E_N) において，n から辺をたどって到達可能な抽象セルすべてに対し，その後方条件がどのように変わりうるか計算し，後方条件を変化させた抽象セルを N に加える．同様に n から辺を逆にたどって到達可能な抽象セルに対して，その前方条件を変化させた抽象セルを N に加える．
- あるセル c からあるセル c' へリンクを作る場合，$n(c)$ となりうるすべての $n \in N$ に対して，(N, E_N) において n から逆に辺をたどって到達可能な抽象セルに対し，その前方条件がどのように変わりうるか計算し，前方条件を変化させてできる抽象セルを N に加える．同様に，$n(c')$ となりうるすべての $n' \in N$ に対して，(N, E_N) において n' から辺をたどって到達可能な抽象セルに対し，その後方条件がどのようになりうるか計算し，後方条件を変化させてできる抽象セルを N に加える．ここで，変化する条件は負の条件が正の条件に変わるものしか考慮しなくてよい．
- あるセル c からあるセル c' へリンクを消す場合もリンクを作る場合と同様であるが，変化する条件は正の条件が負の条件に変わるものしか考慮しなくてよい．

この方針で与えられる抽象遷移は，抽象セルの集合をふくらませる一方であり，妥当性が成り立つことが容易に分かる．

実際の状態遷移にはリンクの張替えといった，リンクの作成と消去などが不可分に行われるものがあり，妥当な抽象遷移をこの方針だけでは決められないかもしれない．ただし，状態遷移によって新たに発生しうる抽象セルを計算し，それを抽象セルの集合に追加するという抽象遷移は自然であり，妥当性を満たしやすい．

しかし，ここにあげた方針で決められる抽象遷移は抽象セルの集合を増やす一方なので，抽象モデル検査において，もとのシステムでは生じないエラー状態が生じる可能性が大変高い．実際の問題においては，抽

象セルの集合を減らすような抽象遷移も必要になるであろう．このような抽象遷移の妥当性が状態遷移の条件から簡単に分かる場合がある．これを自然な抽象遷移と呼ぶことにする．しかし自然な抽象遷移では，抽象化の木目が粗すぎて抽象モデル検査において，エラー状態を生じてしまう場合がある．そこで，矛盾した抽象セルの除去を行ってより木目の細かい抽象遷移を構成する．

抽象遷移 t_A が妥当ならば，$t_A^R(N) = R(t_A(N))$ として定義される抽象遷移 t_A^R は妥当である．なぜならば，定理 1 から $N \supseteq N(X)$ が満たされていれば $t_A(N) \supseteq t_A^R(N) \supseteq N(t(X))$ が満たされているからである．抽象遷移 t_A^R は t_A より木目が細かく抽象モデル検査に成功する（エラー状態に到達しないことが示せる）可能性が増加する．

3. 並列ごみ集めの抽象モデル検査

並列ごみ集めの 2 つの主要なアルゴリズムを本稿で定義した抽象化を用いて抽象モデル検査を行う．本質的には我々が行った抽象モデル検査[13] と同じであるが，そこで使われていた場当たり的な抽象遷移を使う必要がなくなっている．

3.1 並列ごみ集めのモデル

2 つの並列ごみ集めのモデルを紹介する．1 つは Dijkstra ら[3] によって定式化されたオンザフライごみ集めと呼ばれるアルゴリズムである．もう 1 つは湯浅[12),14)] によるスナップショットごみ集めと呼ばれるアルゴリズムである．本稿ではこれらを我々のモデルに適合するように単純化したものを扱う．

3.1.1 セルとレジスタ（図 1）

セルのヒープは形式的には `cell_index` から `cell` への関数（もしくは配列）として定義される．ヒープは `C` と書き表し，`C[i]` によって `i` 番目のセルを表す．

```
C : heap = cell_index -> cell
```

セルはその色と他のセルへのポインタを保持するフィールドから構成される．ポインタはセルインデックスかヌルポインタ `nil` である．本研究では単純化のためセルは 1 つだけしかフィールドを持たない．セルの型 `cell` は次のように定義される．

```
cell = color * (cell_index + {nil})
color = {free, white, gray, black}
```

以下，`i` 番目のセルの唯一のフィールドが保持するポインタは `C[i].f` と書く．

このモデルでは，フリーセルのリストを扱う代わりに，フリーセルはセルの色が**フリー**であるとする．このほかにセルの色として**白色**，**灰色**，**黒色**がある．

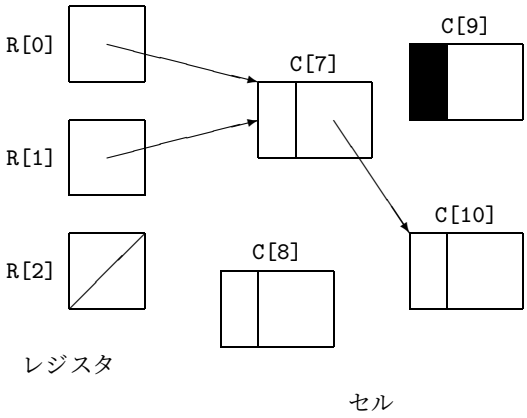


図 1 レジスタとセル
Fig. 1 Registers and cells.

レジスタはセルを操作するミューテータによって使用される．レジスタはセルのマーキングのルートとしてコレクタにも使用される．モデルを実際に近づけるため，レジスタの集合を導入した．以下では `R` は `register_index` から `register` への関数（もしくは配列）を表し，`R[i]` は `i` 番目のレジスタを表す．各レジスタはセルへのポインタを保持する．

```
R : register = register_index ->
                (cell_index + {nil})
```

セルが**直接到達可能**とはそのセルへのポインタがあるレジスタに格納されていることをいう．セルが**レジスタから到達可能**とはあるレジスタからポインタを順にたどっていくことによって到達可能なことをいう．

3.1.2 ミューテータ

ミューテータは**表 1** に示されている操作を持つ．各操作はレジスタもしくはセルのフィールドを変化させる．いくつかの操作は表 1 に記されているようにセルの色も変化させる．表の 2 列目がオンザフライごみ集めであり，3 列目がスナップショットごみ集めである．表に現れる `C[R[i]].f`，`C[R[j]].f` は操作が実行される前，すなわち代入される前のセルフィールドに格納された値を示している．

これらの操作は非可分であり，各々はシステム全体に対して 1 つの状態遷移を引き起こす．

3.1.3 コレクタ

コレクタはオンザフライごみ集めと，スナップショットごみ集めで違いはない．

コレクタは 4 つの状態を持つ．システムの状態と区別するため，コレクタの状態は**コレクタステップ**と呼ぶ．コレクタは 4 つのステップ：`shade`，`mark`，`append`，`unmark` を持つ．各ステップにおいてどのよ

表 1 ミューテータの操作
Table 1 Operations of mutator.

	オンザフライ	スナップショット
`R[i] := nil`		
`C[R[i]].f := nil`		もし `C[C[R[i]].f]` が白色なら，それを灰色にする．
`R[i] := j`（セルの割当て）	（`C[j]` がフリーのときのみ．）`C[j]` を灰色にする．	（`C[j]` がフリーであり，コレクタが `shade` でないときのみ．）`C[j]` を黒色にする．
`R[i] := R[j]`		
`R[i] := C[R[j]].f`	もし `C[C[R[j]].f]` が白色なら，それを灰色にする．	
`C[R[i]].f := R[j]`		もし `C[C[R[i]].f]` が白色なら，それを灰色にする．

表 2 コレクタの操作
Table 2 Operations of collector.

`shade`	各直接到達可能なセルが白色ならそれを灰色にする．終わったら `mark` ステップへ進む．
`mark`	灰色セルを選びそれを黒色にする．もしそのセルが白色セルを参照していたら，それを灰色にする．
`mark`	もし灰色セルが存在しなければ，`append` ステップへ進む．
`append`	白色セルを選びそれをフリーにする．
`append`	もし白色セルが存在しなければ，`unmark` ステップへ進む．
`unmark`	黒色セルまたは灰色セルを選び，それを白色にする．
`unmark`	もし黒色セルも灰色セルも存在しなければ，`shade` ステップへ進む．

うな操作が行われるかは，**表 2** にまとめてある．

`shade` ステップの操作は分割不可能であることに注意する．すべての直接到達可能な白色セルは一度に灰色になる．スナップショットごみ集めでは `shade` ステップでは新しいセルの割当てが許されていない．つまり，`unmark` ステップが終了した後で，`shade` ステップが開始される前に新しいセルの割当てが起こることがない．

3.2 検証する性質

検証すべき性質は

安全性（safety）（初期状態から到達可能な）各状態において，フリーセルがレジスタから到達可能になることはない．

である．活性（liveness）に関しては公平性（fairness）を仮定したモデル検査が必要であり，複雑である．この例では，本稿の抽象化の方法の有効性を示すことが目的なので活性は取り扱わない．

3.3 有限モデル検査

Havelund [4]，Bruns [1]，筆者ら [13] がそれぞれ並列ごみ集めのモデルの有限モデル検査を行っている．ただし，いずれもレジスタの数とセルの数を小さな数に限っている．

3.4 ごみ集めの抽象ヒープと抽象状態

まず，オンザフライごみ集めについて考察する．スナップショットごみ集めについては 3.6 節において述べる．並列ごみ集めモデルでは，レジスタとヒープをあわせたものがリンク構造をなしている．レジスタもセルと考え，2 章の手法を用いて，レジスタとヒープの状態を抽象化する．それを抽象ヒープと呼ぶ．

抽象ヒープを定義するために，いくつかの準備が必要である．まず文字集合についてであるが，リンク構造のセルがレジスタであることを示す r と，セルの色，すなわち白色（w），灰色（g），黒色（b），フリー色（f）を用意する．すなわち $\Sigma = \{r, f, b, g, w\}$ とする．

本方法において一番難しいのは，条件に用いる正則表現の定め方である．前方条件はポインタの指しているセルの色だけを考えて $D_F = \{f, b, g, w, r\}$ とする．後方条件については，安全性を議論するために必要な「レジスタから到達可能」であるというものを入れる必要がある．またミューテータの操作が直接レジスタから指されているセルを対象とするため，「レジスタから直接到達可能」というものも用意する．すなわち $D_B = \{r, [bgw]^* r\}$ とする．

我々のモデルではリンクは 1 つだけなので，抽象セル $n(c)$ の前方条件はせいぜい 1 つしか正の正則表現を持たない．また，後方条件が $r, \neg[bgw]^* r$ を両方持つことはない．一般に 2 つの正則表現 f_1，f_2 につい

表 3 ステップ遷移のフィルタ
Table 3 Filters of step transitions.

shade → mark	割り当てられている文字が r であり，正の前方条件に w を含む抽象セルの除去.
mark → append	割り当てられている文字が g である抽象セルの除去. 正の前方（または後方）条件に g が出現する抽象セルの除去.
append → unmark	割り当てられている文字が w である抽象セルの除去. 正の前方（または後方）条件に w が出現する抽象セルの除去.
unmark → shade	割り当てられている文字が g または b である抽象セルの除去. 正の前方（または後方）条件に g または b が出現する抽象セルの除去.

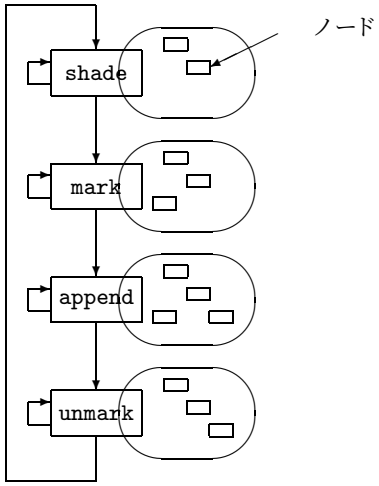


図 2 抽象状態
Fig. 2 Abstract state.

て $f_1 \subseteq f_2$ だった場合，f_1，$\neg f_2$ の両方を前方（後方）条件に持つ抽象セルは必ず矛盾する．

たとえば図 1 を抽象化した抽象ヒープには以下の抽象セルが含まれている．

$$\{\neg r, \underline{\underline{\neg[bgw]^*}} r\} \overset{r}{\bigcirc} \{\underline{\underline{\neg f, \neg b, \neg g}}, w, \neg r\}$$
$$\{\neg r, \underline{\underline{\neg[bgw]^*}} r\} \overset{r}{\bigcirc} \{\underline{\underline{\neg f, \neg b, \neg g}}, \neg w, \neg r\}$$
$$\{r, \underline{\underline{[bgw]^*}} r\} \overset{w}{\bigcirc} \{\underline{\underline{\neg f, \neg b, \neg g}}, w, \neg r\}$$
$$\{\neg r, \underline{\underline{\neg[bgw]^*}} r\} \overset{b}{\bigcirc} \{\underline{\underline{\neg f, \neg b, \neg g}}, \neg w, \neg r\}$$
$$\{\neg r, \underline{\underline{[bgw]^*}} r\} \overset{w}{\bigcirc} \{\underline{\underline{\neg f, \neg b, \neg g}}, \neg w, \neg r\}$$
$$\{\neg r, \underline{\underline{\neg[bgw]^*}} r\} \overset{w}{\bigcirc} \{\underline{\underline{\neg f, \neg b, \neg g}}, \neg w, \neg r\}$$

システムの**抽象状態**はコレクタステップと抽象ヒープの対である．

3.5 抽象遷移

抽象モデル検査を行うためには，抽象状態間の妥当な抽象遷移を定義しなければならない．

我々が使ったモデル検査器では同一のコレクタステップの抽象ヒープはすべて合併させ，1 つのコレクタステップが 1 つの抽象ヒープのみを持つようにした（**図 2**）．

コレクタステップを変化させないミューテータやコレクタによる操作の抽象化は 2.3 節の方針に従った抽象セルの追加として定義する．

コレクタステップを変えるコレクタ遷移の抽象化は自明ではない．この遷移を抽象化するため，フィルタという手続きを導入した．**フィルタ**はコレクタ遷移が満たすべき条件に従って，抽象セルを抽象ヒープから取り除く手続きである．それらを**表 3** に示しておく．ステップ遷移に対応する抽象遷移は，たとえば，mark から append へコレクタステップが変化する場合，mark に付随した抽象ヒープからフィルタを使って減らした抽象セルの集合を append に付随した抽象ヒープに付け加える．この抽象遷移が妥当であるのは，コレクタステップが変化するための条件から簡単に分かる．しかし，このままでは抽象モデル検査したときにエラー状態に到達してしまう．

我々の抽象モデル検査では，このように得られた抽象遷移から 2.3 節で定義した矛盾抽象セルを除去し，より木目の細かい抽象遷移を用いた．この抽象遷移を用いることによって，エラー状態に到達しないことが確かめられた．なお，抽象セルの集合から辺を計算する E_N^i の i の値は 1 を用いた．

3.6 スナップショットの場合

オンザフライの場合，上述した抽象化で抽象モデル検査によって，エラー状態に到達しないことを示せたが，このままではスナップショットの場合はうまくいかなかった．そこで我々は，「白色セルがある灰色セルから白色セルの連鎖を通って到達可能かどうかのフラグ」というものを考えた[13]．このフラグも，簡単に正則表現 w^*g として表すことができ，D_B にこれを追加するだけでよい．ただし，抽象遷移もこの正則表現について修正する必要がある．

このようにどのような抽象化を行うかということと，どの正則表現を選択するかがほぼ同値になり，さまざまな抽象化をテストすることが容易になっている．

3.7 結　　果

我々は並列ごみ集めに特化した抽象モデル検査のプログラムをスクリプト言語 Python[11] を用いて新たに作り検証を行った．抽象モデル検査の結果は

表 4 結果
Table 4 Results.

	繰返し数	抽象セルの数			
		shade	mark	append	unmark
オンザフライ	6	40	69	47	77
オンザフライ（変種）	5	58	69	35	73
スナップショット	7	53	85	40	91
スナップショット（変種）	6	53	85	52	95

表 4 にまとめてある．我々は $D_F = \{f, b, g, w, r\}$, $D_B = \{r, [bgw]^* r, w^* g\}$ として両方のアルゴリズムの抽象モデル検査を行った．

このプログラムでは shade ステップの抽象ヒープをミューテータとコレクタの抽象遷移によって変化させ，次に mark ステップの抽象ヒープ，append ステップの抽象ヒープ，unmark ステップの抽象ヒープと順に同じように変化させた後，shade ステップに戻って繰り返すというループを持っている（図 2 を参照）．表 4 の繰返し数はすべてのステップの抽象ヒープが飽和状態になるまでにかかったループの繰返し数を表している．また，文献 13) で発見されたそれぞれの変種についても抽象モデル検査によって，エラー状態に到達しないことを検証できた．

4. 結　　論

本稿ではリンク構造を正則表現を用いて抽象化する方法を提案した．この方法で得られる抽象化の大きさはもとのリンク構造の大きさに依存しないことが特徴である．このことにより，リンク構造を持つシステムをリンク構造の大きさにかかわらずに抽象モデル検査することが可能となる．実際に本稿では，並列ごみ集めの複数のアルゴリズムの安全性を抽象モデル検査によって検証することに成功した．本稿で扱ったごみ集めのモデルでは各セルがポインタを 1 つしか持たなかったが，複数のポインタを持つモデルもまったく同様に抽象モデル検査可能である．

本稿で提案した方法のもう 1 つの特徴は，正則表現の選び方によってさまざまな抽象化が可能になる点である．抽象モデル検査によってエラー状態に到達しないことを示すためには，システムをどのように抽象化するかが最も肝要である．正則表現を選び直すことによってさまざまな抽象モデル検査を簡単に実行できることは大きな利点であろう．

なお，抽象遷移の妥当性も厳密に証明する必要がある．これは PVS や HOL などの定理証明システムを使って行うべきである．

謝辞 本研究は，文部省科学研究費補助金・基盤研究（B）（2）「抽象モデル検査とその応用」（11480062）の援助を受けている．

参 考 文 献

1) Bruns, G.: *Distributed Systems Analysis with CCS*, Prentice Hall (1997).
2) Clarke, E.M., Grumberg, O. and Long, D.E.: Model checking and abstraction, *ACM Trans. Prog. Lang. Syst.*, Vol.16, No.5, pp.1512–1542 (1994).
3) Dijkstra, E.W., Lamport, L., Martin, A., Scholten, C. and Steffens, E.: On-the-Fly Garbage Collection: An Exercise in Cooperation, *Comm. ACM*, Vol.21, No.11, pp.966–975 (1978).
4) Havelund, K.: Mechanical Verification of a Garbage Collector, *4th International Workshop on Formal Methods for Parallel Programming: Theory and Applications* (*FMTPTA '99*), pp.1258–1283 (1999).
5) Long, D.E.: Model Checking, Abstraction, and Compositional Reasoning, Ph.D. Thesis, Carnegie Mellon University (1993).
6) Müller, O. and Nipkow, T.: Combining Model Checking and Deduction for I/O-Automata, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, Vol.1019, pp.1–16 (1995).
7) Pnueli, A.: Deductive vs. Model-Theoretic Approaches to Formal Verification, *Automated Deduction, CADE-15*, Lecture Notes in Artificial Intelligence, Vol.1421, pp.301–301 (1998).
8) Sagiv, M., Reps, T. and Wilhelm, R.: Solving Shape-Analysis Problems in Languages with Destructive Updating, *POPL'96: The 23th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp.16–31 (1996).
9) Schmidt, D. and Steffen, B.: Program Analysis *as* Model Checking of Abstract Interpretations, *Static Analysis*, Lecture Notes in Computer Science, Vol.1503, pp.351–380 (1998).
10) Schmidt, D.A.: Data-flow Analysis is Model Checking of Abstract Interpretations, *POPL'98:*

The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp.38–48 (1998).
11) van Rossum, G.: Python Tutorial (1998). `http://www.python.org/doc/tut/tut.html`.
12) Yuasa, T.: Real-Time Garbage Collection on General-Purpose Machines, *Journal of Systems and Software*, Vol.11 (1990).
13) 高橋孝一，萩谷昌己：抽象モデル検査による並列ごみ集めの検証，第 2 回プログラミングおよび応用のシステムに関するワークショップ SPA 99 (1999).
14) 湯浅太一：実時間ごみ集め，情報処理学会論文誌，Vol.35, No.11, pp.1006–1013 (1994).

(平成 12 年 5 月 26 日受付)
(平成 12 年 9 月 8 日採録)

高橋　孝一

昭和 38 年生．昭和 63 年名古屋大学大学院理学科数学専攻修士課程修了．同年電子技術総合研究所入所．使用記述言語，証明支援系のユーザインタフェース，形式的証明技術等の研究に従事．

萩谷　昌己（正会員）

昭和 32 年生．昭和 57 年東京大学大学院理学系研究科情報科学専攻修士課程修了．京都大学数理解析研究所を経て，現在，東京大学大学院理学系研究科教授（情報科学専攻）．基本的に，演繹的推論を計算機上に実装することに興味を持っている．また，最近では，生命情報関連の研究（特に，分子計算）も行っている．