情報処理学会論文誌　数理モデル化と応用　Vol. 3　No. 1　73–86 (Jan. 2010)

# 物流システムに対する Ambient Logic モデル検査システム

加藤　暢[†1]　樋口昌宏[†1]　植田直人[†2]

本論文では物流システムの満たすべき性質の形式的な記述体系とそのモデル検査について述べる．物流の世界では，貨物の流通量の増加にともないコンテナ管理の重要性が高まっている．我々は Ambient Calculus による物流システムの記述と，実際の物流がその記述どおりに行われているかを監視するシステムを開発した．この監視システムによる物流管理の正しさを確認するためには，書類から生成されたプロセス式そのものの正当性を確認する必要がある．しかし，プロセス式生成システムで生成されるプロセス式は一般に複雑なものになり，さらに非決定的な動作も多くこの確認作業を人手で行うのは困難であると考えられる．本論文では，この課題を解決するために，物流システムに要求される，“いつか必ず貨物は目的地に輸送される”というような性質を Ambient Logic の論理式で表現し，その性質をプロセス式がすべて満たしていることを形式的に検査する手法を提案する．また，提案手法に基づくモデル検査システムを構築し検証実験を行った結果についても述べる．

# An Ambient Logic Model Checking System for Freight Systems

Toru Kato,[†1] Masahiro Higuchi[†1] and Naoto Ueda[†2]

We present a variant of Ambient Logic to describe desired properties of freight systems and a model checking algorithm for the logic. In the field of distribution, the increase in cargo handling errors is a serious problem as the amount of freight circulation increases. We study a freight inspection system based on Ambient Calculus and we have already built a prototype system. The system can derive process formulas from several freight documents. The system also checks that the actual freight movement conforms the derived formula. To show the correctness of freights transportation, we have to ensure the validity of process formula. We provide an Ambient Logic to present desired properties of freight systems, such as “every cargo will be eventually transported to its destination”. We also present an algorithm to show process formula satisfy Ambient Logic formula by exploring state transition graph. We conducted a verification experiment using the model checking system.

## 1. はじめに

近年，物流の世界では，貨物の流通量の増加にともなうコンテナ管理の重要性が高まっており，コンテナをどのように管理していくかについてさまざまな方法が模索されている．現状では国土交通省が IC タグで貨物をトレースする実験を 2007 年から 2008 年にかけて行った[8]．また国土交通省と米国土安全保障省が共同して，GPS による測位機能を持つ IC タグを使った船荷追跡を行う実証実験も行われている[9]．しかしこれらの方法では，貨物の通った経路のトレースはできるが，どのコンテナがどの船に積まれ，積み下ろされるか，誤った積み込みや積み降ろしが生じた際に警報を発する等の監視は想定していない．

物流はさまざまな貿易書類に基づいて行われており，そのため IC タグ等で取得した情報を，書類と照らし合わせて管理することが考えられる．しかし書類だけでは形式的かつ自動的な監視を行うことができないので，書類に基づき物流システムを形式的な取扱いが可能なモデルで表現することを考えた．我々は，この形式的なモデルに Ambient Calculus を利用し，Ambient Calculus による物流システムの記述と，実際の物流がその記述どおりに行われているかを監視するシステムの研究を行っている．

すでに送り状，B/L Instructions，Container Packing List[7] といった実際に貿易で使われる書類をもとに自動的にプロセス式の生成を行い，RFID で検知した物の移動と，プロセス式の遷移とを関連付けることで記述どおりの貨物輸送が行われていることを監視するシステムを構築している[11]．この監視システムでは，実際の貨物等の動作が正しいかどうかを，貿易書類から生成したプロセス式と照らし合わせることで確認する．しかしこのプロセス式が，物流システムを正しくモデル化できているということは示されていない．監視システムによる物流管理の正しさを示すためには，書類から生成されたプロセス式そのものの正当性を確認することが必要である．

しかし，プロセス式生成システムで生成されるプロセス式は一般に複雑なものになり，さ

†1 近畿大学理工学部情報学科
School of Science and Engineering Department of Informatics, Kinki University
†2 株式会社 JSOL
JSOL Corporation

らに非決定的な動作も多く，この確認作業を人手で行うのは困難であると考えられる．そこで本論文ではこの課題を解決するために，物流システムの満たすべき性質の記述体系を定式化し，さらにそのモデル検査について述べる．具体的には，物流システムに要求される，“いつか必ず貨物は目的地に輸送される” というような性質を様相論理の一種である Ambient Logic[3] のサブセットを用いて記述し，生成されたプロセス式が記述した性質すべてを満たしていることを形式的な手法で検査する方法を提案する．

我々はすでに文献 10) において，生成されたプロセス式のすべての実行可能性を網羅する単純な方法を用いたモデル検査システムに関する報告を行った．この報告において，そのような単純な方法によるモデル検査では簡単に状態空間爆発を起こし，多数のコンテナを運搬する物流システムをモデル化したプロセス式に対しては，実用的なモデル検査が不可能であることを実験により示した．本論文では，状態空間爆発問題を回避するための具体的な方法を提案し，これを用いたモデル検査システムと検証実験の結果について述べる．

## 2. 物流システムのモデル化のための Ambient Calculus

Ambient Calculus[2] は，Microsoft Research の Luca Cardelli と Andrew D. Gordon によって開発されたプロセス代数であり，動的な階層構造を持つシステムを形式的に記述することができる言語である．この特徴から物流システムの持つ階層構造を簡潔に表現することができる．さらに，様相論理の一種である Ambient Logic の公理系を用いて，プロセス式そのものが意図した性質を持っているかどうかを検証することが可能である．

### 2.1 Ambient Calculus の構文規則と遷移規則

本研究ではコンテナ海上輸送を行う物流システムを Ambient Calculus のプロセス式によってモデル化するが，モデル化するにあたって，文献 2) で定義されている Ambient Calculus の構文規則の一部分のみを利用する．本研究で使用する言語の構文規則，遷移規則を本節で定義する．

**定義 2.1（構文規則）**

| | | | |
|---|---|---|---|
| $n$ | names | $P, Q ::=$ | processes |
| $M, N ::=$ | capabilities | $0$ | inactivity |
| $in\ n$ | can enter $n$ | $P \mid Q$ | composition |
| $out\ n$ | can exit $n$ | $n[P]$ | ambient |
| $open\ n$ | can open $n$ | $M.P$ | action |
| $\epsilon$ | null | $(\nu n)P$ | restriction |
| $M.N$ | path | | |

□

定義 2.1 の ambient と capability が Ambient Calculus 特有のものである．ambient “$n[P]$” とは，$n$ という名前のついた何らかの “もの” を表し，括弧は “$n[m[\cdots l[P]]]$” のように任意の深さに階層化できる．これによりネットワークや物流システムの持つ階層構造を表現できる．また，capability によりその振舞いを表現する．例 2.4 でその一例を示す．

定義 2.1 で規定される Ambient Calculus の部分言語に対応し，遷移規則も文献 2) で定義されているものの一部のみを使用する．

**定義 2.2（遷移規則）**

$$n[in\ m.P \mid Q\ ] \mid m[\ R\ ] \rightarrow m[\ n[\ P \mid Q\ ] \mid R\ ] \quad (1)$$

$$m[n[\ out\ m.P \mid Q\ ] \mid R\ ] \rightarrow n[\ P \mid Q\ ] \mid m[\ R\ ] \quad (2)$$

$$open\ n.P \mid n[\ Q\ ] \rightarrow P \mid Q \quad (3)$$

□

定義 2.1 の composition $P \mid Q$ とは，プロセス $P$ と $Q$ が並列な位置に存在していることを示している．定義 2.2 の (1) において，capability “$in\ m$” により，それを囲んでいる ambient “$n[]$” が ambient “$m[]$” の中に移動すること，およびこの capability が遷移により消費され遷移後は消えることを表している．この遷移は ambient “$n[in\ m\cdots]$” と ambient “$m[]$” が並列な位置に存在するときのみ可能となる．そうでなければ，ambient “$n[]$” は ambient “$m[]$” が並列な位置に現れるまで待つ．この規則により，たとえば「コンテナはコンテナ船が港に到着してからコンテナ船に積載される」といったオブジェクト間の同期を簡潔に表現できる．

同様に (2) において，capability “$out\ m$” は，それを囲んでいる ambient が ambient “$m[]$” の中から外へ移動できることを，(3) において，capability “$open\ n$” は，それと並列な位置に存在する ambient “$n[Q]$” を消滅させ，その代わりに “$Q$” が並列な位置に現れるという動作をそれぞれ表す．

Ambient Calculus の本来の構文規則[2] には，$x$（変数），$(x)$（入力動作），$\langle M \rangle$（出力動作），$!P$（複製）等，定義 2.1 では省略した規則が含まれる．これらを省略し，Ambient Calculus の部分言語を使用する理由について述べる．本論文においてモデル化の対象となるものは，海，港，船，コンテナヤード，コンテナの 5 種類のみとする．これらの物を，一意の名前を持った ambient で表現する．また，これらの物の性質に関しては，移動のみをモデル化するため，物と物との間の通信動作を表現する入出力は不必要である．物の移動に

関しては，船やコンテナの目的地までの有限回の移動をモデル化するため，再帰は不必要である．現実の物流システムにおいて，輸送の途中で物が増えることはありえないので，複製は不必要である．Ambient を消滅させる *open* は，物の移動を制御するために必要となる．

以上の分析より，本研究で使用する Ambient Calculus の構文規則を定義 2.1 のように与えた．

文献 2) に準じ，$M.0$ を $M$，$n[0]$ を $n[\,]$ と略記する．演算子間の優先順位を $|<\nu<.$ とする．プロセス間の構造合同を以下のように定義する．

**定義 2.3（構造合同）**

$P \equiv P$　　$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$

$Q \equiv P \Rightarrow P \equiv Q$　　$n \notin fn(P) \Rightarrow (\nu n)(P|Q) \equiv P|(\nu n)Q$

$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$　　$n \neq m \Rightarrow (\nu n)m[P] \equiv m[(\nu n)P]$

$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$　　$P|0 \equiv P$

$P \equiv Q \Rightarrow P|R \equiv Q|R$　　$(\nu n)0 \equiv 0$

$P \equiv Q \Rightarrow M[P] \equiv M[Q]$　　$\epsilon.P \equiv P$

$P \equiv Q \Rightarrow M.P \equiv M.Q$　　$(M.M').P \equiv M.M'.P$

$P|Q \equiv Q|P$

$(P|Q)|R \equiv P|(Q|R)$

□

**例 2.4**　遷移規則に基づく式の遷移を，コンテナ船（$SHIP$）の中に存在しているコンテナ（$CO$）が，船の中から出ていき，さらにコンテナヤード（$CY$）の中に入ることを記述した式を例に説明する．

$$SHIP[\,CO[\,out\ SHIP.in\ CY\,]\,] \mid CY[\,] \tag{4}$$
$$\xrightarrow{out\ SHIP} SHIP[\,] \mid CY[\,] \mid CO[\,in\ CY\,] \tag{5}$$
$$\xrightarrow{in\ CY} SHIP[\,] \mid CY[\,CO[\,]\,]. \tag{6}$$

$$PORT[SHIP[open\ ctrl.out\ PORT|ctrl[\,]]] \tag{7}$$
$$\xrightarrow{open\ ctrl} PORT[\,SHIP[\,out\ PORT\,]\,]. \tag{8}$$

*out* は ambient を今いる ambient の外へ移動させる能力，*in* は ambient を他の ambient の中へ移動させる能力を表現している．式 (4) では $SHIP$ の中に $CO$ があり，最初の遷移が行われた式 (5) では $SHIP$ の中にあった $CO$ が $SHIP$ の外へと移動し，次の遷移が行われた式 (6) では，$CO$ が $CY$ の中へと移動した状態になっている．他の動作として，ambient の境界を消滅させる *open* がある．式 (7) において，$SHIP$ の中の *open ctrl* は，$SHIP$ が港（$PORT$）の外に移動するのを妨害しており，この *open ctrl* の並列な位置に *ctrl*[ ] が存在するので，遷移が行われた後の式 (8) では *open ctrl* が消費され，$SHIP$ が出航できる状態になっている．上記の式の中で，$SHIP$ はコンテナ船そのものを，$CO$ はコンテナそのものを，$CY$ はコンテナヤードそのものをそれぞれ表す ambient である．そして，それぞれの親子関係は，式 (4) ではコンテナ船とコンテナヤードが並列に存在し，コンテナ船の中にコンテナが入っていることを表している．このように，Ambient Calculus を用いることにより記述対象の階層構造を正確に，かつ直観的に表現することができる．　□

これ以降，コンテナやコンテナ船等具体的な物を表す ambient に対して，“物を表現する ambient” という表現を使う．また，*ctrl* のように物を表現する ambient の移動を制御している ambient に対しては，制御用 ambient という表現を使う．

### 2.2　Ambient Calculus を用いたモデル化の利点

物流システムのモデル化に，Ambient Calculus を用いることには次の利点がある．まず，プロセス代数であることから，構文規則，遷移規則が簡潔かつ厳密に定義されている．したがって，物の動作に関する性質を簡潔に，正確に表現できる．たとえば，例 2.4 で示したとおり，ambient は実体のある物を直接表すことが可能であることから，実際の物の動きと式の動きを簡単に関連付けることが可能である．また *composition* により，並列に式を書き連ねていくことが可能である．よって，運ぶ対象が増えた場合でも，容易にシステムを拡大することが可能である．さらに，Ambient Calculus に対応した様相論理の一種である Ambient Logic を用いることで，モデル化したプロセス式が本当に意図した性質を満たしているかを検査することが可能になる．

以上のように，物流システムの構造や貨物の動きを Ambient Calculus で表現することには多くの利点がある．

## 3.　Ambient Calculus を用いた物流監視システム

我々は Ambient Calculus を用いて，貨物の運送が輸送計画に沿って行われているかどうかを監視するシステムの開発を行っている[11]．監視システムの目的は以下のとおりである．

- 荷物を収容したコンテナや，コンテナを積んだコンテナ船が正しく移動することを確か

める．

- 正しくない移動を行った場合に知らせる．
- 現在のコンテナやコンテナ船の位置を把握する．

それぞれの目的を達成するために以下の実装を行った．

- 貿易書類からプロセス式を自動生成するシステム
- 物を表現する ambient ごとの分散処理
- 物の動きとプロセス式遷移との関連付け
- 上記関連付けにともなう動作エラーの警告

本章では，この物流監視システムについて述べる．

### 3.1　プロセス式生成システム

物流システムを表現するプロセス式を自動生成するために，どの船でどこの港からどこの港まで貨物を輸送するかを記述している送り状，コンテナの情報を記述している B/L Instructions と Container Packing List という 3 種類の貿易書類と，船の航路表を用いる．プロセス式は物を表現する ambient ごとに分かれて生成される．まず送り状から荷物を積み込む港と積み下ろす港の情報とコンテナ船の名前を取得する．B/L Instructions と Container Packing List からコンテナ情報を取得する．物を表現する ambient に関する必要な情報を読み込んだ後，「すべてのコンテナを積み込んでから船が出港する」等の制約を表現するための制御用 ambient を追加する．以上より，**図 1** のような物流を記述した，Ambient Calculus 式が生成される．図 1 は式全体を表示しているが，実際に物流監視を行うときは，物を表現する ambient ごとにプロセス式を分散処理させることで監視を行う．またプロセス式生成システムでは，個々の物流を記述対象としているため，生成されたプロセス式には再帰は含まれない．

ところで，図 1 中の “$checkin(SHIP\_v1)\ CY$” は，$in\ SHIP\_v1.out\ SHIP\_v1.in\ CY$ の略記，“$checkout(load\_v1\_TOKYO)\ CY$” は，$in\ load\_v1\_TOKYO.out\ load\_v1\_TOKYO.out\ CY$ の略記である．前者は港 $TOKYO$ に船 $SHIP\_v1$ がついたことをコンテナに知らせるために $CY$ に入る制御用 ambient $load\_v1\_TOKYO$ の動作を表している．後者は，その制御用 ambient がコンテナヤード $CY$ に入ってきたことを知った後にコンテナ $CO\_a1$ が $CY$ から出るという動作を表している．図 1 のプロセス式はコンテナを 1 個 $TOKYO$ のコンテナヤードから $KOBE$ のコンテナヤードへ運ぶ物流を表しており，コンテナを増やす場合は，下線を引いた部分がその数に応じて増加する．

なお，Ambient Calculus では，構文上 name に現れる大文字と小文字の区別に意味はない．しかし本論文で提案するプロセス式生成システムは，物を表現する ambient には必ず大文字で始まる name をつけ，制御用 ambient には小文字で始まる name をつけるという制約を導入する．これは，次の 3.2 節で述べるように，物流システムを監視する際にプロセス式を遷移させるタイミングが，物を表現する ambient と制御用 ambient では異なるからである．

$$
\begin{array}{l}
SEA[\\
\quad KOBE[uload\_v1\_KOBE[in\ SHIP\_v1] \mid CY[]]\\
\quad \mid SHIP\_v1[in\ TOKYO.\underline{open\ lcomp\_a1}\\
\qquad\qquad .out\ TOKYO.in\ KOBE]\\
\quad \mid TOKYO[\\
\qquad load\_v1\_TOKYO[checkin(SHIP\_v1)\ CY]\\
\qquad \mid CY[\underline{CO\_a1}[\\
\qquad\qquad \underline{checkout(load\_v1)CY.in\ SHIP\_v1.lcomp\_a1[out\ CO\_a1]}\\
\qquad\qquad \underline{\mid checkout(uload\_v1\_KOBE)SHIP\_v1.in\ CY}]\\
\qquad ]\\
\quad ]\\
].
\end{array}
$$

**図 1**　物流システムを記述したプロセス式
Fig. 1　A formula for a freight system.

### 3.2　物流と式の遷移との関連付け

監視システム内では，capability を実行するタイミングが物を表す ambient と制御用 ambient では異なる．監視対象の動作を表す物を表す ambient では，実際の物が動いたことを確認した後に，その動作に対応する capability が実行可能な場合のみ遷移が行われる．そのような capability が存在しない場合は，物の移動が本来の物流計画に沿っていないことになり警告を発する．一方，制御用 ambient は遷移可能になったときにただちに遷移させる．そのため物を表す ambient と制御用 ambient の命名規則に区別を与えた．

物を表現する ambient の遷移と現実の物の移動を関連付けるには，RFID を用いる．まず RF タグをコンテナやコンテナ船に取り付ける．各タグごとにその物を表現する ambient のプロセス式が書き込まれている．そして物が出入りする所に設置されたリーダ/ライタが，物の移動時に RF タグを感知し移動が可能かを監視する．可能であれば RF タグのプロセス式を書き換え，不可能であれば警告を行う．また実際に関連付けを行うには，プロセス式を解釈し，遷移させる処理系が必要になるが，この処理系は HORB を用いてすでに実装されている[6]．このように，実際の物とプロセス式の関連付けを行うことで，物の流れを監視

することができる．

## 4. 物流システムのための Ambient Logic

プロセス式生成システムにより生成された式が物流システムの所期の性質を満たしているか検証することを考える．このことを検証するために，様相論理の一種である Ambient Logic[3] を用いることが考えられる．しかし，Ambient Logic に対応した検証系の構築は容易ではないので，本研究では物流システムのための限定的な Ambient Logic を導入する．

### 4.1 検証対象となる性質

物流システムのための限定的な Ambient Logic を導入するために，物流システムの持つ性質を考え，その性質を Ambient Logic で記述するのに必要になる演算子を考える．プロセス式の満たすべき性質として物流システムの所期の性質を満たしていることと，現実の世界では起こりえないことを記述していないことの 2 つをあげることができる．物流システムの所期の性質は，プロセス式を生成するために使用する貿易書類から読み取ることができる．本研究で対象とする上記のような性質を，定義 4.1 においてそれぞれ p1〜p3，s1〜s4 で定義する．

**定義 4.1（対象とする性質）**

p1 いつか必ず貨物（コンテナ）は目的地に輸送される．
p2 特定の場所以外での貨物（コンテナ）の積み下ろしは行われない（不正な貨物の移動の禁止）．
p3 コンテナ船には決められた貨物（コンテナ）が積み込まれる．
s1 物は移動過程で消えることはない．
s2 海，港，コンテナヤードは，移動することはない．
s3 コンテナ船は，海と港の間のみを移動する．
s4 コンテナに別のコンテナが入ることはない．　□

### 4.2 物流システムのための Ambient Logic

ここでは，定義 4.1 で示した 7 つの性質を表現するために必要になる様相記号のみを持つ，部分的な Ambient Logic を定義する．

**定義 4.2（Logical Formulas）**

| | |
|---|---|
| $\eta$ | names |
| $\mathcal{A}\ \mathcal{B}$ ::= | |
| $T$ | *true* |
| $\neg\mathcal{A}$ | *negation* |
| $\mathcal{A} \vee \mathcal{B}$ | *disjunction* |
| $\mathcal{A} \mid \mathcal{B}$ | *composition* |
| $\eta[\mathcal{A}]$ | *location* |
| $\Diamond\mathcal{A}$ | *sometime modality* |
| $\blacklozenge\mathcal{A}$ | *somewhere modality* [⋆1] |

□

定義 4.2 において，*composition*，*location*，*somewhere modality* 以外は通常の時相論理で用いられる論理記号である．これら 3 つを含め，Ambient Logic では 17 個の独自の論理記号，様相記号が導入されている[3]．本研究ではそのうちこれら 3 つのみを通常の時相論理に追加した部分的な Ambient Logic を用いる．この論理の意味は，定義 4.3 の規則 Satisfaction により与えられる．

**定義 4.3（Satisfaction）**

| | |
|---|---|
| $\Pi$ | *sort of processes* |
| $\Phi$ | *sort of formulas* |
| $\Lambda$ | *sort of names* |
| $P \downarrow P'$ | iff $\exists n, P''.P \equiv n[P'] \mid P''$ |
| $\downarrow^*$ | reflexive and transitive closure of $\downarrow$ |
| $\forall P \in \Pi$ | $P \vDash T$ |
| $\forall P \in \Pi, \mathcal{A} \in \Phi$ | $P \vDash \neg\mathcal{A} \triangleq \neg(P \vDash \mathcal{A})$ |
| $\forall P \in \Pi, \mathcal{A}, \mathcal{B} \in \Phi$ | $P \vDash \mathcal{A} \vee \mathcal{B} \triangleq P \vDash \mathcal{A} \vee P \vDash \mathcal{B}$ |
| $\forall P \in \Pi, \mathcal{A}, \mathcal{B} \in \Phi$ | $P \vDash \mathcal{A} \mid \mathcal{B} \triangleq \exists P', P'' \in \Pi.P \equiv P' \mid P'' \wedge P' \vDash \mathcal{A} \wedge P'' \vDash \mathcal{B}$ |
| $\forall P \in \Pi, n \in \Lambda, \mathcal{A} \in \Phi$ | $P \vDash n[\mathcal{A}] \triangleq \exists P' \in \Pi.P \equiv n[P'] \wedge P' \vDash \mathcal{A}$ |
| $\forall P \in \Pi, \mathcal{A} \in \Phi$ | $P \vDash \Diamond\mathcal{A} \triangleq \exists P' \in \Pi.P \rightarrow^* P' \wedge P' \vDash \mathcal{A}$ |
| $\forall P \in \Pi, \mathcal{A} \in \Phi$ | $P \vDash \blacklozenge\mathcal{A} \triangleq \exists P' \in \Pi.P \downarrow^* P' \wedge P' \vDash \mathcal{A}$ |

□

直観的に $P \vDash \mathcal{A} \mid \mathcal{B}$ は，$P$ が $\mathcal{A}$ という論理式を満たすプロセスと，$\mathcal{B}$ という論理式を満たすプロセスの，*composition* 演算子による並列合成で構成されているプロセス式であることを示している．*location* $P \vDash n[\mathcal{A}]$ は，$P$ が $\mathcal{A}$ という論理式を満たすプロセスを持つ $n$ という ambient であることとを示している．*somewhere modality* $P \vDash \blacklozenge\mathcal{A}$ は，$P$ が持つ階層構造のどこかに $\mathcal{A}$ という論理式を満たすプロセスが存在することを示している．

⋆1 ◇ との識別を容易にするため，本論文では *somewhere modality* の記号に ♦ を用いる．

定義 4.4 では，定義 4.2 に示した論理演算子から派生する演算子をまとめる．さらに，定義 4.4 に基づく関係 $\models$ について，定義 4.5 にまとめる．

**定義 4.4（Derived Connectives）**

$F \triangleq \neg T \quad false$

$\mathcal{A} \wedge \mathcal{B} \triangleq \neg(\neg\mathcal{A} \vee \neg\mathcal{B}) \quad conjunction$

$\mathcal{A} \Rightarrow \mathcal{B} \triangleq \neg\mathcal{A} \vee \mathcal{B} \quad implication$

$\Box\mathcal{A} \triangleq \neg\Diamond\neg\mathcal{A} \quad everytime\ modality$ □

**定義 4.5（Expanded Definitions）**

$F \triangleq \neg T$

$\forall P \in \Pi, \mathcal{A}, \mathcal{B} : \Phi \quad P \models \mathcal{A} \wedge \mathcal{B}$
iff $P \models \mathcal{A} \wedge P \models \mathcal{B}$

$\forall P \in \Pi, \mathcal{A}, \mathcal{B} : \Phi \quad P \models \mathcal{A} \Rightarrow \mathcal{B}$
iff $P \models \mathcal{A} \Rightarrow P \models \mathcal{B}$

$\forall P \in \Pi, \mathcal{A} : \Phi \quad P \models \Box\mathcal{A}$ iff $\forall P' \in \Pi. P \rightarrow^* P' \Rightarrow P' \models \mathcal{A}$ □

このような限定的な Ambient Logic を用いる理由について述べる．定義 4.1 の p1 を表現するため等に，一般的な時制論理で用いられる “いつかは” を表現する *sometime modality* や “つねに” を表現する *everytime modality* が必要である．また Ambient Logic 特有の，プロセス式中のある ambient 内のプロセス式が，ある性質を満たしていることを表現する *location* や，プロセス式の部分式が，ある性質を満たしていることを表現する *somewhere modality* が必要である．文献 3) で導入された 17 個ある Ambient Logic 特有の演算子のうち *composition*，*location*，*somewhere modality* の 3 個を使うことで，以上のように示したい性質を記述することができ，これらに限定することでモデル検査システムの実装が比較的容易になる．

なお，本研究で定義している Ambient Calculus の subcalculus と限定的な Ambient Logic の関係は，定義 4.3 で定義された関係 $\models$ によって規定される．この関係は，定義 4.3 に示されているように，文献 3) で導入された 17 個の演算子の中で今回使用しなかった演算子と，文献 2) で規定されたプロセスの構文要素のうち，リプリケーション（複製）や入出力等今回使用しなかった構文要素にはまったく依存していない．したがって文献 3) で規定されているオリジナルの Ambient Calculus と Ambient Logic の関係を変えるものではない．

## 5. モデル検査システム

本研究では，4 章で示した公理系に基づくモデル検査システムを，Java 言語を用いて構築した．システムの規模は，クラス数 15，総行数 3,000 ほどである．本章では，このモデル検査システムについて詳述する．

### 5.1 検査システム

本検査システムでは，与えられたプロセス式に対して初めに**図 2** のようなプロセス式の遷移グラフを作成する．遷移グラフの各ノードは，ノードを識別するためのノード ID，遷移経路を知るための親ノード ID，そして**図 3** に示すようなプロセス式の構造を表す構文木を持っている．構文木の各ノードは，ambient の名前，各 ambient を識別するための ID，親子関係等を識別するための親 ambient の ID，*capability action* のリストを持っている．この構文木の *capability action* のリストと木の親子関係を見ることで，可能な遷移を見つけ，構文木を遷移させることで図 2 のような遷移グラフの作成を行う．可能な遷移が複数ある場合（Ship1 が先に Port に入る場合と Ship2 が先に Port に入る場合等）に枝分かれが生じる．この遷移グラフを深さ優先で探査し，各ノードの構文木に対して *location* や *somewhere modality* のような ambient の空間的な検査を木の親子関係を見ながら行い，*sometime modality* や *everytime modality* のような時間的な検査を経路を見ながら行う．

たとえばプロセス式生成システムにより生成されたプロセス式 (9) のモデル検査をするために遷移グラフを作る場合，まずプロセス式を構文解析し構文木を作る．この構文木に各種

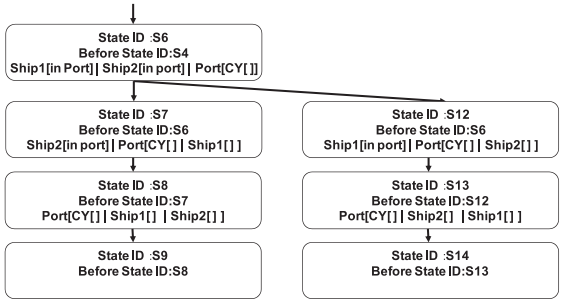


**図 2**　遷移グラフ
Fig. 2　A transition graph.

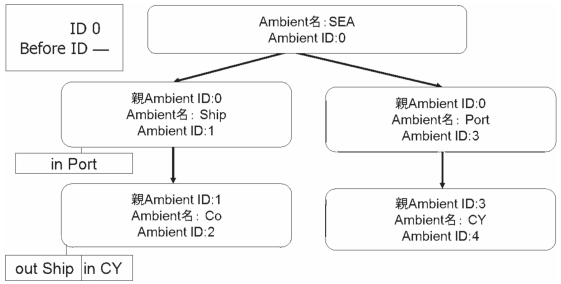


**図 3**　構文木
Fig. 3　A syntax tree.

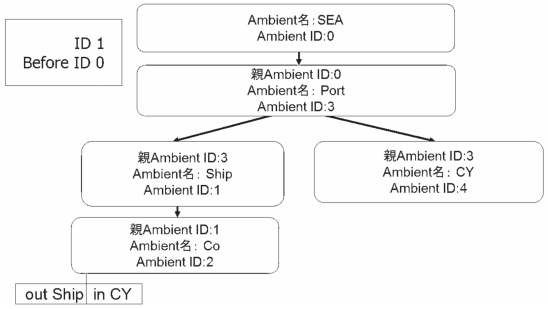


**図 4**　遷移後の構文木
Fig. 4　A syntax tree after a transition.

ノード情報を付加する．これが遷移グラフにおける初期ノード $S0$ になる．

$$SEA[\ SHIP[\ in\ PORT|\ CO[\ out\ SHIP\ .\ in\ CY\ ]\ ]\ |\ PORT[\ CY[\ ]]]. \tag{9}$$

構文木の各ノードが持つ *capability action* のリストから，そのノードで実行可能な遷移を見つけ，遷移後のノードを作成する．図 3 の場合，ambient ID 1 の *SHIP* の持つ *capability action* は *in PORT* なので，この ambient の兄弟に *PORT* という ambient が存在するか検査する．この例の場合は *PORT* があるので *capability action* を削除し，*SHIP* の親 ID を *PORT* の ID である 3 に変えることで，**図 4** のように遷移後の構文木に変える．

*location* や *somewhere modality* 等の空間的な検査は，指定された場所に ambient の階層構造があるかを構文木をたどることで検査することができる．たとえば，$P \vDash \blacklozenge CO[T]$ なら図 3 の構文木を探索し，*CO* という名前の ambient をこの構文木が持っているかどうかを調べる．この場合は *SHIP* の中に *CO* を見つけることができるのでこのノードで性質を満たしていることを示すことができる．このようにして作成した遷移グラフに対して，4 章で示した公理系に基づくモデル検査を行う．

### 5.2　検 査 内 容

定義 4.1 で示した p1〜s4 の 7 つの性質を表す論理式が，遷移グラフの各ノードの持つプロセス式によって満たされるかどうかを検査する．本節ではこれらの論理式について，輸出港 *PORT_A* から港 *PORT_C* を経由し輸入港 *PORT_B* にコンテナ *CO* を輸送する物流システムを例に説明する．

- いつか必ず *CO* は *PORT_B* に輸送される．

$$P \vDash \Box\Diamond\blacklozenge PORT\_B[CY[CO[T] \mid T] \mid T]. \tag{10}$$

式 (10) の $CY[CO[T] \mid T]$ の部分はコンテナヤード *CY* の中にコンテナ *CO* が存在しており，また $CO[T] \mid T$ でコンテナヤード *CY* の中に，さらにコンテナ *CO* 以外が存在してもよいことを示している．$\blacklozenge PORT\_B[CY[CO[T] \mid T] \mid T]$ は，プロセス式のどこかで $PORT\_B[CY[CO[T] \mid T] \mid T]$ という階層構造が存在することを示しており，この式に $\Box\Diamond$ をつけることで，どのような遷移が行われたとしてもどこかでその状態が必ず成り立つことを示している．p1 の性質は，コンテナが輸入港のコンテナヤードにあればよいので式 (10) で表すことができる．

- *PORT_A*，*PORT_B* 以外では *CO* の積み下しは行われない．

$$\begin{aligned} P \vDash \Box(&\neg\blacklozenge PORT\_A[SHIP[T] \mid T] \wedge \neg\blacklozenge PORT\_B[SHIP[T] \mid T] \\ &\Rightarrow \neg\blacklozenge(SHIP[T] \mid CO[T])). \end{aligned} \tag{11}$$

p2 の性質は，コンテナを積み込む直前または積み下した直後は $(SHIP[T] \mid CO[T])$ が成り立ち，積み込み，積み下ろしができるのは，それぞれ輸出港と輸入港だけなので “輸出港，輸入港以外ではコンテナの積み下しは行われない” といい換えることができ，式 (11) で表すことができる．

- *SHIP*（コンテナ船）には指定された *CO* のみが積み込まれる．

$$\begin{aligned} P \vDash \Box(&\neg\blacklozenge PORT\_A[\blacklozenge CO[T]] \wedge \neg\blacklozenge PORT\_B[\blacklozenge CO[T]] \\ &\Rightarrow \blacklozenge(SHIP[CO[T] \mid T])). \end{aligned} \tag{12}$$

p3 の性質は，輸出港か輸入港のどこかにコンテナがない場合には必ず指定されたコンテナ船にコンテナが積み込まれていることを要求しているので，式 (12) で表現できる．この 3 つの式で，物流システムの所期の性質を Ambient Logic で表現できる．

- つねに $PORT\_A$ は $SEA$（海）に存在する（プロセス式に含まれる各港ごとにこの性質を満たす）．
$$P \vDash \Box(SEA[PORT\_A[T] \mid T]). \tag{13}$$
- つねに $PORT\_A$ にはそれぞれ $CY$（コンテナヤード）が存在する（プロセス式に含まれる各港ごとにこの性質を満たす）．
$$P \vDash \Box\blacklozenge(PORT\_A[CY[T] \mid T]). \tag{14}$$
- $PORT\_A$，$PORT\_B$，$PORT\_C$ に寄港する輸送の場合，$SHIP$ は $SEA$（海）の上か航路上の港に存在する．
$$P \vDash \Box(SEA[SHIP[T] \mid T] \vee \blacklozenge PORT\_A[SHIP[T] \mid T] \vee \blacklozenge PORT\_B[SHIP[T] \mid T] \vee \blacklozenge PORT\_C[SHIP[T] \mid T]). \tag{15}$$
- つねに $CO$ は存在する．
$$P \vDash \Box(\blacklozenge CO[T] \mid T). \tag{16}$$
- コンテナに別のコンテナは入らない．
$$P \vDash \neg\Box(\blacklozenge CO\_a1[CO[T] \mid T] \mid T). \tag{17}$$
$$P \vDash \neg\Box(\blacklozenge CO[CO\_a1[T] \mid T] \mid T). \tag{18}$$

物理的に起こりえない性質である s1 は，海，港は式 (13)，コンテナヤードは式 (14)，コンテナ船は式 (15)，コンテナは式 (16) でそれぞれつねに存在することを表現できる．式 (13) と式 (14) で階層構造がつねに固定されており，かつ式 (13) で海とパラレルに存在するものはないと表現しているので，s2 の性質がいえる．s3 は，コンテナ船が，海上か，輸出港，輸入港，経由港のいずれかに存在するということなので式 (15) で表現できる．コンテナが複数ある場合には，式 (17)，(18) でそれぞれコンテナが互いの中に入らない s4 の性質が表現できる．

### 5.3　検査方法の実装

本研究では，定義 4.2，4.4 で示した各演算子の評価を実現するメソッドを実装した．and，or，not 等一般の論理演算子は，対応する Java 言語の boolean 型の演算子を用いて単純に実装できる．Ambient Logic 特有の論理演算子の実装について述べる．以下で，$s$ を検査対象の状態（ノード），$P$ を状態 $s$ におけるプロセス式とする．

**Composition 演算子メソッド**

定義 4.3 の $P \vDash \mathcal{A} \mid \mathcal{B}$ の規則に従い，$P$ を $P \equiv P' \mid P''$ という形に分割できるプロセス式 $P'$，$P''$ の組を探す．その組に対し $P' \vDash \mathcal{A}$ と $P'' \vDash \mathcal{B}$ の真偽の確認作業を行い，それらの両方が真であれば真を返す．偽であれば，$P$ を $P \equiv P' \mid P''$ という形に分割できるプロセス式 $P'$，$P''$ の別の組を探し，見つからなければ偽を返す．見つかれば同様の操作を行う．

**Location 演算子メソッド**

$P \vDash n[\mathcal{A}]$ の規則に従い，あるプロセス式 $P'$ に対して $P$ が $n[P']$ の形をしていることを確認し，さらに $P' \vDash \mathcal{A}$ の真偽の確認作業を行う．

**Sometime 演算子メソッド**

$P \vDash \Diamond\mathcal{A}$ の規則に従い，$P \vDash \mathcal{A}$ を確認する．真ならば真を返す．偽の場合，$s$ において可能な遷移が存在すれば未定を返し，存在しなければ偽を返す（Sometime 演算子は，$P \vDash \mathcal{A}$ が現時点で偽であっても，終了状態までのどこかの時点で真になれば真となる．また，本研究で対象としているプロセス式には再帰や複製が含まれないことから遷移グラフに閉路が存在しないため，グラフを生成しながら真偽を判定する検査は，有限回の手続きで終了する）．

**Somewhere 演算子メソッド**

$P \vDash \blacklozenge\mathcal{A}$ の規則に従い，$P$ に現れるすべての ambient（プロセス $P'$）に対して，1 つずつ $P' \vDash \mathcal{A}$ を確認する．1 つでも真であることが確認できれば確認作業を中止し，真を返す．すべてが偽であれば偽を返す．

**Everytime 演算子メソッド**[⋆1]

定義 4.5 の $P \vDash \Box\mathcal{A}$ の規則に従い，$P \vDash \mathcal{A}$ が偽ならば偽を返す（Everytime 演算子は，終了状態までのどこかの時点で $P \vDash \mathcal{A}$ が偽であれば偽になるので，現時点で $P \vDash \mathcal{A}$ の偽が確認できた段階で偽とする）．

Sometime 演算子では，現時点で偽であっても，将来真になる可能性があるので，真，偽以外の真理値「未定」を返す必要がある．そのため，and，or も含め，すべての演算子メソッドは真，偽，未定のいずれかを引数とし，返り値として返すよう実装している．式 (10)〜(18) の真偽は，これらの演算子用メソッドを組み合わせて判定される．複数の演算子の複雑な組合せの例として，式 (12) がどのように判定されるかを，例 5.1 に示す．

⋆1 □ の実装は必ず次も必要ではないが，頻繁に使用するので効率化のために実装している．

**例 5.1**　式 (12) の右辺は，□(logic1∧logic2⇒logic3) の形をしているので，まず以下のように logic1，logic2，logic3 のそれぞれの真偽を個別に判定する．

```
logic1
 = not 演算子 (
     somewhere 演算子 (
       location 演算子 (
                港名，somewhere 演算子 (location 演算子 (コンテナ名，true))))).
```

logic2，logic3 も同様に判定する．最後に各結果を組み合わせ，下記のように判定する．

```
everytime 演算子 (imply 演算子 (and 演算子 (logic1, logic2),logic3)).
```

□

### 5.4　状態空間爆発問題と Partial Order Reduction

遷移グラフには，非決定的な動作が原因となる枝分かれが生じる．たとえば，式 (19) のようにコンテナヤードにある 2 つの $CO\_a1$，$CO\_a2$ を $SHIP$ に積み込むようなプロセス式の場合，このプロセス式からは $CO\_a1$ を先に運ぶ式 (20) と $CO\_a2$ を先に運ぶ式 (21) に遷移することができる．

$$SEA[\cdots CY[CO\_a1[out\ CY.in\ SHIP] \mid CO\_a2[out\ CY.in\ SHIP]] \mid SHIP[]\cdots]. \tag{19}$$

$$SEA[\cdots CY[CO\_a2[out\ CY.in\ SHIP]] \mid SHIP[] \mid CO\_a1[in\ SHIP]\cdots]. \tag{20}$$

$$SEA[\cdots CY[CO\_a1[out\ CY.in\ SHIP]] \mid SHIP[] \mid CO\_a2[in\ SHIP]\cdots]. \tag{21}$$

したがって，図 2 に示した 2 隻の船の入港順序の場合と同様の枝分かれが生じる．しかもコンテナ数は一般に数百個から数千個のオーダなので，その枝分かれ数は膨大になり，その枝分かれに沿った検証は現実には不可能である．実際，6 個のコンテナを輸送するプロセス式の場合でさえ状態空間爆発が発生し，数 GB のメモリを持つ計算機では検証不可能であった[10]．

このような問題を解決するために，partial order reduction と呼ばれる方法がある[1),5)]．文献 5) で述べられている方法は次のとおりである．遷移グラフにおいて各ノード $s$ からの遷移を，そのノードで実行可能な遷移の集合 $enabled(s)$ の部分集合 $ample(s)$ に制限することによって，遷移グラフの状態空間爆発を抑制することができる．partial order reduction において検証結果に影響を及ぼさないよう，$ample(s)$ は以下の条件を満たすように選ばなければならない．

**C0** $ample(s) = \emptyset$ のときまたそのときに限り $enabled(s) = \emptyset$.

**C1** $ample(s)$ 中の遷移と独立でない[⋆1]遷移は，$ample(s)$ 中のいずれかの遷移を実行してからでないと実行可能にならない．

**C2** $ample(s)$ 中の遷移によって，検証対象の論理式のすべての部分原子式の真偽が変化しない．

遷移グラフ中の各ノードに対して，最小もしくはそれに近い $ample(s)$ を効率的に求めることができれば検証時間の大幅な削減が期待される．しかしながら，最小の $ample(s)$ を正確に求めるためには全状態空間の探索が必要となるため，通常は何らかのヒューリスティクスに基づいて，なるべく小さい $ample$ 集合を求める方法がとられる．

なお，文献 5) ではさらに条件 C3 を満たす必要であるとされているが，C3 は遷移グラフにループが現れる場合の条件であり，本研究で扱うプロセス式では再帰や replication がないため本研究では考慮しない．

### 5.5　Ambient Calculus に対する Partial Order Reduction

状態 $s$ において可能な遷移の集合 $enabled(s)$ の中から，モデル検査に影響を及ぼさないように必要最小限の遷移の集合 $ample(s)$ を選ぶ際，5.4 節で述べた条件 C0，C1，C2 を満たしているかどうかを確認する．C0 と C2 の確認は容易であるが，C1 の確認は，一般には full state graph の全状態を検査する必要があり，数百〜数千のコンテナを扱う物流システムの検査を行う場合事実上不可能である．そこで本節では，Ambient Calculus のプロセス式の持つ性質に着目して，$enabled(s)$ とプロセス式の持つ情報のみから C1 を満たすような $ample(s)$ を決定できる方法を示す．

**定義 5.2（open 可能な ambient，不可能な ambient，illegal なプロセス式）**　小文字で始まるすべての制御用 ambient を *open* 可能な ambient，大文字で始まる物を表す ambient を *open* 不可能な ambient とする．*open* 不可能な任意の ambient $A[\,]$ に対し，もしあるプロセス式中に *open* $A$ という capabilty が出現する場合，このプロセス式を illegal なプロセス式と呼ぶ．□

**定義 5.3（遷移 $T$）**　すべての遷移の集合 $T$ を次のように定める．$T = T_{out} \cup T_{in} \cup T_{open} \cup$

⋆1 すべての状態 $s$ に対して，$\alpha$，$\beta \in enabled(s)$ ならば $\alpha(\beta(s)) = \beta(\alpha(s))$ が成り立つとき，$\alpha$，$\beta$ は独立と呼ばれる．

$T_{checkin} \cup T_{checkout}$．ただし，プロセス式全体の集合 $\Pi$ に対して，

$T_{in} = \{(n, in\ m) \mid n[in\ m.P_1 \mid P_2] \in \Pi\}$.

$T_{out} = \{(n, out\ m) \mid n[out\ m.P_1 \mid P_2] \in \Pi\}$.

$T_{open} = \{(open\ m) \mid n[open\ m.P_1 \mid P_2] \in \Pi\}$.

$T_{checkin} = \{(n, checkin(y)\ m) \mid n[checkin(y)\ m.P_1 \mid P_2] \in \Pi\}$.

$T_{checkout} = \{(n, checkout(y)\ m) \mid n[checkout(y)\ m.P_1 | P_2] \in \Pi\}$. □

C1 を満たすような $ample(s)$ を作るためには，$ample(s)$ 中の遷移と独立でない遷移をプロセス式の中から同定する必要がある．

**定義 5.4（関数 dependent）** 遷移 $t$ に対して $t$ と独立でない遷移の集合を返す関数 $dependent$ $(T \to 2^T)$ は次のように定めることができる．ただし，以下で $\mathcal{N}$ を名前の集合とする．

$dependent((n, in\ m)) = dependent((n, out\ m)) =$
$\{(n, in\ x) \mid x \in \mathcal{N}\} \cup \{(n, out\ x) \mid x \in \mathcal{N}\} \cup \{(n, checkin(y)\ x) \mid x, y \in \mathcal{N}\} \cup$
$\{(n, checkout(y)\ x) \mid x, y \in \mathcal{N}\} \cup \{(m, in\ x) \mid x \in \mathcal{N}\} \cup \{(m, out\ x) \mid x \in \mathcal{N}\} \cup$
$\{(m, checkin(y)\ x) \mid x, y \in \mathcal{N}\} \cup \{(m, checkout(y)\ x) \mid x, y \in \mathcal{N}\} \cup \{(open\ m)\}$
$\cup \{(open\ n)\}$.

$dependent((n, checkin(y)\ m))$
$= dependent((n, checkout(y)\ m)) =$
$dependent((n, in\ m)) \cup$
$\{(y, in\ x) \mid x \in \mathcal{N}\} \cup \{(y, out\ x) \mid x \in \mathcal{N}\} \cup \{(y, checkin(z)\ x) \mid x, z \in \mathcal{N}\} \cup$
$\{(y, checkout(z)\ x) \mid x, z \in \mathcal{N}\} \cup \{(open\ y)\}$.

$dependent((open\ m))) =$
$\{(m, in\ x) \mid x \in \mathcal{N}\} \cup \{(m, out\ x) \mid x \in \mathcal{N}\} \cup \{(m, checkin(y)\ x) \mid x, y \in \mathcal{N}\}$
$\cup \{(m, checkout(y)\ x) \mid x, y \in \mathcal{N}\} \cup \{(open\ m)\}$. □

遷移 $t$ に対し $dependent(t)$ が上式に限定される理由は，たとえば $t = (n, in\ m)$ と，$m$，$n$ に関係ない遷移 $t'$ がともに実行可能なとき，$t(t'(s)) = t'(t(s))$ となるからである．

**定義 5.5（検査用論理式集合 $\mathcal{F}$，状態論理式集合 $\mathcal{F}_{st}$）** モデル検査を実施するために，検査対象となるコンテナ輸送の所期の性質を表す Ambient Logic の論理式の集合 $\mathcal{F}$ を生成する．$\mathcal{F}$ は，5.2 節で説明した論理式 (10)〜(18) をテンプレートとして，各式の $SHIP$，$PORT$，$CO$ 部分を，検査対象のコンテナ輸送で用いられる船名，港名，コンテナ名に置き換えることで生成する．また，$\mathcal{F}$ 内の各論理式の，時相記号を含まない部分式の集合を状態論理式集合 $\mathcal{F}_{st}$ とする．

**定義 5.6（ample(s) 生成手順）** 状態 $s$ に対し $enabled(s)$ の要素が複数ある場合，C0，C1，C2 を満たすような $ample(s)$ が，極力単一の要素を持つ集合となるよう以下の手順で生成する．

(1)　$enabled(s)$ の中から遷移 $t$ を任意に選択し，遷移 $t$ の実行後の状態 $s'$ を求める．

(2)　定義 5.5 で定めた論理式の集合 $\mathcal{F}_{st}$ の中で，$s$ が満たす論理式の集合（$\mathcal{F}_{st}$ の部分集合）が，$\mathcal{F}_{st}$ の中で $s'$ の満たす論理式の集合と一致するかどうか確認する．もし一致すれば $t$ に対し (3) を実施する．もし一致しなければ，$enabled(s) = enabled(s)/\{t\}$ とし，この新たな $enabled(s)$ が空集合でなければ (1) に戻る．空集合ならば，C2 を満たすようには単一の要素を持つよう $ample(s)$ を生成できないので，$ample(s) =$ “最初に与えられた $enabled(s)$” とし，$ample(s)$ 生成手順を終了する．

(3)　状態 $s$ におけるプロセス式を $P$ とする．$P$ の中に現れるすべての遷移の集合を $T_P$ とする．集合 $T_P \cap dependent(t)$ が空集合の場合は $ample(s) = \{t\}$ とし，$ample(s)$ 生成手順を終了する．そうでなければ以下の手順で，$ample(s) = \{t\}$ が条件 C1 を満たすかどうか確認する．

**(a)** $T_P \cap dependent(t)$ 内の遷移の中で，$t$ が実行された後でないと実行可能にならないことが構文的に確認可能である遷移の集合を求める．この集合を $followers(t)$ とする．

**(b)** $followers(t) = T_P \cap dependent(t)$ の場合は $\{t\}$ が条件 C1 を満たすことが確認できたことになり，$ample(s) = \{t\}$ とし，$ample(s)$ 生成手順を終了する．

**(c)** $followers(t) \neq T_P \cap dependent(t)$ の場合は，$(T_P \cap dependent(t))/followers(t)$ 内の各要素 $t'$ に対し $t$ must happen before $t'$ が成り立つかどうかを確認する．これらすべての $t'$ に対して成り立てば $\{t\}$ が条件 C1 を満たすことになり，$ample(s) = \{t\}$ とし，$ample(s)$ 生成手順を終了する．そうでなければ $enabled(s) = enabled(s)/\{t\}$ とし，この新たな $enabled(s)$ が空集合でなければ (1) に戻る．空集合ならば，C1 を満たすようには単一の要素を持つよう $ample(s)$ を生成できないので，$ample(s) =$ “最初に与えられた $enabled(s)$” とし，$ample(s)$ 生成手順を終了する．must happen before の定義は定義 5.8 で与える． □

定義 5.6 の (1)〜(3) の手順を実行した後，$ample(s)$ が単一の要素を持つ集合 $\{t\}$ になっていれば，C1，C2 を満たすような最小の遷移集合が見つかったことになり，状態 $s$ からは

この遷移のみを枝として残した縮小された遷移グラフを用いてモデル検査を継続することができる．$ample(s)$ = “最初に与えられた $enabled(s)$” となっていれば，状態 $s$ において可能なすべての遷移を持つ遷移グラフを用いてモデル検査を継続する．

**例 5.7**　定義 5.6 で示した手順 (3) の (a)，(b) について例を示す．式 (22) は，複数のコンテナのうち $CO\_a1$ がコンテナ船に積み込まれ，積み込まれたことを船側が確認しようとしている状態を表している．

$$
\begin{array}{l}
SEA[ \\
\quad KOBE[\ uload\_v1\_KOBE[in\ SHIP\_v1] \mid CY[]] \\
\quad \mid TOKYO[ \\
\quad\quad \mid SHIP\_v1[ \\
\quad\quad\quad open\ lcomp\_a1.\cdots.open\ lcomp\_an \\
\quad\quad\quad .out\ TOKYO.in\ KOBE \\
\quad\quad\quad \mid CO\_a1[\ lcomp\_a1[out\ CO\_a1] \\
\quad\quad\quad\quad\quad \mid checkout(uload\_v1\_KOBE)SHIP\_v1.in\ CY] \\
\quad\quad\quad ] \\
\quad\quad \mid CY[load\_v1\_TOKYO[] \\
\quad\quad\quad\quad \mid CO\_a2[\cdots] \mid \cdots \mid CO\_an[]] \\
\quad\quad ] \\
\quad ]
\end{array}
\tag{22}
$$

・$t$ はすでに (2) によって条件 C2 を満たすことが確認されている．本研究で対象としているプロセス式の場合，$t$ は物を表す ambient の構造に変化を与えない，制御用 ambient 名に関する遷移に限定されている．

多数の枝分かれが発生する場面においては，このような $t$ は $(open\ \ lcomp\_a1)$ と $(lcomp\_a1, out\ CO\_a1)$ に限定される（添字の数字はコンテナ数分だけ増加する）．たとえば $(uload\_v1\_KOBE, in\ SHIP\_v1)$ という遷移は，$SHIP\_v1$ ambient が港に到着したことを確認するためのものであるが，この遷移が実行可能な場面で，ほかに実行可能な遷移がないことは構文的に確認できる．したがって，これら $(open\ lcomp\_a1)$ と $(lcomp\_a1, out\ CO\_a1)$ という遷移 $t$ に対してのみ $T_P \cap dependent(t)$ を考える．

・$t = (lcomp\_a1, out\ CO\_a1)$ の場合：$T_P \cap dependent(t)$ =$\{(open\ lcomp\_a1)$，$(CO\_a1, checkout(uload\_v1\_KOBE)\ SHIP\_v1)\}$ となる．$(open\ lcomp\_a1)$ が $t$ の実行後でないと実行できないことは，以下のように構文的に確認できる．$lcomp\_a1$ ambient の親 ambient である $CO\_a1$ と，$open\ lcomp\_a1$ capability を持つ ambient である $SHIP\_v1$ は $open$ 不可能な ambient であることから，$(open\ lcomp\_a1)$ が実行できる可能性があるのは，$lcomp\_a1$ ambient が $SHIP\_v1$ ambient の中に移動し，$(open\ lcomp\_a1)$ と兄弟の位置に来たとき，つまり $(lcomp\_a1, out\ co\_a1)$ が実行された後のみだということが構文的に確認できる．プロセス式 $P$ が illegal でないことより，以上のように $\{t\}$ が条件 C1 を満たしていることが保証される．

・以上が確認できた場合は，$ample(s) = \{t\}$ とし，$s'$ のみを遷移グラフの中の新たなノードとする．

$(CO\_a1, checkout(uload\_v1\_KOBE)\ SHIP\_v1)$ に対しては，これが $t$ の実行後にのみ実行可能になることの構文的な確認は困難である．しかし，定義 5.8 に沿って調べることで，全状態探査をすることなく確認できる．□

**定義 5.8（must happen before）**　状態 $s$ 中のプロセス式 $P$ の任意の遷移 $t$ と，$t' \in (T_P \cap dependent(t))/followers(t)$ に対し，以下のすべての条件が満たされるとき，$t$ must happen before $t'$ を真とする．

(1)　$t' \notin enabled(s)$

(2)　$t' = (n, in\ m)$ または $(n, chechout(m)\ y)$ の形をしている場合：

**(a)** これらの遷移の capability を $act$ とする．これらの $act$ を持っている ambient $n$ の中に，$n[act \mid act'']$ となるような他の capability $act''$ が存在しない．存在すれば $t$ とは無関係に $t'$ が実行可能になる可能性があるため，偽とする．

**(b)** $n[M.act]$ のように，$act$ の前に path $M$ がある場合，$M$ に $t$ の capability または，$followers(t)$ の各遷移の持つ capablity が含まれていれば，$t$ must happen before $t'$ は真となる．そうでなければ，以下の手順に進む．

**(c)** $t'$ が状態 $s$ で実行できないのは，ambient $m$ が，ambient $n$ と並列な位置にいないためである．任意の action と target に対し，遷移 $(m, action\ target)$ が $enabled(s)$ ならば，その実行により $t$ とは無関係に $t'$ が実行可能になる可能性があるため，$t$ must happen before $t'$ を偽とする．そうでなければ，以下の手順に進む．

**(d)** $(m, action\ \ target)$ に対し，これを $t'$ に読みかえて，(1) に戻る．ただし $followers(t)$ はそのまま同じものを用いる．

**(e)** 以上を，$t'$ の実行を妨害している他の遷移が $P$ 中に見付からなくなるまで繰り返す．

本研究で対象としているプロセス式には replication や再帰が含まれないこと，および条

件 ( 2-a ) により探査に枝分かれが生じないことより，遷移 $t$ に対する must happen before の検査はプロセス式の長さで終了する．□

本検査システムは，以上の手順で状態数を減らすよう実装されている．

### 5.6　対称性を利用した partial order reduction

コンテナ $CO\_a1$ と $CO\_a2$ の積込港と積降ろし港が同じ場合，それぞれのコンテナを表す ambient 名が異なるだけで，ambient 内の capability はまったく同じである．また，それぞれのコンテナに対応した制御用 ambient も同様である．このように本研究で扱っている物流システムは，コンテナの数は多いがそれらの多くは名前をつけ換えれば同じものになる，すなわち名前の対称性に由来する均質性が見られるという特徴を持つ．したがって，プロセス式の中のこのような対称性さえ確認できれば，これらの ambient の非決定的な動作による違いから生じる複数の遷移は，どれか 1 つを代表として残し，ほかの遷移は遷移グラフから削除してもモデル検査には影響しない．つまり，プロセス式中の対称性を利用した partial order reduction が可能となる．この理由を示す．

**図 5** は，遷移グラフの中で，コンテナ $CO\_a1$ と $CO\_a2$ の両方に関する動作が可能になった状態を表すノード S6 を親ノードとする部分グラフを表している．ノード S6 の持つプロセス式を $P$ とする．また，この状態からコンテナ $CO\_a1$ に関する動作をさせた状態を S7（プロセス式を $Q$），コンテナ $CO\_a2$ に関する動作をさせた状態を S12（プロセス式を $Q'$）とする．また，所期の性質を表す任意の論理式を $A$ とする．

ここで，$Q$ の中に現れるすべての $a1$ という名前と $a2$ という名前を置き換えてできるプロセス式を $Q\{a1 \rightleftharpoons a2\}$，論理式 $A$ に対して同様の操作を行ってできる論理式を $A\{a1 \rightleftharpoons a2\}$ とする．この操作は単なる名前のつけ換えなので，式 (23) は自明である．

$$Q \models A \text{ iff } Q\{a1 \rightleftharpoons a2\} \models A\{a1 \rightleftharpoons a2\}. \tag{23}$$

ところで，プロセス $P$ と論理式 $A$ において名前 $a1$ と $a2$ に関し対称性が確認できた場合，すなわち $P \equiv P\{a1 \rightleftharpoons a2\}$ かつ $A$ と $A\{a1 \rightleftharpoons a2\}$ が等価な論理式であった場合を考える．この場合 $P \equiv P\{a1 \rightleftharpoons a2\}$ より $Q' \equiv Q\{a1 \rightleftharpoons a2\}$ となる．したがってこの場合式 (23) より $Q \models A$ iff $Q' \models A$ となる．

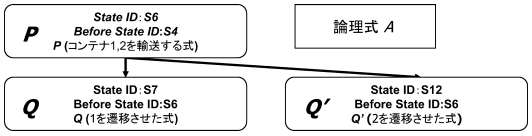


**図 5**　対称なノード
Fig. 5　Nodes with symmetric property.

以上より，プロセス $P$ と論理式 $A$ において名前 $a1$ と $a2$ に関し対称性が確認できた場合は，図 5 において状態 S7 を親ノードとする部分グラフを用いたモデル検査と，状態 S12 を親ノードとする部分グラフを用いたモデル検査の結果は同じになるため，前者のみの検査をすればよいことが分かる．

**例 5.9**　たとえば式 (19) が，「$CO\_a1$ と $CO\_a2$ が必ず目的地に到着する」という性質を表す式 (24) を満たすかどうかの検査について考える．

$$\begin{aligned}&\Box\Diamond\blacklozenge PORT\_B[CY[CO\_a1[T] \mid T] \mid T]\\ &\quad\wedge\Box\Diamond\blacklozenge PORT\_B[CY[CO\_a2[T] \mid T] \mid T].\end{aligned} \tag{24}$$

本来，式 (20) へ遷移した場合と式 (21) へ遷移した場合の両方でその性質が成り立つことを示す必要がある．しかし先に述べた理由から，以下のように式 (20) へ遷移した場合のみを検査するだけでよいことが分かる．

式 (20) が式 (24) を満たすことが検査できたとする．ここで式 (20) と式 (24) に対し $CO\_a1$ と $CO\_a2$ の名前を入れ換えてできるプロセス式と論理式をそれぞれ式 (20)′，式 (24)′ とする．これは単なる名前の入れ換えであり，したがって式 (20)′ が式 (24)′ を満たすことは自明である．ところで，式 (21) と式 (20)′ は構造合同であり，また，式 (24)′ と式 (24) は等価である．式 (20)′ が式 (24)′ を満たしていることはすでに示されているので，結局，式 (21) が式 (24) を満たすことになる．以上より，式 (20) が式 (24) を満たすことが確認できれば，式 (21) が式 (24) を満たす確認をする必要はなくなる．□

本検査システムは，この方法による枝刈りを行うよう実装されている．この方法では，プロセス式内のコンテナ ambient，およびコンテナ ambient が内部に持つ制御用 ambient の名前に対してのみ，どの ambient 名とどの ambient 名が対称になるかを構造合同に基づき調べるだけでよく，枝刈のための確認手続きは比較的単純に実装可能である．また，これらの ambient が有限個であり，プロセス式に再帰や replication がないことから，確認手続きは短時間で終了する．

論理式の対称性に関しては，すべての論理式は式 (10)〜(18) をテンプレートとして作成され，どのコンテナとどのコンテナが同じ港から同じ船で同じ港に輸送されるかも前もって分かっているので，式 (24) のようにそれらのコンテナについて対称性があることは自明である．したがって実際のモデル検査に際しては，プロセス式に対してのみ対称性を自動的に確認するよう実装した．

### 5.7　検証実験 1

図 1 で示したような，ある港からある港へコンテナを輸送する式を用意した．ただし輸送されるコンテナ数は 3〜150 とした．つまり，図 1 における $CO\_a1$ が，$CO\_a1$〜$CO\_a150$ であるような式である．このプロセス式に対し，前節までで示した手法を用いて簡約化した遷移グラフを作りながら，式 (10)〜(18) の論理積を，プロセス式が満たすかどうかという検査実験を行った．実験環境は次のとおりである：OS：Vine Linux 5.0$\beta$2 64 bit 版，CPU：Core2Duo 3 GHz，メモリ：4 GB，JDK1.6.10．結果は**表 1** のようになった．

たとえば 150 個の欄では，150 個のコンテナ輸送を表す式が所期の性質を満たしていることを，本論文で示した 2 つの partial order reduction を併用すると約 4 時間 35 分（16,503,219 ms）で検証できたことを示している．計測不能と書かれた個所は，5 時間以上経過しても結果が返ってこなかったことを示している．

なお式 (16)，つまり輸送途中で物がなくなることがないという性質は，遷移グラフをたどった検査を行わなくても初期ノードに対する構文検査だけで可能である．つまり物を表す ambient に対する *open* 動作が存在しないことを確認できれば十分である．しかし *open* 動作があったとしても，遷移中にそれと並行な位置に物を表す ambient が移動する可能性がまったくない場合もあり，また，実際式 (16) を検査からはずした場合の実験を行ってみたところ検査時間にほとんど変化がみられなかったことと，式 (16) を別に検査するための実装を追加することのコストを考慮し，本研究では式 (16) を他の式と同等に扱うことにした．

### 5.8　検証実験 2

物流書類に誤りがあった場合，あるいはプロセス式生成システムに不具合があった場合を想定し，以下のような誤った式を用いた検証実験を行った．

**表 1**　状態数と検査時間
Table 1　Number of states and elapsed time.

| | 遷移グラフのノード数 | | | |
|---|---|---|---|---|
| 輸送するコンテナ数 | 3 個 | 10 個 | 100 個 | 150 個 |
| 全状態探査（秒） | 2,475 (26.6) | 計測不能 | 計測不能 | 計測不能 |
| ample(s) のみ（秒） | 413 (0.6) | 計測不能 | 計測不能 | 計測不能 |
| 対称性のみ（秒） | 56 (0.2) | 8,170 (11.1) | 計測不能 | 計測不能 |
| ample(s) と対称性を併用 | 36 (0.2) | 246 (1.1) | 17,131 (38 m53 s) | 41,795 (4 h35 m) |

（1）　あるコンテナを表す ambient が，誤った船に乗せられる capability を含んでいる．
（2）　あるコンテナを表す ambient が船に乗せられることを確認せずに船が出港してしまう．

（1）の場合，船が出港できないことから，他のコンテナを積んだ時点でそれ以上の遷移ができなくなり，検査が終了する．性質 p1 がすべてのコンテナに対し偽となり誤りが起こったことが検出できる．検査終了時点での各コンテナの位置を確認することにより，どのコンテナの式が誤っているかの特定が可能であった．

（2）の場合，積み忘れられたコンテナに対してのみ性質 p1 が偽となったことが表示される．

いずれの場合も式中の誤りの部分の特定は容易であり，その後その部分に対し目視による物流書類の検査を行う．物流書類に誤りがなければ，プロセス式生成方法自体に問題があることが分かる．

## 6. 結　　論

本論文では，Ambient Calculus によりモデル化された物流システムに対するモデル検査システムのための公理系，モデル検査システム，および実験結果について述べた．本モデル検査システムにより，プロセス式生成システムから物流書類をもとに生成されたプロセス式が所期の性質を満たすことを形式的に示すことが可能となった．

また，単にプロセス式生成システムの生成方法の正しさを示すだけでなく，物流書類に内在する誤りに対しても，輸送が始まる前に訂正を促すことが期待できる．これは，輸送業者の意図，たとえば貨物はどの港で積み込み，どの港で積み下ろすのか等を反映した論理式を与えることができれば，その式を用いて物流書類から生成されたプロセス式を検査することで可能となる．ただし現状では，実際的な時間で検査できるプロセス式が，コンテナ百数十個にとどまっており，さらなる検査アルゴリズムや実装方法の効率化が必要である．

本論文では 1 つの港から 1 つの港へのコンテナを輸送する場合を想定した単純な実験のみの結果を示したが，数十個は港 A で，数十個は港 B で積み下ろす，あるいは途中の港で数十個をさらに積み込む，積み替える等の複雑な経路を想定した実験を今後予定している．

## 参　考　文　献

1) Affeldt, R. and Kobayashi, N.: Partial Order Reduction for Verification of Spatial Properties of Pi-Calculus Processes, *Proc. 11th International Workshop on Expressiveness in Concurrency* (*EXPRESS 2004*) (2004).

2) Cardelli, L. and Gordon, A.D.: Mobile Ambients, *LNCS*, Vol.1378, pp.140–155 (1998).
3) Cardelli, L. and Gordon, A.D.: Anytime Anywhere Modal Logics for Mobile Ambients, *POPL'00*, *Proc. 2000 ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp.365–377 (2000).
4) Cardelli, L. and Gordon, A.D.: Mobile Ambients, *Theoretical Computer Science*, Vol.240, pp.177–213 (2000).
5) Clarke, E., Grumberg, O. and Peled, D.: *Model Checking*, MIT Press (2000).
6) Kato, T., Ikeda, D. and Okada, Y.: The Implementation of Ambient Calculus with HORB for Mobile Agents, *Proc. 7th World Multiconference of Systemics, Cybernetics and Informatics*, Vol.II, pp.367–372 (2003).
7) 山口範高：貿易書類の見方・書き方，同文舘出版 (2007).
8) 国土交通省：メコン地域陸路実用化実証走行試験 (2007). http://www.meti.go.jp/press/20071018006/press_mMM.pdf
9) 国土交通省：海上貨物追跡タグシステム（MATTS）(2008). http://www.mlit.go.jp/report/press/port02_hhh_000006.html
10) 植田直人，加藤　暢，樋口昌宏：Ambient Calculus による物流システム記述に対するモデル検査，FIT2008 第 7 回情報科学技術フォーラム，pp.13–16 (2008).
11) 森本大輔，加藤　暢，樋口昌宏：Ambient Calculus を用いた物流検査システム，情報処理学会論文誌：プログラミング，Vol.48, No.SIG 10 (PRO33), pp.151–164 (2007).

(平成 21 年 8 月 13 日受付)
(平成 21 年 9 月 30 日再受付)
(平成 21 年 10 月 27 日採録)

**加藤　　暢**（正会員）
平成 9 年岡山大学大学院自然科学研究科知能開発科学専攻博士課程修了．日本学術振興会特別研究員として並行計算理論に関する研究に従事．平成 12 年より近畿大学理工学部講師．博士（工学）．並行論理型言語の意味論，およびプロセス代数に関する研究に従事．電子情報通信学会，ソフトウェア科学会各会員．

**樋口　昌宏**（正会員）
昭和 58 年大阪大学基礎工学部情報工学科卒業．昭和 60 年同大学院博士前期課程修了．（株）富士通研究所，大阪大学講師等を経て，現在近畿大学理工学部准教授．博士（工学）．分散システムの記述，検証，試験に関する研究に従事．

**植田　直人**（正会員）
平成 21 年近畿大学大学院総合理工学研究科博士前期課程修了．（株）JSOL 入社，修士（工学）．プロセス代数による，システムのモデル化およびモデル検査に関する研究に従事．