修 士 論 文

証明譜を用いたバケット同期法の形式検証

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

永浦 尊信

2011 年 3 月

修 士 論 文

証明譜を用いたバケット同期法の形式検証

指導教官　二木 厚吉 教授

審査委員主査　二木 厚吉 教授
審査委員　青木 利晃 准教授
審査委員　緒方 和博 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

0910040 永浦 尊信

提出年月: 2011 年 2 月

概 要

一般に，分散型システムでは因果順序制御と呼ばれるプロトコルが用いられている．因果順序制御は，ネットワークイベントについて，送信側と受信側で，その反映の順序を一致させる．バケット同期法は，ひとつの因果順序制御であり，ネットワークイベントの反映までにかかる時間が一定という性質があるため，ネットワークゲームや，ネットミーティングシステムで用いられている．バケット同期法が，イベントの受理にかかる時間を一定に保証することは，シミュレーション実験が行われており，その性能も評価されている．しかし，そもそもバケット同期法が正しく因果順序制御をおこなっているという，正当性の問題は検証されていない．そこで本論文では，OTS/CafeOBJ を用いてバケット同期法をモデル化し，正当性の検証を行う．

目 次

図 目 次

第1章　はじめに

1.1　背景

ネットワークゲームは，ネットワークを隔てて複数のプレイヤーで遊ぶゲームの総称である．家庭においてインターネット回線を通じて遊ぶことが可能なネットワークゲームにも需要があり，そのような物も含めて数多くのネットワークゲームが作成され，商業的な成功をおさめており，また娯楽・芸術として高い価値を認められている．また，ネットワークゲームは多くの人数で用いることを生かして，シリアスゲームと呼ばれる教育・訓練のための種類のゲームとしても作成されており，初等教育からパイロット養成のツールとしてまで幅広く用いられている． [MC05] このようなネットワークゲームに対する多くの需要に反して，実際にネットワークゲームを開発することは容易ではない．それは，ネットワークゲームが他のコンピュータゲームと同じく，高度な娯楽・芸術としての人間の感性に依存した側面に加え，高度なコンピュータプログラムとしての側面があるということも一因である．現にネットワークゲームでは，バグの発生により娯楽・芸術としての側面が秀でているにもかかわらずプレイヤーを取り逃してしまうという事態が起きる．またシリアスゲームの文脈では，バグの発生でゲームの続行が不可能になると，望まれる教育効果を達成できないため，なんとしてもバグの発生を避けたい．

ネットワークゲームはプログラムとしては，一種の分散型システムを成している．そのため他の分散型システムと同様の問題をネットワークゲームも抱えており，これが制作の困難の一つになっている．一般に分散型システムに存在する問題として，因果順序問題がある．因果順序問題が発生すると，一般にはノード間で情報の一貫性が保たれなくなり，システムの破綻に繋がる．同じようにネットワークゲームにおいても因果順序問題が発生することで，ゲームの続行が不可能になるような致命的な問題が発生する．このイベントの反映の順序を全てのノードで一致させることを，因果順序を保証するといい，因果順序を保障するためのプロトコルは，因果順序制御プロトコルと呼ばれる．一般の分散型システムでは，この因果順序制御プロトコルを導入することで，因果順序問題に対処する．しかしネットワークゲームでは，ネットワークゲームの持つ娯楽・芸術としての側面が，一般の分散型システムの求める条件に加え，人間の感性に依存した条件を因果順序制御プロトコルに求める．そのような条件を満たすプロトコルは，数多く提案されているが，その一つにバケット同期法 (Bucket Synchronization) が存在し，実際にネットワークゲームで用いられている． [GD98]

ところで，ソフトウェアのバグの発生を抑えるための手法として，形式手法 (Formal

Method) が注目されている．一般にソフトウェアは，仕様記述，アーキテクチャ決定，実装，テストという工程を経て作成される．ここで仕様記述の段階で欠陥が紛れ込むと，実装やテストの工程になり初めてそのような欠陥が露呈し，開発工期を大きく延長することになる．そのため，仕様記述を正しく行うことはスムーズなソフトウェア開発において非常に重要になる．形式手法では，仕様記述の時点からソフトウェアの開発を数学的な議論の元で行う手法で，厳密性の上で優れ，高信頼性のあるソフトウェア開発に効果が高い．また形式手法の導入は，仕様記述の工程から無矛盾で厳密な開発を行えるため，開発工程の後戻りを防ぐ効果があると考えられる．そのためネットワークゲーム開発においても，形式手法を導入することで開発効率が向上すると考えられる．しかし，ネットワークゲームに用いられている概念を形式手法で取り扱う方法は知られていない．

1.2 目的

バケット同期法が保証するとされるうち，人間の感性に依存した性質は，このようなアンケート・シミュレーション実験を行って調査しなければならない．このため先行研究では，バケット同期法に対する検証は，シミュレーション実験とアンケート調査によって行われている．しかしバケット同期法が保証するとされる性質は，必ずしも人間の感性に依存した性質ではない．特に因果順序性は，非常に珍しいケースで問題が生じることも考えられ，またそのようなケースが偶然発生しただけでも，ゲームの続行が不可能になる種類の性質である．そのためシミュレーション実験だけでバケット同期法が因果順序性を持つと結論することはできない．

本論文ではバケット同期法を OTS/CafeOBJ を用いて形式化し，バケット同期法が因果順序性を持つことを証明する． [OF03] これによってどのようなケースにおいても，バケット同期法が因果順序性を持つ事を保証し，バケット同期法がネットワークゲームで期待される性質を持つことを保証する．

またその過程で，ネットワークゲームに特有な概念である，フレームをモデル化する手法について提案し，ネットワークゲームに対する形式手法の適用を可能にする．

1.3 本論文の構成

本研究では，バケット同期法と因果順序性を形式仕様を用いてモデル化し，そのモデルを用いて形式的検証を行うことで，バケット同期法の妥当性について論じる．

2 章 本研究で対象とするネットワークゲームの定義と，離散的な時間の概念であるフレームについての考察を行う．これらの概念の元で，ネットワークゲームが求める性質を定義する．また，バケット同期法の定義を述べ，バケット同期法がどのような性質を保証すると期待されるかを確認する．このバケット同期法の定義を用いて，次章以降でモデル化と検証を行う．

3 章 本研究において用いる形式仕様記述言語 CafeOBJ における一般のモデルの作成について触れる．またここで，観測遷移機械の定義を説明し，観測遷移機械を CafeOBJ で記述するモデル化技法として OTS/CafeOBJ 法を説明する．

4 章 OTS/CafeOBJ を用いてフレーム，リアルタイムゲーム，ネットワークゲームをモデル化する指針を示し，この指針の元で実際にバケット同期法を導入したネットワークゲームをモデル化する．

5 章 5 章のモデルを用いて，因果順序性を CafeOBJ で記述し，バケット同期法が因果順序性を保証することを検証する．

6 章 本研究で得られた知見をまとめ，関連研究と本研究との関係や今後の課題について触れる．

第2章　ネットワークゲーム・フレーム・バケット同期法

本章では，本研究でモデル化の対象になるネットワークゲームを定義し，その定義の上でフレームの概念について説明する．その後，ネットワークゲームが，その通信プロトコルに求める性質について考察し，そのような性質を有すると考えられているプロトコルとしてバケット同期法を導入する．

2.1　ネットワークゲーム

本研究で扱うネットワークゲームは，家庭用のインターネット回線を通じて遊ぶことのできるリアルタイムゲームである．リアルタイムゲームとは，以下のようなゲームである．

定義 2.1 (リアルタイムゲーム) 一定の時間間隔で情報が更新されるゲーム世界を持っており，プレイヤーが行った操作が，いつ反映されたのかプレイヤーが認識できない程度に即座にゲーム世界へ反映されるコンピュータゲームをリアルタイムゲームと呼ぶ．

ここで，ゲーム世界という言葉が登場したが，それは以下のような意味である．

定義 2.2 (ゲーム世界) コンピュータゲームのルールに関する計算に必要な情報の集まりをゲーム世界と呼ぶ．

通常，ゲーム世界はプレイヤーに対する映像や音声の提示のためにその情報が用いられる．プレイヤーが入力装置を通じて操作を行うと，操作を受けて即座にゲーム世界の状態が変化するため，リアルタイムゲームではプレイヤーがゲーム世界の中で直接行動しているかのように遊ぶことができる．
リアルタイムゲームの定義を用いると，本研究で扱うネットワークゲームは以下のように定義できる．

定義 2.3 (ネットワークゲーム) 二人以上のプレイヤーが，ネットワーク回線を通じて，相互に一貫したゲーム世界を用いて遊ぶリアルタイムゲームを，ネットワークゲームと呼ぶ．

また，ネットワークゲームのプレイヤーは，それぞれに計算機端末を用いている．このことを踏まえて，プレイヤーの操作を受け取り，ゲーム世界を計算し，ネットワークに情報を送受信するような計算機端末を，**ノード**と呼ぶ．ネットワークゲームでは，ノードそれぞれを区別するための一意の識別子として，ユーザ ID が用いられている．**ユーザ** *ID* はネットワークゲームの開始時に，一意に決定され，その後変更されることはないような値である．

2.2 イベント

ネットワークゲームで，一貫したゲーム世界を作るためには，操作の情報を送受信する必要がある．しかしここで，入力装置としてマウスを考える．マウスはマウスカーソルを動かす操作と，クリックの操作を行うことができる入力装置である．しかしマウスの操作情報をネットワークゲームで用いるときは，ネットワーク回線の帯域の問題があり，マウスカーソル位置の全ての情報を送受信することは望ましくない．そこで，実際のネットワークゲームでは，攻撃を行う指示を出した，というような観点で操作を抽象化した情報を送受信している．このことから，イベントという概念を定める．

定義 2.4 (イベント) ゲーム世界に対して影響のある操作 *1* 回分の情報をイベントと呼ぶ.

通常のネットワークゲームで送受信されるのは，このイベントである．

2.3 ネットワーク

本研究で扱うネットワークゲームは，トランスポートレイヤーまでに以下のようなネットワーク環境を想定している．

- ネットワークの遅延は不安定に変化する．
- ネットワークメッセージは必ず届く
- ネットワークメッセージの届く順序は保証されない
- 一度受信したネットワークメッセージは二度は受信されない

実際にこのような性質を持つトランスポートレイヤーのプロトコルとして，標準化は成されていないものの，RUDP(Reliable UDP) が存在しており，現実のネットワークゲームでも，しばしばこれが用いられる． [BK99] また UDP のような，ネットワークメッセージが届くかについて保証のないトランスポートレイヤーの上位に独自のプロトコルレイヤーを設けることで，このような性質を保証することもある．

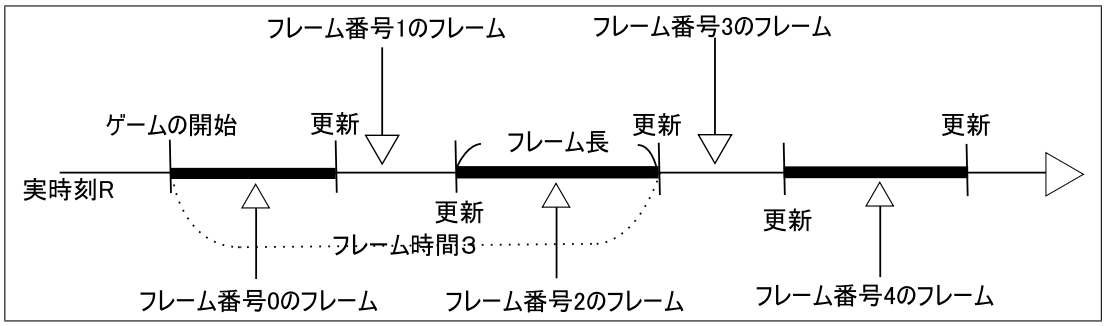

図 2.1: フレームに関する各種概念

2.4 リアルタイムゲームにおけるフレーム

一般にリアルタイムゲームでは，アニメーションのある映像を生成しプレイヤーに提示しなければならない．ある瞬間のゲーム世界は，プレイヤーに一枚の画像を提示するのに十分な情報を持っている．しかしアニメーションは，一定の時間ごとに画像が更新されることで実現される．このような要請のため，リアルタイムゲームは，一定の時間間隔でゲーム世界の情報が更新されるが，この間隔をしばしばフレームと呼び，これにともなって以下のような各種の概念が定義される．

定義 2.5 (フレーム) リアルタイムゲームにおいて，ゲーム世界を更新してから次に更新するまでの間をフレームと呼ぶ．

定義 2.6 (フレーム長) 実時間で計ったフレームの長さをフレーム長と呼ぶ．

定義 2.7 (フレーム番号) リアルタイムゲームが開始してから，ある時刻までに何度目のゲーム世界の更新を行ったかをその時刻におけるフレーム番号と呼ぶ．

定義 2.8 (フレーム時間) 二つのフレーム番号の差の絶対値をフレーム時間と呼ぶ．

これらのフレームに関するそれぞれの概念を図 2.1 に示した．この図の横軸は実時刻である．実時刻の軸に対して縦線が交差しているが，この縦線でゲーム世界の更新が発生したとして読む．

フレーム長はしばしば，FPS という 1 秒間に何回のゲーム世界の更新があるかという単位で表現されている．フレーム長が安定しない場合，映像が乱れ，プレイヤーは不快を感じたり，ゲームを操作することが不可能になるため，フレーム長は，なるべく一定であることが望ましい． [PW02] そのため通常のリアルタイムゲームは，フレーム長を 60FPS や 30FPS に合わせるように努力されている．しかし多くのリアルタイムゲームで，しばしばフレーム長は，ある程度上下することが許容されている．これはゲーム世界の更新には高い負荷がかかる場合があり，そのときにはトレードオフとして，フレーム長を保持するよりも正しいゲーム世界の更新を優先するように選択されているためである．このよ

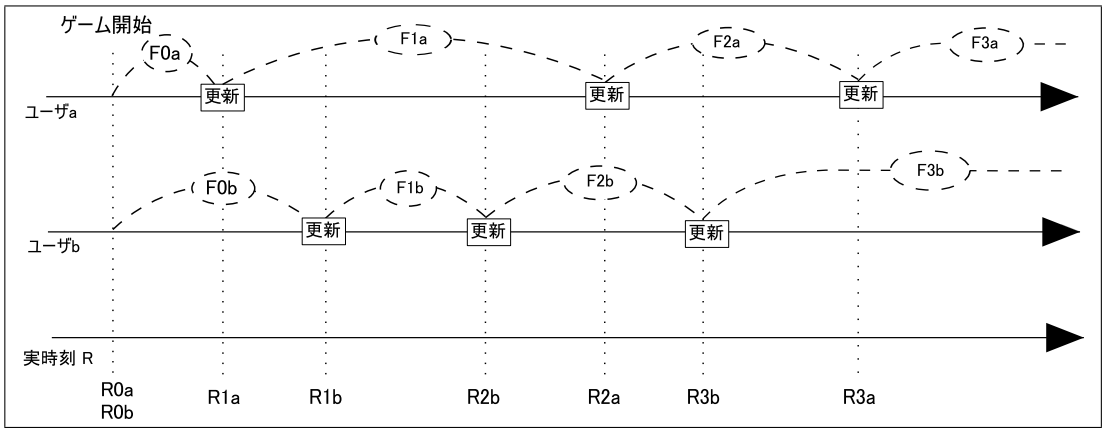

図 2.2: ネットワークゲームにおけるフレームの例

うなフレーム長の不安定性がどの程度まで許容できるのかは，人間の感性と，ゲームの用途に依存している．このため，もしフレームの概念に依存した性質を議論するならば，フレーム長は，実時間の上で一定ではない，不明の長さであるとすることが望ましい．

ここで，フレーム長とフレーム時間は混乱を招きやすいが，別の概念であることに注意を要する．フレーム長は，ある単一のフレームに対して，どれだけの実時間が対応するのかという事であり，フレーム時間は，あくまで二つのフレーム番号の差でしかない．このため，特にフレーム長を不定であると考える場合には，フレーム番号から実時刻を求めたり，フレーム時間から実時間を求めたりすることは出来ない．

また，フレーム長が一定だとしても，操作を行ってから，その操作がゲーム世界に反映されるまでのフレーム時間が一定でないと，やはりプレイヤーは不快を感じたり，ゲームを操作することが不可能になるため，これも一定であることが望ましい．

なお，フレーム時間の単位としても，フレームという単語が用いられる．

例 2.1 フレーム番号 *5* のフレームとフレーム番号 *8* のフレームの間のフレーム時間は，*3* フレームである．

2.5 ネットワークゲームにおけるフレーム

ネットワークゲームでは，ゲーム世界の情報は，それぞれのノードで独立に更新されるため，フレームもそれぞれのノードが個別に持つ事になる．そのため，フレーム番号は，同じ時刻でもノードごとに別々の値になる．

プレイヤー A とプレイヤー B が参加しているネットワークゲームでのフレームの例を図 2.2 に描いた．この図も，横軸は実時刻である．ここで注意するべきことは，互いに 1 フレーム目でも，ユーザ間でフレームの開始・終了の実時刻は一致していないということである．これはネットワークゲームは実際には実時刻ではなく，フレーム番号ごとに同期が計られているためである．

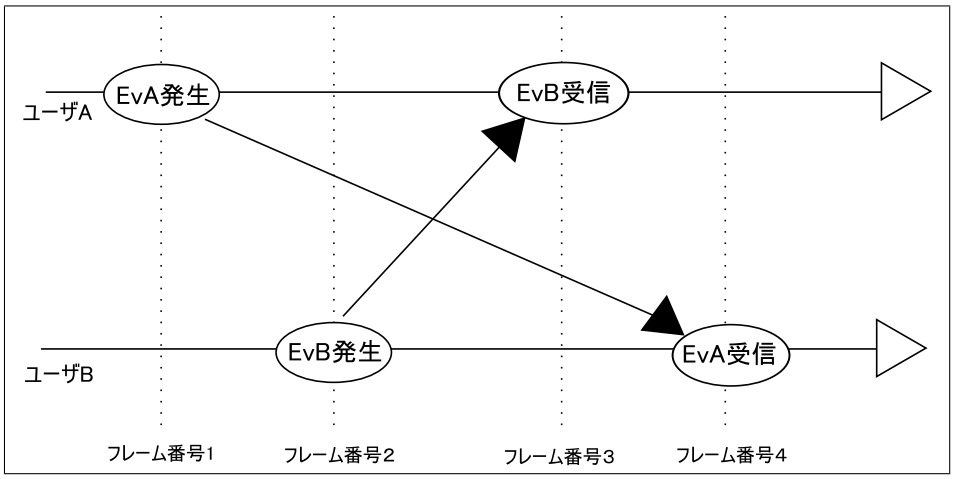

図 2.3: ゲーム世界が非共有

2.6 ゲーム世界の共有

ネットワークゲームでは，各ノードは遅延のあるネットワークを通して互いにイベントを送受信している．しかし単純に送受信するだけでは，ノード間でイベントの反映順序が異なる場合がある．イベントの反映順序が入れ替わる場合，ゲーム世界の更新の結果は，異なった物になる．

定義 2.9 (ゲーム世界の共有) どのようなフレーム番号においても，各ノードがそのフレーム番号に到達したとき，ゲーム世界が等しいならば，各ノードはゲーム世界を共有すると言う．

ネットワークゲームでは，各ノードはゲーム世界を共有している必要がある．

ゲーム世界の共有を崩すプロトコルの例として，発生したイベントを即座に反映および送信するというプロトコルを用いたと仮定し，UserA と UserB が下記で示す通信を行う場合を考える (図 2.3)．

1. ユーザ A が時刻 1 に EvA を発生，受理を行い，EvA をユーザ B に送信．
2. ユーザ B が時刻 2 に EvB を発生，受理を行い，EvB をユーザ A に送信．
3. ユーザ A が時刻 3 に EvB を受信し受理．
4. ユーザ B が時刻 4 に EvA を受信し受理．

ここで，ユーザ A とユーザ B の操作の反映順序は，それぞれ，EvA→EvB，EvB→EvA となり，反映の順序が変わったことで，ゲーム世界の更新における計算結果が異なってくる．そのため，フレーム番号 4 におけるユーザ A とユーザ B のゲーム世界が異なる．よって，ユーザ A とユーザ B はフレーム番号 4 においてゲーム世界を共有していない．したがって，このようなプロトコルをネットワークゲームで利用することはできない．

2.7　プロトコルへの要求

ここまでに見てきた定義と問題点から，少なくとも次の三種類の性質を満たすプロトコルがネットワークゲームには必要になることが解る．

定義 2.10 (因果順序性) あるプロトコルを用いた時，全てのフレーム番号で，全てのノードがゲーム世界を共有しているならば，プロトコルは因果順序性を持つと言う．

定義 2.11 (フレーム一定性) あるプロトコルを用いた時，フレームレートを人間に不快でない程度に，一定に保つことが可能ならば，プロトコルはフレーム一定性を持つと言う．

定義 2.12 (操作安定性) あるプロトコルを用いた時，操作を行ってから実際に反映されるまでのフレーム時間を，人間が不快でない程度に一定に保つことが可能ならば，プロトコルは操作安定性を持つと言う．

2.8　バケット同期法

バケット同期法は，前節で示した 3 種類の性質 (因果順序性，フレーム一定性，操作安定性) を，通信ディレイがある一定値を超えないという条件の下で，保証するとされるため，ネットワークゲームにおいて広く用いられている．これら性質のうち，フレーム一定性，操作安定性は，既存研究でシミュレーションによる調査が行われているが，因果順序性はシミュレーション実験では不十分である．そのため本研究では，バケット同期法が因果順序性を持つかについて検証を行う．

バケット同期法では，イベントの取り扱いに関して，三種類の区別するべき概念がある．

定義 2.13 (イベントの発生) イベントが実際の操作によって確定することをイベントの発生と呼ぶ．イベントが発生した時点では，まだゲーム世界にイベントを反映しない．

定義 2.14 (イベントの受信) イベントをネットワークから受信することをイベントの受信と呼ぶ．イベントを受信した時点では，まだゲーム世界にイベントを反映しない．

定義 2.15 (イベントの受理) 保有しているイベントを用いて，実際にゲーム世界を更新することを，イベントの受理と呼ぶ．

バケット同期法では，バケットという操作の一時的な保管場所を作ることで，イベントの発生・受信・受理の区別を可能にする．

定義 2.16 (バケット) バケットは，ユーザが受信しているが，まだ受理していないイベントの集合である．

バケット同期法では次の方法によって，因果順序性を保証する．

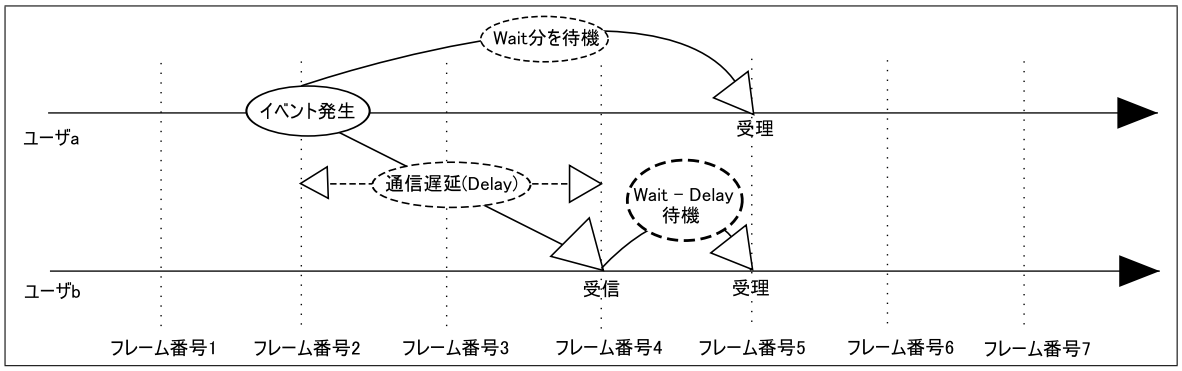

図 2.4: バケット同期法

- ネットワークゲームの開始時に，待ちフレーム時間 Wait を決定し，全ユーザ間で共有する．その後 Wait の値は変更しない．
- あるノード A でイベントが発生した場合，直ちに他の全てのノードに対して，A のフレーム番号とイベントのペアをブロードキャストする．A は，イベントをバケットに保存し，Wait を待機した後に，そのイベントを受理する．
- あるノード B が，フレーム番号 F とイベント E のペアを受信した場合．B は E を，B の持つバケットに追加する．B における現在のフレーム番号 F' と，F のフレーム時間 Delay を求め，Wait と Delay の差のフレーム時間だけ待機した後に，E を受理する．

図 2.4 では，ユーザ A のフレーム番号 2 で発生したイベントの受理を行うフレーム番号をユーザ A と B の間でバケット同期法が一致させる様子を示している．ただし，ここでの横軸は，フレーム番号である．

2.9 保障プロトコル

ところで，図 2.5 のような，Delay が Wait を超える状況では，バケット同期法は因果順序性を保証しない．そこで実際のネットワークゲームの実装では，Wait よりも高い Delay が検出された場合，バケット同期法の利用を停止し，補助的なプロトコルを用いて，ゲーム世界が共有されている状態に復元する対策が用いられる．本研究では，このような状況で呼び出されるバケット同期法の補助プロトコルを総称して，保障プロトコルと呼ぶ．

定義 2.17 (保障プロトコル) ネットワークディレイがバケット同期法で用いている *Wait* を超える時，バケット同期法の代わりに呼び出されるプロトコルを保障プロトコルと呼ぶ．

例として，すべてのノードのフレーム更新を停止して，どれか一つのノードが持つゲーム世界を全体に配信し，ゲーム世界の共有を復旧するという方法がある．また別の例として

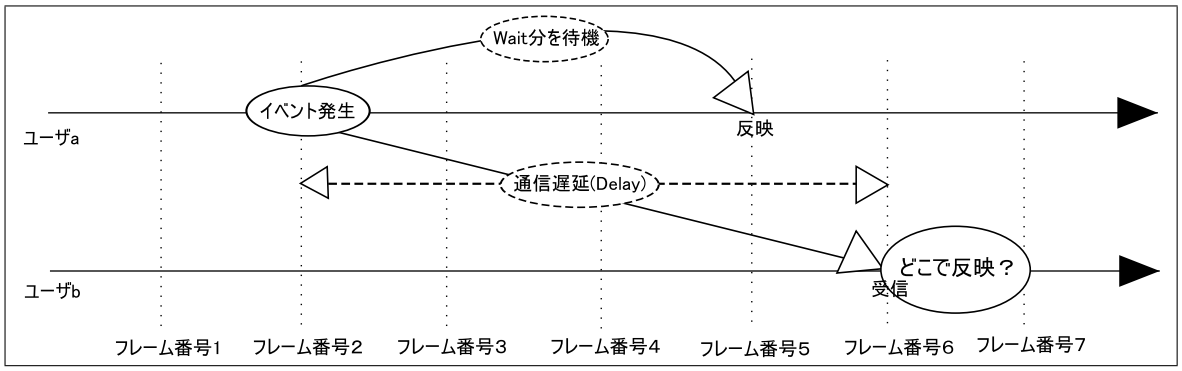

図 2.5: Delay $<$ Wait 仮定の必要性

は，ゲームをその場で終了させ，ゲームを終了させてしまうことも一つの保障プロトコルと捉えられる．

各種提案されている補償プロトコルには，それぞれトレードオフがあり，ゲームに併せて選択されている．そのため，本研究では単一のプロトコルを対象にすることはせず，補償プロトコルは Delay $<$ Wait であることを保証するようなプロトコルであると考え，これを仮定した元でバケット同期法を検証することで，Delay $>$ Wait であるような状況から，ある補償プロトコルがゲーム世界の共有を復元することを検証するだけで，汎用に，バケット同期法とその補償プロトコルを合わせたプロトコルに対して，因果順序性を検証できる枠組みを作成することにする．

第3章　代数的仕様記述言語 CafeOBJ

本章では，CafeOBJ を用いた形式化の手法について解説する．

3.1　CafeOBJ について

形式仕様 (Formal Specification) は，厳密な数学や論理によって定義された言語を用いて表現された仕様のことである．形式仕様では，仕様の持つ性質を解析できるため，仕様が望ましいものであるかを検証することができ，そのための高い仕様を作成できる．このような解析は，計算機による機械的検証によって可能なため，ソフトウェアの開発効率でも利点を持っている．

特に代数的な考え方に基づいて形式仕様の記述を行う言語は，代数的仕様記述言語 (Algebraic Specification Language) と呼ばれている．代数的仕様記述言語では，ソート (Sort)，演算 (Operator)，等式 (Equation) のような基本要素を使って現実世界を表現する．CafeOBJ は，代数的仕様記述言語の一つで，強力なモジュールシステムを備え，項書き換えによって仕様を実行かのうで，始代数 (Initial Algebra) や隠蔽代数 (Hidden Algebra) をサポートするなどの特色を備えている． [Fut06] [NNS03]

3.2　始代数と隠蔽代数

CafeOBJ では，始代数は主に自然数や実数，リスト構造などの抽象データ型の記述を行うために用いられる．始代数では，可視ソート (Visible Sort) を定義し，この可視ソートに対する演算 (Operator) を定義することで，抽象データ型を表現する．

また CafeOBJ において隠蔽代数は，抽象機械を記述するために用いられている．隠蔽代数で抽象機械を記述するためには，まず抽象機械の状態を隠蔽ソート (Hidden Sort) によって表現する．その後，この隠蔽ソートに対して抽象機械の状態遷移を表現するための遷移演算 (Action Operator) と，それぞれの状態を観察するための観測演算 (Observation Operator) を定義することで，抽象機械を表現する．

3.3 OTS/CafeOBJ

OTS/CafeOBJ とは，CafeOBJ を用いて観測遷移機械 (Observational Transition System) を記述する方法である．[OF03] 観測遷移機械では，モデル化の対象になるシステムの値を観察し，その値がどのように変化するのかを記述することによって，モデルを作成する．モデル化の対象となるシステムの状態空間 U と，その各要素，状態 u の存在を仮定し，観測遷移システムは以下のように定められる．

定義 3.1 (観測遷移機械) 観測遷移機械 S とは，以下の条件を満たす $\langle O, T, I \rangle$ の組である．

- O ： O は関数 $o : U\ V_{o0}\ V_{o1}...V_{on} \rightarrow V_o (n \geq 0)$ の集合である．また，この o を観測と呼び，観測の返り値を観測値と呼ぶ．
- T ： T は関数 $t : U\ V_{t0}\ V_{t1}...V_{tn} \rightarrow U (n \geq 0)$ の集合である．また，この t を遷移規則と呼ぶ．各遷移規則 t は，遷移関数と同じ引数を取る述語，すなわち $e[t] : U\ V_{t0}\ V_{t1}...V_{tn} \rightarrow Bool$ を持っている．この述語 $e[t]$ は，効力条件と呼ばれる．
- I ： I は U の部分集合である．また，I の要素のことを初期状態と呼ぶ．
- $=_S$ ： S は同値関係 $=_S$ が存在し，以下の 3 つの条件を満たす．
  - $\forall u_1, u_2 \in U. \forall o \in O. u_1 =_S u_2 \Leftrightarrow o(u_1, x_1, x_2, ..., x_n) = o(u_2, x_1, x_2, ..., x_n)$ (ただし，$x_i (i \geq 0)$ は V_{oi} 型．)
  - $\forall u_1, u_2 \in U. \forall t \in T. u_1 =_S u_2 \Leftrightarrow t(u_1, x_1, x_2, ..., x_n) = t(u_2, x_1, x_2, ..., x_n)$ (ただし，$x_i (i \geq 0)$ は V_{ti} 型．)
  - $\forall u \in U. \forall t \in T. e[t](u, x_1, x_2, ..., x_n) \Rightarrow t(u, x_1, x_2, ..., x_n) = u.$ (ただし，$x_i (i \geq 0)$ は V_{ti} 型．)

観測遷移機械 S の実行とは，いずれかの初期状態から始まり，遷移規則を非決定的に適用することで得られる状態の無限列である．ここで，非決定的な適用とは，全ての遷移規則が実行において無限に適用されるということである．

より正確には，**観測遷移機械 S の実行**は，以下の条件を満たす状態の無限列 $u_0 u_1 \ldots$ である．

開始性 $u_0 \in I$ である

連続性 すべての $i \in \{0, 1, 2, \ldots\}$ に対して，$u_{i+1} =_S t(u_i)$ を満たす $t \in T$ が存在する．

公平性 各 $t \in T$ に対して，$u_{i+1} =_S t(u_i)$ を満たす $i \in \{0, 1, 2, \ldots\}$ が無限に存在する．

ある状態 $u \in U$ が S のいずれかの実行に現れるとき，**状態 u は S で到達可能**であるという．

このような状態遷移機械を記述するために，OTS/CafeOBJ では次のような方法をとる．

- 状態空間 U は，隠蔽ソートを作成することで表現する．これは*[U]*のように記述する．
- 観測 o は，CafeOBJ の観測演算で表現する．これは `bop o : U V1 V2 ... Vn -> V` のようにして記述する．
- 遷移規則 t は，CafeOBJ の作用演算で表現する．これは `bop t : U V1 V2 ... Vn -> U` のようにして宣言する．
- 効力条件は，対応する遷移規則と同じ引数を取り，Bool を返す演算を定義することで宣言する．
- 状態に対する観測値は，観測演算に関する等式で記述する．ある遷移規則を適用したときの状態の変化は，等式において，状態空間を表す隠蔽ソートの CafeOBJ 変数を，遷移規則の第一引数に与えた物を使って表現できる．また効力条件を反映するため，観測値を表す等式を条件付き等式にし，条件として効力条件を与え，効力条件が true の場合は遷移規則を適用したときの状態の変化を記述し，false の場合は状態を変化させないように記述する．
- 初期状態 I は，状態空間の隠蔽ソートを返す，0 個以上の始代数の引数を持った演算を定義することで表現する．たとえば，`op init : -> U .` のようにして宣言する．

3.4 安全性と検証

観測遷移機械 S が安全性 (Safety property)P を持つとは，S の到達可能な全ての状態において P が成り立つことである．安全性を OTS/CafeOBJ に対して表現するためには，まず述語 P を記述するためのモジュール INV を宣言する．

このモジュールで P は，次のように記述する．

```
mod INV
{
    op p : HidSrt X1Srt X2Srt ... XnSrt -> Bool .
    eq p(H, X1, X2, ..., Xn) = P(H, X1, X2, ..., Xn) .
}
```

ここで，HidSrt は隠蔽ソート，XkSrt($0 \leq k \leq n$) は，全て P のもつ自由変数に対応する可視ソートである．またそれぞれ，H は HidSrt の，Xk($0 \leq k \leq n$) は XkSrt の CafeOBJ 変数を表す．

このモジュール INV では，op によって P の CafeOBJ での記述である p を作成し，この p の性質を eq によって等式として定義している．

観測遷移機械 S が安全性を持つ事は，証明譜を作成することで検証を行う事ができる．

この証明譜の作成では，init は任意の初期状態，trans は任意の遷移規則，$xi(1 \leq i \leq n)$ は，それぞれ $XiSrt$ の項，HidSrt は隠蔽ソート，XkSrt$(0 \leq k \leq n)$ は，全て P のもつ自由変数に対応する可視ソート，またそれぞれ，H は HidSrt の，Xk$(0 \leq k \leq n)$ は XkSrt の CafeOBJ 変数を表すとして以下のように行う．

まず安全性 P の CafeOBJ 記述 p をモジュール INV で作成し，次のモジュール ISTEP を定義する．

```
mod ISTEP
{
    pr( INV ) .
    op s s' : -> HidSrt .
    op istep : X1Srt X2Srt ... XnSrt -> Bool .
    eq istep(X1, X2, ..., Xn) =
        p(s , X1, X2, ..., Xn) implies
        p(s', X1, X2, ..., Xn) .
}
```

次に任意の初期状態で，証明したい性質が成り立つことを示すために次のように証明節を記述する．

```
open INV .
    red p(init, x1, x2, ..., xn) .
close .
```

さらに，任意の遷移規則に関して，

```
open ISTEP .
    eq s' = trans s .
    red istep(x1, x2, ..., xn) .
close .
```

red コマンドは，等式を書き換え規則と見なし，与えられた項を簡約するコマンドである．このコマンドが Bool の項 True を返してくるならば，性質 P は初期状態で成り立っていることが確認され，この証明節は有効な証明節としてかんがえる事ができる．このような証明譜の作成は，S の初期状態を帰納基底，遷移規則を帰納段階として，構造的帰納法を行うことに対応する．また，この時点で検証が完了しない場合には，補題を立てるか，条件分けを行う．

第4章　バケット同期法のモデル化

前章までに，ネットワークゲーム，フレーム，そしてバケット同期法の定義を行った．本章ではバケット同期法が因果順序性を保証するかを形式検証するため，OTS/CafeOBJを用いてバケット同期法の形式仕様記述を行う．

4.1　モデル化の指針

4.1.1　フレーム番号

フレームのモデル化には，フレームを区別するために，フレーム番号を計測する必要がある．ただし，フレーム番号はただの有限の基数であるため自然数で代用することが可能である．したがって自然数のモジュールPNATが存在するならば，次のCafeOBJ上の記述でフレーム番号を示すモジュールを記述とすることができる．

```
mod FRAME {
    pr( PNAT * {
        sort Nat -> Frame,
        sort NzNat -> NzFrame,
        sort Zero -> ZeroFrame
    } ) .
}
```

ここでそれぞれ，Natは自然数，Zeroは0，NzNatは非0の自然数のソートである．

また，フレーム時間は，フレーム番号の差の絶対値であるため，自然数が差の絶対値について閉じているために，結局Frameソートで表現できる．

4.1.2　ノードとイベント

ネットワークゲームに登場するノードは，ユーザIDで区別されているため，ユーザIDの記述を作ることでノードを表現する．ユーザIDはゲームごとに特別な情報が付与されている場合もあるが，一般に少なくとも比較が可能であるという性質を持っている．このようなモデルは，CafeOBJでは緩い意味論を用いることで記述できるため，以下のようなモジュールを本研究では用いた．

```
mod* UID {
    [ Uid ]
    pred _=_ : Uid Uid {comm} .
    eq (U:Uid = U) = true .
}
```

また，イベントについてもゲームごとに特別な情報が付与されている場合もあるが，やはり一般に比較が可能であるという性質を持っている．ユーザ ID と同様に，緩い意味論で以下のようなモジュールを作成し表現した．

```
mod* EVENT {
    [ Event ]
    pred _=_ : Event Event {comm} .
    eq (E:Event = E) = true .
}
```

4.1.3 フレーム・リアルタイムゲーム・ネットワークゲーム

フレームは，フレームレートの慣例から 60FPS などの一定の実時間での間隔を持つと考えられているが，実際には実時間に依存して性質を論ずることが望ましくないことは二章で説明した．したがって，本研究ではゲーム世界は，実時間で考えるといつでも更新されうる物として，これをモデル化する．

観測遷移機械でこれを表現するために，各遷移が発生することは適当な実時間の経過があるものと仮定する．しかし，ここで各遷移でどれだけの時間が変化したのかを調べるような観測を一切定義しない．このように記述することで，モデル上の記述が直接実時間へ依存することを防ぐことができる．そして，ゲーム世界の更新 1 回分を，遷移 (update 遷移) として記述し，この遷移のみフレーム番号を次に進めるとする．このように記述することで，実時間で見ると，どの時刻においてもゲーム世界の更新が発生するというモデルを得ることができる．

この方針でモデル化を行う場合，リアルタイムゲームは，フレーム番号のモジュール FRAME を用いて，次のように CafeOBJ 上で記述できる．

```
mod REALTIMEGAME {
    pr( FRAME ) .
    ...
    *[ Game ]*
    op init : SrtI1 SrtI2 ... SrtIn -> Game .
    ...
    -- ゲーム世界の更新は遷移で記述する.
    bop update : Game -> Game .
```

```
    -- これは遷移なので効力条件を以下のように記述する.
    op c-update : Game -> Bool .
    ...
    -- フレーム番号は観測によって得られるようにする
    bop frame : Game -> Frame .
    ...
    -- 初期状態は常にフレーム番号が0
    eq frame( init(I1:SrtI1, I2:SrtI2, ..., In:SrtIn) ) = 0 .
    ...
    -- 効力条件が成り立たないならば遷移しない
    ceq update(G:Game) = G if not( c-update( G ) ) .
    -- ゲーム世界の更新を表現する遷移では，フレーム番号を進める.
    ceq frame( update(G:Game) ) = s( frame(G) ) if c-update( G ) .
    ...
}
```

ただし，...は，一般の記述の省略である.

さらにこれを，ネットワークゲームに拡張する場合，それぞれのノードがフレームを持つため，ユーザIDのモジュールUIDを追加で用いて，次のような記述を行うことが出来る.

```
mod NETWORKGAME {
    pr( FRAME ) .
    pr( UID ) .
    ...
    *[ Game ]*
    op init : SrtI1 SrtI2 ... SrtIn -> Game .
    ...
    -- ユーザごとにゲーム世界の更新を表現するための引数に追加されている
    bop update : Game Uid -> Game .
    -- 引数を追加したので，効力条件も引数に追加する.
    op c-update : Game Uid -> Bool .
    ...
    -- フレーム番号はノードごとに存在するため引数が追加されている
    bop frame : Game Uid -> Frame .
    ...
    -- 初期状態は，どのノードでも常にフレーム番号が0
    eq frame( init(I1:SrtI1, I2:SrtI2, ..., In:SrtIn), U:Uid ) = 0 .
    ...
    -- ゲーム世界の更新が起きたノードのフレーム番号を進める.
```

```
    ceq frame( update(G:Game, U1:Uid ), U2:Uid ) = s( frame(G, U2) ) if U1 = U2 .
    -- ゲーム世界の更新が起きなかったノードのフレーム番号は変化しない.
    ceq frame( update(G:Game, U1:Uid ), U2:Uid ) = frame(G, U2) if not(U1 = U2) .
    ...
}
```

4.1.4 メッセージとネットワーク

バケット同期法に登場するネットワークメッセージは，イベントと送信時のフレーム番号のペアである．メッセージからは，イベントとフレーム番号が取り出せ，イベントとフレーム番号の双方が一致するならば，同じメッセージであると考えられるため，以下のようにCafeOBJで記述した．

```
mod! MESSAGE {
    pr( FRAME ) .
    pr( EVENT ) .
    -- sort
    [ Message ]
    -- operators
    op pack : Event Frame -> Message {constr} .
    op event : Message -> Event .
    op sendtime : Message -> Frame .
    pred _=_ : Message Message {comm} .
    -- variables
    vars M1 M2 : Message .
    var E : Event .
    var T : Frame .
    -- equations
    eq event( pack( E, T ) ) = E .
    eq sendtime( pack( E, T ) ) = T .
    eq (M1 = M2) = ( (event(M1) = event(M2)) and (sendtime(M1) = sendtime(M2))) .
}
```

さらにこのメッセージの記述を用いて，ネットワークの記述を行った．ネットワークとは未受信のメッセージのリストであると考え，ノードごとにネットワークを持つ事で，モデル化を行った．(図 4.1)

本研究で対象にするネットワークは遅延時間が不明であるため，既に送信されたメッセージならば，いつでも受信されうるとしてモデル化を行う．また一度受信したメッセージは識別され二度は受信されないため，受信とはネットワークからメッセージを除去す

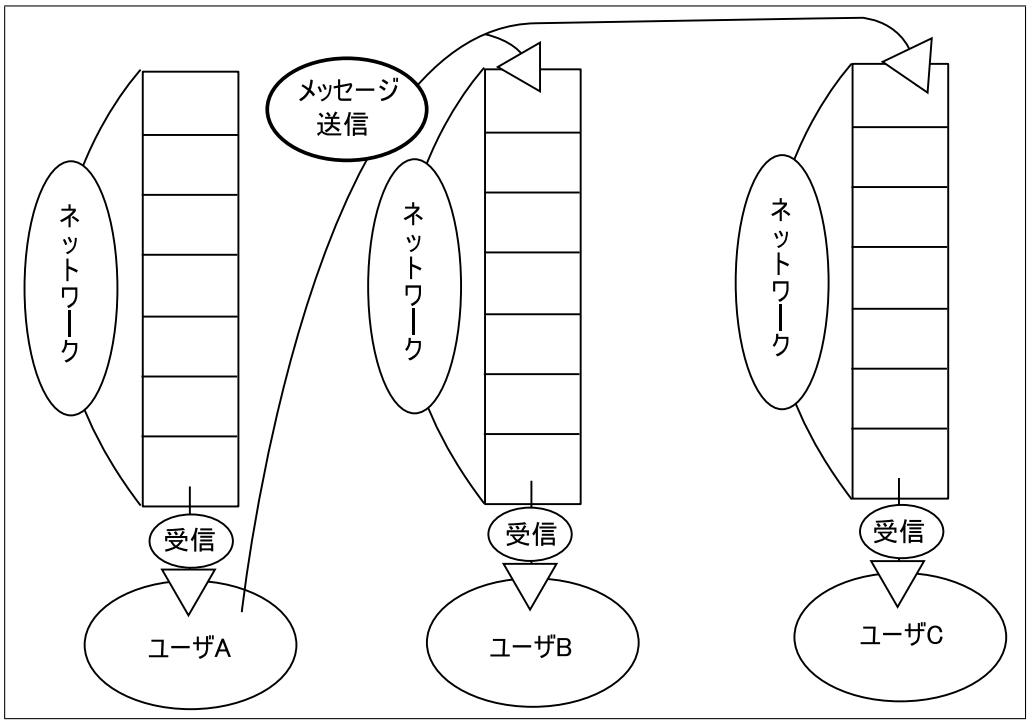

図 4.1: ネットワークのモデル

ることであると考える．あるノードへのメッセージの送信は，そのノードのネットワークに，メッセージを追加することで行える．あるノードからブロードキャストを行うことのモデル化は，そのノード以外の全てのノードの持つネットワークに，メッセージを追加することで行える．

4.1.5 バケット

バケットは，バケット同期法において，受信済み・発生済みだが，まだゲーム世界に受理していないイベントが待機するデータ構造である．これは，受理するべきイベントの集合をフレーム番号ごとに保存した物であると考える事ができる．そこでまず，イベントの集合 EventSet を CafeOBJ モジュールの EVENTSET で定義し，このモジュールを用いて，ネットワークゲームのモデルに，bucket という観測を追加した．bucket は引数として，ゲームの状態，Uid，Frame を取り，フレーム番号で受理を待つ EventSet を返す．

4.2 バケット同期法のモデル

これらの指針の下で，バケット同期法のモデルを作成すると，以下のシグネチャを持つ SESSION モジュールになる．

```
-- hidden sort
 *[Session]*
 -- initial state
 op init : NzFrame -> Session {constr} .

 -- actions
 bop send : Session Uid Event -> Session {constr} .
 bop receive : Session Uid Message -> Session {constr} .
 bop update : Session Uid -> Session {constr} .

 -- observations
 bop wait : Session -> NzFrame .
 bop frame : Session Uid -> Frame .
 bop network : Session Uid -> Network .
 bop gameworld : Session Uid Frame -> EventSet .
 bop bucket : Session Uid Frame -> EventSet .

 -- effective conditions
 pred c-send : Session Uid Event .
 pred c-receive : Session Uid Message .
 pred c-update : Session Uid .
```

それぞれの意味を解説していく.

4.2.1 観測演算

観測演算は，s においての次のような意味を持つ.

- `frame(s, uid)` : uid のユーザのフレーム番号
- `network(s, uid)` : uid のユーザのネットワーク
- `gameworld(s, uid, f)` : uid のユーザがフレーム番号 f においてゲーム世界に反映した EventSet. (次章で説明するゲーム世界更新情報)
- `wait(s)` : 全体で共有されている Wait の値
- `bucket(s, uid, f)` : uid のユーザの f で反映されることを待つ EventSet. (バケット)

4.2.2 初期状態: init(time)

time を Wait として共有し，ネットワークゲームを開始したことを示す遷移演算である．init では，すべてのユーザの frame は 0 に，network，gameworld，bucket は全て空に初期化する．また wait の値は time として初期化する．

これらを SESSION モジュール内部で実際に記述すると次のようになった．

```
-- init
eq wait( init( W ) ) = W .
eq frame( init( W ), ID1 ) = 0 .
eq network( init( W ), ID2 ) = nil .
eq gameworld( init( W ), ID2, F ) = empty .
eq bucket( init( W ), ID2, F ) = empty .
```

4.2.3 遷移演算: send(s, uid, event)

uid のユーザが Event を発生させたことを示す．他のすべてのユーザの network に，Message を追加する．この Message は送信時刻を保持するために，event と frame(s, uid) の組みとして生成する．また bucket(s, uid, frame(s, uid) + wait(s)) に event を追加する．ここではイベントが発生しているが，受理は行わないため gameworld は変更しない．さらに frame の値は増やさず，そのままの値を保つ．これは実時間が過ぎているとしても，あくまでフレーム更新によってのみフレーム番号は増えると考えるためである．send はいつでも発生しうるため，効力条件 c-send は常に true である．

send の SESSION モジュールにおける記述は以下のようになった．

```
-- c-send
eq c-send( S, ID1, E ) = true . -- anytime, send is happen.

-- send
ceq wait( send( S, ID1, E ) ) = wait( S ) if c-send( S, ID1, E ) .
ceq frame( send( S, ID1, E ), ID2 ) =
       frame( S, ID2 ) if c-send( S, ID1, E ) .
ceq network( send( S, ID1, E ), ID2 ) =
    (if ID1 = ID2
       then network( S, ID1 )
       else pack( E, frame( S, ID1 ) ) | network( S, ID2 )
    fi) if c-send( S, ID1, E ) .
ceq gameworld(
       send( S, ID1, E ),
       ID2, F
```

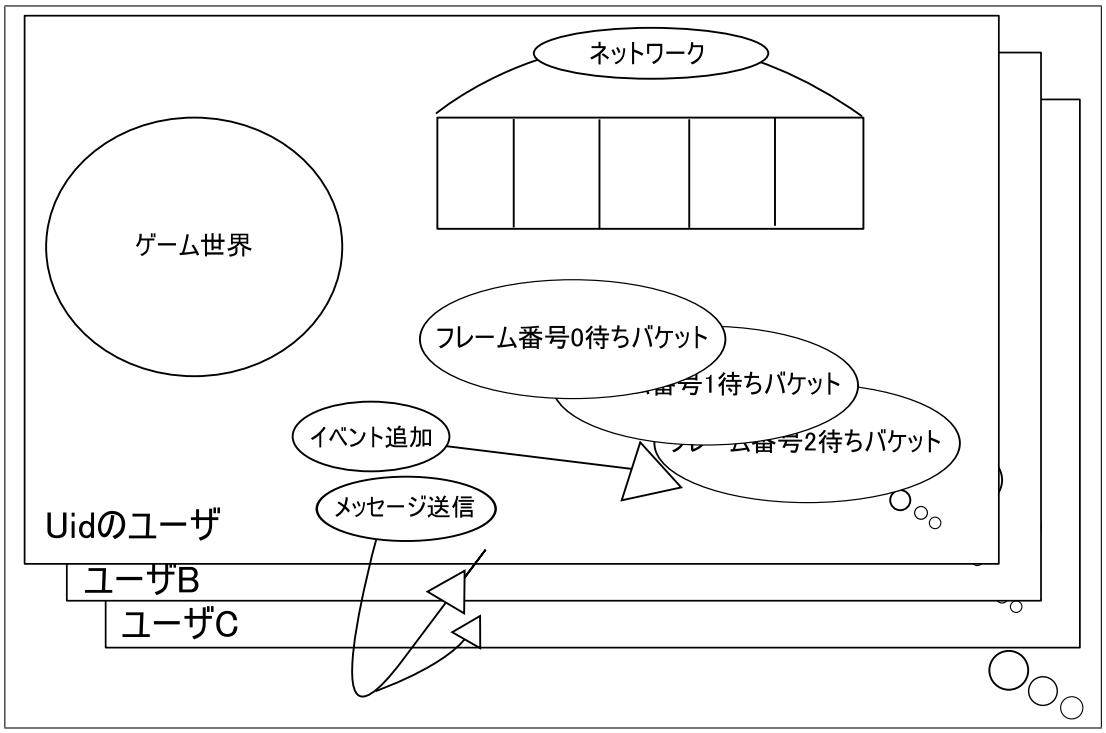

図 4.2: 遷移演算: send(s, uid, event)

```
    ) = gameworld( S, ID2, F ) if c-send( S, ID1, E ) .
ceq bucket( send( S, ID1, E ), ID2, F ) =
    (if (ID1 = ID2) and (F = frame( S, ID2 ) + wait( S ))
        then E bucket( S, ID2, F )
        else bucket( S, ID2, F )
    fi) if c-send( S, ID1, E ) .
```

図 4.2 には，send で実際に起きることを模擬的に描いた．この図で，右下への小さな円の繰り返しは，その項目がそれ以後も無数に存在することを示す．つまり，それぞれのユーザは，あるフレーム番号を待つイベントを持つバケットを無数に有しており，また，ノードもモデル全体で無数に存在しているという図になっている．

4.2.4 遷移演算: receive(s, uid, mes)

receive(s,uid,mes) は uid のユーザが mes を受信したことを示す．(図 4.3) メッセージを受信した場合，バケット同期法では Wait から Delay を引いただけのフレーム時間，待機してからメッセージのもつイベントを受理する．これは bucket(s, uid, frame(s,uid) + (wait(s) - (frame(s,uid) - sendtime(mes)))) に mes のもつイベントを追加することで表現される．send の場合と同様に，receive においても frame の値は変更しない．またここでもイベントの受理は行わないため，gameworld は変更しない．この receive には，c-receive(s,

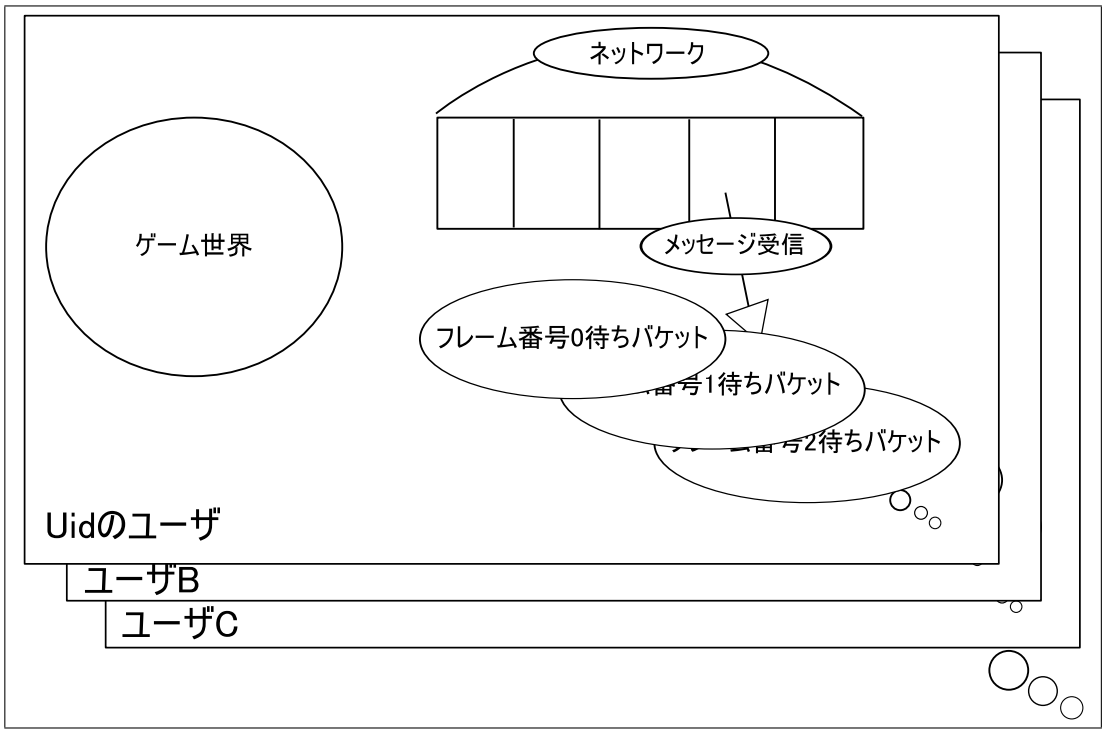

図 4.3: 遷移演算: receive(s, uid, mes)

uid, mes) という効力条件をおいた．c-receive(s, uid, mes) は，network(s, uid) に mes が含まれているという条件で，存在しないメッセージを受信しないための物である．

receive の SESSION モジュールにおける記述は以下のようになった．

```
-- c-receive
eq c-receive( S, ID1, M ) = (M \in network( S, ID1 )) .

-- receive
ceq receive( S, ID1, M ) = S if not( c-receive( S, ID1, M ) ) .
ceq wait( receive( S, ID1, M ) ) = wait( S ) if c-receive( S, ID1, M ) .
ceq frame(
       receive( S, ID1, M ),
       ID2
   ) = frame( S, ID2 ) if c-receive( S, ID1, M ) .
ceq network( receive( S, ID1, M ), ID2 ) =
   (if (ID1 = ID2)
       then remove( network( S, ID2 ), M )
       else network( S, ID2 )
   fi) if c-receive( S, ID1, M ) .
ceq gameworld(
```

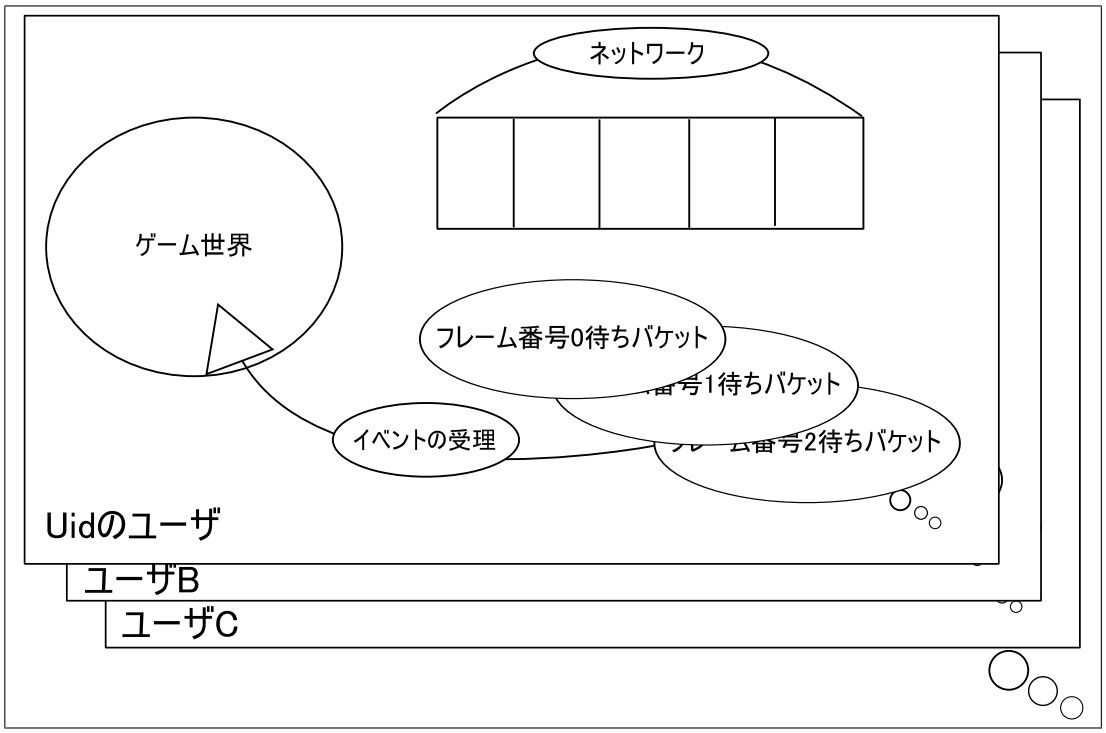

図 4.4: 遷移演算: update(s, uid)

```
        receive( S, ID1, M ),
        ID2, F
    ) = gameworld( S, ID2, F ) if c-receive( S, ID1, M ) .
ceq bucket( receive( S, ID1, M ), ID2, F ) =
    (if ((ID1 = ID2) and (F = sendtime(M) + wait(S)))
        then (event(M) bucket( S, ID2, F ))
        else bucket( S, ID2, F )
    fi)
    if c-receive( S, ID1, M ) .
```

4.2.5 遷移演算: update(s, uid)

uid のユーザがゲーム世界の更新を行ったことを示す．(図 4.4) bucket(s, uid, frame(s, uid)) を gameworld(s, uid, frame(s, uid)) に対して反映する．また，frame(s, uid) の値を次に進める．update の効力条件 c-update も常に true である．

update の SESSION モジュールにおける記述は以下の通りである．

```
-- update
ceq wait( update( S, ID1 ) ) = wait( S ) if c-update( S, ID1 ) .
```

```
ceq frame( update( S, ID1 ), ID2 ) =
    (if (ID1 = ID2)
       then s( frame( S, ID2 ) )
       else frame( S, ID2 )
    fi) if c-update( S, ID1 ) .
ceq network(
       update( S, ID1 ),
       ID2
    ) = network( S, ID2 ) if c-update( S, ID1 ) .
ceq gameworld( update( S, ID1 ), ID2, F ) =
    (if (ID1 = ID2) and (frame( S, ID2 ) = F)
       then bucket( S, ID2, F )
       else gameworld( S, ID2, F )
    fi) if c-update( S, ID1 ) .
ceq bucket(
       update( S, ID1 ),
       ID2, F
    ) = bucket( S, ID2, F ) if c-update( S, ID1 ) .
```

4.3 Delay $<$ Wait 仮定

本研究では，常にメッセージの受信に掛かるフレーム時間 Delay は，Wait 以下であると仮定し，検証を行う．Delay $<$ Wait 仮定は，より明瞭に述べると，メッセージ M が，あるユーザ A のネットワークに含まれているならば，そのユーザの現在のフレーム番号よりも，M の送信時フレーム番号と，Wait 値を足した物のほうが大きい，という仮定になる．この仮定を CafeOBJ で記述するために，これは以下の SESSION+DELAYWAIT モジュールを作成した．

```
mod! SESSION+DELAYWAIT
{
    -- imports
    pr( SESSION ) .

    -- variables
    var S : Session .
    var ID : Uid .
    var M : Message .
```

```
    -- delay<wait assumption
    ceq (frame(S,ID) < sendtime(M) + wait(S)) = true if M \in network(S,ID) .
}
```

検証はSESSION+DELAYWAITモジュールに対して行う.

第5章　バケット同期法の検証

本章では，前章までに作成したバケット同期法のモデルを用いて，バケット同期法が因果順序性を持つことを検証する．

5.1　ゲーム世界の共有と同値

第二章でゲーム世界の共有を定義した．そこではゲーム世界の共有は，ゲーム世界が全てのノードで**一貫している**という定義であった．しかし一貫しているとはどの様な物か，モデル化を行う為には明らかにしなければならない．

そこでモデル化にあたり，ゲーム世界は，ゲームの種類ごとに異なる実装が行われるが，ネットワークゲームにおけるゲーム世界は，ゲーム世界を共有するため，常に次のような性質を持つことに注目した．

1. ゲーム開始時点 (フレーム番号 0) では，全てのノードのゲーム世界は等しい．
2. 同じフレーム番号の間であれば，どの順序でイベントを反映しても，また，何度同じイベントを反映しても，同じゲーム世界に更新される．
3. 同じイベントを，同じゲーム世界に反映すると，やはり同じゲーム世界に更新される．

2 の性質で，同じイベントを何度反映してもよいとあるが，これはある行動を同じフレームの間にプレイヤーが 100 回繰り返し入力しても，実際には 1 回の行動しか発生しないためである．また，同一フレームの間に発生したイベントは，リアルタイムゲームでは同時に発生したものと処理されるため，同じフレーム番号で発生したイベントには順序は関係ない．

ところで，2 の性質を持つ物は，まさに集合として表現できる．この点を踏まえて，ゲーム世界更新情報を定義した．

定義 5.1 (ゲーム世界更新情報) あるフレーム番号で受理したイベントの集合を，ゲーム世界更新情報と呼ぶ．

ゲーム世界更新情報の同値性を調べることで，ある更新と別の更新が，同じ更新になるのかを調べることができる．

このゲーム世界更新情報を使うと，第 1 の性質よりフレーム番号 0 の時点では，常にゲーム世界は等しく，一貫していることから，結局フレーム番号 n でのゲーム世界が等し

いかは，フレーム番号 n-1 までのゲーム世界更新情報が，全て等しいかという事に帰着する．したがって，ゲーム世界の同値を以下のように定めることができる．

定義 5.2 (ゲーム世界の同値) あるユーザ a と b について，フレーム番号 *0* からフレーム番号 *n-1* までに，a が受理したゲーム世界更新情報と，b が受理したゲーム世界更新情報が等しいならば，フレーム番号 n で a と b のゲーム世界は同値であるという．

ゲーム世界の同値を用いると，ある実時刻におけるゲーム世界の共有は以下のように再定義できる．

定義 5.3 (ゲーム世界の同値を用いたゲーム世界の共有の再定義) 全てのユーザ a と b が，a も b もゲーム世界を更新済みの全てのフレーム番号 f に関して，f での a と b のゲーム世界が同値ならば，ゲーム世界は共有されているという．

したがって，これを CafeOBJ で記述すると以下の inv1 になる．

```
eq inv1( S, ID1, ID2, F ) =
   ((F < frame( S, ID1 )) and (F < frame( S, ID2 ))) implies
   (gameworld( S, ID1, F ) = gameworld( S, ID2, F )) .
```

ここで gameworld(S, IDn, F) は，ユーザ IDn のノードの元で，フレーム番号 F において受理を行ったイベントによって構成されるゲーム世界更新情報である．

因果順序性とは，全てのフレーム番号において，ゲーム世界が共有されている事であったため，検証はこの inv1 が安全性を持つことを証明する証明譜を作成することに帰着される．

5.2 証明譜の作成

実際に inv1 に対して，Session の構造に関する帰納法を用いて，実際に証明譜の作成を試みた．

Session の構造に関する帰納法を用いるため，inv1(s, id1, id2, f) に対して s が init(w) の場合を帰納基底として，send(s', uid3, ev) の場合，receive(s', uid3, mes) の場合，update(s', uid3) の場合をそれぞれ帰納段階として証明節を立てた．

このうち，init(w) の場合，send(s', uid3, ev) の場合については，すぐに有効な証明節が得られた．

receive(s', uid3, mes) の場合は，c-receive(s, uid3, mes) について条件分けを行うことで，すべての場合について有効な証明節が得られた．

update(s,uid3) の場合については，id1,id2,id3 と f の関係を考え，条件分けによる証明を試みた．その結果，id1=id3 and id2≠id3 and f=frame(s,id3) and f<frame(s,id2) の場合と，id1≠id3 and id2=id3 and f=frame(s,id3) and f<frame(s,id1) であるという条件を除いて，有効な証明節を得た．

この 2 つの場合は，それぞれ

```
((
    (f < (s f)) and
    (bucket(s,id3,f) = gameworld(s,id2,f))
) xor (
    (f < (s f)) xor true)
):Bool
```

および,

```
((
    (f < (s f)) and
    (gameworld(s,id1,f) = bucket(s,id3,f))
) xor (
    (f < (s f)) xor true)
):Bool
```

という Bool の項を返してくるため，有効な証明節にならない.

そこで補題として，以下の inv2 を立てた.

```
eq inv2( S, ID1, ID2, F ) =
    ((F = frame(S, ID1)) and (F < frame(S, ID2))) implies
    (bucket(S, ID1, F) = gameworld(S,ID2,F)) .
```

inv2 は，ユーザ ID1 のノードが，フレーム番号 F にあり，ユーザ ID2 のノードが，そのフレーム番号よりも先までゲーム世界の更新を行っているときに，ユーザ ID1 のノードが持つバケットの情報と，ユーザ ID2 のノードが F フレームで実際にゲーム世界に反映したゲーム世界更新情報が一致することを示すと考えられる補題である.

この inv2 を用いると inv1 の全てのケースで有効な証明節が得られることを確認した. したがって inv2 を証明できれば，inv1 が証明できたことになるため，inv2 に対しても構造的帰納法を用いた証明を試みることにした.

inv2 については update の条件分けで，not(id1=id3) and id2=id3 and not(f¡frame(s,id3)) and f=frame(s,id1) and f=frame(s,id3) という場合を除いて，有効な証明節を得た.

この場合は

```
((
    (f < (s f)) and
    (bucket(s,id1,f) = bucket(s,id3,f))
) xor (
    (f < (s f)) xor true)
):Bool
```

という Bool の項を CafeOBJ 処理系が返してくる.

そこで，さらに追加の補題として以下の inv3 を立てた.

```
eq inv3( S, ID1, ID2, F ) =
    ((F = frame( S, ID1 )) and (F = frame( S, ID2 ))) implies
    (bucket( S, ID1, F ) = bucket( S, ID2, F )) .
```

inv3 は，ユーザ ID1 のノードもユーザ ID2 のノードも，フレーム番号 F にあるならば，ユーザ ID1 のノードもユーザ ID2 のノードも，互いにバケットの内容が等しいということを示すと考えられる補題である．

ここでもやはり，inv3 を用いると，inv2 の全てのケースで有効な証明節が得られることを確認した．そのため，inv3 を証明することで，inv2 が証明でき，inv2 が証明できることで inv1 が証明できることになる．

そのため inv2 と同様に，この inv3 について構造的帰納法と条件分けを用い証明譜の作成を試みたところ，全ての証明節が有効な物として得られた．

これらの結果を合わせると inv1 に対する有効な証明譜が得られたことになるため，Delay $<$ Wait の仮定の下で，因果順序性の検証に成功したといえる．

第6章　結論

6.1　まとめ

本研究では，ネットワークゲームをに対するフレーム概念を中心にした観測遷移機械によるモデル化を提案し，実際にバケット同期法を導入したネットワークゲームを対象に，OTS/CafeOBJ による形式仕様記述を得た．その後，ゲーム世界の共有を得られたモデルに対して記述し，これに対する証明譜を作成した．そして，この証明譜を CafeOBJ 処理系により実行し，証明の正しさを確認した．

観測遷移機械でのモデル化では，まずユーザ，ゲーム世界などのネットワークゲーム上に登場する概念を記述した．次に，各遷移では，ある実時間が進んでいると仮定し，この更新された実時間は具体的に得られないよう記述を行った．これによって，フレームレートが不安定であり，いつ更新が発生するのか不明であるという性質を記述とした．最後に，バケット同期法の振る舞いを遷移規則としてモデル化した．

検証では，検証したい性質を CafeOBJ 上に項として記述し，この項に対する証明譜を作成，実行させ検証が行えた．

本研究で検証した性質は，Delay < Wait という仮定の下で，バケット同期法が因果順序性について，すなわち，バケット同期法を導入することで，ネットワークゲームのゲーム世界を共有できるかについてである．この検証に成功したため，Delay < Wait の仮定が補償アルゴリズムにより保証されるならば，バケット同期法を用いることで，ゲーム世界を共有できることを確認することができた．

6.2　関連研究

バケット同期法は 1998 年に Gautier によって，MiMaze というインターネット回線を通じたネットワークゲームにおいて，トランスミッションレイヤーのプロトコルとして提案され，広く用いられるようになった．バケット同期法に限らず，ネットワークゲームの因果順序性を保証するプロトコルの研究が行われている．Cronin らは，特に高速な応答が必要なネットワークゲームのための Trailing State Synchronization(TSS) プロトコルを提案している．[CKFJ04] また Ferretti らは，低速なノードが存在する可能性の高いネットワークゲームでも，高速なノードが自然なフレーム更新を行える Optimistic Obsolescence-based Synchronization(OSS) プロトコルを提案している．[FR05]

バケット同期法に対する数学的検証では，Baughman らが 2007 年にバケット同期法に推測航法 (Dead Reckoning Protocol) を補償プロトコルとして導入したプロトコルを対象として Suppress-correct cheat という不正の存在を明らかにしている．[BLL07] Suppress-correct cheat は，意図的にネットワークメッセージの送信を Delay $<$ Wait が破られるところまで遅らせることで，相手のプレイヤーの側におけるゲーム世界の更新を不安定にし，表示を破綻させる不正である．この Baughman らの研究では，結論として補償プロトコルとして推測航法の代わりに固定進行法 (Lockstep Protocol)，非同期的同期法 (Asynchronous Synchronization Protocol) などの導入を奨めている．

6.3 今後の課題

本研究の今後の課題としては，次のような事が考えられる．

- 補償プロトコルの検証 (Delay $<$ Wait 仮定の消去)
  - 本研究では，バケット同期法のみに焦点を絞って検証を行った．つまり，一般にバケット同期法は，Delay $<$ Wait 仮定があるならば，因果順序性を保証することが検証された．このような方法によって，Delay $<$ Wait であることをある補償プロトコルが保証するならば，その補償プロトコルとバケット同期法を合わせたプロトコルも因果順序性を保証することが確認された．しかし，本研究では補償プロトコルを用いれば Delay $<$ Wait が保証されることを検証していない．そのため，何らかの具体的な補償プロトコルが Delay $<$ Wait を保証することを検証しなければならない．
- セッションへ参加するプレイヤー数の増加・減少への対応
  - 本研究では，セッションに参加しているプレイヤー数は，開始時から増加も減少もしないという仮定でバケット同期法を検証している．このような仮定を満たすネットワークゲームは実際に多く存在しているため，決して無理な仮定ではない．しかし，プレイヤー数の増減があるゲームもやはり存在しており，増減がありうるという一般的な仮定の下で検証を行う事が望ましい．

謝辞

本研究を進めるにあたり，ご指導賜りました二木厚吉教授に厚くお礼を申し上げます．また，有益な助言を頂いた同大学院 青木利晃准教授，緒方和博准教授，千葉勇輝助教に感謝の意を示します．また，本研究を進めるために，相談にのって頂いた，二木研究室　有本泰仁さん，大井聡史さん，TRINH, BAO NGOC QUOC さん，青木研究室　谷崎裕明さん，波多野秀行さん，陳江さんに感謝いたします．最後に，研究に関する議論にも応じてくださった，二木研究室，青木研究室，緒方研究室の皆さんにも感謝いたします．

参考文献

[BK99] T. Bova and T. Krivoruchka. Reliable udp protocol. *Network Working Group INTERNET-DRAFT*, Feburary 1999.

[BLL07] N. E. Baughman, M. Liberatore, and B. N. Levine. Cheat-proof playout for centralized and peer-to-peer gaming. *IEEE/ACM Trans. Netw.*, Vol. 15, No. 1, pp. 1–13, 2007.

[CKFJ04] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools and Applications*, Vol. 23, No. 1, pp. 7–30, 2004.

[FR05] S. Ferretti and M. Roccetti. Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games. In *Proc. of Advances in computer entertainment technology*, pp. 405–412. ACM, 2005.

[Fut06] K. Futatsugi. Verifying specifications with proof scores in CafeOBJ. In *Proc. of Advances in computer entertainment technology*, pp. 3–10, 2006.

[GD98] L. Gautier and C. Diot. Design and evaluation of MiMaze, a multi-player game on the internet. In *Proc. of IEEE Multimedia Systems Conference*, pp. 233–236, 1998.

[MC05] David R. Michael and Sandra L. Chen. *Serious Games: Games That Educate, Train, and Inform.* Muska & Lipman/Premier-Trade, 2005.

[NNS03] E. Najm, U. Nestmann, and P. Stevens, editors. *Formal methods for open object-based distributed systems*, Vol. 2884 of *LNCS*. Springer, 2003.

[OF03] K. Ogata and K. Futatsugi. Proof scores in the OTS/CafeOBJ method. In *Proc. of Formal Methods for Open Object-Based Distributed Systems*, pp. 170–184, 2003.

[PW02] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '02, pp. 23–29, New York, NY, USA, 2002. ACM.

付録A　バケット同期法モデルの記述

A.1　PNAT.mod

```
-- PNAT Model

mod! PNAT principal-sort Nat
{
    [ Zero NzNat < Nat ]

    op 0 : -> Zero {constr} .
    op s _ : Nat -> NzNat {constr} .

    pred _ = _ : Nat Nat { comm } .
    pred _ < _ : Nat Nat .
    op _ + _ : Nat Nat -> Nat { assoc comm } .
    op sd : Nat Nat -> Nat { comm } .

    vars N M : Nat .
    var NN : NzNat .

    -- + definitions
    eq s(N) + M = s(N + M) .
    -- 0 is left idempotency of +
    eq 0 + N = N .
    -- 0 is right idempotency of +
    eq N + 0 = N .

    -- = definitions
    eq (NN = 0) = false .
    eq (N = N) = true .
    eq (s(N) = N) = false .
    eq (s(N) = s(M)) = (N = M) .
```

```
    -- < definitions
    eq N < N = false .
    eq 0 < s(N) = true .
    eq s(N) < 0 = false .
    eq s(N) < s(M) = N < M .
}
```

A.2 PNAT-thm0.mod

```
mod PNAT-THM0
{
    pr( PNAT ) .

    vars N : Nat .
    op thm0 : Nat -> Bool .
    eq thm0(N) = (not(0 = N) implies (0 < N)) .
}

mod PNAT-THM0-ISTEP {
    pr( PNAT-THM0 ) .

    ops n n' : -> Nat .

    op thm0-istep : -> Bool .
    eq thm0-istep = thm0(n) implies thm0(n') .
}

in PNAT-thm0-proof.mod

mod! PNAT+thm0
{
    pr( PNAT ) .

    vars N : Nat .

    -- thm0
    ceq (0 < N) = true if not(0 = N) .
}
```

A.3 PNAT-thm0-proof.mod

```
--> Proof PNAT-thm-0

--> ind on n
--> [thm0-base]
open PNAT-THM0 .
   op n : -> Nat .
-- ind. base case: 0
   eq n = 0 .

   red thm0(n) .
close .

--> [thm0-istep]
open PNAT-THM0-ISTEP .
-- ind. case: s(n)
    eq n' = s(n) .

    red thm0-istep .
close .

--> Q.E.D.
```

A.4 PNAT-thm1.mod

```
mod PNAT-THM1 {
    pr( PNAT+thm0 ) .

    vars N M : Nat .
    op thm1 : Nat Nat -> Bool .
    eq thm1(N, M) = ((N = M) implies not(N < M)) .
}

mod PNAT-THM1-ISTEP {
    pr( PNAT-THM1 ) .

    ops n n' : -> Nat .
    var M : Nat .
```

```
    op thm1-istep : Nat -> Bool .
    eq thm1-istep( M ) = thm1(n, M) implies thm1(n', M) .
}

in PNAT-thm1-proof.mod

mod! PNAT+thm1
{
    pr( PNAT+thm0 ) .

    vars N M : Nat .

    -- thm1
    eq ((N = M) implies not(N < M)) = true .
}
```

A.5 PNAT-thm1-proof.mod

```
--> Proof PNAT-thm-1

--> ind on n
--> [thm1-base]
open PNAT-THM1 .
   op n : -> Nat .
   op m : -> Nat .
-- ind. base case: 0
   eq n = 0 .
-- pre conditions
    eq 0 = m .

   red thm1(n, m) .
close .

--> [thm1-istep]
open PNAT-THM1-ISTEP .
    op m : -> Nat .
-- ind. case: s(n)
    eq n' = s(n) .
```

```
-- pre conditions
    eq s(n) = m .

    red thm1-istep( m ) .
close .

--> Q.E.D.
```

A.6 PNAT-thm2.mod

```
mod PNAT-THM2 {
    pr( PNAT+thm1 ) .

    vars N M : Nat .
    op thm2 : Nat Nat -> Bool .
    eq thm2(N, M) =
        (not(N = M) and not(N < M) and not(N = s(M))) implies
        not(N < s(M)) .
}

mod PNAT-THM2-ISTEP {
    pr( PNAT-THM2 ) .

    ops n n' : -> Nat .
    var M : Nat .

    op thm2-istep : Nat -> Bool .
    eq thm2-istep( M ) = thm2(n, M) implies thm2(n', M) .
}

-- proof
in PNAT-thm2-proof.mod

-- define new module
mod! PNAT+thm1+thm2
{
    pr( PNAT+thm1 ) .

    vars N M : Nat .
```

```
    -- thm2
    ceq N < s(M) = false if not(N = M) and not(N < M) and not(N = s(M)) .
}
```

A.7 PNAT-thm2-proof.mod

```
--> Proof PNAT-thm-2

--> ind on n
--> [thm2-base]
open PNAT-THM2 .
   op n : -> Nat .
   op m : -> Nat .
-- ind. base case: 0
   eq n = 0 .
-- pre conditions
    eq (m = 0) = false .

   red thm2(n, m) .
close .

--> [thm2-istep]
open PNAT-THM2-ISTEP .
    op m : -> Nat .
-- ind. case: n
    eq n' = s(n) .
-- pre conditions
    eq (s(n) = m) = false .
    eq (n < m) = false .
    eq (s(s(n)) < m) = false .

    red thm2-istep( m ) .
close .

--> Q.E.D.
```

A.8 PNAT-thm3.mod

```
mod PNAT-THM3 {
    pr( PNAT+thm1+thm2 ) .

    var N : Nat .
    var NN : NzNat .
    op thm3 : Nat NzNat -> Bool .
    eq thm3(N, NN) = not((N + NN) = N) .
}

mod PNAT-THM3-ISTEP {
    pr( PNAT-THM3 ) .

    ops n n' : -> Nat .
    var NN : NzNat .

    op thm3-istep : NzNat -> Bool .
    eq thm3-istep( NN ) = thm3(n, NN) implies thm3(n', NN) .
}

in PNAT-thm3-proof.mod

mod! PNAT+thm1+thm2+thm3
{
    pr( PNAT+thm1+thm2 ) .

    var N : Nat .
    var NN : NzNat .

    eq ((N + NN) = N) = false .
}
```

A.9 PNAT-thm3-proof.mod

```
--> Proof PNAT-thm-3

--> ind on n
--> [thm3-base]
open PNAT-THM3 .
   op n : -> Nat .
```

```
   op m : -> NzNat .

   eq n = 0 .

   red thm3(n, m) .
close .

--> [thm3-istep]
open PNAT-THM3-ISTEP .
    op m : -> NzNat .
    eq n' = s(n) .
-- assumptions

    red thm3-istep( m ) .
close .


--> Q.E.D.
```

A.10 PNAT-thm4.mod

```
mod PNAT-THM4 {
    pr( PNAT+thm1+thm2+thm3 ) .

    vars N M : Nat .
    op thm4 : Nat Nat -> Bool .
    eq thm4(N, M) = (N < M) implies not(N = M) .
}

mod PNAT-THM4-ISTEP {
    pr( PNAT-THM4 ) .

    ops n n' : -> Nat .

    var M : Nat .

    op thm4-istep : Nat -> Bool .
    eq thm4-istep( M ) = thm4(n, M) implies thm4(n', M) .
}
```

```
in PNAT-thm4-proof.mod

mod! PNAT+thm1+thm2+thm3+thm4
{
    pr( PNAT+thm1+thm2+thm3 ) .

    vars N M : Nat .

    eq ((N < M) implies not(N = M)) = true .
}
```

A.11 PNAT-thm4-proof.mod

```
--> Proof PNAT-thm-4

--> ind on n
--> [thm4-base, m=0]
open PNAT-THM4 .
   ops n m : -> Nat .
-- ind. base case: 0
   eq n = 0 .
-- case split assumptions
    eq m = 0 .

   red thm4(n, m) .
close .

--> [thm4-base, ~(m=0)]
open PNAT-THM4 .
   ops n m : -> Nat .
-- ind. base case: 0
   eq n = 0 .
-- pre conditions
    eq (m = 0) = false .

   red thm4(n, m) .
close .
```

```
--> [thm4-istep, s(n)=m]
open PNAT-THM4-ISTEP .
    op m : -> Nat .
-- ind. case: s(n)
    eq n' = s(n) .
-- pre conditions
    eq (n < m) = true .
    eq (s(n) < m) = true .
-- case split assumptions
    eq s(n) = m .

    red thm4-istep( m ) .
close .

--> [thm4-istep, ~(s(n)=m)]
open PNAT-THM4-ISTEP .
    op m : -> Nat .
-- ind. case: s(n)
    eq n' = s(n) .
-- pre conditions
    eq (n < m) = true .
    eq (s(n) < m) = true .
-- case split assumptions
    eq (s(n) = m) = false .

    red thm4-istep( m ) .
close .

--> Q.E.D.
```

A.12 PNAT+theorems.mod

```
-- PNAT Model

mod! PNAT+theorems
{
    -- imports
    pr( PNAT+thm1+thm2+thm3+thm4 ) .
}
```

A.13 bucketsync.mod

```
-- Bucket Synchronization Models
mod! FRAME
{
    -- imports
    pr( PNAT+theorems *{
            sort Nat -> Frame,
            sort NzNat -> NzFrame,
            sort Zero -> ZeroFrame
        } ) .
}

mod* EVENT
{
    [ Event ]
    pred _=_ : Event Event {comm} .
    var E : Event .
    eq (E = E) = true .
}

mod* UID
{
    [ Uid ]
    pred _=_ : Uid Uid {comm} .
    eq (U:Uid = U) = true .
}

mod! EVENTSET
{
    -- imports
    pr( EVENT ) .

    -- sorts
    [ Event < EventSet ]

    -- operators
    op empty : -> EventSet {constr} .
```

```
    op _ _ : EventSet EventSet -> EventSet {constr assoc comm} .

    pred _ \in _ : Event EventSet .
    pred _ issubset _ : EventSet EventSet .
    pred _=_ : EventSet EventSet {comm} .

    -- variables
    vars AE AE' : EventSet .
    vars E E' : Event .

    -- equations : _ _
    eq empty AE = AE .
    eq AE AE = AE .

    -- equations : _ \in _
    eq E \in empty = false .
    eq E \in E' = (E = E') .
    eq E \in (E' AE) = (E = E') or (E \in AE) .

    -- equations : _ issubset _
    eq (empty issubset AE') = true .
    eq (AE issubset AE) = true .
    eq ((E AE) issubset AE') = (E \in AE') and (AE issubset AE') .

    -- equations : _ = _
    eq (AE = AE) = true .
}

mod! MESSAGE
{
    -- imports
    pr( FRAME ) .
    pr( EVENT ) .

    -- sorts
    [ Message ]

    -- operators
```

```
    op pack : Event Frame -> Message {constr} .
    op event : Message -> Event .
    op sendtime : Message -> Frame .

    pred _=_ : Message Message {comm} .

    -- variables
    vars M1 M2 : Message .

    -- equations
    eq event( pack( E:Event, T:Frame ) ) = E .
    eq sendtime( pack( E:Event, T:Frame ) ) = T .

    eq (M1 = M2) = ((event(M1) = event(M2)) and (sendtime(M1) = sendtime(M2))) .
}

mod! NETWORK
{
    -- imports
    pr( MESSAGE ) .

    -- sorts
    [ NilNetwork NnNetwork < Network ]

    -- operators
    op nil : -> NilNetwork {constr} .
    op _ | _ : Message Network -> NnNetwork {constr} .

    op _ \in _ : Message Network -> Bool .
    op remove : Network Message -> Network .

    -- variables
    vars N1 N2 : Network .
    vars M1 M2 : Message .

    -- equations : _ \in _
    eq M1 \in nil = false .
    eq M1 \in (M2 | N2) = (M1 = M2) or M1 \in N2 .
```

```
    -- equations : remove(_, _)
    eq remove( nil, M2 ) = nil .
    ceq remove( (M1 | N1), M2 ) = remove(N1, M2) if M1 = M2 .
    ceq remove( (M1 | N1), M2 ) = M1 | remove(N1, M2) if not(M1 = M2) .
}

mod! SESSION
{
    -- imports
    pr( FRAME ) .
    pr( UID ) .
    pr( EVENT ) .
    pr( EVENTSET ) .
    pr( MESSAGE ) .
    pr( NETWORK ) .

    -- hidden sort
    *[Session]*

    -- initial state
    op init : NzFrame -> Session {constr} .

    -- actions
    bop send : Session Uid Event -> Session {constr} .
    bop receive : Session Uid Message -> Session {constr} .
    bop update : Session Uid -> Session {constr} .

    -- observations
    bop wait : Session -> NzFrame .
    bop frame : Session Uid -> Frame .
    bop network : Session Uid -> Network .
    bop gameworld : Session Uid Frame -> EventSet .
    bop bucket : Session Uid Frame -> EventSet .

    -- effective conditions
    pred c-send : Session Uid Event .
    pred c-receive : Session Uid Message .
```

```
pred c-update : Session Uid .

-- variables
var S : Session .
var F : Frame .
var W : NzFrame .
var E : Event .
var M : Message .
vars ID1 ID2 : Uid .
-- equations
    -- init
    eq wait( init( W ) ) = W .
    eq frame( init( W ), ID1 ) = 0 .
    eq network( init( W ), ID2 ) = nil .
    eq gameworld( init( W ), ID2, F ) = empty .
    eq bucket( init( W ), ID2, F ) = empty .

    -- c-send
    eq c-send( S, ID1, E ) = true . -- anytime, send is happen.

    -- send
    ceq wait( send( S, ID1, E ) ) = wait( S ) if c-send( S, ID1, E ) .
    ceq frame( send( S, ID1, E ), ID2 ) =
           frame( S, ID2 ) if c-send( S, ID1, E ) .
    ceq network( send( S, ID1, E ), ID2 ) =
        (if ID1 = ID2
           then network( S, ID1 )
           else pack( E, frame( S, ID1 ) ) | network( S, ID2 )
        fi) if c-send( S, ID1, E ) .
    ceq gameworld(
           send( S, ID1, E ),
           ID2, F
        ) = gameworld( S, ID2, F ) if c-send( S, ID1, E ) .
    ceq bucket( send( S, ID1, E ), ID2, F ) =
        (if (ID1 = ID2) and (F = frame( S, ID2 ) + wait( S ))
           then E bucket( S, ID2, F )
           else bucket( S, ID2, F )
        fi) if c-send( S, ID1, E ) .
```

```
-- c-receive
eq c-receive( S, ID1, M ) = (M \in network( S, ID1 )) .

-- receive
ceq receive( S, ID1, M ) = S if not( c-receive( S, ID1, M ) ) .
ceq wait( receive( S, ID1, M ) ) = wait( S ) if c-receive( S, ID1, M ) .
ceq frame(
       receive( S, ID1, M ),
       ID2
    ) = frame( S, ID2 ) if c-receive( S, ID1, M ) .
ceq network( receive( S, ID1, M ), ID2 ) =
    (if (ID1 = ID2)
       then remove( network( S, ID2 ), M )
       else network( S, ID2 )
    fi) if c-receive( S, ID1, M ) .
ceq gameworld(
       receive( S, ID1, M ),
       ID2, F
    ) = gameworld( S, ID2, F ) if c-receive( S, ID1, M ) .
ceq bucket( receive( S, ID1, M ), ID2, F ) =
    (if ((ID1 = ID2) and (F = sendtime(M) + wait(S)))
       then (event(M) bucket( S, ID2, F ))
       else bucket( S, ID2, F )
    fi)
    if c-receive( S, ID1, M ) .

-- c-update
eq c-update( S, ID1 ) = true . -- anytime, update is happen.

-- update
ceq wait( update( S, ID1 ) ) = wait( S ) if c-update( S, ID1 ) .
ceq frame( update( S, ID1 ), ID2 ) =
    (if (ID1 = ID2)
       then s( frame( S, ID2 ) )
       else frame( S, ID2 )
    fi) if c-update( S, ID1 ) .
ceq network(
```

```
            update( S, ID1 ),
            ID2
        ) = network( S, ID2 ) if c-update( S, ID1 ) .
    ceq gameworld( update( S, ID1 ), ID2, F ) =
        (if (ID1 = ID2) and (frame( S, ID2 ) = F)
            then bucket( S, ID2, F )
            else gameworld( S, ID2, F )
        fi) if c-update( S, ID1 ) .
    ceq bucket(
            update( S, ID1 ),
            ID2, F
        ) = bucket( S, ID2, F ) if c-update( S, ID1 ) .
}

mod! SESSION+DELAYWAIT
{
    -- imports
    pr( SESSION ) .

    -- variables
    var S : Session .
    var ID : Uid .
    var M : Message .

    -- delay<wait assumption
    ceq (frame(S,ID) < sendtime(M) + wait(S)) = true if M \in network(S,ID) .
}
```

A.14 inv1.mod

```
mod INV1
{
    pr( SESSION+DELAYWAIT+INV3+INV2 ) .

    op inv1 : Session Uid Uid Frame -> Bool .
    var S : Session .
    var F : Frame .
    vars ID1 ID2 : Uid .
```

```
    -- invariant candidate
    eq inv1( S, ID1, ID2, F ) =
        ( (F < frame( S, ID1 )) and
            (F < frame( S, ID2 ))
        ) implies
        (gameworld( S, ID1, F ) = gameworld( S, ID2, F )) .
}

mod INV1-ISTEP
{
    pr( INV1 ) .
    ops s s' : -> Session .
    op inv1-istep : Uid Uid Frame -> Bool .

    var F : Frame .
    vars ID1 ID2 : Uid .
    eq inv1-istep( ID1, ID2, F ) =
        inv1( s, ID1, ID2, F ) implies
        inv1( s', ID1, ID2, F ) .
}

in inv1-proof.mod
```

A.15 inv1-proof.mod

```
--> Proof inv1


--> [1-init]
open INV1 .
    op w : -> NzFrame .
    op f : -> Frame .
    ops id1 id2 : -> Uid .
-- ind. base case: init
    op s : -> Session .
    eq s = init(w) .

    red inv1( init( w ), id1, id2, f ) .
close .
```

```
--> [1-send]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op e : -> Event .
-- ind. case: send
    eq s' = send( s, id3, e ) .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-receive, c-receive]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .
-- ind. case: receive
    eq s' = receive( s, id3, m ) .
-- case split assumptions
    eq c-receive( s, id3, m ) = true .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-receive, ~c-receive]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .
-- ind. case: receive
    eq s' = receive( s, id3, m ) .
-- case split assumptions
    eq c-receive( s, id3, m ) = false .
```

```
    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, id1=id3, id2=id3]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq id1 = id3 .
    eq id2 = id3 .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, id1=id3, ~(id2=id3), f=frame(s,id3), f<frame(s,id2)]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq id1 = id3 .
    eq (id3 = id2) = false .
    eq frame(s,id3) = f .
    eq (f < frame(s,id2)) = true .
-- from inv2
    -- eq inv2(s, id3, id2, f) = true .
    eq bucket(s, id3, f) = gameworld(s, id2, f) .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, id1=id3, ~(id2=id3), f=frame(s,id3), ~(f<frame(s,id2))]
```

```
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq id1 = id3 .
    eq (id3 = id2) = false .
    eq frame(s,id3) = f .
    eq (f < frame(s,id2)) = false .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, id1=id3, ~(id2=id3), ~(f=frame(s,id3)), f<frame(s,id3)]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq id1 = id3 .
    eq (id3 = id2) = false .
    eq (f = frame(s,id3)) = false .
    eq (f < frame(s, id3)) = true .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, id1=id3, ~(id2=id3),
--> ~(f=frame(s,id3)), ~(f<frame(s,id3)), f=s frame(s,id3)]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
```

```
-- assumptions
    eq id1 = id3 .
    eq (id3 = id2) = false .
    eq (f = frame(s,id3)) = false .
    eq (f < frame(s,id3)) = false .
    eq f = s frame(s,id3) .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, id1=id3, ~(id2=id3),
--> ~(f=frame(s,id3)), ~(f<frame(s,id3)), ~(f=s frame(s,id3))]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- assumptions
    eq id1 = id3 .
    eq (id3 = id2) = false .
    eq (f = frame(s,id3)) = false .
    eq (f < frame(s,id3)) = false .
    eq (f = s frame(s,id3)) = false .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, ~(id1=id3), id2=id3, f=frame(s,id3), f<frame(s,id1)]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq (id3 = id1) = false .
    eq id2 = id3 .
```

```
    eq frame(s,id3) = f .
    eq (f < frame(s,id1)) = true .
-- from inv2
    -- eq inv2(s, id3, id1, f) = true .
    eq bucket(s, id3, f) = gameworld(s, id1, f) .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, ~(id1=id3), id2=id3, ~(f=frame(s,id3)), ~(f<frame(s,id1))]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq (id3 = id1) = false .
    eq id2 = id3 .
    eq frame(s,id3) = f .
    eq (f < frame(s,id1)) = false .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, ~(id1=id3), id2=id3, ~(f=frame(s,id3)), f<frame(s,id3)]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq (id3 = id1) = false .
    eq id2 = id3 .
    eq (f = frame(s,id3)) = false .
    eq (f < frame(s,id3)) = true .
```

```
    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, ~(id1=id3), id2=id3,
--> ~(f=frame(s,id3)), ~(f<frame(s,id3)),f=s frame(s,id3)]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq (id3 = id1) = false .
    eq id2 = id3 .
    eq (f = frame(s,id3)) = false .
    eq (f < frame(s,id3)) = false .
    eq f = s frame(s,id3) .

    red inv1-istep( id1, id2, f ) .
close .


--> [1-update, ~(id1=id3), id2=id3,
--> ~(f=frame(s,id3)), ~(f<frame(s,id3)),~(f=s frame(s,id3))]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq (id3 = id1) = false .
    eq id2 = id3 .
    eq (f = frame(s,id3)) = false .
    eq (f < frame(s,id3)) = false .
    eq (f = s frame(s,id3)) = false .

    red inv1-istep( id1, id2, f ) .
close .
```

```
--> [1-update, ~(id1=id3), ~(id2=id3)]
open INV1-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
-- ind. case: update
    eq s' = update( s, id3 ) .
-- case split assumptions
    eq (id3 = id1) = false .
    eq (id3 = id2) = false .

    red inv1-istep( id1, id2, f ) .
close .
```

A.16 inv2.mod

```
mod INV2
{
    pr( SESSION+DELAYWAIT+INV3 ) .

    op inv2 : Session Uid Uid Frame -> Bool .
    var S : Session .
    var F : Frame .
    vars ID1 ID2 : Uid .

    -- invariant candidate
    eq inv2( S, ID1, ID2, F ) =
        ((F = frame( S, ID1 )) and (F < frame( S, ID2 ))) implies
        (bucket( S, ID1, F ) = gameworld( S, ID2, F )) .
}

mod INV2-ISTEP
{
    pr( INV2 ) .
    ops s s' : -> Session .
    op inv2-istep : Uid Uid Frame -> Bool .

    var F : Frame .
```

```
    vars ID1 ID2 : Uid .
    eq inv2-istep( ID1, ID2, F ) =
        inv2( s, ID1, ID2, F ) implies
        inv2( s', ID1, ID2, F ) .
}

in inv2-proof.mod

mod! SESSION+DELAYWAIT+INV3+INV2
{
    pr( SESSION+DELAYWAIT+INV3 ) .

    var S : Session .
    vars ID1 ID2 : Uid .
    var F : Frame .

    eq (((F = frame(S, ID1)) and (F < frame(S, ID2))) implies
        (bucket(S, ID1, F) = gameworld(S, ID2, F))) = true .
}
```

A.17 inv2-proof.mod

```
--> Proof inv2

--> [2-init]
open INV2 .
    op w : -> NzFrame .
    op f : -> Frame .
    ops id1 id2 : -> Uid .

    -- |=
    red inv2( init( w ), id1, id2, f ) .
close .

--> [2-send, id1=id3]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op e : -> Event .
```

```
    eq s' = send( s, id3, e ) .
-- assumptions
    eq id3 = id1 .
    eq (f = frame(s,id1) + wait(s)) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-send, ~(id1=id3)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op e : -> Event .

    eq s' = send( s, id3, e ) .
-- assumptions
    eq (id1 = id3) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .


--> [2-receive, c-receive, id1=id3, f=sendtime(m)+wait(s)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- pre conditions
    eq frame(s,id1) = f . -- PropA
    eq (f < frame(s,id2)) = true .
-- assumptions
    eq c-receive(s, id3, m) = true .
    eq id1 = id3 . -- PropB
```

```
    eq sendtime(m) + wait(s) = f .

    -- from PropA, B
    eq frame(s,id3) = f .

-- from delaywait assumption
-- eq (c-receive(s,id3,m) implies
-- (frame(s,id3) < sendtime(m) + wait(s))) = true .
-- Move to RHS of |=

    -- |=
    red (c-receive(s,id3,m) implies (frame(s,id3) < sendtime(m) + wait(s)))
       implies inv2-istep( id1, id2, f ) .
close .

--> [2-receive, c-receive, id1=id3, ~(f=sendtime(m)+wait(s))]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- pre conditions
    eq frame(s,id1) = f .
    eq (f < frame(s,id2)) = true .
-- assumptions
    eq c-receive( s, id3, m ) = true .
    eq id1 = id3 .
    eq (f = sendtime(m) + wait(s)) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .


--> [2-receive, c-receive, ~(id1=id3)]
open INV2-ISTEP .
    op f : -> Frame .
```

```
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- assumptions
    eq c-receive( s, id3, m ) = true .

    eq (id3 = id1) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-receive, ~c-receive]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- assumptions
    eq c-receive( s, id3, m ) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, id1=id3, id2=id3, f=s frame(s,id3)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
    eq id1 = id3 .
    eq id2 = id3 .
    eq f = s frame(s,id3) .
```

```
    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, id1=id3, id2=id3, ~(f=s frame(s,id3))]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
    eq id1 = id3 .
    eq id2 = id3 .
    eq (f = s frame(s,id3)) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, id1=id3, ~(id2=id3), f=frame(s,id3)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
    eq id1 = id3 .
    eq (id2 = id3) = false .
    eq f = frame(s,id3) .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, id1=id3, ~(id2=id3), ~(f=frame(s,id3)), f=s frame(s,id3)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
```

```
    eq s' = update( s, id3 ) .
-- assumptions
    eq id1 = id3 .
    eq (id2 = id3) = false .
    eq (frame(s,id3) = f) = false .
    eq s frame(s,id3) = f .
-- from PropA, B
    eq (f < frame(s,id2)) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .


--> [2-update, id1=id3, ~(id2=id3), ~(f=frame(s,id3)), ~(f=s frame(s,id3))]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
    eq id1 = id3 .
    eq (id2 = id3) = false .
    eq (f = frame(s,id3)) = false .
    eq (f = s frame(s,id3)) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, ~(id1=id3), id2=id3, f<frame(s,id3)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
```

```
    eq (id1 = id3) = false .
    eq id2 = id3 .

    eq (f < frame(s,id3)) = true . -- PropA
-- from PNAT-THM1
    eq (f < s frame(s,id3)) = true .
-- from PropA
    eq (frame(s,id3) = f) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, ~(id1=id3), id2=id3,
--> ~(f<frame(s,id3)), f=frame(s,id1), f=frame(s,id3)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
    eq (id1 = id3) = false .
    eq id2 = id3 .
    eq (f < frame(s,id3)) = false .
    eq frame(s,id1) = f .
    eq frame(s,id3) = f .

-- from INV3
    eq bucket(s,id1,f) = bucket(s,id3,f) .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, ~(id1=id3), id2=id3,
--> ~(f<frame(s,id3)), f=frame(s,id1), ~(f=frame(s,id3))]
open INV2-ISTEP .
    op f : -> Frame .
```

```
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
    eq (id1 = id3) = false .
    eq id2 = id3 .
    eq (f < frame(s,id3)) = false . -- PropA
    eq frame(s,id1) = f .
    eq (frame(s, id3) = f) = false . -- PropB
-- from PropA, B
    eq f < s frame(s,id3) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, ~(id1=id3), id2=id3, ~(f<frame(s,id3)), ~(f=frame(s,id1))]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- assumptions
    eq (id1 = id3) = false .
    eq id2 = id3 .
    eq (f < frame(s,id3)) = false .
    eq (frame(s,id1) = f) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .

--> [2-update, ~(id1=id3), ~(id2=id3)]
open INV2-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
```

```
-- assumptions
    eq (id1 = id3) = false .
    eq (id2 = id3) = false .

    -- |=
    red inv2-istep( id1, id2, f ) .
close .
```

A.18 inv3.mod

```
mod INV3
{
    pr( SESSION+DELAYWAIT ) .

    op inv3 : Session Uid Uid Frame -> Bool .
    var S : Session .
    var F : Frame .
    vars ID1 ID2 : Uid .

    -- invariant candidate
    eq inv3( S, ID1, ID2, F ) =
        ((F = frame( S, ID1 )) and (F = frame( S, ID2 ))) implies
        (bucket( S, ID1, F ) = bucket( S, ID2, F )) .
}

mod INV3-ISTEP
{
    pr( INV3 ) .
    ops s s' : -> Session .
    op inv3-istep : Uid Uid Frame -> Bool .

    var F : Frame .
    vars ID1 ID2 : Uid .
    eq inv3-istep( ID1, ID2, F ) =
        inv3( s, ID1, ID2, F ) implies
        inv3( s', ID1, ID2, F ) .
}

in inv3-proof.mod
```

```
mod! SESSION+DELAYWAIT+INV3
{
    pr( SESSION+DELAYWAIT ) .
    var S : Session .
    vars ID1 ID2 : Uid .
    var F : Frame .

    eq (
        (F = frame(S, ID1) and F = frame(S, ID2))
            implies
        (bucket(S, ID1, F) = bucket(S, ID2, F))) = true .
}
```

A.19 inv3-proof.mod

```
--> [3-init]
open INV3 .
    op w : -> NzFrame .
    op f : -> Frame .
    ops id1 id2 : -> Uid .

    red inv3( init(w), id1, id2, f ) .
close .

--> [3-send, id1=id3, id2=id3]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op e : -> Event .

    eq s' = send( s, id3, e ) .

-- pre conditions
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
-- assumptions
    eq id1 = id3 .
    eq id2 = id3 .
```

```
    red inv3-istep( id1, id2, f ) .
close .

--> [3-send, id1=id3, ~(id2=id3)]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op e : -> Event .

    eq s' = send( s, id3, e ) .

-- pre conditions
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
-- assumptions
    eq id3 = id1 .
    eq (id2 = id3) = false .

    red inv3-istep( id1, id2, f ) .
close .

--> [3-send, ~(id1=id3), id2=id3]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op e : -> Event .

    eq s' = send( s, id3, e ) .

-- pre conditions
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
-- assumptions
    eq (id1 = id3) = false .
    eq id3 = id2 .

    red inv3-istep( id1, id2, f ) .
```

```
close .

--> [3-send, ~(id1=id3), ~(id2=id3)]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op e : -> Event .

    eq s' = send( s, id3, e ) .

-- pre conditions
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
-- assumptions
    eq (id1 = id3) = false .
    eq (id2 = id3) = false .

    red inv3-istep( id1, id2, f ) .
close .


--> [3-receive, c-receive, id1=id3, id2=id3]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- pre conditons
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
-- case split assumptions
    eq c-receive(s,id3,m) = true .
    eq id1 = id3 .
    eq id2 = id3 .

    red inv3-istep( id1, id2, f ) .
close .
```

```
--> [3-receive, c-receive, id1=id3, ~(id2=id3)]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- pre conditons
    eq frame(s,id1) = f . -- PropA
    eq frame(s,id2) = f .
-- case split assumptions
    eq c-receive(s,id3,m) = true .
    eq id1 = id3 . -- PropB
    eq (id2 = id3) = false .

    -- from PropA, B
    eq frame(s, id3) = f .

-- from delay-wait assumptions
    -- eq delaywait(s, id3, m, f) = true .
-- from def of delaywait
    -- eq (c-receive(s,id3,m) implies
    -- (frame(s,id3) < sendtime(m) + wait(s))) = true .
-- from c-receive(s,id3,m) = true
    -- eq (frame(s,id3) < sendtime(m) + wait(s)) = true .
-- from frame(s,id3) = f
    -- eq (f < sendtime(m) + wait(s)) = true .
-- from PNAT-thm4
    eq (f = sendtime(m) + wait(s)) = false .

    red inv3-istep( id1, id2, f ) .
close .


--> [3-receive, c-receive, ~(id1=id3), id2=id3]
open INV3-ISTEP .
```

```
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- pre conditons
    eq frame(s,id1) = f .
    eq frame(s,id2) = f . -- PropA
-- case split assumptions
    eq c-receive(s,id3,m) = true .
    eq (id1 = id3) = false .
    eq id2 = id3 . -- PropB

    -- from PropA, B
    eq frame(s,id3) = f .

-- from delay-wait assumptions
    -- eq delaywait(s, id3, m, f) = true .
-- from def of delaywait
    -- eq (c-receive(s,id3,m) implies
    -- (frame(s,id3) < sendtime(m) + wait(s))) = true .
-- from c-receive(s,id3,m) = true .
    -- eq (frame(s,id3) < sendtime(m) + wait(s)) = true .
-- from frame(s,id3) = f .
    -- eq (f < sendtime(m) + wait(s)) = true .
-- from PNAT-thm4
    eq (f = sendtime(m) + wait(s)) = false .

    red inv3-istep( id1, id2, f ) .
close .


--> [3-receive, c-receive, ~(id1=id3), ~(id2=id3)]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .
```

```
    eq s' = receive( s, id3, m ) .
-- pre conditons
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
-- case split assumptions
    eq c-receive(s,id3,m) = true .
    eq (id1 = id3) = false .
    eq (id2 = id3) = false .

    red inv3-istep( id1, id2, f ) .
close .

--> [3-receive, ~c-receive]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .
    op m : -> Message .

    eq s' = receive( s, id3, m ) .
-- pre conditons
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
-- assumptions
    eq c-receive(s,id3,m) = false .

    red inv3-istep( id1, id2, f ) .
close .


--> [3-update]
open INV3-ISTEP .
    op f : -> Frame .
    ops id1 id2 id3 : -> Uid .

    eq s' = update( s, id3 ) .
-- pre conditons
    eq frame(s,id1) = f .
    eq frame(s,id2) = f .
```

```
   red inv3-istep( id1, id2, f ) .
close .
```

A.20 wholeproof.mod

```
-- We use PNAT to model of Bucket Synchronization.
in PNAT.mod

-- Our verification needs some PNAT theorems.
-- And, We start to proof theorems.

in PNAT-thm0.mod
in PNAT-thm1.mod
in PNAT-thm2.mod
in PNAT-thm3.mod
in PNAT-thm4.mod

-- because of above proofs, we can say PNAT+thm0+thm1+thm2+thm3+thm4
in PNAT+theorems.mod

-- Import Bucket Synchronization Model.
in bucketsync.mod

-- proof lemmas.
in inv3.mod

in inv2.mod

-- proof "Synchronize" theorem(we call inv1).
in inv1.mod

--> Q.E.D.
```