電子現金の分割利用可能性の形式化と帰納的証明

吉丸 始須雄[†1] 高橋 和子[†1]

本研究では，二分木構造を持つ電子現金について帰納的なモデルを与え，定理証明器 Isabelle/HOL を用いて仕様の検証を行った．

電子現金方式のうち,「完全情報化」,「安全性」,「プライバシー」,「オフライン性」,「譲渡可能性」,「分割利用可能性」の 6 条件を満足するものを「理想的電子現金方式」と呼ぶ．ここでは，理想的電子現金方式として提案されている一方式を形式化し，分割利用可能性についての検証を試みた．

分割利用可能性とは，一度発行された電子現金を，利用合計金額が額面の金額になるまで何度でも使うことができる，という性質である．本研究で扱う方式では，電子現金を二分木によって構成し，二分木上の操作を定義することにより，これを実現している．モデル化の際も同様に，二分木に基づくデータ構造や関数を帰納的に定義した．また，分割利用可能性については「ある電子現金から任意の金額を任意の回数支払った結果，残りの電子現金の金額はその差額に等しい」と解釈し，これを証明した．

なお，自然数と二分木という異なる帰納スキームを持つデータ間の対応関係を帰納法を使って証明するために，中間的なデータ構造として二進数を利用した．そのため，本論文では Isabelle/HOL での二進数の扱い方についても言及する．

Formalization of Divisibility of an Electronic Cash Scheme and Its Inductive Proof

Shizuo Yoshimaru[†1] and Kazuko Takahashi[†1]

We formalize an electronic cash scheme that uses a binary tree structure as an inductive model, and prove its divisibility using Isabelle/HOL.

An electornic cash sheme that satisfies the following six properties is called an ideal e-cash protocols: independence, security, untraceability, offline operation, transferability, and divisibility. Divisibility means that a user can spend an electronic cash in several separate transactions by dividing its value without over-spending. In the target sheme, a coin of some monetary value is encoded as a kind of binary tree, and a payment function is defined as an operation on it. In the formalization, we give an inductive definition to the data structure and functions based on the binary tree.

The correctness of the divisibility is interpreted as follows: when an arbitrary amount is paid from an electronic cash several times, the amount remaining after payment is the difference between the original value and the payment value.

In the proof, we use a bit sequence as an intermediate data structure to successfully apply induction on the lemma on a natural number and a tree which have the different induction schemes. We also discuss the treatment of a bit sequence in Isabelle/HOL that is used in our approach.

1. はじめに

システムが仕様を満たしているかどうかを確認する手段として，数理的技法と呼ばれる検証手法が存在する．システムを実際に動かし，膨大なテストケースによって安全性を保証する手法と異なり，計算機を利用することによって強力かつ効率的に検証を行えることが特徴である．この数理的技法は，モデル検査と定理証明という 2 つの枠組みに分けられる．モデル検査とは，システムを状態遷移モデルとして記述し，起こり得る全ての遷移について自動的に探索を行うことによって，システムに求められる性質が成り立つかどうかを検証するというものである．一方定理証明とは，システムを数学的モデルとして記述し，システムに求められる性質が成り立つことを，推論によって証明するというものである．代表的な定理証明器として Isabelle/HOL[10], ACL[8], PVS[6], Coq[5] などがあるが，本研究では Isabelle/HOL を用いて，定理証明による検証を扱う．

Isabelle/HOL は，高階論理 HOL (High-Order Logic) の体系に基づき，帰納的推論を用いて命題を半自動的に証明する，対話型証明支援システムである．Isabelle/HOL を用いた検証手法としては，Paulson らによる帰納的アプローチが有名である[13]．この手法は，Kerberos や TLS など，暗号プロトコルの検証に幅広く用いられている[2),14)]．これらの証明は，暗号は決して破られないという Dolev-Yao モデル[7] に基づく前提を仮定し，プロトコルレベルの検証を行っている．

一方，暗号の有効性を示すため，暗号分野において計算量理論に基づく安全性の検証がなされている[1]．しかし，このような計算論的手法において，一般にその安全性の証明は複雑であり，誤りを含む証明例も多い．そのため，最近では，暗号プロトコルの証明を数理的技法によって簡単化するといった統合手法も研究されているが，まだ成功例は少ない[3]．

†1 関西学院大学大学院理工学研究科
School of Science&Technology, Kwansei Gakuin University

暗号プロトコルの重要な応用分野のひとつとして，電子現金が挙げられる．電子現金とは，硬貨や紙幣などの現金と同じ価値を持った電子データのことであり，多くの場合，何らかのプロトコルによって発行・支払い・決済といった処理が実現されている．なお，それら電子現金方式には，偽造防止やプライバシー保護などの観点から暗号が利用される[4),12)]．本研究では，電子現金方式に対して，プロトコルレベルではなく，そこで使われるデータ構造に着目した検証を扱う．

電子現金方式には統一された仕様や条件などがないため，これまでに様々な実現方式が提案されている[1),4)]．NTT の岡本氏は，理想的な電子現金方式というものを，6 つの条件と共に提案している[11)]．いずれも電子現金の利便性や現在の問題点の解決のための条件であり，これらの 6 条件は多くの文献で引用されている[9),15)]．ここでは，条件のひとつである分割利用可能性についての検証を行った．なお，著者らの知る限りでは，電子現金方式において，このような安全性以外の性質に関して数理的技法を適用した事例はない．

岡本氏の提案する電子現金方式において，分割利用可能性を効率的に実現するために，電子現金は二分木構造によって実現される[12)]．二分木の各ノードには，それぞれ該当金額と利用可能性が設定されており，それらの値に基づいて残額計算や支払い操作などを行うよう設計されている．ただし，二分木構造と自然数は帰納的に対応しないため，自然数から二分木構造への変換に工夫が必要となる．本研究では，自然数を二進数へ変換し，二進数を二分木へ変換することにより，自然数から二分木構造への変換に成功した．なお，自然数から二進数への変換の際にナイーブな変換アルゴリズムを用いた場合，二進数の下位ビットから値が定まっていくことになる．これを上位ビットの値から順に定まるようにするために，二進数の桁数を利用する定義方法を考案した．また，これらの定義の上で，分割利用可能性が成り立つことを証明した．

本論文の構成は以下の通りである．まず，第 2 節では理想的電子現金方式の 6 条件と，二分木構造の電子現金の仕組みについて述べる．第 3 節では二分木構造の電子現金を形式化する手法について，第 4 節ではその上で分割利用可能性を検証する手法についてそれぞれ述べた後，第 5 節でまとめる．

2. 電 子 現 金

現在世の中で使われている電子現金方式には，IC カードなどに電子情報として収納することによってオフラインで利用する方法や，クレジットカードのように後日決済する方法，小売店が銀行にリアルタイムで問い合わせることによって電子現金を利用する方法などがある．しかし，それぞれ「物理媒体に依存している」,「利用者のプライバシーが保障されない」,「通信コストや処理時間が大きくなってしまう」などの問題点が存在する．本研究では，それらの問題を解決するために提案された，理想的電子現金方式について扱う．

2.1 理想的電子現金方式

理想的電子現金方式とは，いかなる物理媒体にも依存せず，情報そのものが電子現金となるような形で電子財布に格納される形態でありながら，オフラインでの利用やプライバシーの遵守などを実現する，まさに理想的な電子現金方式のことである．しかしながら，情報は極めて容易に全く同一のコピーが可能であり，何の痕跡も残さずにデータを改竄することさえ可能であるため，不正利用の防止にも力を入れる必要がある．

以上を整理すると，理想的電子現金方式とは，以下の 6 条件を全て満たす電子現金方式のこととなる．

(1) 完全情報化　：電子現金が完全に情報のみで自立して実現されていること．
(2) 安全性　：電子現金の複製，偽造等による不正利用ができないこと．
(3) プライバシー　：利用者の購買に関するプライバシーが，小売店や銀行が結託しても露見しないこと．
(4) オフライン性　：小売店での電子現金支払い時の処理が，オフラインで完結すること．
(5) 譲渡可能性　：電子現金を他人へ譲渡可能であること．
(6) 分割利用可能性：一度発行された電子現金を，利用合計金額が額面の金額になるまで何度も使えること．

最後の 2 条件 (5), (6) は，電子現金の利便性を考慮すると当然要求されるものである．しかし，特に (6) の分割利用可能性については，比較的厳しい条件とされている．そのため，ここでは分割利用可能性について検証を行うことにする．

2.2 二分木構造

岡本氏の提案する方式において，理想的電子現金は二分木構造によって構築されている．これにより，分割利用可能性を効率的に実現しているという．

二分木の各ノードには該当金額というものが定められており，それぞれのノードの該当金額は直下の子ノードの該当金額を合計したものになっている．たとえば，左右の子ノードがどちらも 1000 円に該当するならば，その親ノードは 2000 円に該当することになる．電子現金の利用精度（利用する金額の最小単位）は用途に応じて変更することができ，たとえば 1 円単位，100 円単位などの設定が考えられる．また，各ノードには利用可能性が設定されており，利用不可能であるノードを利用することは当然できない．なお，利用可能であ

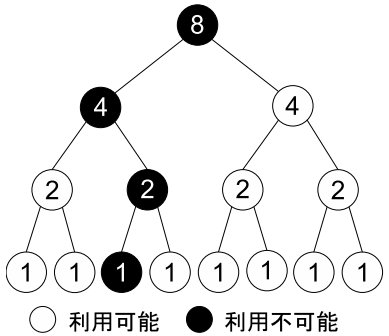

図 1 二分木構造と利用可能性
Fig. 1 Binary tree structure and usability

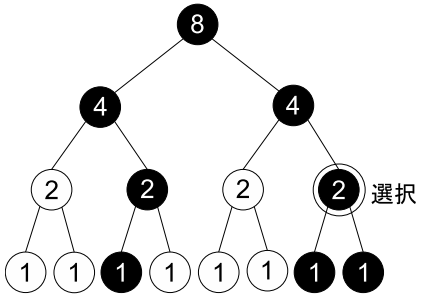

図 2 二分木構造における支払い
Fig. 2 Payment from a binary tree structure

るノードの子孫ノードは全て利用可能であり，利用不可能であるノードの祖先ノードは全て利用不可能である（**図 1** 参照）．

電子現金全体としての金額は，ノードの該当金額と利用可能性によって決定される．ルートノードから順に利用可能性を判定し，ノードが利用可能であれば該当金額をそのままノードの金額とし，利用不可能であれば左右の子ノードの金額の合計を自身の金額とする．そして最終的に求まるルートノードの金額が，その電子現金の金額となる．たとえば，図 1 は 7 円の電子現金として扱われ，**図 2** は 5 円の電子現金として扱われる．

次に，支払い操作の規則について説明する．

(1) 利用者は利用可能ノードの中から，利用したい金額のノードを選択する（図 2 参照）．
(2) 選択したノードと，連結する全ての祖先ノードと子孫ノードを利用不可能とする．
(3) 以上の操作を，合計金額が支払い額に相当するまで実行する．

これにより，支払い金額と支払い後の電子現金の残額の合計が，支払い前の電子現金の残額と常に一致するようになる．また，各ノードの利用可能性についても適切に再設定されているため，何度も同じ電子現金を利用することが可能である．

なお，安全性やプライバシーを守るため，全てのノードに暗号化が施されている．このため，支払いに利用するノードについて，解読のための鍵を小売店に公開し，小売店はその全てのノードについて正当性の検証と支払い処理を行うことになる．仮に，金額の最小単位をリスト構造として並べたような単純な構造の電子現金があった場合，処理に掛かる時間は単純に金額に比例することになる．しかし，二分木を利用する本方式の場合，選択したノードに対してだけ処理を行えば良く，処理時間は格段に短縮される．逆に，二分木構造を採用したことによるデメリットは特になく，理想的電子現金方式を実現する上で最も理に適った構造だと言える．

3. 形 式 化

帰納的推論による証明を基本とする Isabelle/HOL に適用するため，電子現金を帰納的モデルとして記述する必要がある．しかし，ここで扱う電子現金は二分木構造であり，その金額は自然数であるため，帰納的に対応しない両者を対応付けなければならないという問題が生じる．そこで，本研究では二進数を利用し，自然数から二分木を作成した．更に，電子現金の残額確認や支払い操作など，必要な関数を適切に定義した．

3.1 電子現金の作成

二分木を表す型を *tree*，電子現金を表す型を *money* とし，次のように定義する．

```
datatype  'a tree = Tip | Node 'a "'a tree" "'a tree"
types  money = "(nat * bool) tree"
```

money は二分木構造であり，各要素は *nat* と *bool* の組になっている．前者はノードの該当金額を，後者はそのノードが利用可能かどうかをそれぞれ表している[⋆1]．

ここでは，n 円の電子現金を作成する関数として，自然数 n を引数に取り，*money* 型の解を返す関数 *cash* を定義したい．しかし，*cash* 0 の場合は解を Tip とすれば良いが，*cash* (*Suc n*)（*Suc n* は $n+1$ を表す）の場合，左右の子ノードの構造について帰納的に定義することは困難である．何故なら，2 つの子ノードに対し，自然数 n という値をどのように扱えば良いのか分からないためである．

ここで，本電子現金方式の利用可能性についての規則に着目する．すなわち，利用可能であるノードの子孫ノードは全て利用可能であり，利用不可能であるノードの祖先ノードは全て利用不可能である，という性質である．仮に，利用可能であるノードが全て左側に集約され，利用不可能であるノードが全て右側に集約された電子現金を作成したとすると，全てのノードが利用可能である場合を除き，左の子ノードの利用可能性から次のように真偽値リストへ変換することができる．

(1) 左の子ノードが利用可能である場合，$True$ を出力して右の子ノードへ進む．

⋆1 ノードの該当金額が $2^n \times a$ 円である場合，*money* 型では n のみを値として持つことに注意．ただし，a は電子現金の利用金額の最小単位（利用精度）とする．

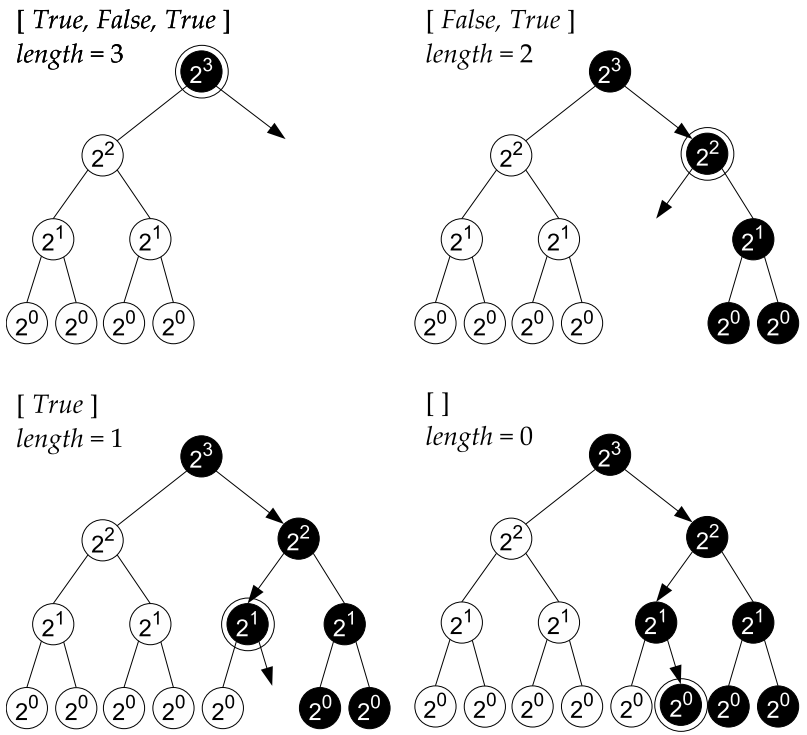

図 3 真偽値リストから二分木への変換
Fig. 3 Convert boolean list to binary tree structure

(2) 左の子ノードが利用不可能である場合，$False$ を出力して左の子ノードへ進む．
(3) 末端のノードまで辿り着いた時点で終了．

(1) では，左の子孫ノードは全て利用可能であると判断できるため，右の子ノードへ進んでいる．逆に (2) では，右の子孫ノードは全て利用不可能であると判断できるため，左の子ノードへ進んでいる．

また，これは逆も成り立つ．すなわち，真偽値リストから二分木構造の電子現金へ変換するアルゴリズムは，次のようになる（**図 3** 参照）．

- リストの先頭が $True$ である場合，左の子孫ノードを全て利用可能とし，リストの残りを右の子ノードへ渡す．
- リストの先頭が $False$ である場合，右の子孫ノードを全て利用不可能とし，リストの残りを左の子ノードへ渡す．
- リストが空になった時点で終了．

ただし，辿ったノードは全て利用不可能ノードとして設定する．

このアルゴリズムに基づき，真偽値リストから電子現金へ変換する関数を帰納的に記述する．ここで，全てのノードが利用可能である電子現金を出力する関数を $full\_money$，全てのノードが利用不可能である電子現金を出力する関数を $empty\_money$ とし，出力する木の深さを引数として与えるものとする．このとき，真偽値リストを電子現金に変換する関数 bs_to_money は，次のように定義できる．

primrec

```
  bs_to_money :: "bool list ⇒ money"
```

where

```
  "bs_to_money [] = empty_money 0" |
  "bs_to_money (b#bs) = (if b
    then Node (length (b#bs), False)
           (full_money (length bs)) (bs_to_money bs)
    else Node (length (b#bs), False)
           (bs_to_money bs) (empty_money (length bs)))"
```

$b\#bs$ は真偽値リストであり，b は先頭要素，bs は残りのリストを表している．また，$length\ bs$ はリスト bs の長さであり，この値がそのまま子ノードの深さに対応している．なお，各ノードが保持する値にも木の深さを採用しているが，この値を n，電子現金の利用精度を a 円単位としたとき，実際のノードの該当金額は $2^n \times a$ 円に設定されていることになる．たとえば末端のノードの場合，深さは 0 であり，該当金額は $2^0 \times a = a$ 円．また，木の深さが 4 であるノードの場合，該当金額は $2^4 \times a = 16a$ 円となる．

この関数を利用することにより，自然数から電子現金を作成する関数 $cash$ を以下のように定義することができる．すなわち，電子現金の金額が 2 の累乗なら $full\_money$ によって電子現金を作成し，そうでなければ一度自然数を真偽値リストに変換し，bs_to_money を利用して電子現金を作成する関数となる．

primrec

```
  cash :: "nat ⇒ money"
```

where

```
  cash_base: "cash 0 = Tip" |
  cash_step: "cash (Suc n) = (if (Suc n) = 2 ^ num_digits (Suc n)
    then full_money (num_digits (Suc n))
    else bs_to_money (bit_seq (Suc n)))"
```

ここでは，自然数を真偽値リストに変換する関数として bit_seq を用意した．この関数の定義については次節で述べる．

3.2 二進数リストの作成

まず，前節で用いた真偽値リストについて考察する．リストの要素が $True$ であり，残りのリストの長さが n であるとき，左の子ノードの金額が $2^n \times a$ 円であることが分かる．一方，リストの要素が $False$ であるとき，右の子孫ノードは全て利用不可能であり，左の子ノードの金額は少なくとも $2^n \times a$ 円未満である．つまり，$True$ なら $2^n \times a$ 円に対応し，$False$ なら対応していないと言える．これは，二進数の性質に酷似している．

ここで，分かりやすさのために利用精度を $a = 1$ 円単位とおいて整理する．つまり，2^n 円に該当するノードについて，$True$ であれば利用可能であり，$False$ であれば利用不可能である．二進数の場合，n 桁目に着目したとき，その値が 1 であれば 2^{n-1} が加算され，0 であれば加算されない．つまり，自然数から目的の真偽値リストへ変換するには，十進数から二進数への変換を行えば良いことになる．

ただし，十進数から二進数へ変換する関数を定義する際には，多少の工夫が必要である．たとえば，二進数変換の最も一般的な方法で関数を定義すると，次のようになる．

fun

```
  bit_seq' :: "nat ⇒ bool list"
```

where

```
  "bit_seq' n = (if n=0 then []
    else (n mod 2 = 1) # bit_seq' (n div 2))"
```

十進数の 13 を二進数に変換すると 1101 であるが，bit_seq' 13 の解は [$True$,$False$,$True$, $True$] となり，逆順で出力されていることが分かる．一般的な二進数変換では，値は最下位から決まっていくため，この定義では逆順リストしか得られないのである．この結果を bs_to_money の引数として利用しようとすると，まず逆順リストを元の順序に直す必要があるが，リストの逆順処理は帰納的な証明との相性が悪く，証明が困難になってしまう．そこで，先頭から値が定まる二進数変換アルゴリズムを考えた．

まず，二進数の桁数に着目する．最上位が何桁目になるのかさえ分かってしまえば，あとは全体の数から 2 の累乗の値を順に引き算していくだけである．たとえば 13 の場合，二進数に直すと 4 桁であるから，次のように 2^{4-1} から引いていけば良い．

(1) $13 - 2^3 = 5$
(2) $5 - 2^2 = 1$
(3) 1 から 2^1 を引くことはできないため，そのまま．
(4) $1 - 2^0 = 0$

このとき，引き算が成立した部分を 1，そうでない部分を 0 に対応させると，1101 という結果が得られる．

また，二進数には，2 で割ると 1 ビット右にシフトし，桁数がひとつ下がるという性質がある．この性質を利用し，二進数の桁数を求める関数 num_digits を次のように定義した．

fun

```
  num_digits :: "nat ⇒ nat"
```

where

```
  "num_digits n = (if n ≤ 1 then 0
    else Suc (num_digits (n div 2)))"
```

定義から分かるように，2 で割った数の桁数を求め，そこに 1 を足した数が解となっている．ただし $n = 1$ の場合，二進数における桁数は 1 であるにもかかわらず，この関数では 0 を解としているため，$n = 2$ 以降の解もそれぞれひとつずつ小さくなっている．これは，二進数変換の際，常に桁数より 1 だけ小さい数を利用するためである．

この関数を利用することにより，十進数を二進数に変換する関数 bit_seq を次のように定義できる．

primrec

```
  calc_bit_seq :: "nat ⇒ nat ⇒ bool list"
```

where

```
  "calc_bit_seq 0 m = [m=1]" |
  "calc_bit_seq (Suc n) m = (if 2 ^ Suc n ≤ m
    then True # (calc_bit_seq n (m - 2 ^ Suc n))
    else False # (calc_bit_seq n m))"
```

primrec

```
  bit_seq :: "nat ⇒ bool list"
```

where

```
bs_base: "bit_seq 0 = []" |
bs_step: "bit_seq (Suc n) = calc_bit_seq (num_digits (Suc n)) (Suc n)"
```

関数 *bit_seq* は，*num_digits* の値を求めて関数 *calc_bit_seq* へ引き渡す役割であり，関数の実質的な本体は *calc_bit_seq* である．第 1 引数が二進数の桁数，第 2 引数が二進数へ変換する自然数を表しており，先に述べたアルゴリズムを忠実に実装している．この関数により，自然数から二進数リストを先頭から順に作成することができるようになった．

3.3 電子現金の残額

次に，与えられた電子現金の残額を計算する関数を定義する．

primrec

```
money_amount :: "money ⇒ nat"
```

where

```
"money_amount Tip = 0" |
"money_amount (Node x l r) = (if usable x
  then 2 ^ amount x
  else money_amount l + money_amount r)"
```

usable はノードの利用可能性を，*amount* はノードの該当金額に対応する値（2 の累乗に対する指数）をそれぞれ返す．ここで定義した関数 *money_amount* は，電子現金の二分木をルートノードから順に辿り，それぞれのパスで最初に現れた利用可能ノードの該当金額を全て合計し，電子現金全体の残額を算出する．

3.4 電子現金での支払い

次に，電子現金から任意の金額を支払う操作を定義する．

primrec

```
pay :: "money ⇒ nat ⇒ money"
```

where

```
"pay Tip n = Tip" |
"pay (Node x l r) n = (if n = 0
  then Node x l r
  else if (usable x ∧ 2 ^ amount x = n)
    then empty_money (amount x)
    else if (money_amount l < n)
      then Node (amount x, False) (empty_money (money_depth l))
        (pay r (n - money_amount l))
      else Node (amount x, False) (pay l n) r)"
```

関数 *money_depth* は，電子現金の二分木構造の深さを表している．関数 *pay* は，左の子ノードから優先的に支払う関数として定義した．左の子ノードだけで支払いを済ますことが出来る場合は左へ進み，そうでない場合は左の子孫ノードを全て利用不可能にした後，その差額を支払う金額として更新し，右へ進んでいる．なお，この関数では，電子現金の残額以上の金額を支払うことはできない．

3.5 電子現金の検査

最後に，証明の際の補助的な関数として，与えられた電子現金が正しく構成されていることを保証する関数 *check_money* を定義する．検査項目は 2 つあり，1 つは二分木の構造に関する規則，もう 1 つは 2.2 節で述べた利用可能性に関する規則である．前者を関数 *check_form*, 後者を関数 *check_rule* としてそれぞれ定義したが，詳しい定義については割愛する．*check_money* の定義は次の通りである．

definition

```
check_money :: "money ⇒ bool"
```

where

```
"check_money c ≡ check_rule c ∧ check_form c"
```

4. 検　　証

本研究で検証しようとしている分割利用可能性とは,「一度発行された電子現金を，利用合計金額が額面の金額になるまで何度も使えること」である．この性質について，自然数リストの要素を足し合わせる関数 *listsum* と，リストに対して畳み込み演算を行う関数 *foldl* を利用し，次のように記述した．なお，*listsum* と *foldl* はいずれもリストを扱うための関数として，Isabelle/HOL に標準で備わっているものである．

theorem `"n ≥ listsum ms`

`⟹ money_amount (foldl pay (cash n) ms) = n - listsum ms"`

$ms = [m1, m2, m3, ..., mn]$ とおいたとき，*listsum* と *foldl* を展開すると次のようになる．

theorem `"n ≥ m1 + m2 + m3 + ... + mn`

`⟹ money_amount (pay ... (pay (pay (pay (cash n) m1) m2) m3) ... mn)`

```
    = n - (m1 + m2 + m3 + ... + mn)"
```

つまり，リスト ms には支払い額がいくつも格納されており，それを n 円分の電子現金から順番に支払っていくとき，最終的な残額が正しいことを表している．また，この定理では残額の正しさだけでなく，同じ電子現金を何度も繰り返して支払いに利用可能であることも示している．

上記の定理を証明するために，以下の補題を用意する．

(1) $check\_money\_for\_fold\_pay$

lemma
```
"⟦check_money c; money_amount c ≥ listsum ms⟧
    ⟹ check_money (foldl pay c ms)"
```

ある電子現金が $check\_money$ を満足するとき，合計が限度額を超えない限りいくら支払っても $check\_money$ を満足する．

(2) $check\_money\_for\_cash$

lemma
```
"check_money (cash n)"
```

$cash$ によって作成された電子現金は $check\_money$ を満足する．

(3) $succeed\_payment$

lemma
```
"⋀n. ⟦check_money c; n ≤ money_amount c⟧
    ⟹ money_amount (pay c n) = money_amount c - n"
```

ある電子現金が $check\_money$ を満足するとき，金額以内であれば支払いは正しく行われる．

(4) $money\_amount\_of\_cash$

lemma
```
"money_amount (cash n) = n"
```

$cash$ によって作成された電子現金は，金額を正しく算出する．

なお，これらを証明する際に様々な補題を用意したが，ここでは特に重要な補題のみを紹介する．

(1) $unfold\_cash\_in\_money\_amount$

lemma
```
"⟦n ≠ 2 ^ num_digits n; bit_seq n = b#bs⟧
    ⟹ money_amount (cash n) = money_amount (Node (length (b#bs), False)
          (full_money (length bs)) (bs_to_money bs))"
```

$cash\ n$ を展開する．

(2) $unfold\_cash\_for\_two\_to\_num\_digits\_in\_money\_amount$

lemma
```
"⟦bit_seq n = b#bs; n ≠ 2 ^ num_digits n⟧
    ⟹ money_amount (full_money (length bs))
      = money_amount (cash (2 ^ num_digits n))
```

$cash\ n$ の左の子ノードを展開する．

(3) $unfold\_cash\_for\_except\_two\_to\_num\_digits\_in\_money\_amount$

lemma
```
"⟦bit_seq n = b#bs; n ≠ 2 ^ num_digits n⟧
    ⟹ money_amount (bs_to_money bs)
      = money_amount (cash (n - 2 ^ num_digits n))
```

$cash\ n$ の右の子ノードを展開する．

つまり，n が 2 の乗数でない場合，$cash\ n$ の左右の子ノードはそれぞれ $cash\ (2^{num\_digits\ n})$, $cash\ (n - 2^{num\_digits\ n})$ に対応しているのである．これを電子現金から自然数へ対応させる際に利用したのであるが，何故このような値となるのかについては，以下のように説明することができる．

関数 $cash$ によって作成された電子現金は，左の子ノードが必ず利用可能ノードとして設定される．利用可能ノードが全て左側に集約されるという $cash$ の性質から，右の子孫にひとつでも利用可能ノードが存在するならば，左の子孫ノードは全て利用可能として設定されるためである．このとき，左の子の金額は「n を超えない最大の 2 の乗数」であると言える．そのような数は，関数 num_digits を用いて $2^{num\_digits\ n}$ となる．また，右の子の金額はその差額であり，$n - 2^{num\_digits\ n}$ となる．

なお，証明のために用意した最終的な補題数は約 60 である．

5. おわりに

本研究では，理想的電子現金方式の一条件である分割利用可能性について，帰納関数による形式化と定理証明による検証を与えた．その際，自然数と二分木を帰納的に対応させるため，二進数を利用した変換規則を定義した．また，自然数から二進数への変換については，ナイーブな手法とは異なる，二進数の桁数から解を導く手法を提案した．帰納的な定義や推論は，ほとんどの定理証明器で採用されているが，このような変換規則を扱った事例は今までにない．したがって，本研究で示した手法が，今後二進数や二分木を自然数とともに扱う際の指針となることが期待される．

今後の方針としては，この二分木を利用した理想的電子現金方式の有用性を示すため，安全性やプライバシーなどの性質を絡めて更に詳しく定式化を行い，検証を試みるつもりである．

謝辞 二分木構造の電子現金について，提案者である岡本氏にご指導を賜りました．謹ん

で感謝の意を表します.

参 考 文 献

1) F. Bao. Colluding attacks to a payment protocol and two signature exchange schemes. In *Advances in Cryptology - ASIACRYPT 2004*, Vol. 3329 of *Lecture Notes in Computer Science*, pp. 417–429, 2004.
2) G.Bella and L.C. Paulson. Kerberos version IV: inductive analysis of the secrecy goals. In J.J. Quisquater, editor, *ESORICS 98: 5th European Symposium on Research in Computer Security*, Vol. 1485 of *Lecture Notes in Computer Science*, pp. 361–375, 1998.
3) R.Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of FOCS*, pp. 136–145, 2001.
4) D.Chaum, A.Fiat, and M.Naor. Untraceable electronic cash. In *Proc. of Crypto'88*, pp. 328–335, 1988.
5) C. Cornes, J. Courant, J. C. Filliatre, G. Huet, C. Murthy, C. Munoz, C. Parent, C.Symbolique, P.Manoury, P.Manoury, A.Saibi, B.Werner, and Projet Coq. *The Coq Proof Assistant - Reference Manual*, 1995.
6) J.Crow, S.Owre, J.Rushby, N.Shankar, and M.Srivas. *A Tutorial Introduction to PVS*, 1995.
7) D.Dolev and A.C. Yao. On the security of public key protocols. In *Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pp. 350–357, 1981.
8) M.Kaufmann and J.S. Moore. *ACL2-Tutorial*, 1997.
9) J.L. Liu, V.K. Wei, and S.H. Wong. Recoverable and untraceable E-cash. EUROCON '2001, Trends in Communications, International Conference on, pp. 132–135, 2001.
10) T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL A Proof Assistant for Higher-Order Logic*. Springer, 2008.
11) T.Okamoto and K.Ohta. Universal electronic cash. In *Proc. of Crypto'91*, pp. 234–337, 1991.
12) T.Okamoto and K.Ohta. A universal electronic cash scheme. *The transactions of the Institute of Electronics, Information and Communication Engineers*, Vol. J76-D-1, No.6, pp. 315–323, 1993.
13) L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Computer Security*, pp. 85–128, 1998.
14) L.C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Computer and System Security*, pp. 332–351, 1999.
15) B.Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons Inc., 1995.