日本ソフトウェア科学会第 26 回大会 (2009 年度) 講演論文集

異なる帰納スキームにまたがる性質の定理証明手法

吉丸 始須雄　高橋 和子

本研究は，自然数と二分木という異なる帰納スキームを対応させる手法として，二進数を利用するアプローチを提案し，その対応関係について定理証明器 Isabelle/HOL による証明を行うものである．

定理証明では一般に，帰納的推論による証明が行われる．そのため，自然数と二分木の対応関係について議論するには，まずそれらの帰納スキームを対応させることが必要となる．

ここでは，二分木構造を用いた具体的な事例として理想的電子現金方式に着目し，その条件のひとつである分割利用可能性についての検証を試みた．これは，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．この性質が，自然数と二分木の対応関係について記述したモデルの上で成り立っていることを証明することにより，二分木構造の電子現金と自然数としての金額が対応していることを示した．

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注目されている．これは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を特徴とする．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 [2][3]．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 [8], ACL2 [6], PVS [5], Coq [4] 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 (Higher-Order Logic) の体系に基づき，与えられた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．証明には帰納的推論が用いることが多く，そのため対象となるシステムを帰納的モデルとして記述するのが一般的である [7] [13]．Paulson らは，暗号プロトコルを Isabelle/HOL で帰納的にモデル化して証明した [11]．その後，Kerberos や TLS などの複雑なプロトコルも，このアプローチを使って証明され [1][12]，暗号プロトコルの安全性の証明は Isabelle/HOL の得意とする分野のひとつとなった．このアプローチでは，プロトコルをイベントの列として捉え，起こり得るイベント列のパターンを集合としてモデル化している．そこでは，イベントが 1 つ増えるとイベント列も 1 つ増えるため，自然な帰納的定義がなされている．

Isabelle/HOL のライブラリには，他にも多くの検証例があるが，用いられるデータとしては自然数やリスト，集合などが主流である．それらの構造を帰納的に定義した場合，Successor は 1 つであり，帰納スキームとしてはどれも同じ構造になる．一方，二分木の場合は Successor が 2 つであり，その上で成り立

Methodology for Proving Properties over the Data with Different Induction Schemes.

Shizuo Yoshimaru, Kazuko Takahashi, 関西学院大学大学院理工学研究科, School of Science&Technology, Kwansei Gakuin University,

つ性質は問題なく証明できる．ところが，現実の問題への適用を試みる場合，たとえば自然数と二分木，リストと二分木のような，異なる帰納スキームをもつデータ間にまたがる性質が必要となることがある．リストと二分木にまたがる性質の検証や両者の変換については，Preorder や Postorder, Inorder などでノードを辿ることによって実現可能であるが，これらはリスト中に二分木構造を埋め込んだ構造を使っており，純粋に Successor が 1 つであるリスト構造を用いた定義にはなっていない．したがって，二分木と自然数にまたがる性質を証明する方法が必要となる．

二分木は様々なアプリケーションで使用されるデータ構造であるが，対象によっては二分木が特殊な性質をもつことがある．たとえば，著者らがモデル化と検証を行った電子現金方式では，電子現金の構造として二分木が用いられており，二分木の各ノードの値が常に子ノードの値の合計となるような性質をもっていた [14] [15]．二分木の親子ノード間に関係が存在するため，ノードを辿るだけでは電子現金の構造とその金額との対応関係を導くことができず，二分木と自然数という異なる帰納スキーム間にまたがる性質の扱いは非常に困難であった．また，Successor が 2 つのデータから 1 つのデータへの変換は比較的容易に行うことができるが，自然数から二分木のように，Successor が 1 つのデータから 2 つのデータへの変換には帰納スキームに関する何らかの工夫が必要となり，一筋縄では解けない問題となっていた．著者らは，自然数から二分木への変換を行う際に，中間データとして二進数リストを用意することにより，これらの問題を解決した．本論文では，この電子現金を事例とし，異なる帰納スキームにまたがる性質を Isabelle/HOL で証明する手法を提案する．これによって，Isabelle/HOL の応用範囲を広げることを目標とする．

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 節で異なる帰納スキームに関する議論を行い，第 6 節でまとめを述べる．

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．それらの問題を解決するため，理想的電子現金方式が提案された．

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．

岡本らは，理想的電子現金方式の満たすべき性質として，以下の 6 条件を定義している [9]．

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

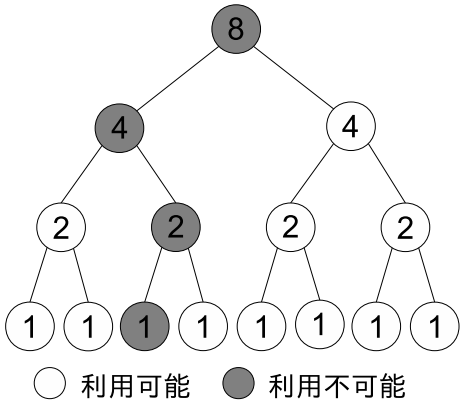

図 1　二分木構造と利用可能性

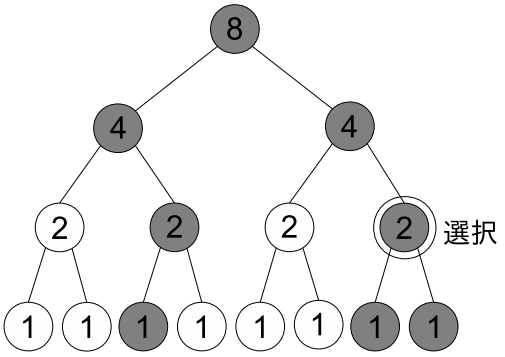

図 2　二分木構造における支払い

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

岡本らの提案した理想的電子現金方式には，データ構造として二分木が採用されている [10]．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．また，これにより，分割利用可能性も効率的に実現されている．

各ノードの該当金額は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，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．当然ながら，利用不可能であるノードを利用することはできない（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，1 は 7 円の電子現金として扱われ，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．

この電子現金は，安全性やプライバシー保護のため，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，処理にか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

Isabelle/HOL において，証明には主に帰納的推論を用いるため，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，金額は自然数であるため，帰納スキームの異なる両者を対応付けなければならない．そこで，本研究では中間データとして二進数を利用し，自然数から二分木への変換を行った．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 を定義する．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．
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．

†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．

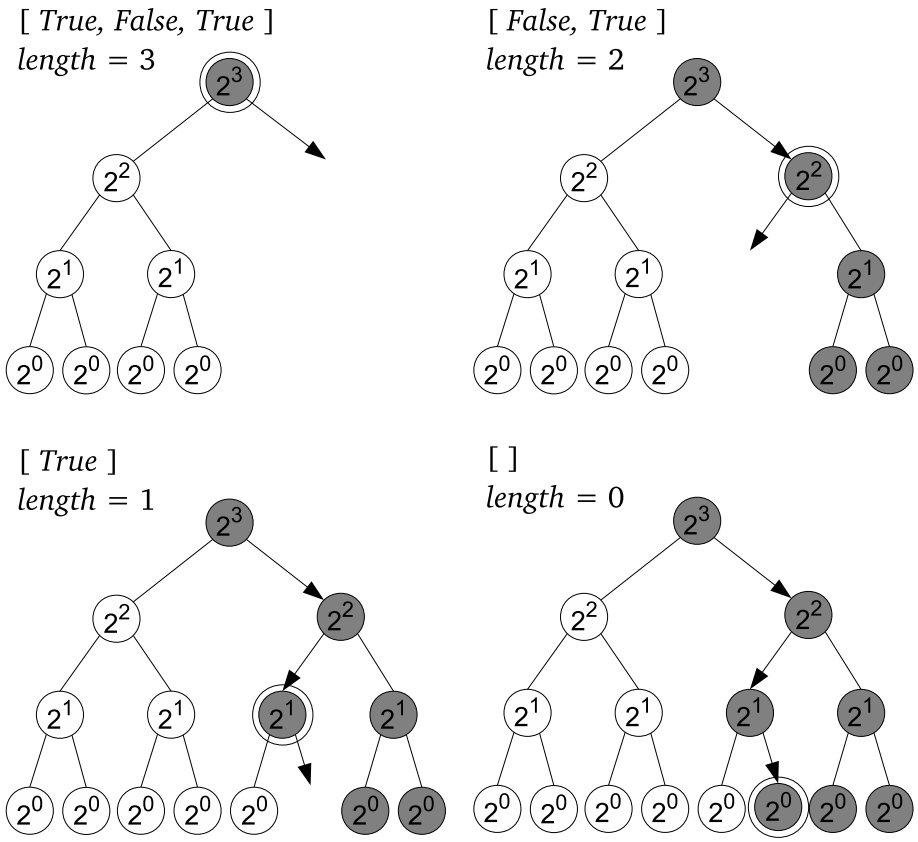

図 3　真偽値リストから二分木への変換

```
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．以下では分かりやすさのため，$a = 1$ として話を進める．

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 0 = Tip" |
 "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 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 円未満である．つまり，True なら 2^n 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．

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．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，この問題を解決するために，先頭から値が定まる二進数変換アルゴリズムを提案する．

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
 "bit_seq 0 = []" |
 "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

本研究で検証す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 *divisibility:*
```
"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．

```
"n ≥ m1 + m2 + m3 + ... + mn
  ⟹ money_amount (pay ... (pay (pay (pay
      (cash n) m1) m2) m3) ... mn)
     = n - (m1 + m2 + m3 + ... + mn)"
```

この定理は，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，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．

上記の定理を以下の手順で証明する．

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．

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

cash によって作成された電子現金は check_money を満足する．

lemma *check_money_for_cash:*
```
"check_money (cash 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．

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

lemma *money_amount_of_cash:*
```
"money_amount (cash n) = n"
```

定理 divisibility を証明するために用意した全補題数は約 60 であるが，ここでは特に重要な補題として，電子現金から自然数へ対応させるために利用したものを紹介する．

自然数 n が 2 の乗数でない場合，関数 cash によって n から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．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，その値は，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．この性質は以下のように記述される．

cash n を展開する．

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 の左の子ノードを展開する．

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 の右の子ノードを展開する．

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))"
```

5 議論

今回証明した電子現金の性質を二分木の問題として一般化すると，次のようになる．与えられた自然数から，各ノードの値が子ノードの値の合計となるような二分木を生成し，その二分木の上でノードを除去す

るという操作が定義されたとき，あるノードを除去した結果得られる二分木の値は，元の二分木と切り離したノードとの値の差である．電子現金の例では，ノードを切り離す操作が電子現金の支払い操作，ノードの値が電子現金の金額にそれぞれ対応している．

この問題のモデル化においては，基本データ構造となる二分木とその上での操作を帰納的に定義するが，証明の際に帰納スキームがうまく働くような工夫が必要になる．そのために，自然数と二分木を対応付ける必要があるが，これらの帰納スキームは異なるため，直接対応付けるのは難しい．著者らはこの問題を，中間データを用意することによって解決した．

今回は二分木の末端ノードの値が $2^0 \times a$ に固定されていると解釈し，自然数から二分木を生成した．この場合，二分木のもつ性質が二進数のものと酷似しているため，二進数を中間データとして利用することができた．一般に二進数はリストとして表現できるため，自然数とも帰納的に対応する．しかし，ナイーブな二進数変換手法ではリストの末尾から要素を決定するため，リスト本来の帰納構造から外れてしまうという懸念があった．そこで著者らは，二進数の桁数を利用した定義によって，先頭から要素を決定する二進数変換アルゴリズムを提案した．また，証明においては，それぞれの対応関係を順次指定していくことにより，自然数と二分木という異なる帰納スキームにまたがる性質を扱うことができた．

一般に，自然数からある解釈や性質を踏まえた二分木を生成する場合，それらに適した中間データを用意することで，中間スキームとして自然数と二分木を繋ぐことができる．

6 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るとともに，異なる帰納スキームにまたがる性質の定理証明手法を提案した．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，このような性質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 Isabelle/HOL における証明の補助となり，応用範囲が広がることが期待される．

今後の方針としては，二分木の末端ノードの値を一般化したモデルに対して，同じように中間データを用意することで証明できることを示すつもりである．

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] Bella, G. and Paulson, L. C. : Kerberos version IV: inductive analysis of the secrecy goals, 5th European Symposium on Research in Computer Security, *Lecture Notes in Computer Science*, Vol. 1485 (1998), pp. 361–375.

[2] Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P. and McKenzie, P. : *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, 2001.

[3] Clarke, E. M., Grumberg, O. and Peled, D. : *Model Checking*, The MIT Press, 2000.

[4] Cornes, C., Courant, J., Filliatre, J. C., Huet, G., Murthy, C., Munoz, C., Parent, C., Symbolique, C., Manoury, P., Manoury, P., Saibi, A., Werner, B. and Projet Coq. : *The Coq Proof Assistant - Reference Manual*, 1995.

[5] Crow, J., Owre, S., Rushby, J., Shankar, N. and Srivas, M. : *A Tutorial Introduction to PVS*, 1995.

[6] Kaufmann, M. and Moore, J. S. : *ACL2-Tutorial*, 1997.

[7] Minamide, Y. : Verified decision procedures on context-free grammars, Proc. of the 20th International Conference on Theorem Proving in Higher Order Logics, *Lecture Notes in Computer Science*, Vol. 4732 (2007), pp. 173–188.

[8] Nipkow, T., Paulson, L. C. and Wenzel, M. : *Isabelle/HOL A Proof Assistant for Higher-Order Logic*, Springer, 2008.

[9] Okamoto, T. and Ohta, K. : Universal electronic cash, Proc. of Crypto'91, 1991, pp. 234–337.

[10] Okamoto, T. and Ohta, K.:A universal electronic cash scheme, *The transactions of the Institute of Electronics, Information and Communication Engineers*, Vol. J76-D-1, No. 6 (1993), pp. 315–323.

[11] Paulson, L. C. : The inductive approach to verifying cryptographic protocols, *Computer Security*, 1998, pp. 85–128.

[12] Paulson, L. C. : Inductive analysis of the internet protocol TLS, *ACM Transactions on Computer and System Security*, 1999, pp. 332–351.

[13] Paulson, L. C. and Susanto, K. W. : Source-level proof reconstruction for interactive theorem proving, Proc. of the 20th International Conference on Theorem Proving in Higher Order Logics, *Lecture*

Notes in Computer Science, Vol. 4732 (2007), pp. 232–245.

[14] 安田武史, 高橋和子：定理証明器による電子現金プロトコルの検証, 情報処理学会第 67 回プログラミング研究会発表資料, 2008.

[15] 吉丸始須雄, 高橋和子：電子現金の分割利用可能性の形式化と帰納的証明, 情報処理学会第 74 回プログラミング研究会発表資料, 2009.