タイトル	定理証明支援系Isabelleにおける関数検索機能の拡張
著者	佐藤，晴彦；SATO，Haruhiko
引用	北海学園大学工学部研究報告(51)：35-48
発行日	2024-01-12

定理証明支援系Isabelleにおける関数検索機能の拡張

佐　藤　晴　彦

Extension of function search functionality for the proof assistant Isabelle

Haruhiko SATO

概　要

定理証明支援系の応用においては，既存の定義・定理ライブラリを効果的に活用することが必要であり，膨大な形式化済みの概念のうち目的に合うものを効率よく探すための検索機能が重要な役割を果たす．本研究では，定理証明支援系Isabelleにおける関数検索機能の拡張として，関数の依存関係に基づく検索を提案し，その実装と実験結果について報告する．型に基づく検索と併用した実験結果より，関数の定義中で用いられている特徴的な概念を推定し検索のクエリとすることで，精度の高い検索結果が得られる場合があることを示す．

1　はじめに

定理証明支援系（interactive theorem prover, proof assistant）は数学の命題や証明を計算機上で厳密に表現し，また証明の正しさを保証するための推論機構を提供することを目的としたソフトウェアである．定理証明支援系の主要な応用先として，ソフトウェア・ハードウェアの設計・実装が正しいことを厳密に保証する形式的検証が挙げられる．現在開発や応用が活発に進められている定理証明支援系としてIsabelle，Lean，Mizar，Coq，Agda，ACL2などがある．このうちIsabelleは様々な論理体系を取り扱える汎用的な定理証明支援系であり，可読性の高い証明を記述するための言語や，定理自動証明器との強力な連携機能などを持ち，幅広い用途で利用されている．

北海学園大学准教授　工学部電子情報工学科
Assistant Professor, Department of Electronics and Information Engineering, Faculty of Engineering, Hokkai−Gakuen University

数学的な概念や推論を，定理証明支援系が基礎とする論理体系および処理可能な形式言語で表現することを形式化（formalization）という．定理証明支援系は通常，多くの応用で必要となる基礎的な数学概念の形式化を提供する．ユーザはこれらの定義・定理を活用することでより高度な形式化を効率的に行うことができる．このためには，利用可能な膨大な形式化済みの概念の中から目的に合うものを見つけ出す必要があり，そのための検索機能も提供されている．定理証明支援系 Isabelle において，形式化された数学概念の個々の要素は関数[*1]として表現される．関数は名前と型情報を基本的な属性として持ち，その関数の意味は論理式によって別途定められる．標準の検索機能は名前と型情報のみに基づいて絞り込むものであり，その意味に基づいた検索機能は提供されていない．

本研究では，定理証明支援系 Isabelle におけるより使いやすい関数検索機能の実現を目的とし，関数の意味に基づいた検索機能を提案する．関数の意味を最も直接的に定めるのはその定義を表す論理式であるが，検索を行うユーザがそれを正確に記述することは困難であることが多い．また，目的の関数がより基本的な関数の組み合わせとして段階的に定義されているとき，それらの関数全ての意味を考慮する必要があるため，検索対象全体の意味を正確に述べることはより困難となる．これらの問題を部分的に解決する緩やかな意味表現が可能な検索として，関数の依存関係に着目した検索機能を提案し，その実装および実験結果について報告する．この機能により，目的とする関数の定義に用いられているより基本的な概念をユーザが推定できるとき，それらに依存する関数のみに絞り込む検索が可能となる．

2 定理証明支援系 Isabelle

定理証明支援系は，数学の形式化を支援するための統合的な証明開発環境を提供する．その主要な構成要素として，証明記述言語およびそれに基づく対話的な証明検証環境，自動証明・反証機能，定理データベースおよびその検索機能などがある．本節では，本研究で提案する関数検索機能に関連が深い概念を中心に，Isabelle の基礎概念について解説する．

2.1 論理体系と理論

論理体系とは，公理系と演繹の規則（推論規則）により，正しいと認める命題および証明を厳密に定めたものである．ある論理体系における定理の証明をこの論理体系に即した形で述べることで，その正しさを機械的に検証可能としたものを形式的な証明（formal proof）という．定理証明支援系は，そのような形式的な証明の作成および検証を計算機上で支援することを目的とする．

[*1] 正式には定数（Constant）と呼ばれているが，本論文では分かりやすさのため関数と呼ぶものとする．従って，本論文における関数は関数型 $\alpha \Rightarrow \beta$ ではない型を持つものも含む．

Isabelleでは純粋論理（Pure logic）を最も基本的な論理体系として提供し，更にそれを基礎として対象論理（Object logic）と呼ばれるより高度な論理体系を定義可能としている．これにより，様々な論理体系の取り扱いを可能としている．Isabelleで最も広く利用されている対象論理がHOL（Higher−Order Logic）であり，Isabelle上でHOLが提供する証明環境全体はIsabelle/HOLと呼ばれる．以降，本論文ではIsabelle/HOLを単にIsabelleと呼び，この環境における関数検索機能の拡張について議論する．このため，以降の節で説明する基礎概念は全ての対象論理に共通するものだけでなく，HOL固有のものも含まれる．

関連する一連の定義や定理を集めたものを理論（theory）と呼ぶ．ある理論はその基礎となる複数の理論を親として包含し，その内容を拡張する形で作成される．先に述べた対象論理も理論として表現され，親理論を選ぶことにより自身が基礎とする対象論理が定まる．ある理論において定理として証明された命題を事実（fact）と呼ぶ．

2.2 関数と型

証明の対象とする命題を表す論理式も含め，全ての数学的な式は項（term）で表現される．項を構成する基本要素は変数と関数であり，変数は自由変数，λ抽象による束縛変数，スキーマ変数の3種類がある．自由変数はその文脈において何らかの仮定を伴う場合がある．一方スキーマ変数は全称量化されたものと同等の意味を持ち，型の制約を満たす任意の項を代入することが認められる．

変数および関数は型を持ち，それらからなる項全体の型整合性は帰納的に定められる．型構築子κは型上の演算子であり，型構築子ごとに定められた一連の型パラメータを受け取る．型構築子κを型$\alpha_1, \ldots, \alpha_n$に適用して得られる型を$(\alpha_1, \ldots, \alpha_n)\kappa$で表す．アリティが0および1の場合は丸括弧を省略して単にκや$\alpha\ \kappa$と表す．アリティが0である型構築子による型の例として，純粋論理における命題を表す型propや，対象論理HOLにおける命題型boolや自然数型natなどがある．アリティが1の型構築子の例として，要素の型がαである集合を表す型α setや，要素の型がαであるリストを表す型α listがある．またアリティが2の型構築子の例として，型α, βそれぞれの値のペアを表す直積型(α, β)prodがある．

特に重要な型構築子として，型αから型βへの関数を表す型(α, β)funがあり，その略記として$\alpha \Rightarrow \beta$を用いる．型$\alpha_1 \Rightarrow (\alpha_2 \Rightarrow (\alpha_3 \Rightarrow \ldots (\alpha_n \Rightarrow \beta)\ldots))$は$\alpha_1, \ldots, \alpha_n$それぞれの型の値$n$個を順に受け取り$\beta$型の値を返す$n$引数関数に対応する．このような深い関数型の構造に

表1 関数とその型の例

関数名	型	意味
HOL.eq	$a \Rightarrow a \Rightarrow$ bool	等価性
Set.member	$a \Rightarrow a$ set $\Rightarrow$ bool	集合の要素か否か
List.map	$(a \Rightarrow b) \Rightarrow a$ list $\Rightarrow b$ list	各要素に関数を適用して得られるリスト

ついては以降 ⇒ が右結合であるものとして括弧を省略し，$\alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n \Rightarrow \beta$と書くものとする．

型はその一部に型変数を含むことができ，これにより多相型を表現する．型変数は後述する型クラス制約が付与され，その制約を満たすような任意の型が代入可能であることを意味する．関数cの型がσとして宣言されていることを$c :: \sigma$で表す．この型σの型変数への制約を満たす代入をθとすると，σにθを適用した型$\tau = \sigma\theta$はcの持つ具体的な型として適切である．このとき型τはσのインスタンスであるといい，またσはτにマッチするという．定数$c :: \sigma$について，σのインスタンスτで具体化したものをc_τで表す．同様に型τを持つ変数xをx_τで表す．Isabelle/HOLの標準ライブラリに含まれる，型変数を含む型を持つ関数の例を**表 1** に示す．

Isabelleにおける項は変数と関数を基本要素とし，項t_1を項t_2に適用する関数適用$t_1 t_2$および項tにおける変数x_τのλ抽象$\lambda x_\tau.\ t$の繰り返しにより構成される．部分項t_1, t_2から構成される関数適用$t_1 t_2$およびλ抽象$\lambda x_\tau.\ t_1$により得られる項の型は以下のように定められる．

$$\frac{t_1 :: \tau \Rightarrow \sigma \qquad t_2 :: \tau}{t_1 t_2 :: \sigma}$$

$$\frac{t :: \sigma}{\lambda x_\tau.\ t :: \tau \Rightarrow \sigma}$$

変数および関数単独の項の型がそれぞれ$x_\tau :: \tau$，$c_\tau :: \tau$であることを基礎として，これらの規則に従い項全体の型が定まるような項は型について整合性を持つといい，システムが推論の対象として扱う項はこのような整合性を持つ項のみとする．

2.3 ロケールと型クラス

関連する定義および定理群をパラメータ化し汎用性を高めるための仕組みとして，Isabelleではロケール（Locales）という機能が提供されている．ロケールはパラメータとする関数群と，それらに関する仮定により宣言される．例として，関数パラメータf，i，eがそれぞれ代数的構造としての群における演算，左逆元，単位元を表すことをロケールgroupとして記述すると次のようになる．

```
locale group =
  fixes f :: "a ⇒ a ⇒ a" and i :: "a ⇒ a" and e :: "a"
  assumes left_neutral : "f e x = x"
    and left_inverse : "f (i x) x = e"
    and assoc : "f (f x y) z = f x (f y z)"
```

ロケールのパラメータに関する仮定に基づき，そのパラメータに関する性質を証明することができる．例として，以下のような証明の記述によりiが右逆元である性質をロケールgroup

の定理として追加することができる．

```
lemma（in group）"f x（i x）= e"
proof –
  have "f x（i x）= f e（f x（i x））" using left_neutral ...
  also have "... = f（f（i（i x））（i x））（f x（i x））" using left_inverse ...
    ⋮
  also have "... = e" ...
  finally show "f x（i x）＝ e" .
qed
```

証明されたロケールに関する事実は，そのインスタンスにも引き継がれる．すなわち，ある関数群がロケールのインスタンスであることを示すことで，その関数群についてロケールの定理を具体化したものが利用可能となる．例として，整数の加算，符号反転，0がこのロケールgroupのインスタンスであることを宣言すると，符号反転が加算の右逆元であることを表す事実が利用可能となる．ロケールのインスタンス宣言においては，パラメータの関数群が確かにロケールが要求する前提を満たしていることの証明が要求される．

ロケールはその包含関係により階層構造をなす．あるロケールl_1を満たすパラメータ群が別のロケールl_2も常に満たす場合，l_1はl_2の部分ロケール（sublocale）であるといい$l_1 \subseteq l_2$で表す．

型クラスはロケールの特別な場合であり，ある型パラメータについて宣言されている一連の関数群と，それらが満たすべき性質を定める．ロケールと異なり，型クラスが要求するパラメータ関数の名前は固定されている点に注意する．型クラスの記述例として，二項演算＋が定義されていることを表す型クラスplusと，その演算が結合則を満たすことを表すsemigroup_addは以下のように記述できる．

```
class plus =
  fixes plus :: "a ⇒ a ⇒ a"（infixl "+" 65）
class semigroup_add = plus +
  assumes add_assoc : "（a + b）+ c = a +（b + c）"
```

型クラスc_1，c_2について部分ロケール関係$c_1 \subseteq c_2$が成り立つとき，c_1はc_2の部分クラスであるという．例えば，semigroup_addはplusのパラメータや仮定をすべて含むものとして定義されているため，semigroup_addはplusの部分クラスである．ロケールと同様に，型クラスにおいてもその前提から導かれる性質を証明することができる．型クラスにより，多くの型に適用可能な汎用性の高い関数を定義することができるだけでなく，それらが満たす性質の証明も高い抽象度で行うことができる．

型変数が属すべき型クラスは通常複数存在し得る．型クラスの集合をソートと呼ぶ．すなわち，型変数への制約は１つのソートで表され，ソートに含まれるすべての型クラスに所属する型のみが代入可能であることを表す．型変数aへのソート制約が，型クラスc_1，...，c_nからなるソートであることを$a :: \{c_1, \ldots, c_n\}$で表す．ソートが１つの型クラス$c$から構成される場合，単に$a :: c$で表す．この意味に基づき，ソート$s_1$がソート$s_2$の部分ソート関係であること$s_1 \subseteq s_2$は$s_1 \subseteq s_2 \equiv \forall c_2 \in s_2.\ \exists c_1 \in s_1.\ c_1 \subseteq c_2$と定められる．$s_1 \subseteq s_2$であるとき，$s_1$を満たす型は$s_2$も満たす．

2.4 関数の定義

関数の定義とは，既に定義済みの関数のみを用いた項tに新しい名前cを割り当てることであり，$c :: \sigma \equiv t$という形式で表される．ここでσは項tの型のインスタンスであり，tはcを含まないものとする．このとき，cの型がσであるという型情報とは別に，$c \equiv t$という事実がデータベースに追加される．cに関する推論を行う際は，この事実を明示的に参照する必要がある．またそのような関数の意味を表す事実を伴わない，型宣言のみの宣言$c :: \sigma$も認められる．

関数の型σが多相型であるとき，その個々の型インスタンスについて異なる定義を定めることも認められている．これは関数のオーバーロードと呼ばれる．型クラスが要求する関数に対し，インスタンスごとに異なる定義を定める際にもこのオーバーロードの仕組みが用いられる．このような場合，１つの関数に対しその定義を定める事実が複数存在することになる．

再帰的な関数について，矛盾なくかつ簡便に定義するための専用の機能が用意されている．これを用いると，再帰呼び出しの構造に基づき関数の停止性を保証するための整礎な順序が自動的に推定される．再帰的な関数の定義においては多くの場合，引数のパターンマッチングに基づく場合分けを行い，それぞれの場合ごとに異なる項を定める必要がある．この場合もオーバーロード関数と同様に，複数の事実によって定義が定められる．

項に新しい名前を与える別の方法として略記（abbreviation）がある．通常の関数については，その関数の意味に基づく推論を行うためにはその定義内容を表す事実を明示的に利用する必要がある．これに対し，略記では内部的に別名とそれが表す項は同一のものとして扱われる．

3 既存の検索ツール

3.1 標準の検索コマンド

Isabelle/HOLに標準の検索コマンドである**find_consts**は，名前と型情報に基づき関数を検索する機能を提供する［３］．デフォルトでは，クエリとして指定した型は検索対象の関数の型$(\alpha_1, \ldots, \alpha_n)\kappa$の一部分，すなわちこの型全体もしくは$\alpha_i$の一部分にマッチすることを要求

する制約として働く．例えば型 int ⇒ bool が該当する関数の例として「整数が素数か否かを返す」Primes.prime_int :: int ⇒ bool があるが，＜で表される整数上の順序関係 Int.ord_int_inst.less_int :: int ⇒ int ⇒ bool もこの型の α_2 が int ⇒ bool であるため該当する．部分に対するマッチを認めない，全体とのマッチのみを表すオプション strict も用意されている．

ソート制約を付与した型変数を含めた型で検索すると，そのインスタンスも結果に含まれる．例えば"(a :: field) ⇒ a ⇒ bool"で検索した場合，体を表す型クラス field のインスタンスとして有理数型 rat，実数型 real，複素数型 complex などが存在するため，二項関係のうち一般の体もしくはこれらの個々のインスタンス型の上で定義されたものが該当する．一方，与えた型をインスタンスとするような型，すなわちより一般的な型を検索する機能は提供されていない．例えば集合の元か否かを判定する関数 Set.member :: a ⇒ a set ⇒ bool は型変数 a を int に具体化することで整数の集合を扱う型 int ⇒ int set ⇒ bool の関数として利用できるが，この具体的な型で検索した場合，Set.member は検索結果に含まれない．

3.2 FindFacts

形式証明アーカイブ（Archive of Formal Proofs, AFP）[4] は，ユーザにより開発された Isabelle 上の形式証明を集積したものである．2004年から継続的に記事が投稿されており，2023年時点では累計450名以上の著者により，750件以上の記事が投稿されている．AFP を構成する証明コード全体は約400万行と巨大であるため，利用者にとって必要な定義・定理を探し出すことは容易ではない．

FindFacts [5] は Huch らにより開発された Web ベースの全文検索システムであり，標準ライブラリおよび AFP を含めた Isabelle で利用可能な理論全体を横断的かつ高速に検索することを目的とする．オープンソースの全文検索エンジン Apache Solr を利用しており文字列ベースの検索を基本とするが，文書中の個々のブロックが型・関数・定理のうちどれを記述したものであるかの情報も付与されており，それらに基づく絞り込みも可能である．一方，標準の find_consts とは異なり項の構造やマッチングに基づく検索は行えない．また，ある時点での標準ライブラリおよび AFP から検索用のデータベースを作成しているため，ユーザが自身の開発中の理論において定義した関数を検索するような，起動中の Isabelle 環境において読み込まれている関数全体を対象とした検索は行えない．

4 提案手法

4.1 インスタンス型による，汎用的な関数の検索

前節で述べた通り，組み込みの検索コマンド **find_consts** は一般的な型 σ を持つ関数をそのインスタンス型 $\tau = \sigma\theta$ から検索する機能を持たない．このため，オーバーロード定義された関

数に依存した関数をはじめとした汎用性の高い関数を検索するには，ユーザの応用上必要となる型が具体的な型τであった場合も，元の一般的な型σを推測し，クエリとする必要がある．ここでσを十分一般的なものとするには適切なソート制約を付与した型変数を含める必要があるが，適切なソート制約を正確に記述することは一般に困難であり，暫定的に空のソート制約を与えて検索した場合，不適当なインスタンス型を持つ関数が検索結果に余分に含まれてしまう問題が生じる．

具体例として，加算演算を集合上に拡張した関数Groups_Big.comm_monoid_add_class.Sumを考える．この関数の型は，二項演算+が交換則と結合則を満たし0を単位元とすることを定める型クラスcomm_monoid_add上の型a :: comm_monoid_addについてa set $\Rightarrow$ aとして定義されている．intはa :: comm_monoid_addのインスタンスであるためaのインスタンスとして適切であるが，int set $\Rightarrow$ intにより検索することはできない．このように，オーバーロードされた関数を定義の一部に用いている関数は，実質的には様々なインスタンス型について利用できるにも関わらず，直接的には個々のインスタンス型に対する定義が存在しないため，インスタンス型による検索が困難な場合がある．

一般に型のマッチング判定，すなわち型τが型σのインスタンスであるかは判定可能であるため，クエリとして与えられた型τに対しそれをインスタンスとするσを型として持つ関数c_σを列挙する機能は容易に実現可能である．しかしながら，単純に型の一般性の観点で該当する関数全体を列挙した場合，ユーザの期待する関数の意味・役割が型で十分表現されておらず，目的に合わない関数が多数該当してしまうことが懸念される．このため，関数の意味の観点から検索条件を追加するなど，型以外の観点の制約を併用することが必要であると考えられる．

4.2 関数の依存関係に基づく検索

関数は通常，$c :: \sigma \equiv t$という形式で，型σを持つ関数cが項tと等価であることが述べられる．この項tが関数cの意味を完全に述べたものであり，この項tもまた関数によって構成される．このとき，項tの中に出現する関数はcが直接依存する関数であり，cの意味の一部を表現するものとして，cを探す検索の中で活用されるべき情報である．間接的な依存関係を持つ関数も同様に元の関数の意味を表す要素として扱い，検索対象の関数がどの関数に依存するか，すなわちどの関数によって意味づけられているかを要件とする検索を提案する．

本研究で提案する，関数の依存関係に基づく検索を形式的に述べる．関数全体の集合をFで表す．関数fの定義に直接出現する関数記号の集合を$D(f)$で表す．関数の直接依存関係を表す二項関係 $\rightarrow$ を$f \rightarrow g \equiv g \in D(f)$で定義し，その推移閉包を $\overset{+}{\rightarrow}$ で表す．$f \rightarrow g$はfの定義に直接gが出現していること，また$f \overset{+}{\rightarrow} g$は$f$の定義が間接的に$g$に依存していることに対応する．よって，$f \overset{+}{\rightarrow} f$ はfが再帰的に定義されていることに対応する．関数fが間接的に依存す

る関数全体を$D^+(f) \equiv \{g \mid f \xrightarrow{+} g\}$と定義すると，関数の集合$G \subseteq F$に対し，$G$のすべての要素に間接的に依存する関数全体は$D(\xrightarrow{+}G) \equiv \{f \mid G \subseteq D^+(f)\}$と表される．関数の依存関係に基づく検索とは，クエリとして与えられたGに対し$D(\xrightarrow{+}G)$を結果として提示する機能である．ユーザは目的の関数fに対し，その意味から$D^+(f)$の部分集合Gのうち$D^+(f)$に特徴的であると思われるものを推測しクエリとすることで，$f \in D(\xrightarrow{+}G)$かつ$D(\xrightarrow{+}G)$を小さく抑えることを試みる．

5 実装・実験

本節では，提案する検索手法の実装に関する細部と，その実装を用いた実験結果について述べる．実装にはIsabelleの実行・開発環境であるIsabelle/MLを利用した．また実験において検索の対象とする関数全体として，解析学分野の概念を集めたライブラリであるHOL−Analysis.Analysisを用いた．これは基本的なライブラリであるComplex_Mainの関数約3000件を基礎として包含し，ライブラリ全体は4816件の関数から構成される．

5.1 依存関係の解析

関数の直接的な依存関係を得るため，各関数の定義を表す事実を参照する必要がある．Isabelleにおける定義を行う主な命令として，非再帰的な定義向けの**definition**命令と再帰的な定義向けの**fun**命令があり，これらで定義された関数fに対しその定義内容を表す事実はf_defまたはf.simpsという名前で与えられる．これに基づき，関数の定義を参照する際もこれらの命名規則に基づき逆引きした事実を用いた．パターンマッチによる定義など，ある関数の定義が複数の事実からなる場合はそれら全ての事実を用いた．また略記による定義についても，通常の定義と同様に扱い検索の対象とした．すなわち，略記aを定義に含んでいる関数fについて$a \in D(f)$であり，fの検索のためにその略記a自体や略記が依存する関数$D^+(a)$の一部による絞り込みが可能となるようにした．

関数の直接依存関係に基づき推移閉包関係を得るため，Isabelle標準の有向グラフの実装Pure/General/graph.MLを利用した．より具体的には，関数名を表す文字列を頂点とし，各頂点同士の直接的な依存関係を辺として与えてグラフを構成する処理を実装し，ある頂点から到達可能なすべての頂点を求める実装済み関数Graph.all_succsを用いて間接依存関係$f \xrightarrow{+} g$を得た．

5.2 依存関係に基づく検索

依存関係に基づく検索の有効性を示すため，いくつかの具体的な検索例を示す．実験においては，ユーザの目的とする関数$c :: \sigma$について，σの特定のインスタンス型τをユーザが正しく

推測できていることを仮定する．これを用いて，まず関数全体Fのうちτをインスタンス型として持つ関数全体F_τのみに絞る．このとき$c \in F_\tau$であり，更に依存関数による絞り込みを行い最終的な検索結果$D(\overset{+}{\rightarrow}G)$を得るものとする．このとき$c$が依存する関数集合$D^+(c)$の部分集合を$G$として選ぶことで$c \in D(\overset{+}{\rightarrow}G)$となるが，このとき検索結果の件数$|D(\overset{+}{\rightarrow}G)|$が小さくなることが望ましい．関数$c$の意味を踏まえたユーザの入力として想定されるいくつかのGについて，それに対する$|D(\overset{+}{\rightarrow}G)|$を調べる．

例 1 ：互いに素である

互いに素であるという関係を表す関数coprimeを検索する場合を考える．関数の詳細および絞り込みに用いるインスタンス型τを以下に示す．

関数名	Rings.algebraic_semidom_class.coprime
σ	$(a$ :: algebraic_semidom$) \Rightarrow a \Rightarrow$ bool
τ	int $\Rightarrow$ int $\Rightarrow$ bool
関連定義	coprime $a\ b \equiv (\forall c.\ c$ dvd $a \longrightarrow c$ dvd $b \longrightarrow$ is_unit $c)$ b dvd $a \equiv \exists k.\ a = b * k$ is_unit $a \equiv a$ dvd 1

定義においては，aがbの約数であることを表す関数a dvd bやある値が単位元であることを表す関数is_unitが用いられており，さらにこれらの関数はより基本的な概念である存在量化∃や乗算＊や定数 1 を用いて定義されている．以上より，D^+(coprime)＝{dvd, ∀, is_unit, ∃, ＊, 1}となる．

これらの部分集合を用いた検索の結果を**表 2 a** に示す．G = {}の場合は$D(\overset{+}{\rightarrow}G) = F_\tau$であるため，型$\tau$をインスタンスとする関数全体は39件である．結果より，ユーザが「互いに素である」の定義を「任意の公約数が単位元である」と推定し，その構成要素の直接的な表現であるdvdまたはis_unitを指定した場合目的の関数 1 件のみに絞り込まれる．また，それらの名前が不明である場合もそれらの更に基本的な概念として存在量化子，乗算，1 を推定できた場合，これら全てを指定することで同じ結果を得ることができる．

例 2 ：リストの要素が互いに異なる

リストの要素同士が互いに異なるか否かを返す関数distinctを検索する場合を考える．

関数名	List.distinct
σ	a list ⇒ bool
τ	int list ⇒ bool
関連定義	distinct [] ≡ True distinct（x # xs） ≡ $x \notin$ set xs ∧ distinct xs $x \notin A \equiv \neg\ (x \in A)$

リストが空ではない場合の「先頭要素xが残りの部分xsに登場しない」ことの定義においては，リスト型を集合型に変換する関数setを用いて集合の非要素関係∉に帰着させている．この集合の非要素関係は要素関係∈の論理否定¬として定義されている．

G	$\lvert D(\overset{+}{\rightarrow}\mathrm{G})\rvert$
{}	39
{1}	8
{∀}	8
{∃}	4
{∗}	4
{1, ∗}	3
{∀, 1, ∗}	3
{∃, 1}	2
{∃, 1, ∗}	1
{dvd}	1
{is_unit}	1

（a）coprime関数

G	$\lvert D(\overset{+}{\rightarrow}\mathrm{G})\rvert$
{}	26
{∧}	12
{¬}	6
{∧,¬}	6
{∈}	3
{∈,¬}	3
{∈, ¬, ∧}	3
{∉}	1

（b）distinct関数

G	$\lvert D(\overset{+}{\rightarrow}\mathrm{G})\rvert$
{}	42
{fold}	16
{0}	10
{fold, 0}	7
{+}	1
{sum}	1

（c）Sum関数

G	$\lvert D(\overset{+}{\rightarrow}\mathrm{G})\rvert$
{}	44
{1}	7
{∗}	5
{1, ∗}	4
{^}	2
{exp}	2
{fact}	2
{LIMSEQ}	2

（d）powr関数

表2　依存する関数の集合Gに対する，検索結果の件数 $\lvert D(\overset{+}{\rightarrow}G)\rvert$

検索結果を**表 2 b**に示す．要素同士が互いに「異なる」ことの表現において登場する関数として推定され得る論理否定¬のみを指定した場合，6 件まで絞り込まれている．更に集合演算への帰着について推定できた場合，集合の要素関係∈や論理否定¬両方を指定するだけでは特定には至らないが，その組み合わせの特別な場合である∉を指定するとdistinct関数のみが該当する．

例 3 ：有限集合の総和

型クラスcomm_monoid_addで表される可換モノイドの公理を満たす演算＋が定義された型に対し，その型を要素型とする有限集合の総和を表す関数Sumを検索する．

関数名	Groups_Big.comm_monoid_add_class.Sum
σ	（a :: comm_monoid_add）set ⇒ a
τ	int set ⇒ int
関連定義	Sum A ≡ sum（$\lambda x.\ x$）A sum g A ≡ comm_monoid_set.F（+）0 g A comm_monoid_set.F f e g A ≡ Finite_Set.fold（$f \circ g$）e A

Finite_Set.foldは型（a ⇒ b ⇒ b）⇒ b ⇒ a set ⇒ bを持つ，集合上の畳み込み演算である．comm_monoid_set.Fはこの畳み込みで用いる一連のパラメータを受け取り，更にその中の二項演算f :: a ⇒ b ⇒ bにおける型aの第一引数を変換する写像gを追加のパラメータとしたものである．sumはこの関数Fにおける二項演算および初期値を＋と 0 に固定したものであり，Sumは更に変換写像gを恒等写像に固定したものである．

検索結果を**表 2 c**に示す．結果より，総和の直接的な基礎概念である畳み込みfoldのみを指定した場合は16件が該当し，その中には総積，最大・最小，上限・下限，最小公倍数・最大公約数が含まれる．一方，畳み込みに用いる演算＋もしくはそれを固定した畳み込み関数sumを指定すると，Sum関数のみが該当する．

例 4 ：指数関数

型クラスlnで表される，対数関数lnが定義された実ノルム代数上で定義される，底と指数の両方をパラメータとする指数関数powrを検索する．

関数名	Transcendental.powr
σ	$(a :: \ln) \Rightarrow a \Rightarrow a$
τ	real ⇒ real ⇒ real
関連定義	x powr $a \equiv$ if $x = 0$ then 0 else exp $(a * \ln x)$ exp $x \equiv$ suminf $(\lambda n.\ x\ \hat{}\ n\ /_R$ fact $n)$ suminf $f \equiv$ THE $s.\ f$ sums s f sums $s \equiv$ LIMSEQ $(\lambda n.$ sum $(\lambda i.\ f\ i)\ \{i.\ i < n\})\ s$

底をパラメータとするpowrは，底をネイピア数とした指数関数expと対数関数lnにより定義されており，expは更に自然数によるべき乗^と階乗関数factを用いた級数として定義されている．級数を求める関数suminfは，無限数列を表す自然数上の関数に対し，もしその総和が収束すればその値を表すものとして述語sumsを用いて定義されている．sumsは無限数列の総和がある値に収束することを表す述語であり，sum関数を用いて表される先頭n項の有限和の極限として表される．LIMSEQ関数は数列とその極限値との関係を表す述語であり，LIMSEQ $f\ s$は数列fの極限がsであることを表す．

検索結果を**表2 d**に示す．件数が2件となった結果の一方は目的のpowr関数であり，もう一方はpowr関数の別名として定義されているpowr_realであり，これらは実質的に同一の関数として扱えるものである．結果より，乗算*やその単位元1の指定のみでは完全な絞り込みには至らないものの，級数による定義に直接現れる概念^，exp，factのいずれかもしくは数列の極限LIMSEQを指定した場合，目的の関数のみが該当する．

6 おわりに

本論文では，定理証明支援系Isabelleにおける関数検索機能の拡張として，検索対象の関数が依存する関数を指定することによる絞り込み機能を提案すると共に，それが有効に働く検索対象の具体例を示した．特に検索対象がロケールのパラメータやオーバーロード関数などの抽象度の高い関数で構成されている例において，それらの具体化において用いられている基本的な関数が既知であるとき，提案手法により十分な絞り込みが可能となる場合があることを示した．

今回の実験においては，対象の関数の型が正確に指定されているという仮定の上で，依存関数を用いた更なる絞り込みの有効性を調べた．一方，引数の多い複雑な関数については，引数の順序を含めた型全体を正確に記述することが困難であるため，順序の違いを許容した場合など，型についての制約が緩い状況における提案手法の有効性を検証することも必要である．

本研究が提案する検索においては，依存関係を定める際の定義として特定の名称を持つ直接

的な定義のみを用いているが，関数に関する性質を表す事実は他にも多数存在するため，それらを更に活用するためにはどのような形式・内容の事実が定義を表すものとして検索上有効かどうかの検討が必要である．また依存関係を辺とする有向グラフから得られる情報のうち，提案手法では単純な到達可能性$D^+(f)$のみを検索に用いているが，より詳細な条件に基づく検索の実現のためには，グラフ構造から得られるより高度な情報を活用することが有効であると考えられる．例えば，特定の述語関数pについて「pを満たさない」という表現を定義の一部に含む関数を検索したい場合，¬(p F)という構造の項を含むことを制約として表現することが必要であるが，単純な到達可能性では¬とpの両方を含むという弱い制約しか表現できない．「¬の直後にpが出現する経路が存在する」のように項の構造の一部をグラフの概念を用いて表現することも含め，検索に必要となる典型的な制約の表現にグラフ上の性質がどのように活用できるかの検討も今後の課題である．

謝辞

本研究は，令和４年度北海学園大学学術研究助成（一般研究）の支援を受けて実施された．

参考文献

［１］ Clemens Ballarin, Tutorial to Locales and Locale Interpretation, Part of the Isabelle documentation : https : //isabelle.in.tum.de/doc/locales.pdf, 2023.

［２］ Florian Haftmann, Haskell−style type classes with Isabelle/Isar, Part of the Isabelle documentation : http : //isabelle.in.tum.de/doc/classes.pdf, 2023.

［３］ Makarius Wenzel et al, The Isabelle/Isar manual, Part of the Isabelle documentation : http : //isabelle.in.tum.de/doc/isar−ref.pdf, 2023.

［４］ Archive of Formal Proofs. https : //www.isa−afp.org

［５］ Fabian Huch and Alexander Krauss, FindFacts : A Scalable Theorem Search, in : Isabelle Workshop, 2020.