Agda によるダイクストラのアルゴリズムの証明

山田麗　(指導教員：浅井健一)

1　はじめに

本研究は、グラフの最短経路を求めるダイクストラのアルゴリズムの正当性について、定理証明系言語 Agda で証明を行ったものである。

Agda は、型チェックを用いて任意の論理式の証明を行うことのできるプログラミング言語である。Agda は依存型を用いていて、関数やデータ型に証明を持たせることができる。このような証明を internal verification という。これに対し、関数やデータ型とは別の部分で証明を行うものを external verification という。

ダイクストラのアルゴリズムの機械による証明の先行研究は、Mizal[1] や ACL2[2]、ダイクストラのアルゴリズムをより拡張させた Floyd のアルゴリズムの証明では Coq などがある。しかし、Agda による証明はなかったため、ACL2 の論文を参考にして Agda で証明を行った。ACL2 は依存型を用いていない言語であるため、証明は全て external だったが、本研究では、証明を internal に行う。

2　ダイクストラのアルゴリズム

ダイクストラのアルゴリズム [3] について説明する。グラフは有向グラフで、辺の長さは非負であるとする。ダイクストラのアルゴリズムでは、頂点に、始点からのパスとその長さを持たせ、最短経路が確定した頂点を確定リスト、未確定の頂点を未確定リストに格納する。アルゴリズムは主に以下の 2 つの段階に分けられる。

- 未確定リストに含まれる頂点の中から、持っているパスの長さが最も短いものを探す (分離)
- 最短であった頂点から繋がっている頂点について、未確定リストに含まれているものの持つパスの長さと、最短点を通るパスの長さを比較し、短い方に更新する。(更新)

以上の計算を終えたら、最短であった頂点を未確定リストから確定リストに移動させ、新しい確定リストと未確定リストで分離、更新を繰り返す。初期状態は、全ての頂点が未確定リストに含まれていて、確定リストは空の状態であり、始点と終点を入力として与える。未確定リストが空になったら終了し、確定リストから終点を探して終点が持つパスを返す。

ダイクストラのアルゴリズムの証明の要点は、未確定リストから確定リストに移動する頂点が、最短経路を持っていることを証明することである。アルゴリズム終了時には、全ての頂点が確定リストに移動した状態となるため、上記が示されれば終点が持つパスがグラフにおいて最短であることが証明される。

証明の説明をする。移動する頂点を v、v が持っているパスを p とする。p はアルゴリズムから、v 以外の頂点は全て確定リストに含まれていることが分かる。ここで、始点から v までの任意のパス p' について考える。v がまだ未確定リストに含まれることから、p' には必ず、初めて確定リストに含まれない頂点となる部分が存在し、その頂点を u とする。始点から u までのパスを q とする。辺の長さは非負なので、u から v までのパスは 0 以上である。また、v と u は未確定リストに含まれる頂点であり、v は未確定リストに含まれる頂点の中で最もパスの長さが短いものである。したがって、q は p より長く、u から v までのパスの長さが 0 以上であることから、p' は p より長いことがわかる。以上より、v までのパスにおいて p が最短であることが示された。

3　データ型の定義

本研究では、グラフは無向グラフで、辺に自然数の重みを持つものとしている。無向グラフにした理由は、実際の交通機関などの最短経路を求めたいと思った場合、どちらの向きにも移動できるのが一般的だからである。また、二点間で辺の繋がっていない部分は長さ infinity の辺を持っていると扱うことで、任意の二点間に必ずパスが存在するようにした。データ型に証明を持たせることで internal に証明を行っていく。以下、証明におけるデータ型の定義を示す。

グラフの型定義は以下の通りである。

```
graph : N → Set
graph n = List (Fin n × Fin n × path-l)
```

グラフは、頂点数を表す自然数 n を受け取り、二頂点とそれらを結ぶ辺の長さの組のリストとして定義した。型 Fin n とは、0 から $n-1$ までの自然数の集合を表していて、頂点の型を Fin n とすることで、頂点数より大きい自然数を頂点が持つ場合を排除でき、頂点がグラフに含まれていることを表すことができる。型 path-l は、infinity か自然数かを表すパスの長さの型である。

パスの型定義は以下の通りである。

```
data path-t (n : N) (g : graph n)
         : Fin n → Fin n → Set where
  vertex : ∀ (x : Fin n) → path-t n g x x
  path   : {a : Fin n} → {b : Fin n}
           → (p : path-t n g a b)
           → (x : Fin n) → path-t n g a x
```

パスは、頂点数、グラフ、始点、終点を引数として持ち、ただ一つの頂点 x からなる vertex、パス p と頂点 x を受け取り、新しいパスを返す path の二つのコンストラクタでできている。パスの定義内で、始点と終点が明示されていて、またパスはグラフに含まれていることが表されている。

確定リストの型定義は以下の通りである。

```
data fs-t (n : N) (g : graph n) (a : Fin n)
          : N → Set where
  empty-fs : fs-t n g a zero
  fs'      : ∀ {l : N}
           → (b : Fin n) -- 含まれる頂点
           → (b-p : path-t n g a b)
                    -- b が持つ最短経路
           → (rest : fs-t n g a l)
           → in-fs? b-p rest ≡ true -- (a)
           → saitan-fs? g b-p       -- (b)
```

```
          → fs-t n g a (suc l)
```
確定リストは、空リストとして empty-fs、fs に頂点を付け加えたものとして fs' の二つのコンストラクタでできている。(a) は、パス b-p が頂点 b 以外は全て rest に含まれている頂点から成ることの証明である。(b) は、パス b-p がグラフにおいて最短である証明である。このように、確定リストにおいて最短である証明を持つことで、アルゴリズムが終了した段階で証明を持った最短経路が得られていることになる。

未確定リストの型定義は以下の通りである。

```
data ts-t {n l : N} {a : Fin n}
          (g : graph n) (fs : fs-t n g a l)
           : N → Set where
  empty-ts : ts-t g fs' zero
  ts'      : ∀ {m : N} (rest : ts-t g fs m)
             → (v : Fin n) -- 含まれる頂点
             → (v-p : path-t n g a v)
                          -- v が持つパス
             → in-fs? v-p fs ≡ true -- (c)
             → saitan-ts fs v-p     -- (d)
             → identity-ts rest v   -- (e)
             → ts-t g fs (suc m)
```
未確定リストは、引数として確定リストを受け取り、空リストとして empty-ts、ts に頂点を付け加えたものとして ts' の二つのコンストラクタでできている。未確定リストも確定リストと同じく証明を持っている。(c) は、パス v-p が頂点 v 以外は全て引数で受け取った確定リストに含まれている証明である。(d) は、パス v-p が引数で受け取った確定リストにおいて最短である証明である。(e) は、rest に、頂点 v と等しい頂点は含まれていないことの証明である。

4 証明

証明について説明する。本研究では、アルゴリズムの証明を internal に行うため、アルゴリズムを行う関数内で同時に証明も行っている。2 章より、アルゴリズムを行う関数も大きく二つに分けて説明する。以下、関数について説明するが、関数内における証明には p1 などの変数名を付けている。

まず、未確定リストからパスの長さが最短である頂点を分離する関数について説明する。以下、この関数を分離関数と呼ぶ。分離関数は、未確定リスト ts を受け取ったら、含まれる頂点について一つずつパスの長さを調べ、最短の頂点 shortest と、それを削除した未確定リスト rest-ts を返す。しかし、後に確定リストへ shortest を移動することになるが、その際 shortest が持つパスがグラフにおいて最短であることを証明する必要が出てくる。そのために必要な証明をこの関数でも行うことになるが、それは後で説明する。

次に、分離された頂点から繋がった未確定リストの頂点についてパスを更新し、分離された頂点を確定リストに追加する関数について説明する。以下、この関数を更新関数と呼ぶ。更新関数の型は以下の通りである。

```
koushin : {n l l' : N} → (g : graph n)
          → (a : Fin n) → (v : Fin n)
          → (v-p : path-t n g a v)
          → v-p-pf → (fs : fs-t n g a l)
          → (ts : ts-t g fs l')
          → p1 → p2 → p3 → p4
          → (k-ts : ts-t g (add v fs) l')
             × p1' × p2'
```
更新関数では、未確定リスト ts の頂点がもともと持っているパスの長さと、分離点 v が持つパスに ts の頂点を加えたパスの長さを比較して、短い方に更新した未確定リスト k-ts を返し、add を使って v を確定リスト fs に追加する。v-p-pf は、v のパス v-p について成り立つ証明を表している。3 節より、確定リストに頂点を追加する際には、その頂点が持つパスがグラフにおいて最短であることを証明しなければならない。そのため add においてその証明を行う。証明は、2 節の内容と同じことを行う。この時、任意のパスを、初めて fs に含まれない頂点になる部分で分割するが、任意のパスが必ず分割できることを保証するために、始点が fs に含まれている証明と、v が fs に含まれていない証明が必要となる。これは、関数の引数の p3、p4 で受け取っている。さらに、v が fs に含まれていないことを示すためには、ts と fs に共通する要素がなく、かつ全ての頂点が ts、もしくは fs に含まれているか v でなければならない。この証明は、fs に含まれていなければ ts に含まれているか v である証明とその逆として、関数では p1、p2 として与えられている。この証明を行うため、未確定リストにおいて頂点が重複しない証明を持たせている。頂点が重複しないことは、紙で証明を行う場合は暗黙で成り立っているものだが、Agda は任意の未確定リストについて証明を行うので、頂点が唯一であることも証明として与えなければならない。また、更新関数によって確定リスト、未確定リスト共に型が変化するため、p1、p2 の証明も更新しなければならないので、更新して p1'、p2' を返している。

分離関数でも、証明を行う必要がある。ts と fs に共通する要素がない証明は、分離関数でも更新する必要があるので、それを行っている。

上記の関数は、証明部分を抜かせば単にアルゴリズムの計算を行う関数であることがわかる。これが、internal に証明するということである。

5 まとめと今後の課題

ダイクストラのアルゴリズムのメインループの正当性について、Agda による証明を完了させた。初期状態と、初期状態における確定、未確定リストに共通部分がない証明、始点が確定リストに含まれてる証明を渡せば、最短である証明の付いたパスを返す事ができるようになった。ここまでの実装で、コードは 2830 行である。

今後は、始点のみが含まれる確定リスト、それ以外の頂点が含まれる未確定リストを作り、初期条件の証明を作り、証明付きのアルゴリズムを完成させたい。

参考文献

[1] Jing-Chao Chen. Dijkstra's Shortest Path Algorithm. *Journal of Formalized Mathematics*, 15:9 pages, 2003.

[2] J Strother Moore and Qiang Zhang. Proof Pearl: Dijkstra's Shortest Path Algorithm Verified with ACL2. *Theorem Proving in Higher Order Logics*, LNCS 3603:373–384, 2005.

[3] 石畑清. **アルゴリズムとデータ構造**. 岩波書店, 1989.