Technical Reports on Information and
Computer Science from Kochi
Vol. 7 (2015), No. 3

システム記述言語Goによるファジィライブラリの実装

片山 淳貴,　森 雄一郎

高知大学 理学部 応用理学科 情報科学コース

概要

本研究ではファジイ論理を従来のコンピュータプログラミングで扱えるようにする為のファジィライブラリを，システム記述言語 Go によって実装した．ファジィ論理は 2 値論理と異なり，曖昧さが扱えるため，曖昧さを持つ現実世界の問題を解決する為に用いられている．本研究では並行計算，ガベージコレクションおよび第一級オブジェクトとしての関数などに対応した Go 言語で実装することで，先行研究よりも表現の精度や処理速度において優れた結果を出すことが出来た．

1　研究目的

本研究の目的はファジィ論理に基づいたプログラミングを行う為に使用するファジィライブラリを，システム記述言語 Go によって実装することである．ファジィ論理は曖昧さを扱うことが可能で，2 値論理とは異なった論理体系であり，現実世界の問題を扱う場面で多く利用されている．なぜなら現実世界は曖昧で不確実な事象が多く，従来の 2 値論理では上手く扱えない為である．

ファジィ論理によってプログラミングを行うには，既存のプログラミング言語では不十分である．なぜならファジィ論理において重要なファジィ集合やメンバーシップ関数といった要素を新たに表現する必要があるからである．その方法の一つにファジィライブラリという形があり，先行研究ではシステム記述言語として主流な C 言語を拡張する目的でファジィライブラリが作成された．

本研究でもファジィライブラリを実装することで既存のプログラミング言語を基にして，ファジィプログラミングを行う．しかし先行研究とは異なり，実装対象言語として Go 言語を選択した．新たに C 言語から Go 言語に変更したのにはいくつかの理由がある．その中でも重要な点として，並行計算に対応したシステム記述言語であり，近代的なプログラミングの考え方を取り入れていることがある．

現在のコンピュータは CPU のマルチコア化によって高速化している．そのためプログラミングの際には並列処理によって，マルチコア CPU の速度を活用することが求められている．Go 言語は並行計算を手軽に行うことが可能であり，ガベージコレクションやクロージャなどの機能も備えている為，新たにファジィライブラリを実装する言語として適していると考えた．本研究では Go 言語によって実装することで，先行研究に対して優れたファジィライブラリを目指す．

2　ファジィ論理

ファジィ論理は曖昧さを許容可能で，2 値論理とは異なる論理体系である．2 値論理が真か偽のどちらかに決めてしまうのに対して，ファジィ論理では「どの程度まで真と言えるか」という曖昧さを含む幅を持つことが出来る．

2.1　ファジィ集合とファジィ真理値

一般的な集合ではある条件に一致するかどうかによって，集合の要素に含まれるか否かを決めている．例えば集合 $A = \{x \subset N | x \text{ は奇数} \}$ という，集合の内包的表記を使えば，自然数上のすべての奇数が含まれた集合 A を定義することが出来る．この集合 A には偶数は一切含まれていない．このように数学的な概念ならば，含まれるか含まれないかという 2 値論理で十分である．

それに対して現実世界には曖昧な概念の方が多く，どちらか一方に決めることは難しい．また主観的な概念であれば，ある人が決めた集合と別の人が決めた集合の定義は異なるかもしれない．そのような曖昧で不確実なものを扱う集合がファジィ集合 [1] である．

ファジィ集合は，集合に属する要素とその真理値から成る．この真理値はある集合に属する度合いのことであり，ファジィ真理値と呼ばれている．このファジィ真理値は 0 ～ 1.0 の実数値を取る．例えばある要素が集合に完全に属していれば 1.0，中程度に属していれば 0.5，まったく属していなければ 0 というファジィ真理値を与える．このファジィ真理値によって，2 値論理とは異なり，曖昧な概念を扱うことが出来る．

2.2 メンバーシップ関数

ファジィ集合は，集合の要素とそのファジィ真理値から成るが，そのファジィ真理値を要素に与えるのがメンバーシップ関数である．ファジィ集合 A を定義するメンバーシップ関数は，全体集合 U の各要素に対して 0 ～ 1.0 のファジィ真理値を割り当てる関数で，

$$\mu_A : U \to [0, 1]$$

と記される．

2.3 ファジィ推論

従来の論理学で行われて来た推論というのは，事実と命題の積み重ねから，結論を導きだすものである．ここでいう命題とは「$A \to B$」のように「A であれば B である」というような形で記述され，この場合「A と B という事実は互いに等しい」という結論が得られる．このような推論を行うものを記号論理という．この記号論理の推論から得られる結論は絶対に正しいものであるが，現実的には A と B は完全に一致せず，この論理が成り立たないことも多い．このように従来の記号論理による推論は，曖昧さを持った現実の問題に対して行うには充分ではない．よって考えだされたのが，命題に曖昧さを許容したファジィ推論である．

2.4 Mamdani 型推論

ファジィ推論を行うモデルの一つに Mamdani 型推論 [2] がある．Mamdani 型推論では以下のような推論過程によって最終的な結論を得る．

1. 任意個のルールを定義する
2. それぞれのルールを処理してそれぞれの結論を得る
3. すべての結論を 1 つに統合して最終的な結論とする

この過程によって得られる最終的な結論はメンバーシップ関数である．メンバーシップ関数のままでは，制御などには使えないので，通常はメンバーシップ関数からある代表値を得て，その数値を結論とする．そのためメンバーシップ関数に対して脱ファジィ化という操作を行うステップが，最後に追加される．

2.4.1 ルールの定義

ルールは記号論理の命題と同じく「もし A ならば B」という形であり，違いはそれらがメンバーシップ関数なことである．例えば室温に応じて空調を制御するならば次のようなルール群が考えられる．

1. もし室温が低いなら出力を上昇
2. もし室温が適度なら出力を維持
3. もし室温が高いなら出力を下降

この「もし前件部なら後件部」という形式を if-then ルールと呼ぶ．また，前件部と後件部はそれぞれ「概念」と「性質」から成る．例の場合の概念とは「室温」と「出力」であり，性質とは「低い」や「上昇する」などである．これら概念と性質を「概念 is 性質」の用に記述すると，次のようになる．

1. if 室温 is 低い then 出力 is 上昇する
2. if 室温 is 適度 then 出力 is 維持する
3. if 室温 is 高い then 出力 is 下降する

このように記述された 1 つ 1 つをルールと呼び，任意個数のルールの集まりをルールベースと呼ぶ．

2.4.2 ルールの処理

先のように記述したルールを処理して，まずはそれぞれのルールから個別の結論を得る．そのために Mamdani 型推論では「頭切り」という方法を用いる．この頭切りによって Mamdani 型推論は推論過程が簡略化され，判り易いものとなる．頭切りというのは，前件部で得たファジィ真理値によって，後件部のメンバーシップ関数の最大値を制限してしまうことに由来する．前件部のメンバーシップ関数 μ_A に値 x_A を入力するとファジィ真理値 $y = \mu_A(x_A)$ が得られる．このファジィ真理値 y によって，後件部のメンバーシップ関数 μ_B の最大値を制限，つまり $\mu'_B(x_B) = \min(\mu_B(x_B), y)$ とする．この μ_B を頭切りした $\mu_B{}'$ が 1 つのルールを処理して得られる結果である．

2.4.3 結果の統合

ルールベースは任意個のルールから成る．そしてそれぞれのルールは常に評価され，何らかの結果をメンバーシップ関数として返す．そのため複数の結果を一つに統合する必要がある．一つのルールベースに属する各ルールは，後件部がどれも同じ概念のため，ルールの評価によって頭切りされたメンバーシップ関数を，同じ集合の要素群に重ね合わせることが可能である．このとき重ね合わせによって要素は 2 つ以上のファジィ真理値を割り当てられる場合がある．その場合には割り当てられ得るファジィ真理値の中で最も大きいものを選択する．つまり重ね合わせるメンバーシップ関数郡 $\mu_1, \mu_2, \ldots, \mu_n$ がある場合，統合されたメンバーシップ関数 $\mu = \max(\mu_1(x), \mu_2(x), \ldots, \mu_n(x))$ となる．この統合されたメンバーシップ関数が推論の最終的な結論である．

2.4.4 脱ファジィ化

ファジィ推論の最終的な結論はメンバーシップ関数である．しかしメンバーシップ関数のままでは，機械制御などに使うのが難しいので，メンバーシップ関数を代表する一つの数値が求められる．そのための操作が脱ファジィ化であり，方法として様々なものがある．その中でも「重心法」という方法がよく知られている．脱ファジィ化するメンバーシップ関数を μ，その要素を x として，

$$center = \frac{\int_{min}^{max} x\mu\,(x)\,dx}{\int_{min}^{max} \mu(x)dx}$$

の式で重心法による脱ファジィ化ができる．なお通常は連続的なメンバーシップ関数を適度にサンプリングすることで，数値積分として求める．

3 Go 言語

Go 言語 [3] は 2009 年から Google が開発しているプログラミング言語である．システムプログラミングを目的の一つとし，並行計算に対応している．

3.1 特徴

Go 言語の目標はコンパイルが速く，並行計算やガベージコレクションを持つシステム記述言語である．その目標のためにシンプルさを重視し，C 言語と構文は完全には一致しない．Go 言語では開発速度も重要視しており，複雑さにつながるオブジェクト指向におけるクラスの概念がなく，予期し難いエラーを生み出す型変換を行わない強い静的型付けの言語である．

3.2 並行計算

Go 言語では並行計算のモデルに Communicating sequential processes(CSP)[4] を採用している．これは並行計算機能の中でも代表的なマルチスレッドによるプログラミングが複雑で，デッドロックなどの深刻なエラーを起こす危険性を持っているため，他の並行計算モデルに基づくことが求められているからである．

CSP では，処理を担うそれぞれのプロセス間での通信によってメモリを共有し，同期を行う．これは各プロセスが処理を完了したことや待ち状態にあることなどをメッセージパッシングによって通知することを意味している．また，あるプロセスが処理したデータを別のプロセスが処理しなければならない場合に，メモリでデータを共有するのではなく，処理したいデータ自体を別プロセスに送信することで，データを受け取

るプロセスは受信完了まで待ち状態を維持するため，安全かつ簡単に実行順番を決定することが出来る．

3.3 並行計算のための機能

Go 言語は言語仕様レベルで並行計算機能を備えている．先述したように Go 言語における並行計算モデルの基本となるのは CSP であり，それを実現するための機能が備わっている．

3.3.1 goroutine

goroutine はスレッドよりも軽量に実装された処理単位であり，呼び出しコストは小さく抑えられている．生成した goroutine はメイン関数とは並行して実行されるが，処理終了を通知する機能はない．そのためメイン関数は他の goroutine の終了を待つことはなく，main 関数が終了した時点で他の goroutine は終了を待たずに破棄される．よって goroutine 間の同期を取ることが必要になるが，そのための機能が後述する channel である．

3.3.2 channel

channel はバッファ付きのメッセージキューであり，goroutine 間で通信を行うためのデータ構造である．channel の使用方法としては，channel からの受信と channel への送信があり，両方へ状態に応じたブロックを行う仕組みがある．

受信では指定 channel にデータがない場合は，データが channel に送信されるまで受信をブロックする．送信では指定 channel のバッファを超えるデータを送信する場合は，channel からデータが受信されてバッファが空くまで送信をブロックする．

バッファを指定しない channel はデータを 1 つだけ保持することができ，このバッファなし channel を 2 つの goroutine に渡すことで，goroutine 間の同期を取ることができる．例えば $goroutine_A$ と $goroutine_B$ があるとき，$goroutine_A$ の処理を終えるまで $goroutine_B$ が待たなければならない場合，$goroutine_A$ は処理が終わったら channel に送信し，$goroutine_B$ は channel から受信するようにすれば，$goroutine_A$ が送信するまで，$goroutine_B$ は待機する (図 1)．

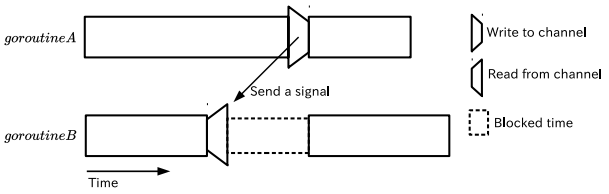

図 1: channel による同期

4 ファジィライブラリ

多くのファジィライブラリがファジィ制御を対象としている．そのため本研究でもまずはファジィ制御を行うためのファジィ推論を対象とし，それに関する機能を先行研究と同等になるように実装した．

4.1 先行研究との違い

Fuzzy system Description Language(FDL) は 1995 年に提案および実装 [5] された，ファジィ制御用の構文とその処理系である．この FDL を本研究では先行研究として参考にしている．

FDL は C 言語で実装された為にメンバーシップ関数の内部表現は配列だった．これは C 言語が第一級オブジェクトとして「関数」を扱えないことに起因する．Go 言語では「関数」は第一級オブジェクトなので，数値型や文字列型と同じように関数の引数や返り値として使用できる．配列で実装した FDL においてメンバーシップ関数は離散的に保持されていたが，Go ファジィライブラリにおけるメンバーシップ関数は，すべての実数上の要素 $x \in \mathbb{R}$ で定義され区間 $(-\infty, \infty)$ で使用できるようになった．そのため配列での実装に比べると，メンバーシップ関数は正確さが向上した．

4.2 メンバーシップ関数

ファジィ制御で使うメンバーシップ関数の形状は，基本的な幾何学図形や数学関数を利用することが広く提案されている [6]．本研究でもそれに従って，台形や三角形など 8 種類のメンバーシップ関数を定義できる．

4.3 ファジィ集合演算

ファジィ集合演算において実装したのは積集合，和集合および補集合である．この中で積集合と和集合はファジィ集合同士の演算であるが，これらは二項演算に限定して実装している．なぜならばファジィ集合の演算では結合法則が成り立つためである．よって 3 つのファジィ集合の積集合 $A \cap B \cap C$ ならば $(A \cap B) \cap C$ として，二項演算を 2 回行うことで表現できる．同様に任意の数の積集合や和集合を二項演算のみで表現できる．

積集合と和集合には代表的な 4 パターンの演算を定義してあり，補集合と合わせて以下の演算を行う関数を定義している．

1. 論理積・論理和：LogicalAND, LogicalOR
2. 代数積・代数和：AlgebraicAND, AlgebraicOR
3. 限界積・限界和：BoundedAND, BoundedOR
4. 激烈積・激烈和：DrasticAND, DrasticOR
5. 補集合：Complement

4.4 推論

推論のためのルール 1 つは前件部と後件部から構成され，前件部が条件，後件部が結論を示している．

まず前件部は基本式と複合式に分かれる．基本式とはメンバーシップ関数とその入力変数の組であり，複合式とは基本式同士，複合式同士または複合式と基本式を二項演算で合わせたものである．これら基本式や複合式は評価すると 1 つのファジィ真理値を返す．これがファジィ推論に使う前件部の評価値となる．

そして後件部は頭切りされるメンバーシップ関数である．単一のルールはこの前件部と後件部を要素にもつ構造体として定義されている．このルールの任意個の集合がルールベースであり，配列として定義されている．

4.5 推論

推論はルールベースのもつ推論用メソッドを実行することで行われる．メソッドを実行すると，推論結果としてメンバーシップ関数が返される．このメンバーシップ関数は，個別ルールの評価をして得た結論を 1 つに統合したものである．このときルールベースに対する入力変数群に変化があった場合は，再びメソッドを実行しなければならない．

4.6 脱ファジィ化

脱ファジィ化には重心法を実装している．重心法を行う Centroid 関数はメンバーシップ関数，有効区間および区間の分割幅を引数に受け取る．このとき脱ファジィ化の精度は分割幅に応じて変化する．

また重心法を並列で行う PCentroid 関数も実装している．この PCentroid 関数は実行時に使用 CPU コア数を最大にして実行する．Go プログラムは実行コア数 1 が標準で設定されているが，この数値を変えることで並列で実行できるようになる．

5 研究成果

本研究で作成したライブラリの有用性の一側面として実行速度がある．比較対象は C 言語で実装された FDL および Go で実装された配列を用いたファジィライブラリである．

5.1 実装方式における速度の違い

まずは関数によってメンバーシップ関数を表現した本研究の方式と，配列によってメンバーシップ関数を表現した方式のそれぞれを Go 言語で実装したもの同士の比較を行う．

関数と配列のそれぞれによりメンバーシップ関数を実装した場合の計測結果が表 1 である．表において Initialize はメンバーシップ関数をメモリに確保するための時間，Inference はファジィ推論を行って推論結果となるメンバーシップ関数を返すまでに必要な時間，Defuzzification は推論結果のメンバーシップ関数を脱ファジィ化するために必要な時間である．ファジィ推論

表 1: 実装方式毎の実行速度

	Initialize	Inference	Defuzzification
function	4,717 ns	3,460 ns	8,069 ns
array	301,785 ns	18,521 ns	253 ns

が使われる場面において重要な数値は，推論時間とその脱ファジィ化に要する時間である．なぜならメモリへの確保は初回の 1 度で済む処理であり，以降は推論を行い，その値を制御などに使用するため脱ファジィ化する処理を繰り返すことになるからである．よって関数を用いた実装では 11,529 ns を，配列を用いた実装では 18,774 ns が比較対象とする時間である．結果として関数による実装は配列による実装に対して，61.4%の時間で終了することができた．

5.2 実装言語における速度比較

C 言語で実装された FDL によって書いたプログラムと，Go 言語で書いた新たなファジィライブラリで書いたプログラムのそれぞれで実行速度を比較する．FDL ではメンバーシップ関数を配列によって実装しているが，通常は同じプログラムを C 言語と Go 言語のそれぞれで実装した場合は C 言語の方が速く動作する．そのため言語間での速度差を計測する．計測は仮想マシン上に表 2 の設定で環境を構築し，OS 付属の time コマンドを用いて測定した．

表 2: 実行環境

仮想化マシン	VirtualBox 4.3.8 on MacOS X
OS	Ubuntu 12.04 (32bit)
RAM	1024 MB
CPU	2.6 GHz Intel Core i7
Go コンパイラ	go 1.4
C コンパイラ	GCC 4.7

計測結果は表 3 のようになった．使用した time コマンドにおいて，real の値が処理の開始から終了までの時間であり，C 言語での実装に比べて Go 言語での実装は 69.4%の時間で終えることが出来た．また user は加算，減算やメモリアクセスなどにかかった時間であり，system は OS のシステムコールを利用した時間である．

表 3: C 言語と Go 言語の実行速度比較

	real	user	system
C	12.338 sec	11.835 sec	0.0064 sec
Go	8.559 sec	4.740 sec	3.682 sec

5.3 並列化

重心法による脱ファジィ化の処理は数値積分を基にしているため，並列化が容易である．本研究のライブラリには Centroid 関数と PCentroid 関数として，逐次処理と並列処理のそれぞれを実装している．これは問題規模が少ない場合には並列化のためのオーバーヘッドの影響が大きく，並列処理の方が遅くなる場合があるという研究 [7] があるためである．

表 4: 並列化の効果計測

CPU's core	$n = 10^2$	$n = 10^3$	$n = 10^4$
1	9,815 ns	94,392 ns	932,886 ns
4	13,521 ns	41,112 ns	284,650 ns
before/after	0.726	2.296	3.277

並列化の効果を測定する為に問題規模ごとに 1 コアを使用する Cnetroid 関数と，4 コアを使用した PCentroid 関数を使い脱ファジィ化を 1 度行う時間を求めた．その結果が表 4 である．

これにより問題規模 $n = 10^2$ では逐次処理の方が速く，$n = 10^3$ より並列化の効果があった．4 コア CPU を用いた実行環境では並列化による高速化率は最大 4 である．$n = 10^4$ で高速化率は 3.277 となり，理想値に近づくことがわかった．

6 まとめ

本研究では C 言語で実装された FDL と比較して，Go 言語で再実装することで速度面など優れた結果を残すことが出来た．しかしファジィライブラリはまだまだ対応すべきファジィ理論の分野を多く残している．現在はファジィプログラミングの代表的な課題であるファジィ制御に対応したが，あくまで基本的な範囲においてであり，より複雑なメンバーシップ関数を用いた推論が行えるように拡張する必要もある．

これからはより複雑な応用分野に対して Go 言語で取り組んでいく．

参考文献

[1] Lotfi Zadeh. Fuzzy sets. *Information and Control*, Vol. 8, pp. 338–353, 1965.

[2] Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the Institution of Electrical Engineers*, Vol. 121, pp. 1585–1588. IET, 1974.

[3] Inc. Google. The go programming language. http://golang.org/.

[4] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, Vol. 21, No. 8, pp. 666–677, 1978.

[5] 鑓田威, 石畑清. ファジィシステム記述言語の試作処理系. *11th Fuzzy System Symposium*, pp. 23–26, 1995.

[6] INDUSTRIAL PROCESS MEASUREMENT and CONTROL. Fuzzy control programming. Technical report, INTERNATIONAL ELECTROTECHNICAL COMMISSION, 1997.

[7] Peiyi Tang. Multi-core parallel programming in go. In *Proceedings of the First International Conference on Advanced Computing and Communications*, pp. 64–69, 2010.