１、＊ はじめに ＊

この特別研究は、八杉満利子教授の計算機数学1.2という授業を基にソフトウェア仕様記述の先進技法であるＺ言語をまとめました。

今回はＺ言語を知ってもらうための説明とそれによっていろいろな制約されたシステムを記述するのに用いるスキーマと言うものについて解説します。

これらを学習した動機は、計算機数学1.2の授業で主に出てきた数学的論理記号とそれを用いたスキーマについての理解を深める事と、今後関連する科目を学習される方々への手助けになるようなものを作りたかった為です。

2、＊ 仕様記述 ＊

仕様記述とは、ソフトウェアシステムを外側から見たときに、そのシステムが実際にどのように動くのかを定義する過程の段階のことをいいます。

システムの内部的な働き、いかにして実現させるかは正しい仕様記述を行った後にすべきことであり、始めにすべきことは、このシステムには

何を備えなければならない

かを記述することなのです。

この仕様記述を明確に示すことによって、仕様の段階で見られる、曖昧な表現や欠かすことのできない詳細なことをチェックすることができ、

後の設計 時点での誤った判断 をなくすことができます。

ソフトウェアのシステム開発におけるエラーの大半は初期段階での不完全さにその原因があります。

つまり、仕様定義と設計の段階に十分な時間と労力を割かないことによって引き起こされているのです。

初期の段階でしっかりとした技術を使用し、多くの時間と労力をかければ、

初期段階のエラーの発生率を抑え、それが全体のエラーの数を減らすことは明らかなことです。

このことから仕様記述の重要性を分かって頂けると思います。

ソフトウェアのライフサイクル

1、要求分析

2、システム仕様定義

3、設計

4、実装

5、テストと統合

３、＊ 形式的仕様記述 ＊

一般的に用いられている自然言語と図や数学的技法などで補った仕様は分かりやすいが、曖昧さ、過不足などを含みやすい。

また、大きな仕様はよく変更を伴うが、これに対応できない部分がでてくる。

形式的仕様記述とは、数学的な厳密さをもつ記法に基づいて書いたソフトウェアの仕様記述のことです。

これを使用することによって、さきほど述べた、問題を解決する手段が与えられます。

また、仕様の正しさの証明、実行可能なプログラムへの詳細化への手段を得る可能性が期待されてもいる。

形式的仕様記述はシステムが、その仕様をみたすことについての信頼性を高めための手段ということになります。

今、この記述法の必要性の認識が高めるのにつれて、様々な形式的記述言語が開発されてきている。

これらのいくつかは、集合論と数学の概念をその基礎とし、システムの仕様記述のための言語として使われている。

そして、その中で最も広く知られているのがZ言語なのです。

４、 ＊ Ｚ言語 ＊

Zにより書かれた仕様記述は、数学的記述と非形式的な説明の文からなります。

数学的記述は仕様で定義されるシステムの正確な表現を与え、非形式的な説明の文はドキュメント（注釈）をより読みやすく把握しやすくし、数学的な記述を実世界に

むすびつけます。

Zの構成要素は集合論にその基礎をおいています。

集合論で成り立っている世界においてZでは対象物は

異なる種類に分類でき、異なる種類の間では重複はないという考えを加えています。

その対象物の種類を１つの 型 として考えます。

そして、次に重要なZの特徴は、スキーマという図表記法です。

これによって、対象物の複数の記述は、１つの名前のついた単位にまとめられ、

仕様記述全体を参照することができます。

これから、この型とスキーマの具体的な説明に入りたいと思います。

集合論の基本的概念

$$\mathrm{EEC} == \{Belgium, France, WestGermany,$$

$$Italy, Luxembourg, Holland, Denmark,$$

$$Greece, Ireland, UnitedKingdom, Spain, Portugal\}$$

$$NATO == \{Belgium, Canada, Denmark, Iceland, Italy,$$

$$Luxembourg, Holland, Norway, Portugal, UnitedKingdom,$$

$$UnitedStates, Greece, Turkey, Spain, WestGermany\}$$

$Scandinavia == \{Denmark, Finland, Norway, Sweden, Iceland\}$

$Benelux == \{Belgium, Holland, Luxembourg\}$

これより、以下の集合は

$EEC \backslash NATO == \{Finland, Swede\}$

$EEC \cup Benelux == \{Belgium, France, WestGermany,$

$Italy, Luxembourg, Holland, Denmark,$

$Greece, Ireland, UnitedKingdom, Spain, Portugal,$

$Canada, Iceland, Norway, UnitedStates, Turkey\}$

$EEC \cap NATO == \{Belgium, Denmark, FranceItaly, Luxembourg,$

$Hollan, Portugal, UnitedKingdom, Greece, Spain,$

$WestGermany\}$

になる。

5、＊ 型について ＊

Z においては対象物の種類を１つの型として考えると説明しました。

この型の考え方により、種類というものの非形式的概念を形式化することになります。

ここから先は種類という言葉ではなく、型という言言葉を用いることにします。

最初に、なぜ型を用いるのか。Z における型の体型系より、型 (種類) を制限する

と２つの利益を得ることができます。

型の体型は仕様書に含まれている単純なエラーやより根の深い問題点を

発見することのできる有用なチェック機構を提供できること。

もう１つは制限をすることによって仕様書の望ましい構造を作ることができ、

書き方の指針を与えてくれます。

では、どのようにして型を用いるのか。

型体系の下では、新しい変数の導入のときに、必ずその変数が属する型の情報と

ともに宣言をしなければならない。

その宣言によって変数が取ることのできる範囲が決まります。

x： Book

これは型が Book であるような変数 x を宣言している例です。

このことのよって変数 x が取ることのできる範囲が型 Book によってさだめられます。

６、＊ 集合表記の定義について ＊

まず、集合において”述語による集合の定義”とはその要素を特徴付ける性質を与えることにより集合を定義するものである。

例として次の例を取り上げてみると

BigCountries== $\{c : country \mid c\ has\ more\ than\ 40\ million\ inhabitants\ \}$

これは『$''c : country|''$なる上の集合は４０００万人以上の住人がいる」と

読むことができる。

“|” の後に続くのが述語であり、定義される集合は真となる命題を生じるものである。

また、述語を書くときには、宣言とともに宣言される変数にさらに制約を加えることもできる。

例として、

$\forall x, y :$ Z $|x > y \bullet x - y \in N$

これは$x > y$を満たす全ての整数xとyについて$x - y$はNに属すると

読むことができる。

ちなみに、ここでのＺは整数型であることを意味し、Nは自然数であることを

意味する。

$x > y$という付け加えられた制限式はxとyが満たされなければならない

もう１つの制限であることに注意してほしい。

ここまでは集合の定義として次の例に示すような宣言文と制限文と

いう形を見てきた。

$\{n : \mathsf{Z} \mid n > 0\}$ は整数型をもつ、0 より大きい整数の集合である。

$\{i : \mathsf{Z} \mid \exists j : \mathsf{Z} \bullet i = (j + j) - 1\}$ は

整数型をもつある j について $i = (j + j) - 1$ を満たす集合である。

どの例においても宣言はただ１つであるが、実はいくつかの宣言を

並べることもできる。

宣言が並んでいるときには集合の要素は全ての値が集合の述語を満たすような宣言

の順序による順序付き組のリストになる。

例えば、$\{i : N;\ s;\ \mathsf{P}\, N | s = \{i\} \wedge i < 4\}$

は i は 4 より小さい自然数とその部分集合の集合であるよって

$\{(0, \{0\}), (1, \{1\}), (2, \{2\}), (3, \{3\})\}$ となる。

また、集合の要素を宣言のリストに対応した組にするのではなく、

Ｚの集合表現では集合の要素を表す項を付け加えることによってより

幅広い集合表現を定義できるようになった。

つまり従来集合表現としては $\{Decls(\text{宣言})|Pred(\text{述語})\}$

という形だったが、

これからＺでは $\{Decls(\text{宣言})|Pred(\text{述語}) \bullet Expr(\text{式})\}$

と定義することにより、より解りやすくなった。

例をあげると、$\{n : N | n < 4 \bullet n^2\}$

は $n < 4$ を満たす $n : N$ を考え、

その n の 2 乗からなる集合であるので $\{0, 1, 4, 9\}$ となる。

ちなみにこの集合の要素の型は集合の定義における式の型

であることに注意してほしい。

７、＊ スキーマ表記について ＊

大規模な仕様書を作成する際に、従来は原始的な集合を利用して仕様書を定義してきたがそれにより大きなスペースが必要となってくる。

それを解決するために型体系によってそれぞれの変数を区別することにより従来より短く定義できるようになったが、それにより仕様書中での宣言の順序が入れ代わると意味が変わってしまうという新たな問題が生じてきた。

そこで仕様書自体を幾つかに分けて定義することができればより見易くでき、かつ簡単にできる。

ここでスキーマと呼ばれる新しい構築要素を採用することにする。

この方法の１つがＺのスキーマである。

スキーマ表記の説明として１つの例を取り上げて進めていきたいと思います。

まず、スキーマ表記において

スキーマの名前は箱の上辺の中に記述すること。

中央の横線はスキーマの宣言部と述語部を仕切るものである。

スキーマの中の改行はアンドの意味を持っている。

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Library}} \\
\hline
stock : Copy \rightarrow Book; \\
issued : Copy \rightarrow Reader; \\
shelved : FCopy \\
readers : FReader \\
\hline
shelved \cup dom\ \ issued = dom\ \ stock \\
shelved \cap dom\ \ issued = \varnothing \\
ranissued \subseteq readers \\
\forall\, r : readers \bullet \sharp(issued \rhd \{r\}) \leq maxloans \\
\hline
\end{array}
$$

Librayスキーマの１行目の宣言文はstockは集合Copyから集合Bookへの部分関数の変数で前で説明した型定義の表現の１つであり、これは在庫にある本にはそれぞれの本のコピーに対してコピー識別子が存在することを意味している。

ここで、それぞれのコピー識別子は最大１つの本に対応し、また通常は使用されていないコピー識別子が多くあるので、stockは部分関数になる。

issuedは発行という意味で同様に２行目はそのコピー識別子は利用者が本を借りると渡されることを意味している。

ちなみに、ある任意の時点において全てのコピーが貸し出し中というわけではないのでissuedも部分関数になる。

ここでissuedの定義域はstockの定義域の部分集合である。

３行目はshelvedという変数は集合Copyの有限部分集合で、shelvedは棚に置くという意味で、これはある本がまだ本棚においてあり、貸し出し可能なコピーの集合の部分集合であることを意味している。

同様に４行目は登録された利用者の集合を意味している。

次に述語部の１行目は shelved と issued の定義域の和集合は stock の定義域であると読み、これは在庫にある全てのコピーは貸し出し中かまたは本棚にあるかどちらかにあることを意味している。

同様に２行目は shelved と issued の定義域の積集合は空集合であると読み、これは本棚にあることと本のコピーが貸し出し中であることは同時には起こらないということを意味している。

３行目は issued の値域は readers の部分集合であると読み、これは貸し出しは登録された利用者にのみ行われることを意味している。

４行目はすべての r という変数は readers の型をもち定義域 issued に対する 値域変数 r のシャープは maxloans より小さいと読み、

$\sharp$ は数を表し、$\rhd$ は定義域制限演算子と呼ばれ、

ここでは２つの集合 $issued$ と $issued$ を制限する集合の間において使われ、

２つの関係を表すのに使われている。

これは登録された利用者のそれぞれへの貸し出し冊数は最大許容貸し出し数を越えてはいけないということを意味している。

本を登録された利用者に貸し出す操作という事柄をIssueというスキーマにし、宣言と状態の事前事後の構成要素に対する制約と共に加えると次の様に定義できます。

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Issue}} \\
\mathit{stock}, \mathit{stock}' : \mathit{Copy} \longrightarrow \mathit{Book} \\
\mathit{issued}, \mathit{issued}' : \mathit{Copy} \longrightarrow \mathit{Reader} \\
\mathit{shelved}, \mathit{shelved}' : \mathsf{F}\ \mathit{Copy} \\
\mathit{readers}, \mathit{readers}' : \mathsf{F}\ \mathit{Reader} \\
\hline
c? : \mathit{Copy};\ r? : \mathit{Reader} \\
\mathit{shelved} \cup \mathit{domissued} = \mathit{domstock} \\
\mathit{shelved}' \cup \mathit{domissued}' = \mathit{domstock}' \\
\mathit{shelved} \cup \mathit{domissued} = \varnothing;\ \mathit{shelved} \cap \mathit{domissued}' = \mathit{set} \\
\mathit{ran}\ \ \mathit{issued} \subseteq \mathit{readers};\ \mathit{ran}\ \ \mathit{issued}' \subseteq \mathit{readers}' \\
\forall\, r : \mathit{readers} \bullet \#(\mathit{issued} \rhd \{r\}) \leq \mathit{qmaxloans} \\
\forall\, r : \mathit{readers}' \bullet \#(\mathit{issued} \rhd \{r\}) \leq \mathit{maxloans} \\
c? \in \mathit{shelved};\ r? \in \mathit{readers};\ \#(\mathit{issued} \rhd \{r?\mathit{ce} < \mathit{maxloans}) \\
\mathit{issued}' = \mathit{issued} \oplus \{c? \mapsto r?\};\ \mathit{stock}' = \mathit{stock};\ \mathit{readers}' = \mathit{readers} \\
\hline
\end{array}
$$

まず最初の囲いの宣言部ではコピーの部分関数である1冊の本のstock,stock’（在庫）、1冊の本を登録利用者に貸し出すissue, issue’（貸し出し）、コピーが集まっているshelved,shelved’（本棚）、利用者の集合であるreaders,readers’（利用者）、コピー識別子の変数をc?利用者の変数をr?と型宣言を入力する。

次の囲いは述語部を表しており、本棚のコピーと貸し出している本の和集合は在庫の定義域と等しくなる。それはダッシュが付いていても同じ事がいえる。（以後同様のことがいえる。）

本棚のコピーと貸し出しのコピー の定義域の共通部分 は空集合になる。貸し出しの値域は登録利用者に含まれる。すべての利用者について貸し出しするコピー冊数は許容最大冊数以下でなければならない 。貸し出すコピーは最初本棚にあり、利用者は登録されてなければならない。利用者にコピーを貸し出したことを記録する。在庫と登録利用者はどちらも変化はせず同じである。

次に、今までに説明してきた事を利用してスキーマRegister New ReaderとRemove Copy Stockを定義する。

Register New Readerは産大関係者が産大図書館を利用するための新規登録、Remove Copy Stockは図書館の在庫から破損した本を捨てるという操作である。

それでは、今までの説明で説明しきれなかった表記法なども説明しながら、２つのスキーマを定義していきます。

１.Register New Readerは

$RegisterNewReader$

$nr? : NewReader$
$i : ID$
$rep! : Report$

$nr? \notin Reader$
$nr? \in KSU$
$reader' = reader \oplus \{i \mapsto nr?\} \wedge nr? \notin banned$
$\exists\, i : ID \mid i \notin domReader$
$rep! = NewReaderAdded$

このスキーマは次のような宣言をしている。

１、新規登録希望者nr?は、型がNew Readerであるような入力オブジェクトを表している。

２、型がIDであるような変数iを宣言している。

３、語彙rep!はReportを型に持つ出力オブジェクトである。

ここで、入力オブジェクトには変数の語尾に?を付け、出力オブジェクトには変数の語尾に!を付けます。

Register New Reader の述語部は非形式的には次のように説明できる。

１、1 行目、変数 nr?は reader の要素ではない、は新規登録者はまだ登録されては

いない事を表している。

２、2 行目、入力オブジェクト nr?は KSU の要素である、は新規登録希望者は

産大関係者でなければならない事を表している。

３、3 行目、reader' は reader と、i は入力変数 nr に対応する集合の和集合であり、

かつ入力変数 nr は banned の要素ではない、は reader' は ID をもら った新規登録希望者

と現登録者の和である。ただし条件として新規登録希望者は貸出禁止者ではないこと

が条件である事を述べている。

ここで、記号$\oplus$は関数の和集合を表している。簡単な例を挙げると、

$f : \{0 \mapsto 1\}, \{1 \mapsto 1\}$

$g : \{2 \mapsto 1\}$

これに対してfとgの和集合は次のようになる。

$f \oplus g = \{0 \mapsto 1, 1 \mapsto 1, 2 \mapsto 1\}$

４、4 行目、あるIDという型をもつ変数iについて、iは$Reader$の定義域の要素では

ない、はまだ登録者に使われていない$IDno$がある事を示しています。

すなわち、新規登録者に与えられる$IDno$がまだ存在することを表しています。

５、5 行目 $rep!$は述語部分の条件を全て満たした場合、「新規登録完了」を

意味する$NewReaderAdded$を出力する。

以上で、$RegisterNewReader$の定義は終りです。

２、RemoveCopyFromStock は

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{RemoveCopyFromStock}} \\
\hline
\Delta \mathit{Library} \\
b : \mathit{Book} \\
\mathit{BC} : \mathit{BrokenCopy} \rightarrow \mathit{Book} \\
\hline
\mathit{shelved} \cap \mathit{dom issued} = \varnothing \\
\mathit{stock}' = \mathit{stock} \setminus \mathit{BC} \\
\mathit{readers}' = \mathit{readers} \quad \mathit{issued}' = \mathit{issued} \\
\hline
\end{array}
$$

このスキーマは次のような宣言をしています。

１、$\Delta Library$は先程の $Library$の宣言、述語及び変数を使用できる事を意味する。

ここで､Δは微分計算でδが小さな変化に対応して使用されるように

変化を示すために使用される。

任意のスキーマ $Library$に対して$\Delta Library$は普通つぎのように表される。

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Delta \mathit{Library}} \\
\hline
\mathit{Library} \\
\mathit{Library}' \\
\\
\hline
\end{array}
$$

これは、Library の全ての宣言とそこで宣言された変数に関係した述語、

及びこれに対応するダッシュ付きの変数の集まりの宣言とこれらの変数に関連した

述語が含まれていることを示している。

２、型が Book であるような b を宣言している。また BC は Broken Copy から Book

への部分関数を表している。

Remove Copy From Stoock の述語部は非形式的に次のように説明できる。

１、1 行目、 shelved と dom issued の積集合は、本棚にある本と貸出している本には共通部分が無い事を表している。

２、2 行目 stock’ は stock から BC の差である、は本棚から破損した本を捨てる事を表している。

３、3 行目は登録利用者と貸し出しは変化しない事を表している。

以上で Remove Copy FRom Stock の定義を終わります。

８、* 推論における証明　次に推論を利用してスキーマを証明し、より詳しい場合ではこのスキーマは正確に

働く分を見ていくことにする。

最初にフットボールファンのために全国識別カードを導入する。

ファンには１つの一意的な識別カードが割り当てられ、スキーマは誰に

どの識別カードを割り当てたかという記録を保存していて、またその試合を

応援することを禁じられている人のコードのリストも保持しているスキーマ Fid を

定義すると、

$$\begin{array}{|l}
\multicolumn{1}{l}{Fid} \\ \hline
members : ID \longrightarrow PERSON \\
banned : P\ ID \\ \hline
banned \subseteq dommembers \\ \hline
\end{array}$$

定理を簡単に述べるときには、初期状態が実際に我々のシステムに対して

存在しているという事実を記述するためにスキーマを使った方が、使用に含まれる

表現をずっと明確 かつ正確にすることができるのでこのシステムの初期状態を

定義する。初期状態では、どの会員もまだ登録されておらず、

また誰も禁止されてないので、

$$\begin{array}{|l}
\multicolumn{1}{l}{InitFid} \\ \hline
Fid' \\
member' = \varnothing \\
banned' = \varnothing \\ \hline
\end{array}$$

上記のように、少なくとも１つの適当な初期状態が存在することに関して

先に述べられた種類の定理を初期設定定理と言い、

$\vdash \exists\, Fid' \bullet InitFid$

今の例では、初期設定定理は単純でほとんど当たり前ともいうべき定理であるが成立するかどうかチェックすることは大切である。

初期設定スキーマは、状態の要素に明示的な値を与えることが多く、そのときには、状態要素が合接的に状態不変性を満足していることを通常は容易に確認できる。

このような定理の証明は、仕様が要求した特性を持っていることをどのようにして確かめられるか、という最初に示された質問に答えることから始まり、次の問題として仕様が要求しているとおりに、本当にプログラムが動作するかどうかを決定するというものである。

一般に証明は、仮定から出発して推論し、推論の各段階に対してその正当性を与え最終的には求められている結論に到達することによって構成される。

もう１つの推論スタイルとしては、事前条件の計算のように、複雑な表現を得て、その意味を保ったままその叙述を簡単化することが必要であるような状況である。

しかし、簡単化を実行するときには、新しい術語が古い述語と同値であることを確かめることで各段階の正当性が示されなければならない。

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\, AddMember \,\rule{6cm}{0.4pt}} \\
\Delta Fid \\
applicant? : PERSON \\
id! : ID \\
\hline
applicant? \notin ranmembers \\
id! \notin dommembers \\
members' = members \cup id! \rightarrow applicant \\
banned' = banned \\
\hline
\end{array}
$$

上記のスキーマより応募者がすでに登録されておらず、識別コードがすべて

利用されてない限りは、操作 AddMember が成功するであろうというように

見えるが、本当にそうなのかチェックするためには、この操作の事前条件を

計算する方法が必要である。

事後状態の変数と出力をスキーマの宣言部分から取り除き、述語内でそれらに

存在記号を付加することで、事前条件を見出せる。

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\, PreAddMember \,\rule{6cm}{0.4pt}} \\
Fid \\
applicant? : PERSON \\
\hline
\exists\, Fid!;\ id! : ID \bullet \\
applicant? \notin ranmembers \land \\
id! \notin dommembers \land \\
members' = members' \cup \{id! \rightarrow applicant?\} \land \\
banned' = banned \\
\hline
\end{array}
$$

このスキーマでは事前条件を記述しているが、Addmember の実行が成功することを

保証するが、存在記号の導入が述語をやたらに複雑にしてしまうことが多いのが問題である。そこで、スキーマの述語部分はずっと簡潔でかつ論理的には同値な叙述を与えるように簡単化できる。

より解りやすい命題として p $\wedge((q \Rightarrow \neg p) \vee q)$

を簡単化すると

$$
\begin{aligned}
& p \wedge ((q \Rightarrow \neg p) \vee q) \\
& \equiv (p \wedge (q \Rightarrow \neg p)) \vee (p \wedge q) \\
& \equiv (p \wedge (\neg q \vee \neg p)) \vee (p \wedge q) \\
& \equiv ((p \wedge \neg q) \vee (p \wedge \neg p)) \vee (p \wedge q) \\
& \equiv ((p \wedge \neg q) \vee false) \vee (p \wedge q) \\
& \equiv (p \wedge \neg q) \vee (p \wedge q) \\
& \equiv (p \wedge (\neg q \vee q) \\
& \equiv p \wedge true \\
& \equiv p
\end{aligned}
$$

次にスキーマ PreAddMember を簡単化をすると、

まず PreAddMember の述語部の Fid’ を展開して

$\exists\, members' : IDPERSON;\ banned' : P\ ID;\ id! : ID \bullet$
$banned' \subseteq dommembers' \wedge$
$applicant? \notin ranmembers \wedge$
$id! \notin dommembers \wedge$
$members' = members \cup \{id! \mapsto applicant?\} \wedge$
$banned' = banned$

$\Xi \exists\, id! : ID \bullet$
$members \cup \{id! \mapsto applicant?\} \in IDPERSON \wedge$
$banned \in \P\ ID \wedge$
$banned \subseteq dom(members \cup \{id! \mapsto applicant?\})e$
$applicant? \notin ranmembers \wedge$
$id! \notin dommembers$

$\Xi \exists\, id! : ID \bullet$ (Fidにより $banned \in PID$は保証されるので冗長である。)
$members \cup \{id! \mapsto applicant?\} \in IDPERSON \wedge$
$banned \subseteq dom(members \cup \{id! \mapsto applicant?\})e$
$applicant? \notin ranmembers \wedge$
$id! \notin dommembers$

$\Xi \exists\, id! : ID \bullet$ (定義域の特性)
$members \cup \{id! \mapsto applicant?\} \in IDPERSON \wedge$
$banned \subseteq dommembers \cup dom\{id! \mapsto applicant?\})edge$
$applicant? \notin ranmembers \wedge$
$id! \notin dommembers$

$\Xi \exists\, id! : ID \bullet$ (Fidにより $banned \subseteq dommembers$は保証されており冗長である。)
$members \cup \{id! \mapsto applicant?\} \in IDPERSON \wedge$
$applicant? \notin ranmembers \wedge$
$id! \notin dommembers$

９、＊ 最後に ＊

特別研究１の授業で初めてＺ言語を見た時は、単に論理記号の集まりであり

計算機数学の授業の延長であると思って取り組み始めました。

しかし実際その中身に触れて行くうちに、その応用性と奥深さに驚きました。

今回取り上げた図書館システム（LibSys）にせよ私たちの身近な物から、業界内で

多くの製品を取り扱うのに使用されているのを考えてもその事がよく解ります。

２年間この研究に携ったせいか、普段の生活のちょっとしたシステム

なら頭の中でスキーマと論理記号が浮かびでる様な気がします。

私たちの胸中ではある程度満足していますが、事実まだまだ奥があり

その先っぽしか齧ってないと思います。

けれど自分達でここまでやってこれたのも大変嬉しいことであり、

自信にもなりました。